Publikationsreihe des John von Neumann-Instituts für Computing (NIC)
NIC-Serie Band 35

John von Neumann-Institut für Computing (NIC)

Tatjana Eitrich

Dreistufig parallele Software zur Parameteroptimierung von Support-Vektor-Maschinen mit kostensensitiven Gütemaßen

NIC-Serie Band 35 ISBN 978-3-9810843-1-3

Zentralinstitut für Angewandte Mathematik

Die Deutsche Bibliothek - CIP-Einheitsaufnahme
Ein Titeldatensatz für diese Publikation ist bei
Der Deutschen Bibliothek erhältlich.

Herausgeber: NIC-Direktorium
Vertrieb: NIC-Sekretariat
Forschungszentrum Jülich
52425 Jülich
Deutschland
Internet: www.fz-juelich.de/nic
Druck: Graphische Betriebe, Forschungszentrum Jülich

NIC-Serie Band 35 ISBN 978-3-9810843-1-3

Dreistufig parallele Software zur Parameteroptimierung von Support-Vektor-Maschinen mit kostensensitiven Gütemaßen

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

am Fachbereich Mathematik und Naturwissenschaften der
Bergischen Universität Wuppertal
genehmigte

Dissertation

von

Dipl.-Math. Tatjana Eitrich

aus Görlitz

Tag der mündlichen Prüfung:	29.01.2007
Referent:	Prof. Dr. Bruno Lang
Korreferent:	Prof. Dr. Dr. Thomas Lippert

"*The prediction accuracy of employed data mining algorithms is of fundamental impact for their successful application. The computational time required for constructing a prediction model is becoming more important as the amount of available data is constantly growing.*" [93]

Vorwort

Fragestellungen der Extraktion wichtiger Informationen und interessanter Zusammenhänge aus Datenmengen, die aufgrund ihrer Größe und Unübersichtlichkeit für den Menschen nicht mehr überschaubar sind, gewinnen in der heutigen Zeit immer mehr an Bedeutung. Die Anzahl derartiger Daten wächst stetig an, wobei gleichzeitig immer höhere Anforderungen an die Güte der Analysen gestellt werden. Es sind genaue und effiziente Verfahren aus dem Bereich des Data-Mining zu entwickeln, um die tatsächlich relevanten Informationen in akzeptabler Zeit zu finden.

Im Rahmen dieser Arbeit beschäftigen wir uns mit einem modernen Verfahren des maschinellen Lernens, der sogenannten Support-Vektor-Maschine. Maschinelles Lernen hat nichts mit „Maschinen“ zu tun, wie wir sie verstehen. Vielmehr geht es dabei um künstliches Lernen – die automatische Wissensextraktion aus Daten. Es werden Algorithmen entwickelt, welche das menschliche Lernen nachempfinden sollen. Möglicherweise ist die in der Community verwendete Übersetzung von “*machine*” (Automat) in „Maschine“ in diesem Zusammenhang nicht besonders vorteilhaft. Diese Problematik ist auch schon von der Turingmaschine bekannt, die von Alan Turing 1936 als reines Gedankenmodell vorgestellt worden war, heutzutage aber stets mit einem Rechner in Verbindung gebracht wird.

Wir setzen uns mit aktuellen Aspekten von Support-Vektor-Maschinen auseinander. Dazu zählen die Bereiche kostensensitiver Klassifikation, automatischer Parameteroptimierung sowie effizienter und paralleler Algorithmen. Diese Arbeit entstand in enger Verbindung mit dem Industrieprojekt GALA (Grünenthal Applied Life Science Analysis) zwischen dem Forschungszentrum Jülich und dem Pharmaunternehmen Grünenthal GmbH in Aachen. Ein wichtiger Teil unserer Arbeit ist daher auch die Anwendung der implementierten Verfahren auf reale Datensätze.

Ich danke meinem Doktorvater Prof. Dr. Bruno Lang für die Unterstützung dieser Dissertation, das entgegengebrachte Vertrauen, die fruchtbaren Diskussionen und die Ratschläge bei der Erstellung zahlreicher Publikationen. Weiterhin danke ich Prof. Dr. Dr. Thomas Lippert, dem Direktor des Zentralinstituts für Angewandte Mathematik (ZAM) am Forschungszentrum Jülich für die Förderung meines Promotionsvorhabens und die Übernahme des Korreferates. Diese Dissertation entstand im Zeitraum Februar 2004 bis November 2006 am ZAM. Allen Kollegen danke ich für die Hilfsbereitschaft in allen Lebenslagen, die fachliche Unterstützung, die hervorragenden Arbeitsbedingungen und die einmalige

Möglichkeit, die Supercomputer des Forschungszentrums Jülich zu benutzen. Ebenfalls danke ich der Arbeitsgruppe Angewandte Informatik an der Universität Wuppertal für die freundliche Aufnahme und Unterstützung. Abschließend bedanke ich mich bei Dr. Achim Kless (Grünenthal GmbH) für die gute Zusammenarbeit im GALA-Projekt.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Data-Mining, künstliche Intelligenz und maschinelles Lernen – diese Begriffe sind derzeit Schlagworte in Forschung und Industrie. Dabei sieht man die künstliche Intelligenz oft als Wurzel, aus der das maschinelle Lernen als Teilgebiet bzw. Anwendung hervorgegangen ist und dann in Kombination mit Statistik und Datenbanktheorie das Gebiet des Data-Mining hervorgebracht hat. Es besteht der Wunsch und die Notwendigkeit, in immer größer werdenden Daten immer komplexere Zusammenhänge zu finden. Die Anwendungen sind zahlreich und ziehen sich durch viele Arbeitsgebiete. Beispielsweise lassen sich in der pharmazeutischen Forschung und der klinischen Entwicklung von Medikamenten nicht alle in Frage kommenden Substanzen experimentell untersuchen, die Zahl der Kandidaten ist zu groß. Mit Hilfe eines Quantitative Structure-Activity Relationship Modells (QSAR) lassen sich aber logische Verknüpfungen der Eigenschaften bekannter Wirkstoffe herstellen. Die dafür notwendigen Algorithmen fallen in den Bereich des überwachten Lernens, einem Teilgebiet des maschinellen Lernens. Sie sollen unter Einbeziehung chemischer Deskriptoren und biologischer Daten multidimensionale Zusammenhänge in Strukturen erkennen, um die enormen Entwicklungszeiten innovativer Medikamente zu verkürzen.

Eine der wichtigsten Methoden des überwachten Lernens ist die binäre Klassifikation. Ein binärer Klassifikator kann Daten einer von zwei Klassen zuordnen. Das klingt zunächst einfach, jedoch muss dieser Klassifikator erst bestimmt werden. Dafür setzt man eine bestimmte Menge an bereits klassifizierten Daten ein, aus denen dann ein Zusammenhang erlernt wird. Neben vielen anderen Algorithmen wurden die sogenannten Support-Vektor-Maschinen (SVM's) für die Klassifikation von Daten konzipiert. Zu Beginn ihrer Entwicklung waren SVM's wegen ihrer einfachen linearen Lernidee wenig akzeptiert. Später, als klar wurde, wie kompliziert die zugrunde liegende Lerntheorie tatsächlich ist, wurden wiederum einfachere und besser interpretierbare Methoden bevorzugt. Die Herausforderungen bei der Implementierung und die wegen der langen Rechenzeit schwierige Nutzung für große Daten verstärkten die Ablehnungshaltung. Nach und nach, insbesondere durch die bessere Verfügbarkeit freier SVM-Software sowie die im Vergleich mit anderen Methoden zuverlässigen Ergebnisse, hat sich das Verfahren der Support-Vektor-Maschinen

durchgesetzt und zählt mittlerweile zu den bekanntesten modernen Klassifikationsmethoden.

Bei der Erstellung von Klassifikatoren ist die Zuverlässigkeit für neue Daten von großer Bedeutung. Eine Herausforderung stellt die Anpassung notwendiger Modellparameter dar. Sie sollen nicht nur für die verwendeten Daten passen, sondern auch dafür sorgen, dass künftige Daten gut klassifiziert werden. Jedoch können bei vielen aktuellen Anwendungen die gewünschten Gütewerte nicht erreicht werden. Oft liegt das daran, dass eine Unausgeglichenheit zwischen den Klassen vorliegt. Die kann zum Einen durch unterschiedliche Größen der Klassen, aber zum Anderen auch durch unterschiedlich hohe Kosten für Fehlklassifikationen entstehen. Typischerweise maximieren Lernmethoden und die dazugehörigen Parameteroptimierungsverfahren ausschließlich die Genauigkeit, das heißt die Wahrscheinlichkeit, einen Punkt korrekt zu klassifizieren. Die Anwender haben nicht die Möglichkeit und das notwendige Wissen, um korrigierend in den Lernprozess einzugreifen. Verbesserte Modelle, aufwändige Parameteroptimierung sowie die immer schneller wachsenden Datenbestände in Forschung und Industrie führen bei Algorithmen des maschinellen Lernens und insbesondere bei den Support-Vektor-Maschinen schnell an die Grenzen der tolerierbaren Rechenzeiten. Bisher versuchte man, das Problem durch Datenreduzierung oder andere Vereinfachungen zu umgehen. Die zunehmende Zahl an Parallelrechnern, insbesondere von SMP[1]-Cluster-Systemen, ermöglicht die Nutzung von parallelen Data-Mining-Algorithmen zur Datenanalyse. Zur Zeit ist die Entwicklung moderner paralleler Klassifikationsverfahren ein sehr junges und dynamisches Forschungsgebiet, dessen weitere Entwicklung noch nicht genau vorherzusehen ist. Diese Arbeit leistet dazu einen Beitrag.

1.1 Überblick

Unsere Arbeit, die sich mit dem modernen Verfahren der Support-Vektor-Maschinen auseinandersetzt, wurde stark durch das Industrieprojekt GALA [85] motiviert. Dieses Projekt wurde ins Leben gerufen, um neue Wege bei der Entwicklung von Medikamenten zu beschreiten. Arbeitsschwerpunkt des Projektes war die Erkennung, Analyse und Interpretation multidimensionaler Zusammenhänge unter Einbeziehung chemischer Deskriptoren und biologischer Daten, mit dem Ziel humane in vivo Daten von Substanzen besser vorhersagen zu können. Hintergrund dieser Bemühungen ist es, mit den entwickelten Verfahren die enormen Entwicklungszeiten (10 - 15 Jahre) für innovative Arzneimittel signifikant zu verkürzen. Dazu wurden von der Grünenthal GmbH in Aachen und dem Forschungszentrum Jülich gemeinsam neue Methoden und Verfahren zur Datenanalyse in der pharmazeutischen Forschung entwickelt. Diese sollen helfen, die Wirkung von Substanzen auf den Menschen im Vorhinein abschätzen zu können. Das Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich übernahm dabei Aufgaben auf dem

[1] Shared-Memory-Prozessoren

Gebiet der Statistik. Es wurden Algorithmen entwickelt, die Abhängigkeiten zwischen den Merkmalen eines Stoffes aufdecken und neue Substanzen in Gruppen unterschiedlicher Erfolgsaussichten einstufen. Neue Methoden der künstlichen Intelligenz, insbesondere Support-Vektor-Maschinen, werden eingesetzt, um die erzielten Ergebnisse weiter zu verbessern [36, 38, 82–84].

Wir beschäftigen uns zunächst mit der Implementierung einer seriellen SVM-Software. Dabei spielen Effizienz und Parallelisierbarkeit eine wichtige Rolle. Zur Lösung des Optimerungsproblems während des SVM-Trainings wählen wir einen Zerlegungsalgorithmus (*decomposition*), der auf einem schnellen Projektionsverfahren aufbaut. Um den sogenannten Trainingsalgorithmus zum Erlernen einer Klassifikationsfunktion herum wird eine Validierungsumgebung zur internen Bewertung des Modells implementiert. Motiviert durch die Anwendungen innerhalb des GALA-Projektes erstellen wir ein Gerüst zur flexiblen und kostensensitiven Klassifikation. Das Zusammenspiel von Optionen erweitert die Möglichkeiten der Modellierung. Für den Anwender ist die Nutzung dennoch schwierig. Wie die meisten maschinellen Lernverfahren sind auch die Support-Vektor-Maschinen durch verschiedene Parameter konfigurierbar. Für erfolgreiche Anwendungen ist es wichtig, diese Parameter möglichst gut auf die jeweiligen Daten einzustellen. Sind die Parameter schlecht gewählt, bringen auch die neuen Techniken keine Verbesserung. Aus diesem Grund setzen wir uns mit dem Problem der Parameteroptimierung auseinander. Zur Bewertung von Zwischenergebnissen werden geeignete Gütemaße eingeführt. Ein vielversprechendes Maß wird mit flexiblen Komponenten ausgestattet, welche die Parameteroptimierung erleichtern. Zudem wird eine frei verfügbare Software zur numerischen Parameteroptimierung ausgewählt und an die SVM-Software gekoppelt. Unser Beitrag zum Gebiet des parallelen Data-Minings ist die Entwicklung einer zweistufig parallelen SVM-Software. Die innere Ebene des Trainings wird im Shared-Memory-Modus realisiert und kann so auf einzelnen Knoten eines SMP-Cluster-Systems ausgeführt werden. Zusätzlich gibt es eine MPI[2]-parallele Validierungsebene, die auch über SMP-Knotengrenzen hinweg eingesetzt werden kann. Die Software zur Parameteroptimierung stellt ebenfalls einen MPI-parallelen Modus zur Verfügung. Über unsere Implementierung eines neuen Kommunikationsschemas ermöglichen wir auch die Kopplung der parallelen Ebenen. Es entsteht eine dreistufig parallele Software zur automatisch optimierten Klassifikation mit Support-Vektor-Maschinen. Jede Ebene kann einzeln oder kombiniert flexibel parallel gewählt werden.

1.2 Struktur der Arbeit

Diese Arbeit besteht aus acht Kapiteln. Kapitel 2 führt in die Grundlagen des überwachten Lernens und der Klassifikation mit Support-Vektor-Maschinen ein. Kapitel 3 beschäftigt sich mit dem unserer Implementierung zugrunde liegenden SVM-Trainingsalgorithmus –

[2]Das Message-Passing Interface ist ein Protokoll, das parallele Berechnungen auf verteilten Systemen ermöglicht.

der Zerlegungsmethode – und den darin enthaltenen inneren Lösern. In Kapitel 4 stellen wir unsere Erweiterungen der Software vor. Diese haben zum Ziel, die Flexibilität des Lernens zu erhöhen. Der wichtigste Aspekt dabei ist kostensensitives Lernen, eine der Zielsetzungen des GALA-Projektes. Im Anschluss daran beschäftigen wir uns in Kapitel 5 mit Parameteroptimierung. Es werden geeignete Gütemaße sowie die von uns verwendete *APPSPACK*-Software [55] vorgestellt. Die Umsetzung der dreistufig parallelen Software wird in Kapitel 6 erläutert. Ausgehend von einem Überblick zu parallelen Methoden des Data-Mining und speziell der Support-Vektor-Maschinen erörtern wir die Implementierung der parallelen SVM-Software und gehen im Anschluss auf die Kopplung mit der parallelen *APPSPACK*-Software ein. In den Kapiteln 2 bis 6 werden keine Testergebnisse besprochen, diese befinden sich ausschließlich in Kapitel 7. Das Kapitel ist analog zum Aufbau der Arbeit geordnet nach Tests zum SVM-Trainingsalgorithmus allgemein, Tests zum kostensensitiven Lernen, Gütemaßen und deren Optimierung mit *APPSPACK* sowie Tests zum Verhalten der parallelen Ebenen.

1.3 Veröffentlichungen

Viele Teile dieser Arbeit wurden auf Workshops, Konferenzen und in Zeitschriften publiziert. Davon ausgenommen sind die Kapitel 2 und 3, in denen wichtige Grundlagen über Support-Vektor-Maschinen und die implementierten seriellen Optimierungsalgorithmen zusammengestellt wurden.

Arbeiten zur kostensensitiven Klassifikation mit Support-Vektor-Maschinen wurden beim Symposium *Knowledge Exploration in Life Science Informatics (KELSI 2004)* [82] sowie bei der *3. IEEE International Conference on Intelligent Systems (IS 2006)* [44] vorgestellt. Erstere Arbeit zeigt Ergebnisse von Anwendungen innerhalb des GALA-Projektes und ist in enger Zusammenarbeit mit Dr. Achim Kless, Grünenthal GmbH, entstanden. Weitere Veröffentlichungen mit Dr. Kless sind ein Artikel zur Projektbeschreibung in der Zeitschrift *Bioworld* [84] sowie ein Poster auf dem *16. European Symposium on QSAR and Molecular Modelling* [83]. Unsere Studie zur kostensensitiven Klassifikation aktueller CYP-Datensätze ist im *Journal of Chemical Information and Modeling* erschienen [38].

Unser neues Gütemaß als Grundlage für kostensensitive Parameteroptimierung haben wir im *Journal of Computational and Applied Mathematics* [43] vorgestellt. Sowohl in dieser Arbeit als auch beim Symposium *Computational Life Sciences (CompLife 2005)* [40] sind erfolgreiche Anwendungen der Optimierung unausgeglichener Klassifikationsprobleme mit *APPSPACK* und dem neuen Gütemaß präsentiert worden.

Im Zusammenhang mit den Arbeiten zum Zerlegungsalgorithmus und den drei parallelen Stufen unserer SVM-Software, ihrer Verbindung und Anwendung wurden Arbeiten auf der *International Conference on Artificial Intelligence and Machine Learning (AIML 2005)* [39], der *11. International Conference on Computer Science (ICCS 2006)* [42],

der *Fourth Biennial International Conference Advances in Information Systems (ADVIS 2006)* [41], der *European Conference on Parallel Computing (EuroPar 2006)* [37], dem *Workshop From Computational Biophysics to Systems Biology (CBSB 2006)* [47], dem *Workshop on Parallel Data Mining (PDM 2006)* [48] und der *Australasian Data Mining Conference (AusDM 2006)* [46] vorgestellt.

Kapitel 2

Klassifikation mit Support-Vektor-Maschinen

In dieser Arbeit werden Support-Vektor-Maschinen zur binären Klassifikation betrachtet. Das zugrunde liegende Modell wird stets das sogenannte überwachte (*supervised*) Lernen [26] sein. Sind funktionale Zusammenhänge zwischen Eingaben und Ausgaben eines Systems nicht bekannt, da man sie entweder gar nicht oder nur unter unverhältnismäßig großem Arbeits- und Zeitaufwand generieren kann, lohnt sich der Einsatz von Algorithmen des überwachten Lernens. Dazu müssen endlich viele Beispiele zur Verfügung stehen, welche die Abhängigkeit so gut wie möglich repräsentieren. Diese Beispiele verwendet man, um die gesuchte Funktionalität zu erlernen.

Definition 2.1 (Trainingspunkt)**.** *Sei $n \in \mathbb{N}$. Als Trainingspunkt bezeichnen wir im Folgenden einen Datenvektor $\boldsymbol{x} \in \mathbb{R}^n$ genau dann, wenn ihm ein Label $y \in \{-1, 1\}$ zugeordnet ist und er zusätzlich zum Erlernen eines Klassifikators verwendet wird.*

Bemerkung 2.1. *Wie auch in Def. 2.1 angegeben, bezeichnen wir die Anzahl der betrachteten Variablen (Attribute, Features, Merkmale, Deskriptoren) mit n. Die Gesamtheit der für einen Lernvorgang betrachteten Variablen wird in der Pharmaforschung oft als Deskriptorset bezeichnet.*

Definition 2.2 (Trainingsdatensatz)**.** *Als rd-elementigen Trainingsdatensatz (Referenzdatensatz) bezeichnen wir im Folgenden die Menge $\left\{(\boldsymbol{x}^1, y_1), \ldots, (\boldsymbol{x}^{rd}, y_{rd})\right\}$. Die zugehörige Indexmenge wird mit $\mathcal{RD}$ bezeichnet.*

Die Herausforderung des überwachten Lernens besteht darin, eine geeignete Menge $\mathcal{H}$ an möglichen Hypothesenfunktionen zu definieren, aus der dann eine spezielle Hypothese $h : \mathbb{R}^n \to \{-1, 1\}$ in Abhängigkeit der konkreten Trainingspaare gewählt und angepasst wird. Support-Vektor-Maschinen bedienen sich dazu affin-linearer Funktionen in einem speziellen Merkmalsraum.

In den Abschnitten 2.2 und 2.3 werden die den SVM's zugrunde liegenden Theorien vorgestellt. Diese werden in den Abschnitten 2.4 und 2.5 zusammengeführt und es wird erläutert, wie die finalen SVM-Lernaufgaben konzipiert sind. Zunächst gehen wir im folgenden Abschnitt auf einige Grundlagen zur Bewertung von Klassifikationsalgorithmen ein.

2.1 Bewertung von Klassifikationsalgorithmen

Jeder Klassifikator, unabhängig von der speziellen Lernmethode, durchläuft im Allgemeinen zwei Phasen – Optimierungs- und Nutzungsphase. Die Optimierungsphase teilt sich in Modell- (Kapitel 4) und Parameteroptimierung (Kapitel 5). Die Nutzungsphase dient der Klassifikation neuer, unbekannter Daten. Letztere ist für unsere Betrachtungen uninteressant.

Modell- und Parameteroptimierung für Algorithmen des überwachten Lernens basieren auf mehrmaligen zeitintensiven Trainingsphasen und der Frage, wie man die Leistung eines Modells bzw. Parametertupels bewerten kann [144]. Prinzipiell gibt es zwei Möglichkeiten der Bewertung:

- Untersuchung charakteristischer Merkmale des Trainings

  Als Kriterien kommen die Anzahl der Trainingsfehler, Charakteristika der Lösung oder auch die Trainingszeit in Frage. Auch wenn diese Merkmale in der Praxis durchaus Verwendung finden, letzteres z.B. als Kriterium zur Variablenselektion mit SVM's in [51], ist die alleinige Bewertung des Trainings jedoch unserer Meinung nach kein akzeptables Kriterium zur Qualitätsmessung von SVM-Modellen, da diese eine besondere Art der Risikominimierung durchführen, die auf der expliziten Fehlertolerierung zugunsten einer gut generalisierenden Lösung beruht (vergleiche Abschnitt 2.4.2).

- Durchführung von internen Tests

  Unter einem internen Test verstehen wir eine Aufteilung des Trainingsdatensatzes in Modellierungs- und Testdaten, wobei erstere für den Lernvorgang verwendet werden und der Klassifikator dann selbstständig die Testdaten klassifiziert, um zu zeigen, wie zuverlässig das Modell auf unabhängigen[3] Daten ist. Dieser Test wird dann als Recall bezeichnet. Die Ergebnisse müssen bewertet werden. Eine verbesserte Variante dieser Methode stellt die v-fache Kreuzvalidierung [1] dar. Dabei werden die Trainingsdaten in v disjunkte Gruppen aufgeteilt. v mal wird mit $v-1$ Gruppen trainiert und mit einer Gruppe getestet, sodass jede Gruppe einmal getestet wurde. In Abbildung 2.1 ist eine 4-fache Kreuzvalidierung dargestellt. Je größer v ist, umso besser passen die gewählten Parameter im Anschluss zum Gesamtdatensatz [135]. Im Extremfall gilt $v = rd$, sodass für jeden Trainingspunkt ein neues Modell erlernt wird.

[3] nicht in den Lernvorgang eingeflossenen

Diese Methode wird *leave-one-out* genannt und führt zu enormen Rechenzeiten. Für große Daten sollte sie nicht verwendet werden.

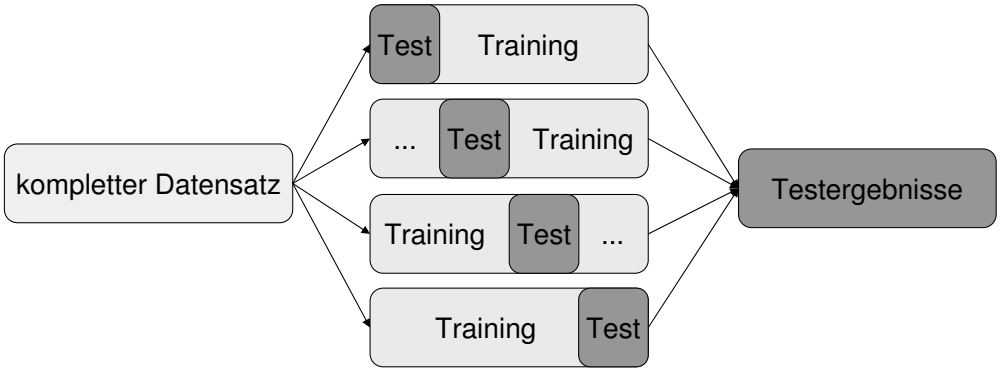

Abbildung 2.1: Vierfache Kreuzvalidierung.

Es stellt sich nun die Frage, wie die Ergebnisse eines internen Tests – einer Kreuzvalidierung oder eines unabhängigen Recalls – bewertet werden können. Dafür stehen innerhalb des Data-Mining verschiedene Kennzahlen zur Verfügung. Wir geben zunächst einige einfache, aber sehr wichtige Kennzahlen an, auf die wir später wieder zurückgreifen. Wir bezeichnen die Menge der betrachteten Punkte im Folgenden stets als $\mathcal{TD}$, die Menge der Testdaten (z.B. Validierungs- oder Recalldaten).

Definition 2.3 (Anzahl positiver und negativer Punkte)**.**

$$pp := |\{i \in \mathcal{TD} : \ y_i = 1\}| \quad , \qquad np := |\{i \in \mathcal{TD} : \ y_i = -1\}| \, .$$

Definition 2.4 (Anzahl falscher Klassifikationen)**.**

$$fk := |\left\{i \in \mathcal{TD} : \ y_i \cdot h(\boldsymbol{x}^i) < 0\right\}| \, .$$

Die vorhergesagte Klasse stimmt nicht mit dem gegebenen Klassenlabel überein.

Ein binärer Klassifikator unterscheidet nur zwischen positiven und negativen Vorhersagen, sodass sich vier mögliche Szenarien für einen Punkt ergeben.

Definition 2.5 (Anzahl richtig positiver Klassifikationen)**.**

$$rp := |\left\{i \in \mathcal{TD} : \ y_i = 1, \ h(\boldsymbol{x}^i) = 1\right\}| \, .$$

Definition 2.6 (Anzahl falsch positiver Klassifikationen)**.**

$$fp := |\left\{i \in \mathcal{TD} : \ y_i = -1, \ h(\boldsymbol{x}^i) = 1\right\}| \, .$$

Definition 2.7 (Anzahl richtig negativer Klassifikationen)**.**

$$rn := \left| \left\{ i \in \mathcal{TD} : \; y_i = -1, \; h(\boldsymbol{x}^i) = -1 \right\} \right| .$$

Definition 2.8 (Anzahl falsch negativer Klassifikationen)**.**

$$fn := \left| \left\{ i \in \mathcal{TD} : \; y_i = 1, \; h(\boldsymbol{x}^i) = -1 \right\} \right| .$$

Klassifikationsergebnisse, auch für mehr als zwei Klassen, können anhand einer sogenannten Konfusionsmatrix[4] $\boldsymbol{KFM}$ dargestellt werden, siehe Tabelle 2.1. Je mehr Fehler aufgetreten sind, umso stärker sind Nichtdiagonalelemente besetzt. Ein fehlerfreier Test führt zu einer reinen Diagonalgestalt.

	Klasse 1	Klasse -1
Hypothese 1	rp	fp
Hypothese -1	fn	rn

Tabelle 2.1: Konfusionsmatrix eines Klassifikators (binärer Fall).

Die wichtigsten Maße zur Bewertung von Testergebnissen sind [82]:

Definition 2.9 (Genauigkeit, *accuracy*)**.** ***Die Genauigkeit ist der Anteil der insgesamt korrekt klassifizierten Testpunkte.***

$$ac = \frac{rp + rn}{pp + np} . \tag{2.1}$$

Definition 2.10 (Sensitivität, *sensitivity*)**.** ***Die Sensitivität gibt an, wieviele der positiven Punkte tatsächlich als positiv klassifiziert worden sind.***

$$se = \frac{rp}{pp} . \tag{2.2}$$

Definition 2.11 (Spezifität, *specificity*)**.** ***Die Spezifität gibt an, wieviele der negativen Punkte tatsächlich als negativ klassifiziert worden sind.***

$$sp = \frac{rn}{np} . \tag{2.3}$$

Definition 2.12 (Präzision, *precision*)**.** ***Die Präzision gibt an, wieviele der als positiv klassifizierten Punkte tatsächlich positiv sind.***

$$pr = \frac{rp}{rp + fp} . \tag{2.4}$$

[4] confusion matrix [106]

Zur Veranschaulichung der mit den obigen Maßen verfolgten Ziele seien die Beispiele in Abbildung 2.2 gegeben. Bei insgesamt 10 Punkten, von denen die Hälfte positiv ist, gelten:

$$ac = 0.9\,, \quad se = 0.8\,, \quad sp = 1.0 \quad \text{und} \quad pr = 1.0 \quad \text{(A)}, \tag{2.5}$$

$$ac = 0.9\,, \quad se = 1.0\,, \quad sp = 0.8 \quad \text{und} \quad pr = 0.8\bar{3} \quad \text{(B)}, \tag{2.6}$$

$$ac = 0.8\,, \quad se = 0.8\,, \quad sp = 0.8 \quad \text{und} \quad pr = 0.8\bar{3} \quad \text{(C)}. \tag{2.7}$$

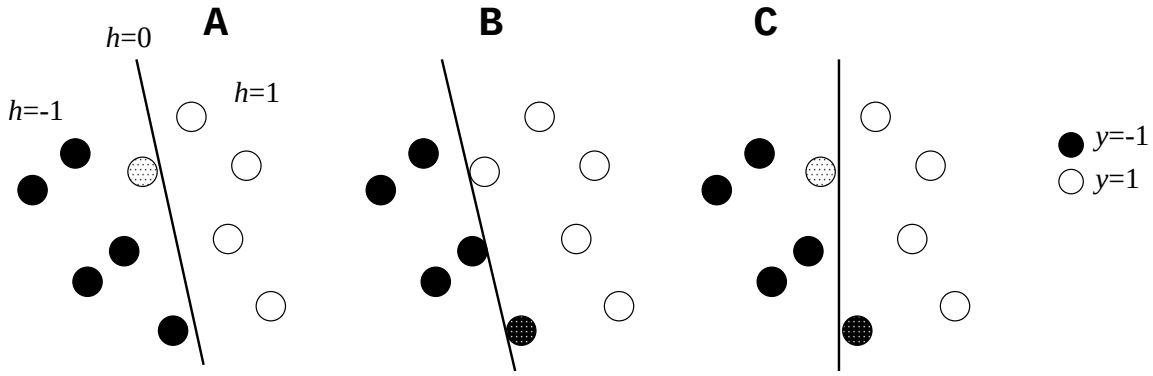

Abbildung 2.2: Beispiele zur Bewertung eines Tests.

Typischerweise wird bei Klassifikationsbewertungen die Genauigkeit betrachtet [120]. Diese Standardmethode ist jedoch oft nicht ausreichend, vor allem, wenn es strukturelle Unterschiede zwischen den Klassen von Datenpunkten gibt oder wenn die Anwendung stark auf eine Klasse ausgerichtet ist [3]. Wir verweisen an dieser Stelle auf Kapitel 5, in dem wir uns mit speziellen Gütemaßen auseinandersetzen werden. Wie auch immer Ergebnisse bewertet werden, es kommt zur Unterscheidung zwischen schwachen und starken Klassifikatoren, sogenannten *weak* und *strong learners* [75]. Schwache Klassifikatoren haben oft den Vorteil, dass die zugrunde liegende Lernmethode einfach zu verstehen und schnell durchzuführen ist. Über eine Kombination mehrerer *weak learners* – sogenannte Ensemble-Methoden (vergleiche Seite 106) – kann auch ein starker Klassifikator geschaffen werden. SVM's gelten als ein sehr starkes Lernverfahren.

2.2 Lineares Lernen

Lineare Modelle bilden die Grundlage der nichtlinearen SVM-Klassifikation. Dieser Abschnitt soll dazu dienen, die wichtigsten Eckpunkte des linearen Lernens zu beleuchten.

Dabei wird auch der Begriff der Marge definiert werden, welcher fundamental für SVM-basierte Methoden ist.

Innerhalb der Theorie der Klassifikation gibt es viele Ansätze, um zwei Klassen mittels einer affin-linearen Funktion zu trennen. Dazu gehören beispielsweise die lineare Diskriminanzanalyse [149], aber auch einfache neuronale Netze [12].

Definition 2.13 (Zielfunktion)**.** *Sei $\boldsymbol{x} \in \mathbb{R}^n$ ein zu klassifizierender Punkt. Die beim linearen Lernen zur Verfügung stehenden Zielfunktionen sind von der Form*

$$f_{\text{lin}}(\boldsymbol{x}) := \sum_{k=1}^{n} w_k x_k + b \, . \tag{2.8}$$

Dabei bezeichnen wir mit $\boldsymbol{w} \in \mathbb{R}^n$ den Gewichtsvektor und mit $b \in \mathbb{R}$ den Schwellwert der affin-linearen Klassifikationsfunktion.

Definition 2.14 (Hypothesenfunktion)**.** *Sei $\boldsymbol{x} \in \mathbb{R}^n$ ein zu klassifizierender Punkt. Die beim linearen Lernen zur Verfügung stehenden Hypothesenfunktionen zur Einordnung dieses Punktes in eine von zwei Klassen sind von der Form*

$$h_{\text{lin}}(\boldsymbol{x}) := \operatorname{sgn}\left(f_{\text{lin}}(\boldsymbol{x})\right) \, . \tag{2.9}$$

Dabei definieren wir die spezielle Signumfunktion als

$$\operatorname{sgn}(a) := \begin{cases} 1 & \text{falls } \; a \geq 0 \\ -1 & \text{sonst} \end{cases} \qquad (a \in \mathbb{R}) \, ,$$

um sicherzustellen, dass eine binäre Klassifikation mit genau 2 Ausgabemöglichkeiten durchgeführt wird.

In Abbildung 2.3 ist eine lineare Trennfunktion dargestellt, die wir im Folgenden als (separierende, trennende) Hyperebene bezeichnen werden. Es stellt sich die Frage nach der Lage der optimalen Hyperebene. Wir werden im Folgenden erklären, was als optimal angesehen wird. Separierbarkeit ist im Zusammenhang mit der Klassifikation von großen, hochdimensionalen Daten ein wichtiger und oft auch problematischer Aspekt.

Definition 2.15. *Ein Trainingsdatensatz wird linear separabel genannt, falls eine Hypothese h_{lin} der Form (2.9) existiert, sodaß*

$$h_{\text{lin}}(\boldsymbol{x}^i) = y_i \quad \forall i \in \mathcal{RD}$$

gilt, also jeder Trainingspunkt durch h_{lin} korrekt zugeordnet wird.

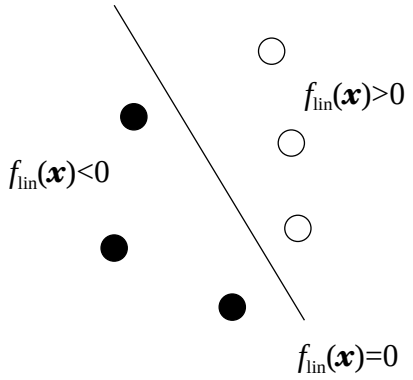

Abbildung 2.3: Hyperebene zur Trennung von zwei Klassen.

Wie bewertet und vergleicht man unterschiedliche Hypothesen, die einen Datensatz separieren können? Die Theorie der Support-Vektor-Maschinen stützt sich dafür stark auf das Konzept der Marge. Dieses besagt, dass eine lineare Trennfunktion als gut angesehen werden kann, wenn sie die vorhandenen Trainingsdaten korrekt klassifiziert und zusätzlich die Punkte beider Klassen möglichst weit von der Trennlinie entfernt sind. In Abb. 2.4 sind zwei lineare Trennfunktionen f^1_{lin} und f^2_{lin} dargestellt, welche die Daten perfekt separieren. Dennoch wird man die zweite Funktion intuitiv bevorzugen – aber warum? Wir betrachten

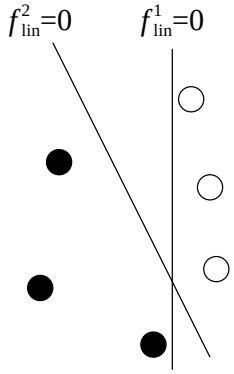

Abbildung 2.4: Zwei mögliche Klassifikationsfunktionen, die den gegebenen Datensatz perfekt trennen.

zunächst ein einzelnes Trainingspaar.

Definition 2.16 (funktionaler Abstand)**.** *Seien $(\boldsymbol{x}^i, y_i)$ ein Trainingspaar und f_{lin} eine gegebene Zielfunktion. Dann ist der funktionale Abstand $\gamma_i \in \mathbb{R}$ zwischen dem Trainingspaar und der Hyperebene definiert als*

$$\gamma_i := y_i \cdot f_{\text{lin}}(\boldsymbol{x}^i) \; . \tag{2.10}$$

Folgerung 2.1. *Falls für alle Trainingspaare $i \in \mathcal{RD}$ $\gamma_i > 0$ gilt, so liegt eine korrekte Klassifikation vor.*[5]

In Abbildung 2.5 sind zwei Fehlklassifikationen dargestellt, die jeweils über die negativen funktionale Abstände zu lokalisieren sind. Bezug nehmend auf das Konzept der Marge

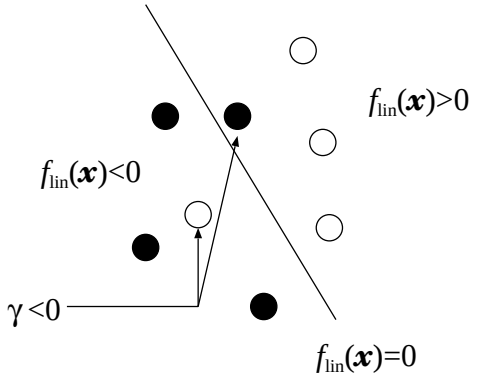

Abbildung 2.5: Negative funktionale Abstände sind Klassifikationsfehler.

kann man festhalten, dass eine gute Zielfunktion keine negativen und möglichst große positive funktionale Abstände realisieren kann. Dabei wird die Gesamtqualität schon durch einzelne schlechte Klassifikationen gemindert, denn der am schlechtesten klassifizierte Punkt bestimmt den Wert der Marge.

Bemerkung 2.2. *Im Folgenden bezeichnet $\|\cdot\|$ stets die euklidische Norm (2-Norm) eines Vektors bzw. einer Matrix. Abweichende Normen werden stets gesondert definiert.*

Definition 2.17 (funktionale Marge)**.** *Die funktionale Marge $\gamma \in \mathbb{R}$ einer Klassifikationsfunktion für einen Trainingsdatensatz wird definiert als*

$$\gamma := \min_{i=1,\ldots,rd} \gamma_i \ .$$

Intuitiver als die funktionalen, sind die sogenannten geometrischen Abstandsmaße. Dazu werden normalisierte Zielfunktionen betrachtet. Diese sind von der Form (2.8), wobei jedoch die normierten Parameter betrachtet werden. Diese erhält man über die Transformation

$$(\boldsymbol{w}, b) \ \longmapsto \ \left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}, \frac{b}{\|\boldsymbol{w}\|} \right) \ .$$

[5] $\gamma_i = 0$ bedeutet bei negativen Punkten einen Fehler.

Diese ändert die Lage der trennenden Hyperebene nicht, ermöglicht jedoch eine geometrische Interpretation der Abstände. Die geometrischen Abstände bezeichnen wir im Folgenden als γ_i^g $(i = 1, \ldots, rd)$.

Definition 2.18 (geometrische Marge)**.** *Die geometrische Marge $\gamma^g \in \mathbb{R}$ einer Klassifikationsfunktion für einen Trainingsdatensatz, im Folgenden auch Trenngüte genannt, wird definiert als*

$$\gamma^g := \min_{i=1,\ldots,rd} \gamma_i^g.$$

Support-Vektor-Maschinen werden oft als *maximal margin classifiers* bezeichnet [26], d.h. die dem Training zugrunde liegende Gütemessung basiert auf der Marge. Der durch die Marge entstehende Grenzbereich ist in Abbildung 2.6 dargestellt. Er ist frei von Punkten. Auf die besondere Bedeutung der Punkte an der Grenze des Margenbereiches für die von Support-Vektor-Maschinen erzeugte Klassifikationsfunktion wird im Abschnitt 2.4 genauer eingegangen.

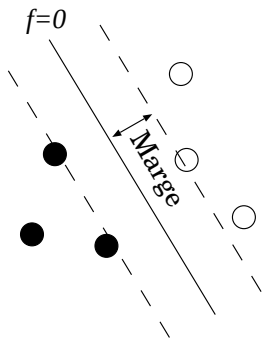

Abbildung 2.6: Perfekte Trennung führt zu einem punktleeren Grenzbereich mit der Breite der doppelten Marge.

Bei der Betrachtung eines Klassifikationsverfahrens steht immer die Frage einer guten Generalisierung im Vordergrund. Die Generalisierungstheorie versucht zu zeigen, dass ein Lernverfahren Hypothesen derartig aus Beispielen erlernen kann, dass sie

- speziell genug sind, um gegebene Funktionalitäten zu erfassen; andererseits aber auch
- allgemein genug sind, um sich durch die vorhandenen Beispiele nicht zu stark auf Einzelfälle festzulegen.

Der Generalisierungsfehler linearer Klassifikatoren, die durch Maximierung der Marge entstehen, kann nach oben abgeschätzt werden [26, 36]. Dadurch ist gesichert, dass die *maximal margin classifiers* gute Hypothesen erlernen. Das Konzept nichtlinearer SVM-Klassifikation, welches im nächsten Abschnitt vorgestellt wird, ändert an diesen Abschätzungen nichts.

An dieser Stelle soll auf die Probleme von Überanpassung sowie Nichtseparierbarkeit von Daten eingegangen werden. Überangepasstes Verhalten einer Klassifikationsfunktion ist das Gegenstück zu guten Generalisierungseigenschaften.

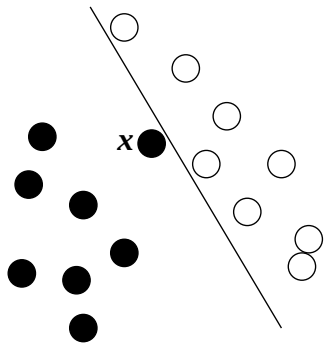

Abbildung 2.7: Ein einzelner ungünstiger Punkt führt bei strikter Trennung zu einer Marge von Null.

Definition 2.19 (Überanpassung)**.** *Eine Klassifikationsfunktion zeigt überangepasstes Verhalten, wenn sie die Trainingsdaten mit sehr wenigen Fehlern klassifiziert, jedoch bei neuen Daten versagt.*

Ursachen von Überanpassung sind vielschichtig und können nicht verallgemeinert werden. Der Begriff ist auch dahingehend kritisch, als dass er oft verwendet wird, jedoch nicht quantisierbar ist. In [62] werden zwei Arten von Überanpassung unterschieden:

- ein Modell ist flexibler als nötig und verkompliziert Zusammenhänge, oder
- ein Modell enthält irrelevante Komponenten.

In Abb. 2.7 kann man erkennen, dass schon ein einzelner Trainingspunkt ($\boldsymbol{x}$) dazu führen kann, dass das alleinige Anwenden des Konzeptes der Marge eine offensichtlich schlechte Klasseneinteilung nach sich ziehen kann. Die dargestellte Hyperebene ist zu stark an die Daten angepasst. Dabei ist zu beachten, dass dieses Verhalten insbesondere für fehlerhafte Daten zu enormen Problemen führen kann. Es ist sehr wahrscheinlich, daß diese Hypothesenfunktion bei neuen Daten schlechte Klassifikationen generieren wird. Wir halten daher fest, dass bei Fällen der Überanpassung einzelne Punkte bei der Maximierung der Marge nicht beachtet werden sollten. Im Gegenzug dazu müssen die entstandenen Fehler jedoch berechnet und bewertet werden, damit nicht beliebig viele Punkte ausgenommen werden. Damit stehen sich nun zwei konkurrierende Konzepte gegenüber:

- die Maximierung der Marge, und
- die Minimierung der Anzahl von Punkten, welche die Marge verletzen.

Definition 2.20 (Schlupfvariablen [26])**.** *Seien $(\boldsymbol{x}^i, y_i)$ ein Trainingspaar, f_{lin} eine Klassifikationsfunktion und $\gamma^* > 0$ eine fest definierte (gewünschte) funktionale Marge. Dann nennen wir*

$$\xi_i := \max\ \left\{\gamma^* - y_i \cdot f_{\text{lin}}(\boldsymbol{x}^i); 0\right\} \overset{(2.10)}{=} \max\ \{\gamma^* - \gamma_i; 0\}$$

Schlupfvariable für das Paar i.

Der Wert einer Schlupfvariablen zeigt, um wieviel die Klassifikationsfunktion das Mindestziel γ^* verfehlt hat. Im Extremfall liegen die Daten derartig, dass eine strikte Trennung überhaupt nicht möglich ist. Das erkennt man an negativen funktionalen Abständen einzelner Punkte, bzw. an den Werten ihrer Schlupfvariablen, für die dann $\xi_i > \gamma^*$ gilt. In solchen Fällen spricht man von Daten, die nicht linear separabel sind. Ist man dennoch auf lineare Funktionen eingeschränkt, muss man einzelne Punkte ausblenden. Im Abschnitt 2.4 werden wir das Konzept der Marge und der Schlupfvariablen für die konkrete Beschreibung des SVM-Lernens für separierbare sowie nichtseparierbare Daten wieder aufgreifen. Zunächst gehen wir im folgenden Abschnitt auf die Theorie der Kerne ein, welche neben der linearen statistischen Lerntheorie fundamental für die hervorragenden Lerneigenschaften von Support-Vektor-Maschinen ist.

2.3 Kerne

Im Abschnitt 2.2 wurde zunächst die dem SVM-Lernen zugrunde liegende Lerntheorie vorgestellt. Typischerweise sind die zu lernenden Probleme hochgradig nichtlinear, sodass lineare Modelle allein keine sinnvollen Hypothesen erbringen können. Es muss daher ein Weg gefunden werden, komplexe Modelle zu erlernen. Fast alle Methoden des maschinellen Lernens können nichtlineare Abhängigkeiten modellieren. Im Zusammenhang mit der Erweiterung des linearen Lernens sollten zunächst die klassischen neuronalen Netze erwähnt werden. Sie überbrücken den Pfad vom linearen zum nichtlinearen Modell mittels mehrerer übereinanderliegender Schichten [36] und werden in Anlehnung an die einfachen Perceptrons als mehrschichtige Perceptrons bezeichnet. Sie werden mittels nichtlinearer Optimierungsfunktionen derart angepasst, dass die entstehenden Modelle komplexe Abhängigkeiten gut darstellen können. Die Theorie der Support-Vektor-Maschinen schlägt einen anderen Weg ein. Da die lineare Lerntheorie mit ihren einfachen Hypothesen, der gut überschaubaren Struktur und den Fehlerabschätzungen, welche die Generalisierungstheorie [135] liefert, erhalten bleiben soll, besteht die einzige Möglichkeit des nichtlinearen Lernens in einer a priori Datentransformation. Anstelle ein Netz einfacher linearer Klassifikatoren zu erzeugen, generiert man einen komplexen Merkmalsraum mit nichtlinearen Attributen. Dazu sei das Schema in Abb. 2.8 gegeben. Die Transformation ϕ muss so

Definiere eine Abbildung $\phi : \mathbb{R}^n \to \mathcal{M}$ (Merkmalsabbildung).
Bestimme für jeden Trainingspunkt $\boldsymbol{x}^i$ den zugehörigen N-dimensionalen Punkt $\boldsymbol{\phi}^i$ im Merkmalsraum $\mathcal{M}$.
Lerne eine Funktion $f_{\text{lin}}^{\mathcal{M}}(\boldsymbol{\phi}) := \sum_{k=1}^{N} w_k \phi_k + b$, die jedem Punkt $\boldsymbol{\phi} \in \mathcal{M}$ analog zu (2.8) im $\mathbb{R}^n$ einen Zielfunktionswert zuordnet. Verwende dazu die Menge der Trainingspaare $\{(\boldsymbol{\phi}^1, y_1), \ldots, (\boldsymbol{\phi}^{rd}, y_{rd})\}$.

Abbildung 2.8: SVM-Datentransformation in einen Merkmalsraum [26].

gewählt werden, dass die neuen Trainingspaare in $\mathcal{M}$ möglichst linear separabel sind, siehe dazu Abb. 2.9. Prinzipiell ist es immer möglich, für eine endliche Anzahl von Punkten durch Transformation in einen höher dimensionalen Raum eine lineare Trennbarkeit zu erzeugen [25]. Die Idee der Trennbarkeit von transformierten Daten ist dahingehend kri-

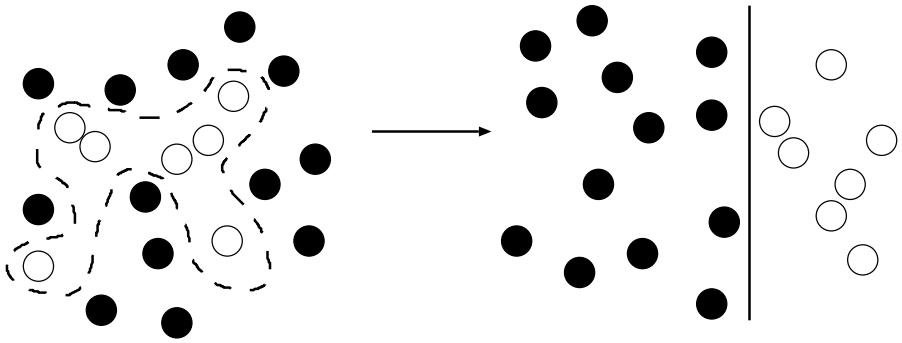

Abbildung 2.9: Lineare Trennbarkeit durch Datentransformation (vereinfachte zweidimensionale Darstellung).

tisch, als dass es bei komplizierten Datensätzen keinerlei Vorgehensweise gibt, um eine geeignete Transformation zu finden. An dieser Stelle kann man sich der Theorie der Kerne bedienen.

Definition 2.21 (Kern)**.** *Ein Kern ist eine Funktion $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, welche ein Skalarprodukt zwischen zwei Punkten in einem hochdimensionalen Raum $\mathcal{M}$ berechnet, d.h.*

$$k(\boldsymbol{x}^i, \boldsymbol{x}^j) = \langle \boldsymbol{\phi}^i, \boldsymbol{\phi}^j \rangle_{\mathcal{M}} \,. \tag{2.11}$$

Die Punkte $\boldsymbol{\phi}^i$ und $\boldsymbol{\phi}^j$ sind die Transformationen der ursprünglichen Daten $\boldsymbol{x}^i$ und $\boldsymbol{x}^j$.

Die Berechnung von Kernen erfolgt mittels der Punkte im ursprünglichen Raum $\mathbb{R}^n$ ohne

Kenntnis von $\mathcal{M}$ [102]. Aus Definition 2.21 folgt zudem, dass Kerne insbesondere symmetrisch sein müssen, d.h. $k(\boldsymbol{x}^i, \boldsymbol{x}^j) = k(\boldsymbol{x}^j, \boldsymbol{x}^i)$ gilt. Zu weiteren Eigenschaften von Kernen sowie Anforderungen an eine Kernfunktion verweisen wir auf [36]. Der sogenannte *kernel trick* [134] ermöglicht es, jeden auf Skalarprodukten basierenden Algorithmus nichtlinear zu erweitern. Ein Beispiel dafür ist – neben den SVM's – die kernbasierte Hauptkomponentenanalyse[6] [135].

Wie nutzen SVM's die Theorie der Kerne? Im Abschnitt 2.4 werden wir zeigen, dass die SVM-Algorithmen, welche auf den Trainingsdaten operieren, in letzter Instanz streng genommen ausschließlich auf Skalarprodukt-Werte zwischen den Punkten zurückgreifen. An diesen Stellen kann man Kerne einsetzen und ein implizites nichtlineares Lernverfahren entsteht. Die Berechnung der Skalarprodukte zielt bei den SVM's darauf ab, Ähnlichkeiten (lokale Abstände) zwischen Punkten zu messen. Durch die Verwendung eines Kerns wird die Ähnlichkeitsmessung implizit in den hochdimensionalen Raum $\mathcal{M}$ verlagert.

Definition 2.22 (Grammatrix, Kernmatrix [26])**.** *Als Grammatrix eines Trainingsdatensatzes bezeichnen wir im Folgenden die Matrix $\boldsymbol{K} \in \mathbb{R}^{rd,rd}$ mit den Einträgen*

$$K_{ij} := k(\boldsymbol{x}^i, \boldsymbol{x}^j) \quad (1 \leq i, j \leq rd) \; . \tag{2.12}$$

Jede positiv semidefinite Matrix kann als Grammatrix in einem speziellen Merkmalsraum interpretiert werden.

An dieser Stelle stellen wir kurz die von uns implementierten Kerne vor.

Definition 2.23 (Gauß-Kern [135])**.**

$$k^G(\boldsymbol{x}^i, \boldsymbol{x}^j) := \exp\left(-\frac{\|\boldsymbol{x}^i - \boldsymbol{x}^j\|^2}{2\sigma^2}\right) \qquad \sigma \in \mathbb{R}_+ \; . \tag{2.13}$$

Der Gauß-Kern, auch als RBF-Kern[7] bezeichnet, ist aufgrund der Verwandschaft zur Theorie der neuronalen Netze (RBF-Netze) sowie seiner hervorragenden Eigenschaften der bekannteste Kern überhaupt. Der Parameter σ wird als Weite des Kerns bezeichnet.

Definition 2.24 (Polynomialkern [135])**.**

$$k^P(\boldsymbol{x}^i, \boldsymbol{x}^j) := \left(c + \langle \boldsymbol{x}^i, \boldsymbol{x}^j \rangle\right)^d \qquad c \in \mathbb{R}_{+,0},\; d \in \mathbb{N}_+ \; . \tag{2.14}$$

Der Parameter d wird als Ordnung des Kerns bezeichnet. Der Polynomialkern wird als inhomogen bezeichnet, wenn $c > 0$ ist. Für die oft verwendete homogene Form gilt $c = 0$.

Die Kerne (2.13) und (2.14) können als Standardkerne angesehen werden und werden in vielen SVM-Programmen angeboten [19, 74]. Zusätzlich arbeiten wir auch mit neuen Kernen, die wir in [38] vorgestellt haben. Das ist zunächst der Slater-Kern.

[6] *kernel PCA*
[7] *radial basis function kernel*

Definition 2.25 (Slater-Kern [38])**.**

$$k^S(\boldsymbol{x}^i, \boldsymbol{x}^j) := \exp\left(-\frac{\|\boldsymbol{x}^i - \boldsymbol{x}^j\|}{2\sigma^2}\right) . \tag{2.15}$$

Wie auch in (2.13) muss $\sigma \in \mathbb{R}_+$ gelten. Für einen allgemeinen Überblick zur Slater-Funktion verweisen wir auf [141, 142].

Die Kerne (2.13)-(2.15) haben jeweils einen (oder zwei) Parameter. Der Wert des Parameters bestimmt den Merkmalsraum, in dem implizit gearbeitet wird. Auf das Problem der Parameterwahl bei SVM's werden wir in Kapitel 5 detailliert eingehen. In der folgenden Definition stellen wir einen weiteren Kern vor, der parameterfrei ist und ausschließlich für binäre Daten verwendet werden kann.

Definition 2.26 (Tanimoto-Kern)**.**

$$k^T(\boldsymbol{x}^i, \boldsymbol{x}^j) := \frac{I_{11}(\boldsymbol{x}^i, \boldsymbol{x}^j) + I_{00}(\boldsymbol{x}^i, \boldsymbol{x}^j)}{2I_{10}(\boldsymbol{x}^i, \boldsymbol{x}^j) + 2I_{01}(\boldsymbol{x}^i, \boldsymbol{x}^j) + I_{11}(\boldsymbol{x}^i, \boldsymbol{x}^j) + I_{00}(\boldsymbol{x}^i, \boldsymbol{x}^j)} . \tag{2.16}$$

*Der Tanimoto-Kern basiert auf dem bekannten Tanimoto-Koeffizienten [155]. Die Funktion $I_{a,b}(\cdot,\cdot)$ berechnet, wie oft der Wert $a \in \{0,1\}$ im ersten Vektor $\boldsymbol{x}^i$ auftritt während **gleichzeitig** der Wert $b \in \{0,1\}$ im Vektor $\boldsymbol{x}^j$ vorkommt. Die Reihenfolge ist von Bedeutung. Dieser Kern kann demnach nur für $0/1$-wertige Daten ausgewertet werden.*

Anhand der Definition der Funktion $I_{a,b}(\cdot,\cdot)$ ist klar, dass folgende Gleichungen gelten:

- $I_{11}(\boldsymbol{x}^i, \boldsymbol{x}^j) = (\boldsymbol{x}^i)^T\boldsymbol{x}^j$,
- $I_{10}(\boldsymbol{x}^i, \boldsymbol{x}^j) = (\boldsymbol{x}^i)^T\boldsymbol{x}^i - I_{11}(\boldsymbol{x}^i, \boldsymbol{x}^j)$,
- $I_{01}(\boldsymbol{x}^i, \boldsymbol{x}^j) = (\boldsymbol{x}^j)^T\boldsymbol{x}^j - I_{11}(\boldsymbol{x}^i, \boldsymbol{x}^j)$,
- $I_{00}(\boldsymbol{x}^i, \boldsymbol{x}^j) = n - I_{11}(\boldsymbol{x}^i, \boldsymbol{x}^j) - I_{10}(\boldsymbol{x}^i, \boldsymbol{x}^j) - I_{01}(\boldsymbol{x}^i, \boldsymbol{x}^j)$.

Damit kann man den Kern wie folgt umformen:

$$\begin{aligned} k(\boldsymbol{x}^i, \boldsymbol{x}^j) &= \frac{(\boldsymbol{x}^i)^T\boldsymbol{x}^j + n + (\boldsymbol{x}^i)^T\boldsymbol{x}^j - (\boldsymbol{x}^i)^T\boldsymbol{x}^i - (\boldsymbol{x}^j)^T\boldsymbol{x}^j}{n + (\boldsymbol{x}^j)^T\boldsymbol{x}^j + (\boldsymbol{x}^i)^T\boldsymbol{x}^i - 2(\boldsymbol{x}^i)^T\boldsymbol{x}^j} \\ &= \frac{n + 2(\boldsymbol{x}^i)^T\boldsymbol{x}^j - (\boldsymbol{x}^i)^T\boldsymbol{x}^i - (\boldsymbol{x}^j)^T\boldsymbol{x}^j}{n - 2(\boldsymbol{x}^i)^T\boldsymbol{x}^j + (\boldsymbol{x}^i)^T\boldsymbol{x}^i + (\boldsymbol{x}^j)^T\boldsymbol{x}^j} \\ &= \frac{n+t}{n-t} , \end{aligned}$$

wobei wir t definieren als $t := 2(\boldsymbol{x}^i)^T\boldsymbol{x}^j - (\boldsymbol{x}^i)^T\boldsymbol{x}^i - (\boldsymbol{x}^j)^T\boldsymbol{x}^j$. Das erleichtert die effiziente Implementierung.

Die Kerne (2.15) und (2.16) sind in der Umgebung von der Support-Vektor-Maschinen nicht bekannt, jedoch wurden sowohl der Slater-, als auch der Tanimoto-Koeffizient schon in der Pharmaforschung verwendet [111, 146]. Die beiden neuen Kerne wurden innerhalb des GALA-Projektes entwickelt und erfolgreich getestet [38, 83].

2.4 Modelle

In diesem Abschnitt werden die bekanntesten drei SVM-Lernmethoden vorgestellt. Wie schon im Abschnitt 2.2 erwähnt wurde, betrachten wir die Methode der Maximierung der Marge über die Minimierung der Norm des Gewichtsvektors einer linearen Klassifikationsfunktion welche, wie im Abschnitt 2.3 motiviert wurde, in einem hochdimensionalen Merkmalsraum operiert und dadurch zu einer nichtlinearen Funktion wird.

2.4.1 Modell der harten Trennung

Die Basis dieses Modells besteht in der Erkenntnis, dass der Generalisierungsfehler einer linearen Lernmaschine mittels der Marge der Hypothese bezüglich eines Trainingsdatensatzes abgeschätzt werden kann. Das Modell der harten Trennung kann nur dann verwendet werden, wenn die Daten im Merkmalsraum auch tatsächlich linear trennbar sind und ist für die Praxis nahezu irrelevant. Es bildet jedoch die Grundlage aller SVM-Modelle. Im folgenden Abschnitt werden wir einige Erweiterungen vorstellen, welche für reale Daten wichtig sind. Diese führen zu neuen Lernansätzen, welche dann auch für nichtseparierbare Trainingsdaten geeignet sind. Zunächst stellen wir das Grundmodell vor. Die Theorie der Datentransformation und der Kerne wird hier als bekannt vorausgesetzt.

Analog zu Abschnitt 2.2 über lineare Lerntheorie gilt, dass für eine lineare Hypothese

$$f_{\text{lin}}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle_{\mathcal{M}} + b \tag{2.17}$$

in einem Raum $\mathcal{M}$ mit konstanter funktionaler Marge $\gamma = 1$ die geometrische Marge γ^g angegeben werden kann als

$$\gamma^g = \frac{1}{\|\boldsymbol{w}\|_{\mathcal{M}}} \,. \tag{2.18}$$

Aus dem Zusammenhang (2.18) sowie der Definition der funktionalen Marge auf Seite 14 folgt die primale Aufgabe für das SVM-Modell mit Margenmaximierung über eine harte Trennung als

$$\min_{\substack{\boldsymbol{w}\in\mathcal{M} \\ b\in\mathbb{R}}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2_{\mathcal{M}} \tag{2.19}$$

unter den Nebenbedingungen

$$y_i(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}^i)\rangle_{\mathcal{M}} + b) \geq 1 \;\; (i = 1, \ldots, rd) \; .$$

Die Aufgabe enthält noch keine Skalarprodukte. Diese ergeben sich erst durch die Umwandlung von (2.19) in das zugehörige duale Optimierungsproblem. Die Umformulierung ist bei SVM's unproblematisch, da wir es ausschließlich mit quadratischen Optimierungsaufgaben zu tun haben, für welche der starke Dualitätssatz gilt [36]. Dieser sichert die Existenz einer globalen Lösung, die sowohl über das primale, als auch über das duale Problem berechnet werden kann. Die Lagrange-Funktion für die Aufgabe (2.19) hat offensichtlich die Form

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) := \frac{1}{2}\|\boldsymbol{w}\|^2_{\mathcal{M}} + \sum_{i=1}^{rd} \alpha_i \left(1 - y_i \left(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}^i)\rangle_{\mathcal{M}} + b\right)\right) \; . \tag{2.20}$$

Aus den Ableitungen der Lagrange-Funktion (2.20) nach $\boldsymbol{w}$ und b ergeben sich die notwendigen Bedingungen für ein Minimum von (2.19) als

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{rd} y_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}^i) \; , \\ 0 &= \sum_{i=1}^{rd} y_i \alpha_i \; . \end{aligned} \tag{2.21}$$

Die duale Zielfunktion, welche durch Einsetzen dieser Gleichungen in (2.20) entsteht, hat die Form

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^{rd}} \quad \sum_{i=1}^{rd} \alpha_i - \frac{1}{2} \sum_{i=1}^{rd} \sum_{j=1}^{rd} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{\phi}(\boldsymbol{x}^i), \boldsymbol{\phi}(\boldsymbol{x}^j)\rangle_{\mathcal{M}} \; . \tag{2.22}$$

Mittels (2.11) entsteht die für uns interessante Zielfunktion

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^{rd}} \quad \sum_{i=1}^{rd} \alpha_i - \frac{1}{2} \sum_{i=1}^{rd} \sum_{j=1}^{rd} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}^i, \boldsymbol{x}^j) \; . \tag{2.23}$$

Die zugehörigen Nebenbedingungen lauten

$$\begin{aligned} \boldsymbol{\alpha}^T \boldsymbol{y} &= 0 \, , \\ \boldsymbol{\alpha} &\geq \mathbf{0} \, . \end{aligned} \tag{2.24}$$

Für die Lösung $\boldsymbol{w}^*$ von (2.19) gilt wegen (2.21) und der starken Dualität der konvexen Optimierung [53]

$$w_k^* := \sum_{i=1}^{rd} y_i \alpha_i^* \phi_k(\boldsymbol{x}^i) \quad (k = 1, \ldots, N) \,. \tag{2.25}$$

Support-Vektor-Lernen mittels Maximierung der Marge hat die beiden Effekte, dass zum Einen eine gute Generalisierungsfähigkeit der Hypothese erreicht und zum Anderen ein dünnbesetzter Lösungsvektor $\boldsymbol{\alpha}^*$ generiert wird. Letzteres zeigen wir im Abschnitt 2.5. Dieser führt dann zu einer sparsamen Hypothese der Form

$$\begin{aligned}
h^*(\boldsymbol{x}) &\overset{(2.9)}{=} \operatorname{sgn}\left(f^*(\boldsymbol{x})\right) \qquad (2.26)\\
&\overset{(2.17)}{=} \operatorname{sgn}\left(\sum_{k=1}^{N} w_k^* \phi_k(\boldsymbol{x}) + b^*\right)\\
&\overset{(2.25)}{=} \operatorname{sgn}\left(\sum_{k=1}^{N}\left(\sum_{i=1}^{rd} y_i \alpha_i^* \phi_k(\boldsymbol{x}^i)\right)\phi_k(\boldsymbol{x}) + b^*\right)\\
&= \operatorname{sgn}\left(\sum_{i=1}^{rd}\sum_{k=1}^{N} y_i \alpha_i^* \phi_k(\boldsymbol{x}^i)\phi_k(\boldsymbol{x}) + b^*\right)\\
&= \operatorname{sgn}\left(\sum_{i=1}^{rd} y_i \alpha_i^* \sum_{k=1}^{N} \phi_k(\boldsymbol{x}^i)\phi_k(\boldsymbol{x}) + b^*\right)\\
&= \operatorname{sgn}\left(\sum_{i=1}^{rd} y_i \alpha_i^* \langle \boldsymbol{\phi}(\boldsymbol{x}^i), \boldsymbol{\phi}(\boldsymbol{x})\rangle_{\mathcal{M}} + b^*\right)\\
&\overset{(2.11)}{=} \operatorname{sgn}\left(\sum_{i=1}^{rd} y_i \alpha_i^* k(\boldsymbol{x}^i, \boldsymbol{x}) + b^*\right) \qquad (2.27)\\
&= \operatorname{sgn}\left(\sum_{\substack{i \in \mathcal{RD}\\ \alpha_i^* > 0}} y_i \alpha_i^* k(\boldsymbol{x}^i, \boldsymbol{x}) + b^*\right) \,, \qquad (2.28)
\end{aligned}$$

und verwendet zur Klassifikation eines neuen Punktes nur bestimmte Trainingspaare $(\boldsymbol{x}^i, y_i)$. Darauf gehen wir später noch ein.

2.4.2 Modelle der weichen Trennung

Für die meisten Klassen von Kernen lassen sich Parameterwerte finden, die einen Trainingsdatensatz perfekt trennen [26]. Eine Hypothese ohne Trainingsfehler führt aber nicht automatisch zu guten Klassifikationen auf neuen Daten, da eine erzwungene Datentrennung sehr schnell zu Überanpassungseffekten führt, insbesondere wenn die Daten verrauscht oder fehlerhaft sind. Es kann auch Familien von Kernen geben, für die überhaupt keine Trennung der Daten realisierbar ist. Das Modell der harten Trennung versucht jedoch immer, eine fehlerfreie Klassifikation der Trainingsdaten zu gewährleisten. Es sollte daher bei Datensätzen aus der Praxis nur mit Vorsicht oder gar nicht eingesetzt werden.

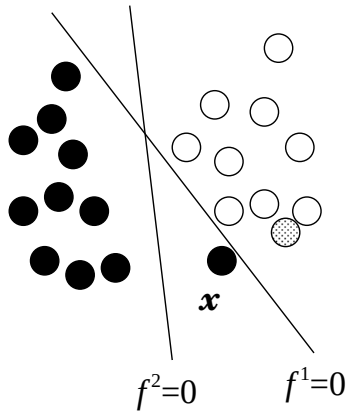

Abbildung 2.10: Die optimale Hyperebene f^1 wird bei Tolerierung eines Fehlers $\boldsymbol{x}$ zu f^2.

In Abbildung 2.10 ist erneut eine Klassifikationsfunktion f^1 dargestellt, welche die Daten perfekt trennt. Die dabei realisierte Marge ist annähernd Null, was auf schwache Generalisierungseigenschaften hindeutet. Ignoriert man hingegen den Punkt $\boldsymbol{x}$ und bestimmt dann eine neue Trennfunktion, erhält man f^2. Dabei wird auf Kosten eines falsch klassifizierten Punktes eine bessere Marge erreicht. Intuitiv entspricht die Lage von f^2 viel stärker der tatsächlichen Verteilung der Daten. Falls der Punkt $\boldsymbol{x}$ nun an der Stelle des grauen Punktes liegen wüde, ergäbe sich der Extremfall, bei dem es keine lineare Trennfunktion gibt, die das Konzept der Marge korrekt umsetzt.

Dieser Abschnitt stellt die sogenannten *softmargin*-Verfahren vor, welche derart verteilte Daten verarbeiten können. Das Hauptproblem beim Modell der harten Trennung ist, dass eine Marge maximiert wird, die erst dann einen positiven Wert annimmt, wenn alle Trainingspunkte korrekt klassifiziert sind. Die Modelle der weichen Trennung des Support-Vektor-Lernens addressieren diese Problematik und stellen eine erweiterte Lerntheorie zur Verfügung, welche es ermöglicht, gute Hypothesen für reale Daten hoher Komplexität zu erlernen, indem Lernfehler in gewissem Maße toleriert werden. Diese weichen Methoden sind robuster, weil sie eine größere Anzahl an Trainingspunkten zur Definition der trennenden Hyperebene nutzen. Im Folgenden stellen wir zwei Methoden der weichen Trennung

vor und geben analog zu Abschnitt 2.4.1 die sich ergebenden dualen Aufgaben an. Diese enthalten ebenfalls Werte von Kernfunktionen anstelle unbekannter Punkte und können in dieser Form direkt implementiert werden. Wie schon erwähnt wurde, sollten bei Abweichungen vom Modell der harten Trennung stets Nutzen und Kosten abgewägt werden, sodass Verletzungen der Marge bewertet werden müssen.

Für alle Trainingspunkte werden die auf Seite 17 definierten Schlupfvariablen ξ_i für eine Zielmarge von 1 betrachtet. Sei f eine Zielfunktion der Form (2.17), dann gilt für alle $i = 1, \ldots, rd$:

$$\xi_i \;=\; \max\left\{1 - y_i \cdot \left(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}^i)\rangle_{\mathcal{M}} + b\right);\, 0\right\} \;.$$

Bei den weichen Modellen werden Verletzungen der Marge toleriert. Die Werte der Schlupfvariablen – die jeweiligen Höhen der Verletzungen der Marge – gehen in die Optimierung mit ein, indem eine Norm des Vektors $\boldsymbol{\xi}$ berechnet, mit einem Parameter $C \in \mathbb{R}_+$ gewichtet und zur Zielfunktion hinzuaddiert wird. Dabei gibt es zwei Ansätze, die im Folgenden beschrieben werden. Beide Methoden lassen sich mittels der Generalisierungstheorie motivieren (vgl. [36], Abschnitt 3.4).

2.4.2.1 Das L_1-Norm-Modell

Bei diesem Modell nehmen die Werte der Schlupfvariablen über die Manhattan-Norm von $\boldsymbol{\xi}$ Einfluß. Das primale Optimierungsproblem des L_1-Norm-Ansatzes hat die Form

$$\min_{\substack{\boldsymbol{w}\in\mathcal{M}\\ b\in\mathbb{R}\\ \boldsymbol{\xi}\in\mathbb{R}^{rd}}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2_{\mathcal{M}} + C\sum_{i=1}^{rd}\xi_i \tag{2.29}$$

mit den Nebenbedingungen

$$\begin{aligned} y_i(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}^i)\rangle_{\mathcal{M}} + b) &\geq 1 - \xi_i \quad (i = 1, \ldots, rd)\,, \\ \xi_i &\geq 0 \qquad (i = 1, \ldots, rd)\,. \end{aligned} \tag{2.30}$$

Die Lagrange-Funktion kann angegeben werden als

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \;:=\; & \frac{1}{2}\|\boldsymbol{w}\|^2_{\mathcal{M}} + \sum_{i=1}^{rd}\alpha_i\left(1 - \xi_i - y_i\left(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}^i)\rangle_{\mathcal{M}} + b\right)\right) \\ & + C\sum_{i=1}^{rd}\xi_i - \sum_{i=1}^{rd}\beta_i\xi_i \;. \end{aligned} \tag{2.31}$$

Aus den Ableitungen der Lagrange-Funktion nach $\boldsymbol{w}$, b und $\boldsymbol{\xi}$ ergeben sich

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{rd} y_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}^i) \,, \\ 0 &= \sum_{i=1}^{rd} y_i \alpha_i \,, \\ \boldsymbol{C} &= \boldsymbol{\alpha} + \boldsymbol{\beta} \,. \end{aligned}$$

Die duale Aufgabe hat dann die Form

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^{rd}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{rd} \alpha_i - \frac{1}{2} \sum_{i=1}^{rd} \sum_{j=1}^{rd} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}^i, \boldsymbol{x}^j) \tag{2.32}$$

unter den Nebenbedingungen

$$\begin{aligned} \boldsymbol{\alpha}^T \boldsymbol{y} &= 0 \,, \\ \boldsymbol{\alpha} &\geq \boldsymbol{0} \,, \\ \boldsymbol{\alpha} &\leq \boldsymbol{C} \,. \end{aligned} \tag{2.33}$$

2.4.2.2 Das L_2-Norm-Modell

Bei diesem Modell nehmen die Werte der Schlupfvariablen über die euklidische Norm von $\boldsymbol{\xi}$ Einfluß. Das primale Optimierungsproblem des L_2-Norm-Ansatzes hat die Form

$$\min_{\substack{\boldsymbol{w} \in \mathcal{M} \\ b \in \mathbb{R} \\ \boldsymbol{\xi} \in \mathbb{R}^{rd}}} \quad \frac{1}{2} \|\boldsymbol{w}\|_{\mathcal{M}}^2 + C \sum_{i=1}^{rd} \xi_i^2 \tag{2.34}$$

mit den Nebenbedingungen

$$y_i(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}^i) \rangle_{\mathcal{M}} + b) \geq 1 - \xi_i \;\; (i = 1, \ldots, rd) \,. \tag{2.35}$$

Die Lagrange-Funktion ist dann definiert als

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) := \frac{1}{2} \|\boldsymbol{w}\|_{\mathcal{M}}^2 + C \sum_{i=1}^{rd} \xi_i^2 + \sum_{i=1}^{rd} \alpha_i \left(1 - \xi_i - y_i \left(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}^i) \rangle_{\mathcal{M}} + b\right)\right) \,. \tag{2.36}$$

Aus den Ableitungen der Lagrange-Funktion nach $\boldsymbol{w}$, b und $\boldsymbol{\xi}$ ergeben sich

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{rd} y_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}^i) \,, \\ 0 &= \sum_{i=1}^{rd} y_i \alpha_i \,, \\ 2\boldsymbol{C} \cdot \boldsymbol{\xi} &= \boldsymbol{\alpha} \,. \end{aligned}$$

Die duale Aufgabe hat die Form

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^{rd}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{rd} \alpha_i - \frac{1}{2} \sum_{i=1}^{rd} \sum_{j=1}^{rd} y_i y_j \alpha_i \alpha_j \underbrace{\left(k\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) + \frac{1}{2C} \delta_{i,j} \right)}_{=: \tilde{k}(\boldsymbol{x}^i, \boldsymbol{x}^j)} \tag{2.37}$$

unter den Nebenbedingungen

$$\begin{aligned} \boldsymbol{\alpha}^T \boldsymbol{y} &= 0 \,, \\ \boldsymbol{\alpha} &\geq \mathbf{0} \,. \end{aligned}$$

Bemerkung 2.3. *Die Aufgabe (2.37) entspicht der Aufgabe (2.23) bis auf die modifizierte Kernfunktion*

$$\tilde{k}(\boldsymbol{x}^i, \boldsymbol{x}^j) := k(\boldsymbol{x}^i, \boldsymbol{x}^j) + \frac{\delta_{i,j}}{2C} \,.$$

Im Allgemeinen wird in der Literatur die Konstante als $\delta_{i,j}/C$ angegeben [26]. Das ist nicht kritisch und entspricht einem modifizierten Strafparameter $\tilde{C} := C/2$ in der Aufgabe (2.34). Da wir im Abschnitt 7.3.2 C-Werte für beide Modelle vergleichen werden, vermeiden wir diese Ungenauigkeit bei der Implementierung.

Das L_2-Norm-Modell wird relativ selten genutzt. Das liegt zum Einen daran, dass es den Ruf hat, zuviele Support-Vektoren zu produzieren und zum Anderen sind L_1-Norm-Implementierungen deutlich besser verfügbar. Mit der Frage, welches Modell zu bevorzugen ist, haben wir uns in [44] auseinander gesetzt. Die Ergebnisse werden in Kapitel 7 präsentiert.

2.5 Schwellwert

Die gesuchte Zielfunktion, welche die Basis für die SVM-Hypothese bildet, hat gemäß (2.28) sowohl für das Modell der harten, als auch bei den weichen Trennungen, die Form

$$f^*(\boldsymbol{x}) = \sum_{\substack{i \in \mathcal{RD} \\ \alpha_i^* > 0}} y_i \alpha_i^* k(\boldsymbol{x}^i, \boldsymbol{x}) + b^* \ . \tag{2.38}$$

Mittels der Lösung eines quadratischen Problems der Form (2.23), (2.32) oder (2.37) wird der Vektor $\boldsymbol{\alpha}^*$ erzeugt. Darüber hinaus ist noch der Wert für b^* zu bestimmen. Die dualen Optimierungsaufgaben beinhalten diesen Schwellwert nicht. Ein Zugang zu b^* ist jedoch über die Dualitätstheorie möglich. Lösungen $\boldsymbol{\alpha}^*$ bzw. $\{\boldsymbol{w}^*, b^*\}$ der primalen bzw. dualen Optimierungsaufgabe müssen die sogenannten Karush-Kuhn-Tucker-Optimalitätsbedingungen (KKT-Bedingungen) [26] erfüllen.[8] Für den Fall harter Trennung lauten diese Bedingungen:

$$\alpha_i^* \left[y_i \cdot f^*(\boldsymbol{x}^i) - 1 \right] = 0 \quad (\forall i \in \mathcal{RD}) \ . \tag{2.39}$$

In (2.39) kann man erkennen, dass für alle Trainingspunkte i gilt:

$$\alpha_i^* > 0 \Longleftrightarrow y_i \cdot f^*(\boldsymbol{x}^i) = 1 \Longleftrightarrow \gamma_i = 1 \ .$$

Punkte mit positiven Lagrange-Multiplikatoren liegen demnach genau auf der Grenze des Margenbereiches. Diese Punkte werden Support-Vektoren genannt und geben den SVM's ihren Namen. Sie sind, wie in Abbildung 2.11 dargestellt, verantwortlich für die Lage der trennenden Hyperebene. In allen anderen Fällen gilt $\alpha_i^* = 0$ und die zugehörigen Trainingspunkte liegen in positiver Entfernung von der Margengrenze.

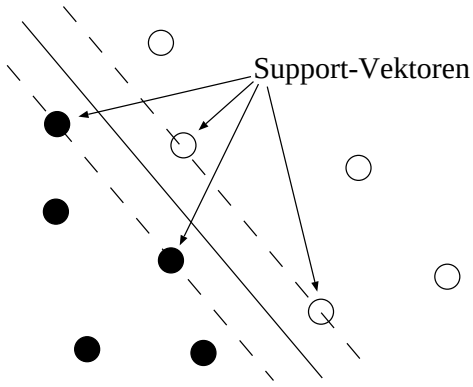

Abbildung 2.11: Support-Vektoren an der Margengrenze.

Definition 2.27. *Die Menge $\mathcal{SV}$ der Support-Vektoren ist definiert als*

$$\mathcal{SV} := \{i \in \mathcal{RD} : \alpha_i^* > 0\} \ .$$

[8] Früher sagte man Kuhn-Tucker-Bedingungen, jedoch hat Karush diese Bedingungen 12 Jahre vor Kuhn und Tucker völlig unabhängig formuliert.

Anwendung von (2.38) auf (2.39) liefert

$$\alpha_i^* \left[y_i \cdot \left(\sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) + b^* \right) - 1 \right] = 0 \quad (\forall i \in \mathcal{RD}) \; . \tag{2.40}$$

Jedes $i \in \mathcal{SV}$ kann daher zur Berechnung von b^* herangezogen werden mittels

$$b^* = y_i - \sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) \quad (i \in \mathcal{SV}) \; . \tag{2.41}$$

Für die Fälle weicher Trennung ergeben sich ähnliche Vorgehensweisen. Zunächst definieren wir zwei wichtige Mengen.

Definition 2.28. *Die Menge der Support-Vektoren für das L_1-Norm-Modell teilt sich in freie und beschränkte Support-Vektoren [135].*

- *$\mathcal{FSV}$ sei die Menge der freien Support-Vektoren, die definiert ist als*

$$\mathcal{FSV} := \{ i \in \mathcal{SV} : \alpha_i^* < C \} \; .$$

- *$\mathcal{BSV}$ sei die Menge der beschränkten Support-Vektoren, die definiert ist als*

$$\mathcal{BSV} := \{ i \in \mathcal{SV} : \alpha_i^* = C \} \; .$$

Für alle Trainingspunkte $i \in \mathcal{RD}$ lauten die KKT-Bedingungen:

$$\begin{aligned}
\alpha_i^* \left[y_i \cdot f^*(\boldsymbol{x}^i) - 1 + \xi_i^* \right] &= 0 \qquad \text{und} \\
\xi_i^* \cdot (\alpha_i^* - C) &= 0 \qquad (L_1\text{-Norm}) \; , \\
&\text{bzw.} \\
\alpha_i^* \left[y_i \cdot f^*(\boldsymbol{x}^i) - 1 + \xi_i^* \right] &= 0 \qquad \text{und} \\
\alpha_i^* &= C \cdot \xi_i^* \qquad (L_2\text{-Norm}) \; .
\end{aligned}$$

Daraus folgt wegen $y_i^{-1} = y_i$ für den L_1-Norm-Ansatz

$$b^* = y_i - \sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) \quad \forall i \in \mathcal{FSV} \; , \tag{2.42}$$

denn für $\alpha_i^* \in (0, C)$ muss $\xi_i^* = 0$ gelten. Wir schließen zusätzlich zu den nicht-Support-Vektoren auch die beschränkten Support-Vektoren aus [135], d.h. (2.41) gilt hier nur für $0 < \alpha_i^* < C$. Beachte, die Summe läuft dennoch über alle $\alpha_j^* > 0$, da es sich um

die Auswertung der Zielfunktion ohne b^* handelt. Dazu werden alle positiven Lagrange-Multiplikatoren benötigt.

Für den L_2-Norm-Ansatz ergibt sich

$$b^* = y_i - \frac{\alpha_i^* y_i}{C} - \sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) \quad \forall i \in \mathcal{SV} \, . \tag{2.43}$$

Eine im Gegensatz zu (2.41) robustere Methode zur Berechnung von b^* stellt der Durchschnitt b_r^* über alle zur Verfügung stehenden Support-Vektoren dar. Wir definieren

$$b_r^* := \frac{\sum_{i \in \mathcal{SV}} \left(y_i - \sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) \right)}{|i \in \mathcal{SV}|} \tag{2.44}$$

für den Fall harter Trennung. Für die Modelle der weichen Trennung erhält man analog

$$b_r^* = \frac{\sum_{i \in \mathcal{FSV}} \left(y_i - \sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) \right)}{|i \in \mathcal{FSV}|} \, , \quad \text{bzw.} \tag{2.45}$$

$$b_r^* = \frac{\sum_{i \in \mathcal{SV}} \left(y_i - \frac{\alpha_i^* y_i}{C} - \sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) \right)}{|i \in \mathcal{SV}|} \, . \tag{2.46}$$

Diese Berechnung ist zwar etwas aufwändiger, aber dennoch unproblematisch, da sie erst nach dem SVM-Training stattfindet.

Abschließend wird in Abbildung 2.12 für positive Punkte exemplarisch gezeigt, wo sich Support-Vektoren bei Modellen weicher Trennung befinden können und welche Auswirkungen die Lage der Punkte auf die Summierung von Fehlern in den Modellen L_1-Norm und L_2-Norm hat. Für positive Punkte gelten

- $\xi_i = 0$ für Punkte der Lage (a),
- $\xi_i \in (0, 1)$ für Punkte der Lage (b),
- $\xi_i = 1$ für Punkte der Lage (c),
- $\xi_i > 1$ für Punkte der Lage (d) oder in weiterer Entfernung.

Analoges gilt für negative Punkte. Man kann erkennen, dass die Summierung der Schlupfvariablen in (2.29) und (2.34) einen zweigeteilten Effekt hat:

- Schlupfvariablen mit Werten kleiner als 1 werden durch die L_1-Norm stärker bewertet, wohingegen

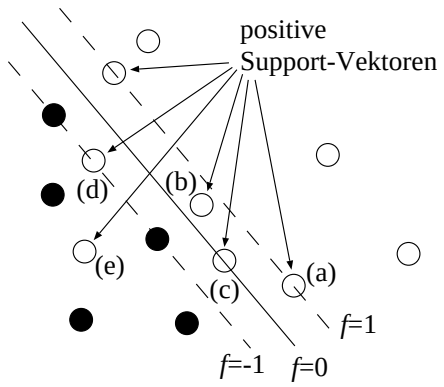

Abbildung 2.12: Positive Support-Vektoren bei Modellen weicher Trennung.

- Schlupfvariablen mit Werten größer als 1 durch die L_2-Norm stärker bewertet werden.

Bei den Modellen weicher Trennung liegen die Support-Vektoren nicht nur an der Grenze des Margenbereiches, sondern auch innerhalb des trennenden Bandes sowie auf der „anderen Seite“ der Hyperebene. Letztere Punkte sind echte Klassifikationsfehler, wohingegen wir die Punkte mit $\xi_i \in (0, 1)$ als schwache Klassifikationen bezeichnen. In den Kapiteln 4 und 5 werden wir auf diese Aspekte eingehen.

2.6 Zusammenfassung

In diesem Kapitel haben wir Grundlagen für das überwachte Lernen mit Support-Vektor-Maschinen vorgestellt. Die wichtigsten Eckpunkte dabei waren

- generelle Bewertung eines Klassifikators,
- lineares Lernen auf Trainingsdaten,
- Konzept der Marge zur strukturellen Risikominimierung,
- Konvexität und starke Dualität,
- Einbettung von nichtlinearen Kernen und
- sparsame Hypothese aus Support-Vektoren.

Weitere Details zu Support-Vektor-Maschinen und der zugrunde liegenden Lerntheorie sind in [26, 135, 151] zu finden. Wir haben gezeigt, dass die gesuchte Hypothesenfunktion mittels der Lösung eines quadratischen Optimierungsproblems sowie der Berechnung eines Schwellwertes generiert werden kann. Für die Implementierung einer Support-Vektor-Maschine zur Klassifikation von Daten stellt sich daher die Frage nach einem effizienten Algorithmus zur Lösung des quadratischen Optimierungsproblems der Form (2.23),

(2.32) oder (2.37). Im folgenden Kapitel betrachten wir einen effizienten Algorithmus zur Lösung der L_1-Norm-Aufgabe (2.32). Dieser Algorithmus wird Grundlage für alle weiteren Betrachtungen, insbesondere für die Parallelisierung in Kapitel 6, sein. In Kapitel 4 werden wir darauf eingehen, wie man den von uns beschriebenen Algorithmus sowohl für das L_2-Norm-Modell, als auch für das Modell harter Trennung flexibel nutzbar machen kann. Die notwendigen Änderungen der Software beschreiben wir im Abschnitt 4.2.1. In diesem Zusammenhang werden wir uns mit der Frage auseinandersetzen, welches der Modelle zu bevorzugen ist. Das Modell der harten Trennung ist für reale Datensätze oft nicht geeignet. Sind in den Daten beispielsweise Punkte mit dem falschen Label enthalten, sind die Daten allgemein verrauscht, gibt es keine Funktionalität in den Daten oder sind die SVM-Parameter falsch gewählt, ist es entweder überhaupt nicht möglich, eine perfekte Trennung durchzuführen, oder aber die Trennung wird überangepasst sein. Beim Vergleich der Modelle weicher Trennung kommen andere Aspekte ins Spiel. Als Kriterien lassen sich beispielsweise die Rechenzeit der Verfahren, die Anzahl der Support-Vektoren und die Genauigkeit der Hypothese auf Trainings- und Testdaten nennen. Eine Studie zu diesem Aspekt der SVM's stellen wir im Abschnitt 7.3.2 vor.

Kapitel 3

Zerlegungsalgorithmus

In Kapitel 2 wurden die Grundlagen von Support-Vektor-Maschinen vorgestellt. Es wurde gezeigt, auf welchen Ideen die Klassifikation basiert und wie sich eine Hypothesenfunktion zusammensetzt. Das Erlernen einer solchen Funktion setzt einen Trainingsdatensatz und einige Parameterwerte voraus, auf deren Grundlage ein quadratisches Optimierungsproblem und eine Gleichung zur Bestimmung des Schwellwertes gelöst werden.

Der Zerlegungsalgorithmus, welchen wir im Rahmen dieser Arbeit einsetzen, wird typischerweise zur Lösung des SVM-Trainingsproblems verwendet. Er basiert auf der Idee, dass das Gesamtproblem auch gelöst werden kann, indem iterativ auf Teilen des Lösungsvektors optimiert wird. Der Zerlegungsalgorithmus ist zunächst nur ein allgemeines Konstrukt, sodass nicht vorgegeben ist, welche Teile in den Iterationen ausgewählt werden und nach welchem Schema die Teiloptimierung stattfinden soll. Wir betrachten in unserer Arbeit eine spezielle Projektionsmethode zur Lösung der quadratischen Teilaufgaben, welche ihrerseits auch ein iteratives Schema aufweist. Es wird schrittweise optimiert, wobei in jedem Schritt ein transformiertes Optimierungsproblem mit Diagonalmatrix entsteht. Der Vorteil dabei ist, dass sich derartige Optimierungsaufgaben mit einem schnellen Löser bearbeiten lassen, der die spezielle Diagonalform ausnutzt. Dieser stellt dann die innerste Ebene dar. Das Szenario ist in Abbildung 3.1 dargestellt, wobei die Nummern den Abschnitten entsprechen, in denen die Methoden vorgestellt werden.

In diesem Kapitel betrachten wir die duale Maximierungsaufgabe (2.32) einer L_1-Norm-SVM

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^{rd}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{rd} \alpha_i - \frac{1}{2} \sum_{i=1}^{rd} \sum_{j=1}^{rd} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}^i, \boldsymbol{x}^j) \tag{3.1}$$

mit den Nebenbedingungen

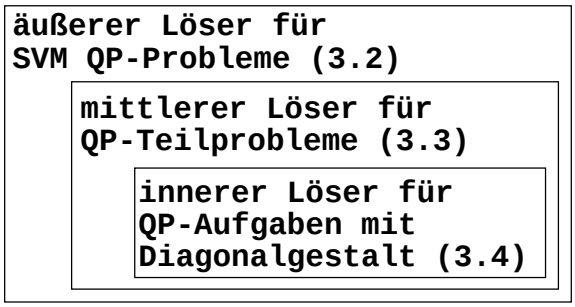

Abbildung 3.1: Schema zum Zerlegungsalgorithmus mit Zuordnung zu den Abschnitten.

$$\begin{aligned} \boldsymbol{y}^T\boldsymbol{\alpha} &= 0\,, \\ \boldsymbol{\alpha} &\geq \mathbf{0}\,, \\ \boldsymbol{\alpha} &\leq \boldsymbol{C}\,. \end{aligned} \tag{3.2}$$

Definition 3.1. *Im weiteren Verlauf der Arbeit werden wir im Allgemeinen nicht mehr mit der Grammatrix $\boldsymbol{K}$, sondern mit der Matrix $\boldsymbol{Q}$ mit den Einträgen*

$$Q_{ij} := y_i y_j K_{ij} \quad (1 \leq i, j \leq rd) \tag{3.3}$$

arbeiten.

Bemerkung 3.1. *Für zulässige Kerne (Mercer-Kerne [102]) ist die Matrix $\boldsymbol{K}$ positiv semidefinit. Das bedeutet, $\boldsymbol{K}$ ist symmetrisch und es gilt*

$$\boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^{rd}\,. \tag{3.4}$$

Daraus folgt direkt, dass $\boldsymbol{Q}$ ebenfalls positiv semidefinit ist, denn $\boldsymbol{Q}$ ist symmetrisch und es gilt

$$\begin{aligned} \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} &= \sum_{i=1}^{rd}\sum_{j=1}^{rd} x_i Q_{ij} x_j \overset{(3.3)}{=} \sum_{i=1}^{rd}\sum_{j=1}^{rd} x_i y_i K_{ij} y_j x_j \\ &\overset{\tilde{x}_i = x_i y_i}{=} \tilde{\boldsymbol{x}}^T\boldsymbol{K}\tilde{\boldsymbol{x}} \overset{(3.4)}{\geq} 0\,. \end{aligned}$$

Im Folgenden minimieren wir die Funktion $W^- := -W$, denn es gilt

$$\boldsymbol{\alpha}^* = \arg\max\,\{W(\boldsymbol{\alpha})\} \quad \Longleftrightarrow \quad \boldsymbol{\alpha}^* = \arg\min\,\{-W(\boldsymbol{\alpha})\}\,. \tag{3.5}$$

Die neue Minimierungsaufgabe lautet

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^{rd}} \quad W^-(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{1} \tag{3.6}$$

unter den Nebenbedingungen

$$\boldsymbol{y}^T \boldsymbol{\alpha} = 0\,, \tag{3.7}$$

$$\boldsymbol{\alpha} \geq \mathbf{0}\,, \tag{3.8}$$

$$\boldsymbol{\alpha} \leq \boldsymbol{C}\,. \tag{3.9}$$

Wir betrachten also ein quadratisches Optimierungsproblem mit oberen und unteren Schranken sowie einer einfachen linearen Nebenbedingung. Die Matrix $\boldsymbol{Q}$ – die Hessematrix des QP-Problems[9] – ist typischerweise dicht besetzt und wächst mit der Anzahl rd der verwendeten Trainingspunkte, sodass sich für große Klassifikationsprobleme die Frage nach der effizienten Lösung von (3.6) stellt. Die Lösung dieses Optimierungsproblems wird oft als der SVM-Flaschenhals bezeichnet und wirkt insbesondere bei sehr großen Daten abschreckend. Für große Datensätze passt die Matrix $\boldsymbol{Q}$ nicht mehr in den Speicher.[10] Zusätzlich ist die Berechnung der Elemente beispielsweise bei Verwendung des Gauß-Kerns sehr teuer.[11] In diesem Kapitel beschäftigen wir uns mit einem effizienten SVM-Trainingsalgorithmus, der auch Basis für die Parallelisierung in Kapitel 6 sein wird.

Die Grundidee des Verfahrens der Zerlegung [112] besteht darin, die Aufgabe (3.6) über eine Folge kleinerer Optimierungsaufgaben zu lösen. Damit wird es möglich, ein SVM-Training auch für große Daten durchzuführen, bei denen die Matrix $\boldsymbol{Q}$ nicht in den Speicher passt. Dabei wird der Vektor $\boldsymbol{\alpha}$ in jeder Iteration in einen aktiven und einen nichtaktiven Teil zerlegt. Die Optimierungsarbeit findet immer nur auf dem aktiven Teil des Vektors (Arbeitsmenge, Working-Set, Chunk) statt. Methoden dieser Art werden als Working-Set-Algorithmen bezeichnet.

Definition 3.2. *Zur Definition einer Arbeitsmenge wird der Vektor $\boldsymbol{\alpha}$ in einen aktiven und einen inaktiven Teil zerlegt. Im Folgenden werden die aktiven (dynamischen) Variablen mit einem d, alle nichtaktiven (festen) Variablen mit f gekennzeichnet. Die zugehörigen Indexmengen werden als $\mathcal{D}$ und $\mathcal{F}$ bezeichnet. Wir gehen davon aus, dass die Daten für theoretische Betrachtungen immer so sortiert vorliegen, dass in jeder Iteration alle aktiven α_i im oberen Teil des Vektors stehen, d.h.:*

$$\boldsymbol{\alpha} := \begin{pmatrix} \boldsymbol{\alpha}_\mathrm{d} \\ \boldsymbol{\alpha}_\mathrm{f} \end{pmatrix}.$$

Analog dazu werden auch der Vektor $\boldsymbol{y}$ und die Matrix $\boldsymbol{Q}$ umsortiert:

$$\boldsymbol{y} := \begin{pmatrix} \boldsymbol{y}_\mathrm{d} \\ \boldsymbol{y}_\mathrm{f} \end{pmatrix},$$

[9] *quadratic programming problem*

[10] Bei 20000 Trainingspunkten wird allein für die Kernmatrix bei 8 Bytes pro Eintrag (doppelt genau reell) ein Speichervolumen von 3.2GB benötigt.

[11] Publikationen über sehr schnelle SVM-Algorithmen basieren sehr oft nur auf linearen SVM's. Diese sind prinzipiell schnell, da nur ein einfaches Skalarprodukt berechnet wird. Weitere Bemerkungen zu linearen SVM's folgen in Kapitel 6.

$$\boldsymbol{Q} := \begin{pmatrix} \boldsymbol{Q}_{\mathrm{dd}} & \boldsymbol{Q}_{\mathrm{df}} \\ \boldsymbol{Q}_{\mathrm{fd}} & \boldsymbol{Q}_{\mathrm{ff}} \end{pmatrix}.$$

Offensichtlich sind $\boldsymbol{Q}_{\mathrm{dd}}$ und $\boldsymbol{Q}_{\mathrm{ff}}$ symmetrisch und für die gemischten, im Allgemeinen nicht quadratischen Matrizen[12] gilt

$$\boldsymbol{Q}_{\mathrm{fd}}^T = \boldsymbol{Q}_{\mathrm{df}} \,. \tag{3.10}$$

An dieser Stelle wollen wir angeben, wie sich das Problem (3.6) verändert, falls ein Teil des Vektors $\boldsymbol{\alpha}$ konstant ist. Wir formen zunächst die Zielfunktion um:

$$\begin{aligned} W^-(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\mathbf{1} \\ &= \frac{1}{2}\left(\boldsymbol{\alpha}_{\mathrm{d}}^T\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{\mathrm{d}} + \boldsymbol{\alpha}_{\mathrm{f}}^T\boldsymbol{Q}_{\mathrm{fd}}\boldsymbol{\alpha}_{\mathrm{d}} + \boldsymbol{\alpha}_{\mathrm{d}}^T\boldsymbol{Q}_{\mathrm{df}}\boldsymbol{\alpha}_{\mathrm{f}} + \boldsymbol{\alpha}_{\mathrm{f}}^T\boldsymbol{Q}_{\mathrm{ff}}\boldsymbol{\alpha}_{\mathrm{f}}\right) - \boldsymbol{\alpha}_{\mathrm{d}}^T\mathbf{1} - \boldsymbol{\alpha}_{\mathrm{f}}^T\mathbf{1} \,. \end{aligned}$$

Da der Vektor $\boldsymbol{\alpha}_{\mathrm{f}}$ konstant ist, können $\frac{1}{2}\boldsymbol{\alpha}_{\mathrm{f}}^T\boldsymbol{Q}_{\mathrm{ff}}\boldsymbol{\alpha}_{\mathrm{f}}$ und $\boldsymbol{\alpha}_{\mathrm{f}}^T\mathbf{1}$ bei der Optimierung entfallen und wir erhalten eine neue Funktion von $\boldsymbol{\alpha}_{\mathrm{d}}$. Diese hat die Form

$$W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{\mathrm{d}}) = \frac{1}{2}\boldsymbol{\alpha}_{\mathrm{d}}^T\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{\mathrm{d}} + \boldsymbol{\alpha}_{\mathrm{f}}^T\boldsymbol{Q}_{\mathrm{fd}}\boldsymbol{\alpha}_{\mathrm{d}} - \boldsymbol{\alpha}_{\mathrm{d}}^T\mathbf{1} \,. \tag{3.11}$$

Die angepassten Nebenbedingungen lauten

$$\boldsymbol{y}_{\mathrm{d}}^T\boldsymbol{\alpha}_{\mathrm{d}} = -\boldsymbol{y}_{\mathrm{f}}^T\boldsymbol{\alpha}_{\mathrm{f}} \,, \tag{3.12}$$

$$\boldsymbol{\alpha}_{\mathrm{d}} \geq \mathbf{0} \,, \tag{3.13}$$

$$\boldsymbol{\alpha}_{\mathrm{d}} \leq \boldsymbol{C} \,. \tag{3.14}$$

Die rechte Seite von (3.12) ist konstant. Jede der schon erwähnten Teilaufgaben wird von der Form (3.11) unter den Nebenbedingungen (3.12) bis (3.14) sein.

3.1 Größe der Arbeitsmenge

Die Größe der Arbeitsmenge wird im Folgenden als ws bezeichnet.[13] Sie ist ein entscheidender Parameter für die Trainingszeit einer Support-Vektor-Maschine. Für das Verfahren der Zerlegung, bei dem iterativ verkleinerte Optimierungsprobleme gelöst werden, beeinflusst ws die Anzahl der Optimierungsschritte sowie die Komplexität jedes einzelnen

[12] Nur für $|\mathcal{D}| = |\mathcal{F}|$ sind alle vier Matrizen quadratisch.

[13] abgeleitet von *working set*

Schrittes. Im Allgemeinen führt eine kleine Arbeitsmenge zu einer großen Anzahl schnell zu lösender Optimierungsaufgaben, während eine große Arbeitsmenge wenige große Optimierungsaufgaben nach sich zieht. Dabei ist die Gesamtlaufzeit jedoch nicht fest, sondern variiert. Die Größe ws ist nach oben begrenzt durch den verfügbaren Speicher des Rechensystems. Wir werden uns in Kapitel 7 mit der Frage nach einer optimalen Größe, insbesondere im Zusammenhang mit Parallelisierungsaspekten, auseinandersetzen. Zur Handhabung der Größe der Arbeitsmenge gibt es unterschiedliche Ansätze, die in diesem Abschnitt kurz vorgestellt werden sollen.

3.1.1 Chunking

Die Grundidee des Chunking [26] besteht in der Überlegung, dass nur die Support-Vektoren die Zielfunktion einer Support-Vektor-Maschine beeinflussen. Wenn man daher ausschließlich diese Vektoren betrachtet, erlernt man die gleiche Funktion, die man bei Verwendung aller Datenpunkte erhalten hätte. Die Support-Vektoren sind zu Beginn des Trainings natürlich nicht bekannt. Chunking setzt eine Methode um, welche iterativ Support-Vektoren bestimmt und zum Training verwendet. Die erste Arbeitsmenge wird zufällig ausgewählt. Nach dem Training auf dieser Menge verbleiben nur die Support-Vektoren im Chunk. Die erlernte Funktion wird auf den inaktiven Daten getestet. Die c Punkte, welche die KKT-Bedingungen am stärksten verletzen, werden dann in den Chunk aufgenommen. Der Parameter $c \in \mathbb{N}$ ist während der Laufzeit konstant. Schwierig beim Chunking ist eine Vorhersage der Größe der Arbeitsmenge. Typischerweise wird sie zu Beginn ansteigen, da sowohl neue Support-Vektoren gefunden werden, als auch die erlernte Funktion nicht optimal ist und so zu vielen Verletzungen der KKT-Bedingungen führt. Zum Ende verbleiben lediglich die Support-Vektoren im Chunk. Im Allgemeinen sinkt damit die Anzahl an Punkten in der Arbeitsmenge kurz vor Beendigung des Chunking. In Fällen, bei denen es viele Support-Vektoren gibt, beispielsweise bei sehr großen Datensätzen oder bei Daten mit sehr hohem prozentualen Anteil an Support-Vektoren, kann es dazu kommen, dass die Matrix $\boldsymbol{Q}_{\mathrm{dd}}$ nicht in den Speicher passt und das Verfahren nicht angewendet werden kann.

3.1.2 Decomposition

Die Methode Decomposition (Zerlegung) [91, 113], welche auch in dieser Arbeit betrachtet wird, ist dem Chunking sehr ähnlich. Der Unterschied liegt darin, dass die Arbeitsmenge in jeder Iteration die gleiche Größe ws hat. Die Konsequenz ist, dass für jeden Punkt, der in diese Menge aufgenommen wird, ein anderer entfernt werden muss. Für die Nutzung dieses Verfahrens ist eine sinnvolle Strategie dafür zu entwickeln, welche Punkte in der Arbeitsmenge durch neue Punkte ersetzt werden. Der große Vorteil des Decomposition liegt tatsächlich in der festen Größe der Arbeitsmenge. Man kann von Beginn an mit der gewünschten Anzahl an Punkten arbeiten, ohne befürchten zu müssen, dass die Dimension

der Matrix $\boldsymbol{Q}_{\mathrm{dd}}$ plötzlich anwächst. Wir werden uns im Abschnitt 3.3 ausführlich mit der Frage beschäftigen, nach welcher Methode die Optimierung der Zielfunktion auf der Arbeitsmenge durchgeführt werden soll. Wir lassen diese Problematik daher zunächst außer Acht und beschäftigen uns im Abschnitt 3.2 mit Fragen zur Aktualisierung der Arbeitsmenge zwischen den einzelnen Optimierungsschritten, zu den Optimalitätskriterien und Abbruchbedingungen sowie zu Effizienzsteigerungsmöglichkeiten und sinnvollen Modifikationen.

3.1.3 Sequential-Minimal-Optimization

Der sogenannte SMO-Algorithmus [118] ist ein interessanter und oft genutzter Spezialfall der Zerlegungsmethode. Hierbei wird die Größe der Arbeitsmenge auf 2 fixiert. Aus diesem Grund ist es nicht mehr notwendig, ein kompliziertes Optimierungsverfahren zur Arbeit auf der Arbeitsmenge anzuwenden, denn die Optimierung der Zielfunktion auf zwei Punkten erfolgt dank der linearen Nebenbedingungen analytisch [36]. Die Implementierung von SMO ist somit einfach zu realisieren. Die Problematik von SMO liegt also weniger bei den Optimierungschritten selbst, als vielmehr in der Wahl der Arbeitsmenge. Da die Arbeitsmenge so klein ist, muss diese sehr oft aktualisiert werden und für große Datensätze ist die Anzahl der Optimierungsschritte extrem hoch. Die Kosten, die man bei der analytischen Optimierung auf kleinen Mengen spart, gleichen sich dadurch aus, dass mehr Optimierungsschritte vorbereitet und durchgeführt werden müssen. Normalerweise werden für alle Punkte, die nicht in der letzten Arbeitsmenge waren, die Optimalitätskriterien geprüft. Das wäre in diesem Fall ein zu großer Aufwand. SMO nutzt daher Heuristiken zur Auswahl der neuen Arbeitsmenge [26]. Diese Heuristiken sind gut durchdacht, führen allerdings in ungünstigen Fällen zu unnötig hohen Rechenzeiten und sind deshalb in [80] verbessert worden. Dennoch ist der SMO-Algorithmus langsam und die Größe der Arbeitsmenge schließt eine Parallelisierung nahezu aus.

3.2 Äußere Schleife des Zerlegungsalgorithmus

In diesem Abschnitt stellen wir die Grundstruktur des Zerlegungsverfahrens zur Lösung der SVM-Trainingsaufgabe (3.6) in der von uns implementierten Form vor. Auf Modifikationen des Verfahrens, die spezielle Lernziele ermöglichen, gehen wir erst später in Kapitel 4 ein. Diese Trainingsmethode ist Grundlage für die in den Kapiteln 5 und 6 beschriebenen Möglichkeiten zu Parameteroptimierung und Parallelisierung und wird für die in Kapitel 7 präsentierten Tests verwendet.

Der Zerlegungsalgorithmus besteht aus einer Schleife, in der jeweils ein Problem mit der Zielfunktion (3.11) und den Nebenbedingungen (3.12)-(3.14) gelöst, die Optimalität geprüft und gegebenenfalls eine neue Arbeitsmenge bestimmt wird. Im Folgenden wird

gezeigt, wie die Arbeitsmengen aktualisiert werden sollten und mittels welcher Optimalitätskriterien man den Zerlegungsalgorithmus beenden kann. An dieser Stelle sei erwähnt, dass die Frage, wie die Zielfunktion auf der Arbeitsmenge optimiert werden soll (innerer Löser), erst in den Abschnitten 3.3 und 3.4 ausführlich behandelt wird, sodass wir diese Problematik hier ausblenden und in diesem Abschnitt davon ausgehen, dass die Optimierung auf jeder Arbeitsmenge problemlos verläuft und das optimale Ergebnis als $\boldsymbol{\alpha}_{\mathrm{d}}^{*}$ zur Verfügung gestellt wird.

3.2.1 Initialisierung

Wie für jedes andere iterative Optimierungsverfahren stellt sich die Frage nach einem geeigneten Startvektor $\boldsymbol{\alpha}^1$ für den Zerlegungsalgorithmus. Ein Startvektor $\boldsymbol{\alpha}^1$ des Zerlegungsalgorithmus muss offensichtlich die Nebenbedingungen der Aufgabe (3.6) erfüllen, das sind gemäß (3.12)-(3.14)

$$\sum_{i=1}^{rd} y_i \cdot \alpha_i^1 = 0 \qquad \text{und} \qquad \mathbf{0} \leq \boldsymbol{\alpha}^1 \leq \boldsymbol{C}. \tag{3.15}$$

Offensichtlich erfüllt der Punkt $\boldsymbol{a}^1 = \mathbf{0}$ diese Nebenbedingungen und stellt damit eine zulässige Wahl dar. Die Wahl des Nullvektors für den Start ist streng genommen nur dann sinnvoll, wenn die Lösung sehr wenige Support-Vektoren beinhalten wird, sodass viele der Nullen auch noch im Vektor $\boldsymbol{\alpha}^*$ wiederzufinden sind. In Fällen, bei denen es viele Support-Vektoren gibt, werden auch entsprechend mehr Einträge des Startvektors geändert. Der Decomposition-Algorithmus kann verbessert werden, indem ein Startvektor gewählt wird, der dem endgültigen Ergebnis mehr ähnelt als der Nullvektor. Leider hat man in den seltensten Fällen eine Idee, welche Punkte für die Klassentrennung verantwortlich sind und möglicherweise Support-Vektoren werden. Sobald man jedoch anfängt, eine Optimierung auf Teildaten durchzuführen, befindet man sich streng genommen schon mitten im Zerlegungsalgorithmus. Eine interessante Idee zur Wahl des Startpunktes, welche man zumindest in einer parallelen Rechenumgebung sinnvoll nutzen kann, wurde kürzlich in [137] entwickelt und wird jetzt kurz vorgestellt.

Die Indexmenge $\{1, \ldots, rd\}$ der gesamten Trainingsdaten wird in p Teile zerlegt, wobei p die Anzahl der zur Verfügung stehenden Prozessoren sei. Dann wird die Aufgabe (3.6) auf jedem Prozessor i ($i \in \{1, .., p\}$) für die ihm zugeteilten Daten gelöst, völlig unabhängig von den anderen, ohne die Nebenbedingungen anzupassen. Vielmehr wird simuliert, dass es die anderen Daten überhaupt nicht gibt. Wir bezeichnen die jeweiligen Lösungen als $\hat{\boldsymbol{\alpha}}_i$. Der Startvektor ergibt sich dann durch Zusammenführung dieser Teillösungen als

$$\left(\boldsymbol{\alpha}^1\right)^T = \left(\hat{\boldsymbol{\alpha}}_1^T, .., \hat{\boldsymbol{\alpha}}_p^T\right) \ .$$

Die p Aufgaben sind völlig unabhängig voneinander und können auf einem Mehrprozessorsystem gleichzeitig gelöst werden. Bei einer ausreichenden Anzahl an Prozessoren wird die Größe der einzelnen Probleme moderat sein. Sie sollten direkt mit einem QP-Löser behandelt werden, siehe dazu beispielsweise Abschnitt 3.3. Ein oft verwendeter Löser ist *LOQO* [150], der jedoch nur für bis zu 1000 Punkte verwendet werden kann. Der so generierte Startvektor impliziert natürlich auch die Wahl des ersten Working-Sets. Es werden diejenigen Indizes j ausgewählt, für welche der zugehörige Wert α_j^1 streng positiv ist. Diese Möglichkeit hat man normalerweise nicht und wählt die Arbeitsmenge zufällig aus. Wie schon erwähnt, kann man dieses Verfahren nicht für $p = 1$ benutzen, da die Lösung eines einzigen Teilproblems der Lösung der gesamten Aufgabe entsprechen würde. Hat man nur sehr wenige Prozessoren zur Verfügung, kann natürlich auch jeder Prozessor mehrere solcher Teilaufgaben hintereinander durchführen. Dabei ist jedoch zu beachten, dass die Rechenzeit des Gesamtverfahrens unter Umständen unverhältnismässig ansteigen kann. Im seriellen Fall ist die Methode komplett abzulehnen.

Definition 3.3 (Notationen)**.** *In Folgenden bezeichne $\boldsymbol{\alpha}^k$ den Vektor aller Lagrange-Multiplikatoren vor dem k-ten Optimierungsschritt. Dieser wird durchgeführt mit einem Teil dieses Vektors – $\boldsymbol{\alpha}_{\mathrm{d}}^k$. Die Lösung der Optimierungsaufgabe sei $\boldsymbol{\alpha}_{\mathrm{d}}^{k+1}$, sodass sich der neue Gesamtvektor $\boldsymbol{\alpha}^{k+1}$ zusammensetzt als*

$$\boldsymbol{\alpha}^{k+1} := \begin{pmatrix} \boldsymbol{\alpha}_{\mathrm{d}}^{k+1} \\ \boldsymbol{\alpha}_{\mathrm{f}}^{k} \end{pmatrix} . \tag{3.16}$$

3.2.2 Optimalitätskriterien

Die iterative Aktualisierung der Arbeitsmenge und die Lösung von (3.11) stellen nur dann eine effiziente Lösungsmethode für (3.6) dar, wenn nach jedem Schritt ein Optimalitätskriterium darüber entscheidet, welche Einträge von $\boldsymbol{\alpha}_{\mathrm{d}}$ im nächsten Schritt nicht weiter bearbeitet werden müssen und welche Teile von $\boldsymbol{\alpha}_{\mathrm{f}}$ als nächstes optimiert werden. Das Optimierungsproblem (3.6) ist konvex, denn $\boldsymbol{Q}$, die Hessematrix der Zielfunktion, ist nach Folgerung 3.1 positiv semidefinit und alle Nebenbedingungen sind linear[14]. Für konvexe Optimierungsaufgaben sind die Karush-Kuhn-Tucker-Bedingungen notwendig und hinreichend dafür, dass $\boldsymbol{\alpha}^*$ optimal ist [57]. Diese Bedingungen haben für den Fall von $2 \cdot rd$ Schranken- und einer Gleichheitsnebenbedingung die Form [10, 36]:

Es existieren $\boldsymbol{\lambda}^* \in \mathbb{R}^{rd}$, $\boldsymbol{\mu}^* \in \mathbb{R}^{rd}$ und $\nu^* \in \mathbb{R}$, sodass

[14] Lineare Funktionen sind sowohl konvex, als auch konkav.

$$
\begin{aligned}
\mathbf{0} &= \nabla W^{-}(\boldsymbol{\alpha}^*) + \sum_{i=1}^{rd} \lambda_i^* \nabla g_i^1(\boldsymbol{\alpha}^*) + \sum_{i=1}^{rd} \mu_i^* \nabla g_i^2(\boldsymbol{\alpha}^*) + \nu^* \nabla h(\boldsymbol{\alpha}^*) \,, \\
\mathbf{0} &\geq \boldsymbol{g}^1(\boldsymbol{\alpha}^*) \,, \\
\mathbf{0} &\geq \boldsymbol{g}^2(\boldsymbol{\alpha}^*) \,, \\
0 &= h(\boldsymbol{\alpha}^*) \,, \\
\mathbf{0} &= \boldsymbol{\lambda}^* \boldsymbol{g}^1(\boldsymbol{\alpha}^*) \,, \\
\mathbf{0} &= \boldsymbol{\mu}^* \boldsymbol{g}^2(\boldsymbol{\alpha}^*) \,, \\
\mathbf{0} &\leq \boldsymbol{\lambda}^* \,, \\
\mathbf{0} &\leq \boldsymbol{\mu}^*
\end{aligned}
\tag{3.17}
$$

gelten. Wir bezeichnen die Schrankennebenbedingungen (3.8) und (3.9) mit g_i^1 und g_i^2 $(i = 1, .., rd)$ und die Gleichheitsnebenbedingung (3.7) mit h, d.h. die Nebenbedingungen von (3.6) werden zu

$$
\begin{aligned}
g_i^1(\boldsymbol{\alpha}) &:= -\alpha_i \leq 0 \quad (i = 1, .., rd) \,, \\
g_i^2(\boldsymbol{\alpha}) &:= \alpha_i - C \leq 0 \quad (i = 1, .., rd) \,, \\
h(\boldsymbol{\alpha}) &:= \boldsymbol{y}^T \boldsymbol{\alpha} = 0 \,.
\end{aligned}
$$

Wir setzen diese Funktionen und deren Ableitungen direkt in (3.17) ein, um das System der Karush-Kuhn-Tucker-Bedingungen in einer für uns interpretierbaren Form zu erhalten:

$$
\begin{aligned}
\mathbf{0} &= \nabla W^{-}(\boldsymbol{\alpha}^*) - \boldsymbol{\lambda}^* + \boldsymbol{\mu}^* + \nu^* \boldsymbol{y} \,, && (3.18) \\
\mathbf{0} &\geq -\boldsymbol{\alpha}^* \,, && (3.19) \\
\mathbf{0} &\geq \boldsymbol{\alpha}^* - \boldsymbol{C} \,, && (3.20) \\
0 &= \boldsymbol{y}^T \boldsymbol{\alpha}^* \,, && (3.21) \\
\mathbf{0} &= -\boldsymbol{\lambda}^* \boldsymbol{\alpha}^* \,, && (3.22) \\
\mathbf{0} &= \boldsymbol{\mu}^* (\boldsymbol{\alpha}^* - \boldsymbol{C}) \,, && (3.23) \\
\mathbf{0} &\leq \boldsymbol{\lambda}^* \,, && (3.24) \\
\mathbf{0} &\leq \boldsymbol{\mu}^* \,. && (3.25)
\end{aligned}
$$

Prinzipiell gilt, dass die Bedingungen (3.19) bis (3.21) nach jedem Zwischenschritt des Zerlegungsalgorithmus erfüllt sind, denn sie werden durch die Optimierungsaufgabe (3.11) automatisch beachtet. Dadurch ergibt sich auch die Gültigkeit der Forderungen (3.22) bis (3.25), denn die Schrankennebenbedingungen der Aufgaben (3.6) und (3.11) sind identisch. Die Prüfung der KKT-Bedingungen reduziert sich demnach auf (3.18). Die Auswertung erfolgt in jedem Schritt des Verfahrens und sollte effizient durchgeführt werden.

Bemerkung 3.2. *Der Gradient von W^- an der Stelle $\boldsymbol{\alpha}^{k+1}$ berechnet sich nach der Formel*

$$\nabla W^-(\boldsymbol{\alpha}^{k+1}) = \boldsymbol{Q}\boldsymbol{\alpha}^{k+1} - \mathbf{1} \tag{3.26}$$

und wird für die Prüfung von (3.18) benötigt. Wir sparen Rechenoperationen, indem wir umformen:

$$\begin{aligned}
\boldsymbol{Q}\boldsymbol{\alpha}^{k+1} - \mathbf{1} &= \begin{pmatrix} \boldsymbol{Q}_{\mathrm{dd}} & \boldsymbol{Q}_{\mathrm{df}} \\ \boldsymbol{Q}_{\mathrm{fd}} & \boldsymbol{Q}_{\mathrm{ff}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}_{\mathrm{d}}^{k+1} \\ \boldsymbol{\alpha}_{\mathrm{f}}^{k} \end{pmatrix} - \mathbf{1} \\
&= \begin{pmatrix} \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} + \boldsymbol{Q}_{\mathrm{df}}\boldsymbol{\alpha}_{\mathrm{f}}^{k} \\ \boldsymbol{Q}_{\mathrm{fd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} + \boldsymbol{Q}_{\mathrm{ff}}\boldsymbol{\alpha}_{\mathrm{f}}^{k} \end{pmatrix} - \mathbf{1} \\
&= \begin{pmatrix} \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} + \boldsymbol{Q}_{\mathrm{df}}\boldsymbol{\alpha}_{\mathrm{f}}^{k} + \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k} - \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k} \\ \boldsymbol{Q}_{\mathrm{fd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} + \boldsymbol{Q}_{\mathrm{ff}}\boldsymbol{\alpha}_{\mathrm{f}}^{k} + \boldsymbol{Q}_{\mathrm{fd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k} - \boldsymbol{Q}_{\mathrm{fd}}\boldsymbol{\alpha}_{\mathrm{d}}^{k} \end{pmatrix} - \mathbf{1} \\
&= \nabla W^-(\boldsymbol{\alpha}^{k}) + \begin{pmatrix} \boldsymbol{Q}_{\mathrm{dd}}\left(\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} - \boldsymbol{\alpha}_{\mathrm{d}}^{k}\right) \\ \boldsymbol{Q}_{\mathrm{fd}}\left(\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} - \boldsymbol{\alpha}_{\mathrm{d}}^{k}\right) \end{pmatrix}
\end{aligned} \tag{3.27}$$

und den alten Gradientenvektor in jeder Iteration wiederverwenden. In unserer Implementierung gilt für den Startvektor $\boldsymbol{\alpha}^1 = \mathbf{0}$ übrigens $\nabla W^-(\boldsymbol{\alpha}^1) = -\mathbf{1}$.

Es ist nicht zwingend erforderlich, die Bedingung (3.18) als Abbruchkriterium des SVM-Trainings zu verwenden. Wir werden später sehen, dass der von uns verwendete Algorithmus zur Aktualisierung der Arbeitsmenge zusätzlich auch dazu geeignet ist, die Optimalität der alten Iterierten $\boldsymbol{\alpha}^k$ zu überprüfen. Die Überlegungen zur effizienten Berechnung des Gradienten werden wir dennoch verwenden, denn der Gradientenvektor wird in jeder Iteration zur Aktualisierung der Arbeitsmenge verwendet werden, siehe dazu Abschnitt 3.2.3.2.

3.2.3 Aktualisierung der Arbeitsmenge

Solange das gewählte Optimalitätskriterium nicht erfüllt ist, teilt der Zerlegungsalgorithmus die Daten in aktive und inaktive Teile und löst eine quadratische Optimierungsaufgabe auf den aktiven Daten, die im Allgemeinen einen kleinen Teil der Gesamtdaten darstellen. Wie klein dieser Teil ist, hängt jeweils von der Gesamtgröße des Problems und den zur Verfügung stehenden Ressourcen ab. Ausgehend von einem nichtoptimalen $\boldsymbol{\alpha}^k$ muss eine Indexmenge $\mathcal{D}^k$ und somit ein Vektor $\boldsymbol{\alpha}_{\mathrm{d}}^k$ gewählt werden, sodass es im Problem (3.11) eine Abstiegsmöglichkeit gibt, denn falls

$$W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{\mathrm{d}}^{k+1}) < W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{\mathrm{d}}^{k})$$

gilt und $\boldsymbol{\alpha}_{\mathrm{f}}^k$ nicht verändert wird, so gilt für die Zielfunktion (3.6)

$$W^-(\boldsymbol{\alpha}^{k+1}) < W^-(\boldsymbol{\alpha}^k) \; .$$

Es sei bemerkt, dass der Vektor $\boldsymbol{\alpha}^{k+1}$ der Form (3.16) offensichtlich zulässig ist für (3.6), sofern $\boldsymbol{\alpha}^k$ zulässig war. Der Algorithmus gewährleistet durch die Lösung eines Teilproblems (3.11) eine Verbesserung des Zielfunktionswertes von (3.6). Der Erfolg des Zerlegungsalgorithmus hängt also stark ab von der Fähigkeit, geeignete Arbeitsmengen zu finden. Gesucht sind Mengen, die jeweils zu großen Schritten in Richtung des Minimums der Zielfunktion W^- führen. Eine geeignete Methode dazu wurde in [73] vorgestellt und konnte sich im Großteil der in den letzten Jahren entwickelten SVM-Softwarepakete durchsetzen [19, 23, 74, 130]. Sie basiert auf der Methode von Zoutendijk [167], die wir auf den folgenden Seiten vorstellen werden. Es sei bemerkt, dass es auch andere Möglichkeiten zur Aktualisierung der Arbeitsmenge gibt. Osuna, Freund und Girosi [114] nutzen dafür ausschließlich die Karush-Kuhn-Tucker-Bedingungen. Sie suchen Nullelemente in der letzten Arbeitsmenge und ersetzen diese durch Punkte aus der nichtaktiven Menge, welche die KKT-Bedingungen verletzen. Hintergrund dieses Vorgehens ist die Tatsache, dass der Großteil der Daten nicht zu den Support-Vektoren gehört. Diese Punkte haben optimale Werte $\alpha_i^* = 0$ und ihnen muss keine Beachtung geschenkt werden. Wir werden zusätzlich zu der Beschreibung der Methode von Zoutendijk zeigen, dass sich dieses Verfahren dennoch mit den KKT-Bedingungen auseinandersetzt. Diese Verbindung wird in der Literatur kaum beachtet. Das führt dazu, dass die Methode von Zoutendijk nicht gedeutet werden kann. Diesen offenen Punkt werden wir im Folgenden beleuchten und interpretieren.

3.2.3.1 Wahl der ersten Arbeitsmenge

Die Wahl einer ersten Arbeitsmenge ist relativ unkritisch. Dabei müssen keine Optimalitätskriterien beachtet werden. Es geht einfach darum, zu bestimmen, auf welchen Indizes zu Beginn optimiert werden soll. Unzulässige Indexmengen können dabei nicht auftreten. Eine triviale Wahl besagt, dass die ersten ws Indizes des Vektors $\boldsymbol{\alpha}$ in die Arbeitsmenge gehen, während die restlichen $rd - ws$ Werte konstant bleiben.[15] Man könnte durch eine günstige Wahl der ersten Arbeitsmenge die Rechenzeit des gesamten Zerlegungsalgorithmus beschleunigen. Die Taktik dabei sollte darin bestehen, genau diejenigen Indizes in das erste Working-Set zu geben, die zu einem großen Sprung des Wertes der Zielfunktion nach der ersten Optimierung beitragen. Im Allgemeinen wird man dazu keine Aussagen treffen können, sodass nur die zufällige Wahl bleibt. Zur Initialisierung der Arbeitsmenge im Zusammenhang mit der sinnvollen Festlegung eines Startvektors $\boldsymbol{\alpha}^1$ für den Zerlegungsalgorithmus hatten wir im Abschnitt 3.2.1 eine interessante Idee aus [137] vorgestellt, welche auf Mehrprozessorsystemen verwendet werden kann.

[15] Beachte: wegen der linearen Nebenbedingung ist es notwendig, dass sowohl positive als auch negative Punkte in jeder Arbeitsmenge enthalten sind, denn sonst ließe sich der Vektor der Lagrange-Multiplikatoren nicht ändern

3.2.3.2 Die Methode von Zoutendijk

Das Verfahren von Zoutendijk [167] gehört zu den Methoden der zulässigen Richtungen, welche ein nichtlineares Optimierungsproblem lösen, indem sie jeweils ausgehend von einem zulässigen Punkt einen neuen zulässigen Punkt berechnen, der den Wert der Zielfunktion verbessert. Wir betrachten die Methode von Zoutendijk ausschließlich für die Aktualisierung der Arbeitsmenge. Für eine nichtlineare Zielfunktion f unter Nebenbedingungen bestimmt die Methode von Zoutendijk eine verbessernde Richtung über die Lösung eines linearen Optimierungsproblems. Die grundlegende Idee dabei ist, die stärkste Richtung $\boldsymbol{v}^*$ des Abstiegs zu finden, wobei $\boldsymbol{v}^*$ natürlich eine zulässige Richtung sein muss.

Für eine allgemeine Aufgabe der Form

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) \tag{3.28}$$

unter den Nebenbedingungen

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &\leq \boldsymbol{a}\,, \\ \tilde{\boldsymbol{A}}\boldsymbol{x} &= \tilde{\boldsymbol{a}} \end{aligned} \tag{3.29}$$

besagt die Methode von Zoutendijk [9]:

Sei $\boldsymbol{x}$ ein zulässiger Vektor und seien

$$\begin{aligned} \boldsymbol{A}_1\boldsymbol{x} &= \boldsymbol{a}_1\,, \\ \boldsymbol{A}_2\boldsymbol{x} &< \boldsymbol{a}_2\,, \end{aligned} \tag{3.30}$$

wobei $\boldsymbol{A}^T = \left[\boldsymbol{A}_1^T, \boldsymbol{A}_2^T\right]$ und $\boldsymbol{a}^T = \left[\boldsymbol{a}_1^T, \boldsymbol{a}_2^T\right]$. Dann ist $\boldsymbol{v}^*$ eine verbessernde zulässige Richtung, falls

$$\begin{aligned} \nabla f(\boldsymbol{x})^T\boldsymbol{v}^* &< 0\,, \\ \boldsymbol{A}_1\boldsymbol{v}^* &\leq \boldsymbol{0}\,, \\ \tilde{\boldsymbol{A}}\boldsymbol{v}^* &= \boldsymbol{0}\,. \end{aligned}$$

Die Richtung $\boldsymbol{v}^*$ wird generiert über die Lösung des Problems

$$\min_{\boldsymbol{v}} \quad \left(\nabla f(\boldsymbol{x})\right)^T \boldsymbol{v}$$

unter den Nebenbedingungen

$$\boldsymbol{A}_1\boldsymbol{v} \leq \boldsymbol{0}\,, \tag{3.31}$$

$$\tilde{\boldsymbol{A}}\boldsymbol{v} = \boldsymbol{0}\,. \tag{3.32}$$

Bemerkung 3.3. *Eine verbessernde Richtung $\boldsymbol{v}^*$ muss zusätzlich normalisiert werden, um zu verhindern, dass ein unendlich großer Abstieg stattfindet. Beispiele für mögliche Normalisierungen sind [167]*

$$\begin{aligned} \boldsymbol{v}^T\boldsymbol{v} &\leq 1\,, \\ -\mathbf{1} \leq \boldsymbol{v} &\leq \mathbf{1}\,. \end{aligned}$$

Ausgehend von $\boldsymbol{v}^*$ wird die neue Iterierte berechnet als

$$\boldsymbol{x} = \boldsymbol{x} + \tilde{\lambda}\boldsymbol{v}^*\,.$$

Dabei ist $\tilde{\lambda} > 0$ eine zu bestimmende Schrittweite. Die Methode von Zoutendijk wird nur teilweise angewendet. Es wird zwar der Vektor $\boldsymbol{v}^*$ bestimmt, jedoch nutzt man diesen für die Generierung einer neuen Arbeitsmenge. Es wird weder eine Schrittweite berechnet, noch wird $\boldsymbol{v}^*$ benötigt, um die Lösung zu aktualisieren. Die neue Arbeitsmenge wird aus allen denjenigen Indizes i zusammengesetzt, für welche der zugehörige Wert v_i^* positiv ist. Um die Methode von Zoutendijk anwenden zu können, muss der Vektor $\boldsymbol{v}^*$ genau ws Einträge haben, die nicht Null sind, sodass die zu diesen Elementen gehörenden Indizes die neue Arbeitsmenge bilden können [73]. Man sollte jedoch bedenken, dass solche Lösungen nicht immer realisierbar sind [97]. Der Vektor $\boldsymbol{v}^*$ kann auch weniger als ws Nichtnull-Elemente haben. $\boldsymbol{v}^*$ hat höchstens ws Einträge, die nicht Null sind und ws ist eine gerade Zahl. Diese Einschränkung wird getroffen, um den Algorithmus zur Generierung von $\boldsymbol{v}^*$ robust zu machen. Die Aufgabe (3.28) unter den Nebenbedingungen (3.29) entspricht offensichtlich der Aufgabe (3.6). Die Zerlegung der Matrix $\boldsymbol{A}$ nach dem Schema (3.30) ist noch anzugeben, um die Methode von Zoutendijk umsetzen zu können. Die Nebenbedingungen (3.8) und (3.9) können aufgeteilt werden in

$$\begin{aligned} \alpha_i = C \quad &\text{oder} \quad -\alpha_i = 0 \quad \text{für} \quad i \in \mathcal{A}\,, \\ \alpha_j < C \quad &\text{und} \quad -\alpha_j < 0 \quad \text{für} \quad j \in \mathcal{B}\,. \end{aligned}$$

Die disjunkten Indexmengen $\mathcal{A}$ und $\mathcal{B}$ haben wir eingeführt, um die aktiven und nicht aktiven Nebenbedingungen zu trennen.[16] Die Indexmenge $\mathcal{A}$ und die jeweiligen Vorzeichen der α_i benötigt man für die Schrankennebenbedingung (3.31). Die Umsetzung der Nebenbedingung (3.32) ist trivial.

Wendet man die allgemeine Form des Verfahrens auf die speziellen Funktionen und Vorgaben an, resultiert das Verfahren von Zoutendijk in der Aufgabe

[16] Aktiv und nicht aktiv benutzen wir an dieser Stelle ausnahmsweise im klassischen Sinne der Optimierung unter Nebenbedingungen.

$$\min_{\boldsymbol{v}} \quad \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)^T \boldsymbol{v} \quad \stackrel{(3.26)}{=} \quad \min_{\boldsymbol{v}} \quad \left(\boldsymbol{Q}\boldsymbol{\alpha}^k - \mathbf{1}\right)^T \boldsymbol{v} \tag{3.33}$$

unter den Nebenbedingungen

$$\begin{aligned}
-v_i &\leq 0 \qquad \text{für } -\alpha_i = 0\,, && (3.34)\\
v_i &\leq 0 \qquad \text{für } \alpha_i = C\,, && (3.35)\\
\boldsymbol{y}^T\boldsymbol{v} &= 0\,, && (3.36)\\
\boldsymbol{v} &\leq \mathbf{1}\,, && (3.37)\\
-\boldsymbol{v} &\leq \mathbf{1}\,, && (3.38)\\
|\{v_i : v_i \neq 0\}| &\leq ws\,, && (3.39)
\end{aligned}$$

wobei (3.37) und (3.38) die gewählte Normalisierungsbedingung darstellen, siehe dazu Bemerkung 3.3.

Folgerung 3.1. *Setzt sich die neue Arbeitsmenge aus Indizes i zusammen, für welche der zugehörige Wert v_i^* nicht Null ist, wobei der Vektor $\boldsymbol{v}^*$ Lösung der Aufgabe (3.33) unter den Nebenbedingungen (3.34) bis (3.39) ist, dann ist abgesichert, dass der Wert der Zielfunktion W^- durch eine Optimierung auf der Arbeitsmenge kleiner wird. Es bleibt jedoch zu klären, wie man den Vektor $\boldsymbol{v}^*$ effizient berechnen kann.*

Die Aufgabe (3.33) zur Generierung einer neuen Arbeitsmenge, die auf Beschreibungen von Zoutendijk [167] beruht und beispielsweise von Joachims für die SVM^{light} Software [74] verwendet wurde, stellt ein lineares Optimierungsproblem dar. Es wird typischerweise nach dem Schema in Abbildung 3.2 gelöst [97]. Wir zeigen im Folgenden, dass der Algorithmus in Abbildung 3.2 die Aufgabe (3.33) unter den entsprechenden Nebenbedingungen überhaupt löst. Dieses Hinterfragen der Vorgehensweise ist im Allgemeinen nicht üblich, wir halten das jedoch an dieser Stelle für wichtig. Wir zeigen zunächst, dass die erzeugte Lösung die Nebenbedingungen (3.36) bis (3.39) erfüllt.

- Es gilt

$$\begin{aligned}
\boldsymbol{y}^T\boldsymbol{v}^* &= \sum_{i=1}^{rd} y_i v_i^* = \sum_{i=1}^{\frac{ws}{2}} y_i v_i^* + \sum_{i=\frac{ws}{2}+1}^{ws} y_i v_i^* + \sum_{i=ws+1}^{rd} y_i \cdot 0\\
&= \sum_{i=1}^{q} y_i \cdot (-y_i) + \sum_{i=q+1}^{\frac{ws}{2}} y_i \cdot 0 + \sum_{i=\frac{ws}{2}+1}^{\frac{ws}{2}+q} y_i \cdot y_i + \sum_{i=\frac{ws}{2}+q+1}^{rd} y_i \cdot 0\\
&= -\sum_{i=1}^{q} y_i^2 + \sum_{i=1}^{q} y_{i+\frac{ws}{2}}^2 \quad \Rightarrow \quad \text{Bedingung } (3.36)\,.
\end{aligned}$$

1. Sortiere den Vektor $\left(\boldsymbol{y} \cdot \left(\boldsymbol{Q}\boldsymbol{\alpha}^k - \mathbf{1}\right)\right)$ in absteigender Reihenfolge, setze $\boldsymbol{v} = 0$, Anfang $= 1$, Ende $= rd$.
2. Ausgehend vom Anfang der Liste suche vorwärtsgerichtet Element i für das gilt:
   (a) $0 < \alpha_i^k < C$ oder
   (b) $\alpha_i^k = 0$ und $y_i = -1$ oder
   (c) $\alpha_i^k = C$ und $y_i = 1$.
   Ausgehend vom Ende der Liste suche rückwärtsgerichtet Element j für das gilt:
   (a) $0 < \alpha_j^k < C$ oder
   (b) $\alpha_j^k = 0$ und $y_j = 1$ oder
   (c) $\alpha_j^k = C$ und $y_j = -1$.
   Für alle während der Suche erfolglos betrachteten Elemente t werden $v_t = 0$ gesetzt. Setze $v_i = -y_i$ und $v_j = y_j$.
3. Aktualisiere Anfang und Ende der Liste und gehe zu 2.

Abbildung 3.2: Schema zur Working-Set-Bestimmung.

Wir haben hier aus Darstellungsgründen eine gewisse Umordnung der Vektoren $\boldsymbol{y}$ und $\boldsymbol{v}^*$ vorausgesetzt, die für das Skalarprodukt keine Relevanz hat. Der Wert $q \in \mathbb{N}$ steht für die Anzahl der erfolgreich generierten Pärchen $(v_i^* = -y_i, v_j^* = y_j)$. An dieser Stelle wird deutlich, warum ws eine gerade Zahl sein sollte. Es gilt $q \leq \frac{ws}{2}$, denn die Anzahl der positiven Elemente in $\boldsymbol{v}^*$ darf die Größe des Working-Sets nicht überschreiten.

- Für alle $\alpha_i^k = 0$ gilt $v_i^* \in \{0; 1\} \quad \Rightarrow \quad$ Bedingung (3.34) .
- Für alle $\alpha_i^k = C$ gilt $v_i^* \in \{-1; 0\} \quad \Rightarrow \quad$ Bedingung (3.35) .
- Es gilt $v_i^* \in \{-1; 0; 1\} \quad \Rightarrow \quad$ Bedingungen (3.37) und (3.38) .
- Es gilt $|v_i^* : v_i^* \neq 0| \leq 2 \cdot \frac{ws}{2} = ws \quad \Rightarrow \quad$ Bedingung (3.39) .

Damit garantiert der Algorithmus auf Seite 47, dass der Vektor $\boldsymbol{v}^*$ zulässig ist für die Aufgabe (3.33).

Satz 3.1 ([97]). *Der Algorithmus bricht bei den Elementen $\tilde{i}$ und $\tilde{j}$ vorzeitig ab, falls*

- $\tilde{i} + 1 = \tilde{j}$*, oder falls*
- $\tilde{i} + 2 = \tilde{j}$ *und* $0 < \alpha_{\tilde{i}+1}^k < C$.

Beweis: Angenommen, der Algorithmus stoppt nach der Untersuchung von $\tilde{i}$ und $\tilde{j}$. Sei $\tilde{m} = \tilde{i} + 1$ der nächste Index nach $\tilde{i}$ in der sortierten Liste, dann gilt entweder $0 < \alpha_{\tilde{m}}^k < C$ oder $\alpha_{\tilde{m}}^k = 0,\ y_{\tilde{m}}^k = -1$ oder $\alpha_{\tilde{m}}^k = C,\ y_{\tilde{m}}^k = 1$, denn ansonsten wäre $v_{\tilde{m}} = 0$ und das nächste Element hätte untersucht werden müssen. Analog

gilt: sei $\tilde{m}' = \tilde{j} - 1$ der direkte Index vor $\tilde{j}$ in der sortierten Liste, dann gilt weder $0 < \alpha^k_{\tilde{m}'} < C$, noch $\alpha^k_{\tilde{m}'} = 0,\ y_{\tilde{m}'} = 1$, noch $\alpha^k_{\tilde{m}'} = C,\ y_{\tilde{m}'} = -1$, denn ansonsten hätten wir mit $(\tilde{m}, \tilde{m}')$ ein neues Paar für die Optimierung gefunden. Deshalb muss $\alpha^k_{\tilde{m}'} = 0,\ y_{\tilde{m}'} = -1$ oder $\alpha^k_{\tilde{m}'} = C,\ y_{\tilde{m}} = 1$ gelten. Das würde jedoch zu $v_{\tilde{m}'} = 0$ und damit einem weiteren Schritt führen. Wir wissen damit, dass sich höchstens ein Element $\tilde{m}$ zwischen $\tilde{i}$ und $\tilde{j}$ befinden kann. Die Fälle $\alpha^k_{\tilde{m}} = 0,\ y_{\tilde{m}} = -1$ und $\alpha^k_{\tilde{m}} = C,\ y_{\tilde{m}} = 1$ sind jedoch auch ausgeschlossen, da sie wiederum zu $v_{\tilde{m}'} = 0$ führen würden. Fazit: falls sich überhaupt ein Element zwischen $\tilde{i}$ und $\tilde{j}$ befindet, handelt es sich dabei um $\tilde{m} = \tilde{i} + 1 = \tilde{j} - 1$ mit $\alpha^k_{\tilde{m}} \in (0, C)$. □

Mittels der bisherigen Analysen läßt sich zeigen:

Satz 3.2. *Ein Vektor $\boldsymbol{v}^*$, der mittels des in Abbildung 3.2 angegebenen Verfahrens generiert wurde, löst die Aufgabe (3.33).*

Für den Beweis dieses Satzes verweisen wir auf [97]. Es wird gezeigt, dass $\boldsymbol{v}^*$ ein KKT-Punkt des linearen Optimierungsproblems (3.33) und somit dessen Lösung ist.

3.2.3.3 Karush-Kuhn-Tucker-Bedingungen

Wir haben bisher erläutert, wie das Verfahren von Zoutendijk arbeitet, in welchem Rahmen man es für die Bestimmung der Arbeitsmenge verwenden kann und nach welchem Algorithmus es behandelt wird. In diesem Abschnitt soll ergänzend gezeigt werden, dass die auf Seite 47 beschriebene Vorgehensweise zur Aktualisierung der Arbeitsmenge, welche durch Überlegungen zur Lösung der Aufgabe (3.33) entsteht, auch unabhängig von der Theorie der zulässigen Richtungen einzig durch Überlegungen zu den Karush-Kuhn-Tucker-Bedingungen entwickelt werden kann. Wir erachten diese Herleitung als interessant, da sich die KKT-Bedingungen als Optimalitätsindikatoren des SVM-Trainings durch die gesamte SVM-Theorie ziehen. Wenn die Methode von Zoutendijk und die lineare Optimierungsaufgabe vernachlässigbar sind, erleichtert das die Interpretation der gewählten Arbeitsmenge, welche vielen Anwendern bisher nicht klar ist. Durch die Betrachtung der KKT-Bedingungen, welche für die globale Lösung unerläßlich sind, gewinnen wir zudem Unabhängigkeit von der Normalisierungsbedingung, vgl. (3.37) und (3.38).

Wir stützen uns bei den Betrachtungen in diesem Abschnitt auf [96, 97]. Diese Arbeiten von Lin beschäftigen sich mit der Wahl der Arbeitsmengen für den Zerlegungsalgorithmus und mit den daraus resultierenden Konvergenzeigenschaften des Gesamtverfahrens.

Definition 3.4. *Für beliebige $\boldsymbol{\alpha} \in \mathbb{R}^{rd}$ definieren wir nun die beiden Indexmengen [96]*

$$\mathcal{I} := \left\{ i \in \{1,\dots,rd\} : \{\alpha_i < C \text{ und } y_i = 1\} \text{ oder } \{\alpha_i > 0 \text{ und } y_i = -1\} \right\}$$

und

$$\mathcal{J} := \left\{ j \in \{1,\dots,rd\} : \{\alpha_j < C \text{ und } y_j = -1\} \text{ oder } \{\alpha_j > 0 \text{ und } y_j = 1\} \right\}.$$

Man beachte, dass die Mengen $\mathcal{I}$ und $\mathcal{J}$ für zulässige $\boldsymbol{\alpha}$ nicht disjunkt sind, sondern die Schnittmenge

$$\mathcal{I} \cap \mathcal{J} := \{i : \alpha_i \in (0, C)\}$$

bilden. Für die Vereinigung gilt

$$\mathcal{I} \cup \mathcal{J} := \{i : i = 1, .., rd\} \ .$$

Satz 3.3. *Für die globale Lösung $\boldsymbol{\alpha}^*$ der Aufgabe (3.6) gilt*

$$\max_{j \in \mathcal{J}} \quad \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_j \cdot y_j \ \leq \ \min_{i \in \mathcal{I}} \quad \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i \cdot y_i \ . \tag{3.40}$$

Diese Bedingung ist hinreichend und notwendig für eine Lösung.

Beweis: Da die KKT-Bedingungen notwendig und hinreichend sind, vgl. S. 40, ist es ausreichend zu zeigen, dass gilt

$$(3.40) \Leftrightarrow \text{KKT-Bedingungen erfüllt} \quad .$$

„⇐“

Sei $\boldsymbol{\alpha}^* \in \mathbb{R}^{rd}$ ein KKT-Punkt der Aufgabe (3.6), dann gelten (vgl. S. 41)

$$\begin{aligned}
\nabla W^-(\boldsymbol{\alpha}^*) - \boldsymbol{\lambda}^* + \boldsymbol{\mu}^* + \nu^* \boldsymbol{y} &= \mathbf{0} \,, && (3.41)\\
-\boldsymbol{\alpha}^* &\leq \mathbf{0} \,, && (3.42)\\
\boldsymbol{\alpha}^* - \boldsymbol{C} &\leq \mathbf{0} \,, && (3.43)\\
\boldsymbol{y}^T \boldsymbol{\alpha}^* &= 0 \,, && (3.44)\\
-\boldsymbol{\lambda}^* \boldsymbol{\alpha}^* &= \mathbf{0} \,, && (3.45)\\
\boldsymbol{\mu}^* (\boldsymbol{\alpha}^* - \boldsymbol{C}) &= \mathbf{0} \,, && (3.46)\\
\boldsymbol{\lambda}^* &\geq \mathbf{0} \,, && (3.47)\\
\boldsymbol{\mu}^* &\geq \mathbf{0} \,. && (3.48)
\end{aligned}$$

Wir stellen (3.41) um und erhalten

$$\nabla W^-(\boldsymbol{\alpha}^*) + \nu^* \boldsymbol{y} = \boldsymbol{\lambda}^* - \boldsymbol{\mu}^* \,. \tag{3.49}$$

Im Folgenden unterscheiden wir für alle α_i die drei möglichen Fälle

- $\alpha_i^* = 0$,
- $\alpha_i^* = C$,
- $0 < \alpha_i^* < C$.

Für $\alpha_i^* = 0$ gilt

$$\left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i + \nu^* y_i \geq 0 \,, \tag{3.50}$$

denn nach (3.46) muss $\mu_i^* = 0$ gelten und $\lambda_i^* \geq 0$ gilt nach (3.47).

Für $\alpha_i^* = C$ gilt

$$\left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i + \nu^* y_i \leq 0 \,, \tag{3.51}$$

denn nach (3.45) muss $\lambda_i^* = 0$ gelten und $\mu_i^* \geq 0$ gilt nach (3.48).

Für $\alpha_i^* \in (0, C)$ gilt

$$\left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i + \nu^* y_i = 0 \,, \tag{3.52}$$

denn nach (3.45) und (3.46) müssen $\lambda_i^* = 0$ und $\mu_i^* = 0$ gelten. Wir stellen zusammenfassend alle möglichen Fälle in Tabelle 3.1 dar, wobei die Werte y_i $(i = 1, .., rd)$ mit einbezogen werden. Man beachte, dass alle Ungleichungen mit y_i multipliziert wurden. Im Fall binärer Klassifikation gilt $y_i^2 = 1$.

	$y_i = 1$	$y_i = -1$
$\alpha_i = 0$	$(\nabla W^-(\boldsymbol{\alpha}^*))_i \cdot y_i + \nu^* \geq 0$	$(\nabla W^-(\boldsymbol{\alpha}^*))_i \cdot y_i + \nu^* \leq 0$
$\alpha_i = C$	$(\nabla W^-(\boldsymbol{\alpha}^*))_i \cdot y_i + \nu^* \leq 0$	$(\nabla W^-(\boldsymbol{\alpha}^*))_i \cdot y_i + \nu^* \geq 0$
$\alpha_i \in (0, C)$	$(\nabla W^-(\boldsymbol{\alpha}^*))_i \cdot y_i + \nu^* = 0$	$(\nabla W^-(\boldsymbol{\alpha}^*))_i \cdot y_i + \nu^* = 0$

Tabelle 3.1: Darstellung der möglichen Fälle für die KKT-Bedingungen.

Ordnet man diese Fälle in die Indexmengen $\mathcal{I}$ und $\mathcal{J}$ ein, so stellt sich heraus, dass

$$\begin{aligned} \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i \cdot y_i &\geq -\nu^* \quad (\forall i \in \mathcal{I}) \,, \\ \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_j \cdot y_j &\leq -\nu^* \quad (\forall j \in \mathcal{J}) \,. \end{aligned}$$

gelten. Anders ausgedrückt bedeutet das: für alle $i \in \mathcal{I}$ und $j \in \mathcal{J}$ gilt

$$-\left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i \cdot y_i \leq \nu^* \leq -\left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_j \cdot y_j \,. \tag{3.53}$$

Diese Bedingung impliziert (3.40).

„⇒“

Sei (3.40) erfüllt. Dann existiert mindestens ein $\nu^* \in \mathbb{R}$ mit

$$\left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_j \cdot y_j \leq -\nu^* \leq \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i \cdot y_i \quad (\forall i \in \mathcal{I}, \forall j \in \mathcal{J}) \,. \tag{3.54}$$

Daraus folgen

$$\begin{aligned} \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i \cdot y_i + \nu^* &\geq 0 \quad (\forall i \in \mathcal{I}) \,, \\ \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_j \cdot y_j + \nu^* &\leq 0 \quad (\forall j \in \mathcal{J}) \,. \end{aligned}$$

Eine Fallunterscheidung nach α_i^*, vgl. (3.50) bis (3.52), liefert die Gültigkeit der KKT-Bedingungen. □

Man beachte, dass $\nu^* \equiv b^*$ gilt [73, 97]. Dieser Wert ist jedoch während der Lösungsphase für das Problem (3.6) nicht bekannt und wird erst im Anschluss berechnet, siehe dazu Abschnitt 2.5.

Folgerung 3.2. *Aus Satz 3.3 folgern wir, dass für zulässige, aber nicht optimale Punkte $\boldsymbol{\alpha}^k$ mindestens ein $i \in \mathcal{I}$ und ein $j \in \mathcal{J}$ mit*

$$-\left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_i \cdot y_i > -\left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_j \cdot y_j \tag{3.55}$$

existieren und damit die Bedingung (3.40) verletzen.

Definition 3.5. *Ein Paar (i, j), welches die Ungleichung (3.55) erfüllt, bezeichnen wir im Folgenden als KKT-Bedingungen verletzendes Punktepaar.*

Um die KKT-Bedingungen zu erfüllen, müssen solche Paare eliminiert werden. Insgesamt dürfen wir in jeder Iteration, d.h. bei einem einzelnen Update-Schritt, $\frac{ws}{2}$ Pärchen auswählen. Aus der Menge $\mathcal{I}$ wählen wir sukzessive Indizes $i_1, .., i_{\frac{ws}{2}}$, gleichzeitig Indizes $j_1, .., j_{\frac{ws}{2}}$ aus $\mathcal{J}$. Dabei müssen gelten:

$$\begin{aligned} \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_{i_1} \cdot y_{i_1} \leq \ldots &\leq \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_{i_{\frac{ws}{2}}} \cdot y_{i_{\frac{ws}{2}}} \\ &< \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_{j_{\frac{ws}{2}}} \cdot y_{j_{\frac{ws}{2}}} \leq \cdots \leq \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_{j_1} \cdot y_{j_1} \,. \end{aligned}$$

Die Paare sind so angeordnet, dass $(\alpha_{i_1}, \alpha_{j_1})$ das Paar mit der stärksten Verletzung der KKT-Bedingungen ist und diese mit wachsenden Subindizes abnehmen. Sobald erstmalig

$$\left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_{i_q} \cdot y_{i_q} \geq \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_{j_q} \cdot y_{j_q} \quad (q \in \{1,..,ws/2\}) \tag{3.56}$$

auftritt, stoppt das Verfahren mit einer Arbeitsmenge der Größe $2(q-1)$, die kleiner ist als maximal zulässig.

Dieser Abschnitt sollte zeigen, dass die Karush-Kuhn-Tucker-Bedingungen der globalen Optimierungsaufgabe (3.6) ganz intuitiv zu einer geeigneten Form der Aktualisierung der Arbeitsmenge führen, sodass die Methode von Zoutendijk mit der speziellen Normierung eher zufällig in dieses Schema passt und sich dahinter auch kein unverständliches Verfahren verbirgt. Wir werden im Abschnitt 3.2.4 zeigen, welchen Unterschied es zwischen den Verfahren gibt und welche Konsequenzen dieser für die Optimierung der Zielfunktion und für ein sinnvolles Abbruchkriterium hat.

3.2.3.4 Modifikation

Neben dem allgemein üblichen Ziel, möglichst ws Elemente zu finden, welche die KKT-Bedingungen verletzen, gibt es eine weitere Idee zur Aktualisierung der Arbeitsmenge. Dabei werden mittels der Methode von Zoutendijk maximal $\tilde{ws} \leq ws$ Elemente gesucht. Der allgemeine Fall ist mit $\tilde{ws} = ws$ enthalten, jedoch wird das Ziel verfolgt, $ws - \tilde{ws}$ Elemente im Working-Set zu belassen und die sich dadurch ergebenden positiven Effekte zu nutzen [139]. Wir greifen diese Idee für unsere Arbeit auf. Das Verfahren arbeitet nach dem in Abbildung 3.3 dargestellten Schema. Unsere Experimente haben dazu geführt, $\tilde{ws} = ws/2$ zu wählen.

1. Bestimme maximal $\tilde{ws}$ Indizes mittels der Methode von Zoutendijk und bilde daraus die Menge $\mathcal{D}^{\text{neu}}$. Gehe zu 2.
2. Suche Indizes $i \in \mathcal{D}^{k-1}$ mit $i \notin \mathcal{D}^{\text{neu}}$ und $0 < \alpha_i^k < C$ und lege diese in die Menge $\mathcal{D}^{\text{neu}}$ bis entweder $|\mathcal{D}^{\text{neu}}| = ws$ gilt oder keine weiteren Elemente gefunden werden. Im letzteren Fall gehe zu $3.$, sonst gehe zu 5.
3. Suche Indizes $i \in \mathcal{D}^{k-1}$ mit $i \notin \mathcal{D}^{\text{neu}}$ und $\alpha_i^k = 0$ und lege diese in die Menge $\mathcal{D}^{\text{neu}}$ bis entweder $|\mathcal{D}^{\text{neu}}| = ws$ gilt oder keine weiteren Elemente gefunden werden. Im letzteren Fall gehe zu $4.$, sonst gehe zu 5.
4. Suche Indizes $i \in \mathcal{D}^{k-1}$ mit $i \notin \mathcal{D}^{\text{neu}}$ und $\alpha_i^k = C$ und lege diese in die Menge $\mathcal{D}^{\text{neu}}$ bis $|\mathcal{D}^{\text{neu}}| = ws$ gilt. Gehe zu 5.
5. $\mathcal{D}^k = \mathcal{D}^{\text{neu}}$.

Abbildung 3.3: Modifizierte Aktualisierung der Arbeitsmenge.

Bemerkung 3.4. *Wie schon dargestellt wurde, kann es vorkommen, dass es nicht ausreichend viele die KKT-Bedingungen verletzenden Paare gibt, aber der aktuelle Vektor* $\boldsymbol{\alpha}^k$

dennoch nicht optimal ist. Für diese Fälle haben wir das Verfahren abgeändert. Sei q die Anzahl der die KKT-Bedingungen verletzenden Paare und es gelte $2 \cdot q < \tilde{ws}$. Dann fülle das Working-Set mit $ws - 2 \cdot q$ Indizes nach dem Schema in Abbildung 3.3 auf und setze

$$\tilde{ws} = 2 \cdot q$$

für alle weiteren Iterationen des Zerlegungsalgorithmus. Diese Einschränkung ist völlig unproblematisch. Wenn in einer Iteration tatsächlich nur $q < \tilde{ws}/2$ Pärchen gefunden werden, dann ist es unmöglich, dass in einer späteren Iteration mehr Pärchen entstehen, welche die KKT-Bedingungen verletzen.

3.2.4 Abbruchkriterium

Wir hatten schon erwähnt, dass man den Algorithmus zur Aktualisierung der Arbeitsmenge gleichzeitig dazu verwenden kann, festzustellen, ob der Zerlegungsalgorithmus den optimalen Vektor $\boldsymbol{\alpha}^*$ generiert hat. Im Abschnitt 3.2.3.3 haben wir gezeigt, wie Pärchen KKT-Bedingungen verletzender Punkte definiert sind und welche Optimalitätsbedingung für einen Abbruch der Suche erfüllt sein muss, siehe (3.56). Daraus kann man schließen, dass der Zerlegungsalgorithmus genau dann stoppen sollte, wenn es kein einziges solcher Pärchen mehr gibt. Wir vergleichen die von uns hergeleitete Methode zur Aktualisierung der Arbeitsmenge und zur Definition eines Abbruchkriteriums nun mit der von Joachims in [73] vorgestellten Methode, siehe Seite 47. Es gelten

$$v_i^* = 0$$

für alle Indizes i, die nicht zu den die KKT-Bedingungen verletzenden Paaren gehören. Daraus kann man schlussfolgern, dass es genau dann keine solchen Paare gibt, wenn $\boldsymbol{v}^* = \mathbf{0}$ gilt. Ein sinnvolles Abbruchkriterium wäre daduch definiert. Das Schema von Joachims operiert auf der Liste, die durch Sortierung des Vektors $\boldsymbol{y} \cdot \nabla W^-(\boldsymbol{\alpha}^k)$ entsteht. In Tabelle 3.1 kann man erkennen, dass für alle $\alpha_i \in (0, C)$

$$y_i \cdot \left(\nabla W^-(\boldsymbol{\alpha}^*)\right)_i + \nu^* = 0$$

gelten muss. Da ν^* für alle i den gleichen Wert hat[17], und es sich auch nicht um eine Ungleichung, wie bei den anderen Fällen handelt, kommt man zu dem Ergebnis, dass im Optimum für alle freien Support-Vektoren die zugehörigen Werte in der sortierten Liste gleich sind. Die Methode von Joachims wählt für den Fall, dass es noch nicht genug Einträge in der Arbeitsmenge gibt, Pärchen aus eben dieser Menge aus, obwohl sie nach Definition 3.5 die KKT-Bedingungen nicht verletzen. Zunächst fragt man sich, warum die vorgestellten Methoden dennoch beide die Aufgabe (3.33) optimal lösen. Der Vektor, der durch Joachims Methode generiert wird, enthält an einer geraden Anzahl von Stellen

[17] ν^* entspricht dem Schwellwert b^*, siehe Seite 51.

die Einträge 1 oder -1 (in gleicher Anzahl $\mathcal{L}$), an denen bei dem Verfahren der Prüfung der KKT-Bedingungen keine Einträge vorgenommen werden. Wenn man aber bedenkt, dass die betreffenden Werte der sortierten Liste gleich sind, beträgt der Unterschied des Zielfunktionswertes von (3.33) genau

$$\mathcal{L} \cdot \left(1 \cdot y_i \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_i + (-1) \cdot y_i \left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_i\right) = 0$$

für einen beliebigen Index i aus der betreffenden Menge. Sobald sich zwei Werte entsprechen, werden auch alle folgenden Werte gleich sein, denn die Liste ist sortiert und wird von unten und oben geordnet durchsucht. Egal an welcher Stelle die gewählten Werte gleich sind, darf sich das bei weiteren Pärchen nicht mehr ändern. Die Änderung der Zielfunktion ist daher stets Null, sodass beide Methoden einen zulässigen und optimalen Vektor $\boldsymbol{v}^*$ generieren. Die Konsequenz dieser Ausführung ist, dass der von Joachims Methode generierte Vektor Indizes enthält, die zu optimalen Teilen der Lösung gehören. Das führt dazu, dass der Zerlegungsalgorithmus nicht terminieren kann, wenn er die Anzahl der Nichtnullelemente in $\boldsymbol{v}^*$ als Abbruchkriterium benutzt. Obwohl Joachims in ein und derselben Arbeit sowohl das Lösungsschema von Seite 47 vorstellt, als auch den Hinweis gibt, dessen Rückgabewert als Optimalitätskriterium zu nutzen, gibt es keinen Hinweis auf diese Problematik. Um die Methode dennoch effizient nutzen zu können, wurde folgende Variante umgesetzt. Sobald ein Indexpaar gewählt wird, dessen Einträge in der sortierten Liste annähernd gleich sind, stoppt das Verfahren von Joachims. Sollte es sich dabei um das erste gewählte Paar handeln, bricht der ganze Zerlegungsalgorithmus mit der optimalen Lösung ab. Andernfalls muss die Arbeitsmenge bis zur Grenze ws mit Indizes aufgefüllt werden. Die Umsetzung des Auffüllens wurde im Abschnitt 3.2.3.4 bereits beschrieben.

Eine sehr fruchtbare Diskussion zu diesem Thema fand mit Chih-Jen Lin[18], Entwickler der mittlerweile sehr populären LIBSVM-Software [19], statt. Er verwendet folgendes Abbruchkriterium. Der Decomposition-Algorithmus stoppt, sobald

$$-\left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_i \cdot y_i < -\left(\nabla W^-(\boldsymbol{\alpha}^k)\right)_j \cdot y_j + \varepsilon \qquad (i \in \mathcal{I}, j \in \mathcal{J}) \tag{3.57}$$

auftritt. Das entspricht einer praktischen Form von Bedingung (3.56). Diese Information bestärkt auch die von uns gewählte Vorgehensweise.

3.2.5 Fazit

Wir haben die äußere Schleife des Verfahrens der Zerlegung vorgestellt. Diese Methode gehört zu den Algorithmen der aktiven Mengen [95] und eignet sich hervorragend zur Lösung des SVM-Optimierungsproblems. Um eine bessere Übersicht zu verschaffen, geben wir in Abbildung 3.4 das Zerlegungsverfahren in der Form an, wie wir es benutzen.

[18] Associate Professor, Department of Computer Science and and Information Engineering, National Taiwan University

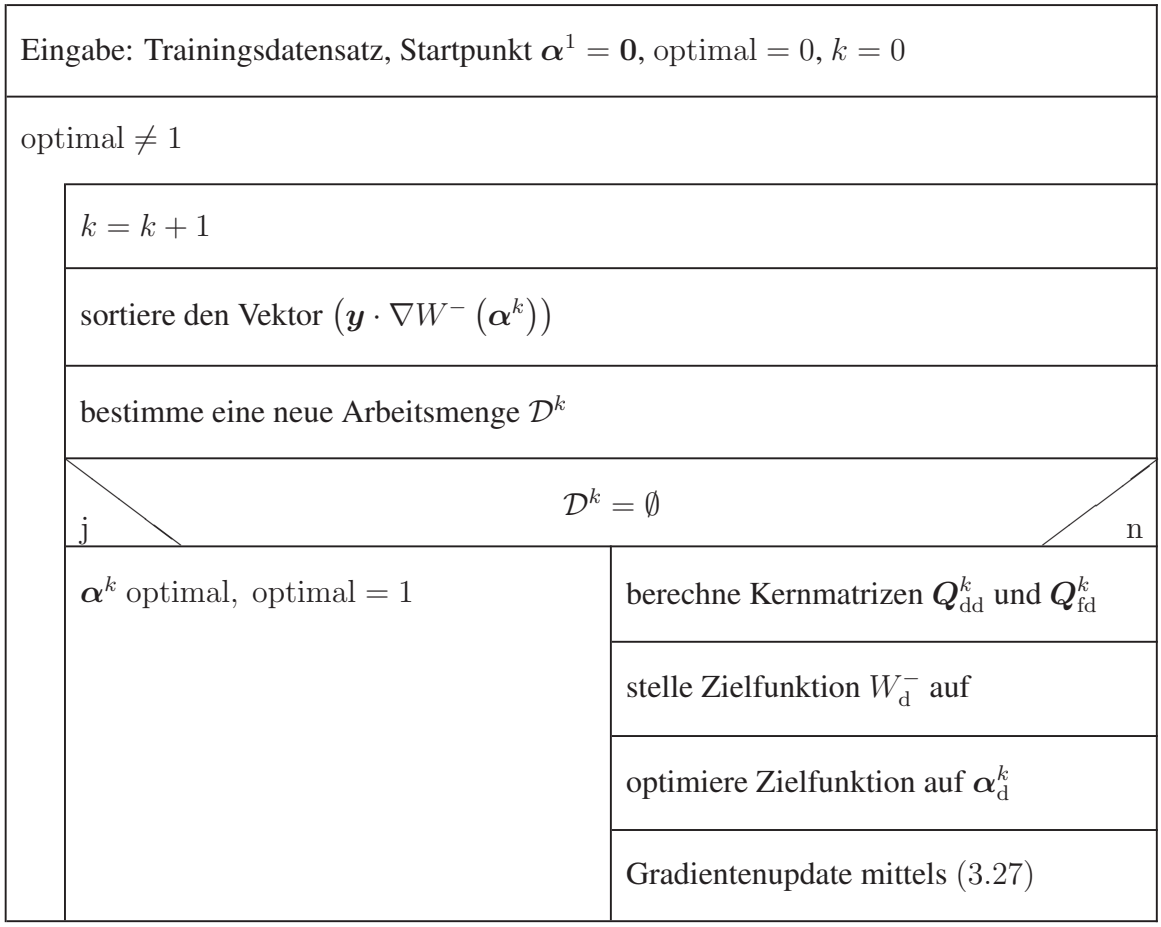

Abbildung 3.4: Äußere Schleife des Zerlegungsalgorithmus.

3.3 QP-Löser für Teilprobleme

Wie schon gezeigt wurde, muss im Zerlegungsalgorithmus nach jeder Generierung einer neuen Arbeitsmenge die konvexe quadratische Optimierungsaufgabe

$$\min_{\boldsymbol{\alpha}^k_{\mathrm{d}}} \quad W^-_{\mathrm{d}}\left(\boldsymbol{\alpha}^k_{\mathrm{d}}\right) = \frac{1}{2}\left(\boldsymbol{\alpha}^k_{\mathrm{d}}\right)^T \boldsymbol{Q}^k_{\mathrm{dd}}\boldsymbol{\alpha}^k_{\mathrm{d}} + \left(\boldsymbol{\alpha}^k_{\mathrm{f}}\right)^T \boldsymbol{Q}^k_{\mathrm{fd}}\boldsymbol{\alpha}^k_{\mathrm{d}} - \mathbf{1}^T\boldsymbol{\alpha}^k_{\mathrm{d}} \tag{3.58}$$

unter den Nebenbedingungen

$$\begin{aligned} \left(\boldsymbol{y}^k_{\mathrm{d}}\right)^T \boldsymbol{\alpha}^k_{\mathrm{d}} &= -\left(\boldsymbol{y}^k_{\mathrm{f}}\right)^T \boldsymbol{\alpha}^k_{\mathrm{f}}\,, && (3.59)\\ \mathbf{0} &\leq \boldsymbol{\alpha}^k_{\mathrm{d}}\,, && (3.60)\\ \boldsymbol{\alpha}^k_{\mathrm{d}} &\leq \boldsymbol{C} && (3.61) \end{aligned}$$

gelöst werden. Dafür benötigt man einen effizienten QP-Löser. Es sei nochmals in Erinnerung gerufen, dass $\boldsymbol{\alpha}_{\mathrm{d}}^{k} \in \mathbb{R}^{ws}$ gilt, wobei $ws \in \mathbb{N}_{+}$ die Größe der Arbeitsmenge ist. Der Index d signalisiert die aktiven Variablen, f steht für die in diesem Abschnitt nicht aktiven Variablen. $\boldsymbol{Q}_{\mathrm{dd}}^{k}$ und y_{d}^{k} ändern sich in jeder Iteration. Sowohl $\left(\boldsymbol{\alpha}_{\mathrm{f}}^{k}\right)^{T} \boldsymbol{Q}_{\mathrm{fd}}^{k}$ als auch $\left(\boldsymbol{y}_{\mathrm{f}}^{k}\right)^{T} \boldsymbol{\alpha}_{\mathrm{f}}^{k}$ sind während der Durchführung der Optimierung in jedem Schritt k konstant. Der Iterationsindex k der Zerlegungsroutine wird im Folgenden stets weggelassen. Wir definieren

$$\boldsymbol{q} := \left(\boldsymbol{Q}_{\mathrm{fd}}\right)^{T} \boldsymbol{\alpha}_{\mathrm{f}} - \mathbf{1} \tag{3.62}$$

und

$$e := -\left(\boldsymbol{y}_{\mathrm{f}}\right)^{T} \boldsymbol{\alpha}_{\mathrm{f}} \,. \tag{3.63}$$

Bemerkung 3.5. *Der Vektor $\boldsymbol{\alpha}_{\mathrm{d}}^{k}$ wird vom QP-Löser aufgenommen und verläßt diese Routine als $\boldsymbol{\alpha}_{\mathrm{d}}^{k+1}$. Innerhalb dieses Kapitels vergeben wir neue Namen, um Verwechslungen zu vermeiden. Der zu optimierende Vektor sei mit $\boldsymbol{\alpha}_{ws}$ und der Iterationsindex mit k' bezeichnet. Konstante Vektoren und Matrizen müssen selbstverständlich nicht umbenannt werden.*

Wir betrachten von nun an die formal neue Aufgabe

$$\min_{\boldsymbol{\alpha}_{ws}} \quad W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}) = \frac{1}{2} \boldsymbol{\alpha}_{ws}^{T} \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}_{ws} + \boldsymbol{q}^{T} \boldsymbol{\alpha}_{ws} \tag{3.64}$$

unter den Nebenbedingungen

$$\begin{aligned} \boldsymbol{y}_{\mathrm{d}}^{T} \boldsymbol{\alpha}_{ws} &= e \,, & (3.65)\\ \mathbf{0} &\leq \boldsymbol{\alpha}_{ws} \,, & (3.66)\\ \boldsymbol{\alpha}_{ws} &\leq \boldsymbol{C} \,. & (3.67) \end{aligned}$$

In diesem Abschnitt stellen wir ein Projektionsverfahren zur Lösung der quadratischen Teilaufgabe innerhalb der Zerlegungsmethode vor. Der Algorithmus stützt sich auf Arbeiten von Serafini, Zanghirati und Zanni [136, 138, 163]. Basis dieser Arbeiten sind wiederum die von Ruggiero und Zanni vorgestellten Methoden [125–128]. Andere Ansätze zur Lösung von (3.64) sind

- Innere-Punkte-Algorithmen für nichtlineare Optimierungsaufgaben [53], beispielsweise implementiert in *LOQO* [150] als Teil der SVM^{light}-Software [74];
- die in *MINOS* [107] implementierten Löser auf Basis von Gradienten- und Newton-Verfahren [52], unter anderem verwendet in [114].

Beide Löser sind nur für kleine Problemgrößen bis etwa 1000 Punkte geeignet [73]. In [163] wurden beide Pakete mit dem speziellen Projektionsverfahren verglichen. Die Tests zeigten, dass das Projektionsverfahren deutlich schneller zur Lösung gelangt, als diese typischerweise in der SVM-Community eingesetzten QP-Löser. Aus diesem Grund setzen wir die Projektionsmethode als Grundlage für die parallele SVM-Implementierung ein.

3.3.1 Grundlagen

Bevor wir damit beginnen, den verwendeten Löser zu beschreiben, rufen wir einige bekannte Verfahren der Optimierung in Erinnerung zurück, auf denen dieser Löser aufbaut. Wir wählen absichtlich die Abspaltung der Präsentation dieser Grundlagen, um später auf die Details, die sich im Hintergrund unseres Lösers befinden, nicht immer eingehen zu müssen. Wir vermeiden damit eine aufgeblähte Struktur des Abschnitts 3.3.2 und glauben, diese Vorgehensweise wird zum Verständnis des speziellen Algorithmus beitragen. Ein weiteres Anliegen soll es sein, einen Überblick zu den Zusammenhängen der einzelnen Methoden zu geben sowie zu zeigen, dass die im Anschluss vorgestellte Methode aus [138] keineswegs neu ist, sondern vielmehr ein sorgfältig aufgearbeitetes Verfahren der restringierten Optimierung darstellt, welches Einzug in das Arbeitsgebiet der Support-Vektor-Maschinen finden soll. Für die auf den folgenden Seiten kurz angerissenen Verfahren werden keine Aussagen zur geeigneten Wahl der Startwerte bzw. -parameter sowie zu den Abbruchbedingungen getroffen. Weiterhin stellt jedes Verfahren spezielle Anforderungen an die Zielfunktion, wie beispielweise (strenge) Konvexität, Stetigkeit oder Differenzierbarkeit. Wir werden uns bei der allgemeinen Aufarbeitung damit nicht beschäftigen, da keine dieser Methoden in der allgemein üblichen Form umgesetzt werden wird. Wir verweisen jedoch an den einzelnen Stellen auf geeignete Literatur.

3.3.1.1 Gradientenverfahren

Das Gradientenverfahren ist eine klassische Methode zur Lösung von Minimierungsproblemen ohne Restriktionen. Wir betrachten eine Aufgabe der Form

$$\min_{\boldsymbol{\alpha}_{ws} \in \mathbb{R}^{ws}} W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}) \; . \tag{3.68}$$

Das Verfahren bestimmt sukzessive Fortschreitungsrichtungen $\boldsymbol{s}^{k'} \in \mathbb{R}^{ws}$ und Schrittweiten $\theta_{k'} > 0$. Daraus entsteht iterativ die Folge $\{\boldsymbol{\alpha}_{ws}^{k'}\}_{k'>0}$ nach der Formel

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \boldsymbol{\alpha}_{ws}^{k'} + \theta_{k'} \boldsymbol{s}^{k'} \; . \tag{3.69}$$

Der Startpunkt $\boldsymbol{\alpha}_{ws}^{1}$ ist extern festzusetzen. Das Gradientenverfahren gehört, wie auch das bekannte Newton-Verfahren, zur Gruppe der Abstiegsmethoden, d.h. $W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'+1}) <$

$W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})$ gilt für alle $k' > 0$, sofern $\boldsymbol{\alpha}_{ws}^{k'}$ nicht optimal war. Eine Fortschreitungsrichtung $\boldsymbol{s}^{k'}$ wird Abstiegsrichtung im Punkt $\boldsymbol{\alpha}_{ws}^{k'}$ genannt, falls sie die folgende gewünschte Eigenschaft besitzt. Es existiert ein $\tilde{\theta} > 0$ mit

$$W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'} + \theta \boldsymbol{s}^{k'}) < W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'}) \quad \forall \theta \in (0, \tilde{\theta}) \,. \tag{3.70}$$

Im Gradientenverfahren wird die Abstiegsrichtung über

$$\boldsymbol{s}^{k'} = -\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'}) \tag{3.71}$$

und im Anschluss daran die streng positive Schrittweite über das Cauchy-Prinzip

$$W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'} + \theta_{k'} \boldsymbol{s}^{k'}) \leq W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'} + \theta \boldsymbol{s}^{k'}) \quad \forall \theta \geq 0 \tag{3.72}$$

bestimmt. Zu Konvergenzaussagen des Gradientenverfahrens sowie zu anderen Schrittweitenmethoden verweisen wir auf [53].

3.3.1.2 Fixpunktiterationen

Wir betrachten jetzt die restringierte Aufgabe

$$\min_{\boldsymbol{\alpha}_{ws} \in \Omega} W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}) \,. \tag{3.73}$$

Dabei bezeichen wir mit Ω den durch die Nebenbedingungen gegebenen zulässigen Bereich. Wir nehmen an, Ω sei konvex. Im Fall einer konvexen Zielfunktion W_{d}^{-} kann man zeigen (Satz 5.57 in [53]), dass aus

$$\left(\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{*})\right)^{T} (\boldsymbol{\alpha}_{ws} - \boldsymbol{\alpha}_{ws}^{*}) \geq 0 \quad \forall \boldsymbol{\alpha}_{ws} \in \Omega \tag{3.74}$$

folgt, $\boldsymbol{\alpha}_{ws}^{*}$ ist das globale Minimum der Aufgabe (3.73).

Definition 3.6 (Projektion, [53])**.** *Sei Ω eine nichtleere, abgeschlossene und konvexe Menge im $\mathbb{R}^{ws}$. Sei $\boldsymbol{x} \in \Omega$ beliebig. Einen Vektor $\tilde{\boldsymbol{x}}$ für den*

$$\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\| \leq \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \quad \forall \hat{\boldsymbol{x}} \in \Omega \tag{3.75}$$

gilt, bezeichnen wir als die Projektion von $\boldsymbol{x}$ auf Ω. Wir notieren diese als $\tilde{\boldsymbol{x}} = Proj_{\Omega}[\boldsymbol{x}]$.

Satz 5.58 in [53] sagt aus, dass die Bedingung (3.74) genau dann gilt, wenn $\boldsymbol{\alpha}_{ws}^*$ der Fixpunktgleichung

$$\boldsymbol{\alpha}_{ws}^* = \text{Proj}_\Omega \left[\boldsymbol{\alpha}_{ws}^* - \theta \nabla W_\text{d}^-(\boldsymbol{\alpha}_{ws}^*) \right] \tag{3.76}$$

genügt. Dabei ist $\theta > 0$ ein beliebiger positiver Parameter (siehe Beweis in [53], S. 296, unter Verwendung des Projektionssatzes). Derartige Fixpunktprobleme lassen sich über Fixpunktiterationen der Form

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \text{Proj}_\Omega \left[\boldsymbol{\alpha}_{ws}^{k'} - \theta \nabla W_\text{d}^-(\boldsymbol{\alpha}_{ws}^{k'}) \right] \quad (k' > 0) \tag{3.77}$$

lösen. Mittels des bekannten Fixpunktsatzes von Banach [68] kann man die Konvergenz der erzeugten Folge gegen die Lösung nachweisen.

3.3.1.3 Projiziertes Gradientenverfahren

Das Projizierte Gradientenverfahren hat seinen Ursprung im klassischen Projektionsverfahren [53] und wird für konvex restringierte Optimierungsprobleme eingesetzt. Das bedeutet, der zulässige Bereich der betrachteten Aufgabe ist konvex. Aufgabe (3.58) hat lineare Gleichungs- und Schrankennebenbedingungen. Diese erfüllen die Voraussetzungen des Lemmas 2.14. in [53], woraus wir schließen, dass der zulässige Bereich von (3.58) konvex ist. Das Projizierte Gradientenverfahren erzeugt iterativ zulässige Punkte[19] und stellt außerdem eine Fixpunktiteration dar.

Das Projizierte Gradientenverfahren löst die Aufgabe (3.73) mittels einer Fixpunktiteration der Gestalt

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \text{Proj}_\Omega \left[\boldsymbol{\alpha}_{ws}^{k'} - \theta_{k'} \nabla W_\text{d}^-(\boldsymbol{\alpha}_{ws}^{k'}) \right] \quad (k' > 0)\,. \tag{3.78}$$

Dabei ist $\theta_{k'} > 0$ eine wählbare Schrittweite. Man kann erkennen, dass für $\Omega = \mathbb{R}^{ws}$ die Vorschrift (3.78) dem klassischen Gradientenverfahren entspricht. Wir verweisen auf [53] für weitere Informationen zu diesem Verfahren, insbesondere zur Wahl wohldefinierter Schrittweiten. Trotzdem die Vorschrift (3.78) der klassischen Fixpunktmethode (3.77) zu entsprechen scheint, unterscheidet sie sich dadurch, dass der Parameter $\theta > 0$ nicht einmalig fest gewählt wird, sondern sich in jeder Iteration ändert.

[19] analog zum Verfahren zulässiger Richtungen

3.3.1.4 Verallgemeinertes Projiziertes Gradientenverfahren

Beim Projizierten Gradientenverfahren handelt es sich streng genommen um den Spezialfall eines Verfahrens der zulässigen Richtungen. Dieses Verfahren wird beispielsweise in [53] völlig übergangen, findet jedoch in [10] Beachtung, wo es unter dem Namen *gradient projection method* vorgestellt wird. Das Verfahren bewegt sich in Richtung der Lösung $\boldsymbol{\alpha}_{ws}^*$ von (3.73) mittels der vom Gradientenverfahren bekannten Iteration (3.69)

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \boldsymbol{\alpha}_{ws}^{k'} + \theta_{k'} \boldsymbol{s}^{k'} \ .$$

Abweichend gelten jedoch $\theta_{k'} \in (0, 1]$ und die Richtung $\boldsymbol{s}^{k'}$ wird nach der Formel

$$\boldsymbol{s}^{k'} = \boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'} \tag{3.79}$$

berechnet. Der Vektor $\boldsymbol{x}_{k'}^*$ ist dabei die Lösung der Aufgabe

$$\boldsymbol{x}_{k'}^* = \mathrm{Proj}_{\Omega} \left[\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'} \nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'}) \right] \ . \tag{3.80}$$

$\rho_{k'} > 0$ ist ein wählbarer Parameter. Bei diesem Verfahren wird also zunächst ein Schritt in die Richtung des negativen Gradienten vollzogen, danach wird das Ergebnis $\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'} \nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'})$ auf Ω projiziert. Daraus entsteht der zulässige Vektor $\boldsymbol{x}_{k'}^*$. Der eigentliche Iterationsschritt erfolgt dann in die zulässige Richtung $\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'}$ mit der Schrittweite $\theta_{k'}$. Eine andere interessante Interpretation dieses Verfahrens [126] besagt, dass das Projizierte Gradientenverfahren (3.78) durchgeführt wird, wobei die neue Iterierte abweichend nach der Formel

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \theta_{k'} \mathrm{Proj}_{\Omega} \left[\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'} \nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'}) \right] + (1 - \theta_{k'}) \boldsymbol{\alpha}_{ws}^{k'} \ . \tag{3.81}$$

berechnet wird. Dieser Schritt wird als Korrektur bezeichnet. Die auszuführenden Schritte sind jeweils gleich, jedoch liegt der Unterschied in der Frage, ob man das Verfahren als eine Art Gradienten- oder als Projektionsverfahren interpretiert. An dieser Stelle kann man sich klar machen, dass der verwendete Parameter $\rho_{k'}$ ebenfalls als Schrittweite angesehen werden kann, denn setzt man beispielsweise in (3.69) $\theta_{k'} = 1$, so gilt

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'} = \boldsymbol{x}_{k'}^* \ , \tag{3.82}$$

und damit

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \mathrm{Proj}_{\Omega} \left[\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'} \nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'}) \right] \ . \tag{3.83}$$

Dadurch ist auch bewiesen, dass das Verfahren der Projizierten Gradienten, wie es in [53] vorgestellt wird, lediglich ein Speziallfall dieser Methode ist.

3.3.1.5 Projektionsverfahren

Ähnlich der einfachen Fixpunktiteration gibt es eine Gruppe von Projektionsverfahren für streng konvexe Probleme[20], welche ausgehend von einem Startvektor $\boldsymbol{\alpha}_{ws}^{1}$ die Folge $\left\{\boldsymbol{\alpha}_{ws}^{k'}\right\}_{k'>0}$ iterativ nach der Vorschrift

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \underset{\boldsymbol{\alpha}_{ws}\in\Omega}{\arg\min} \quad \frac{1}{2}\boldsymbol{\alpha}_{ws}^{T}\frac{\boldsymbol{P}}{\theta}\boldsymbol{\alpha}_{ws} + \left(\boldsymbol{q} + \left(\boldsymbol{Q}_{\mathrm{dd}} - \frac{\boldsymbol{P}}{\theta}\right)\boldsymbol{\alpha}_{ws}^{k'}\right)^{T}\boldsymbol{\alpha}_{ws} \tag{3.84}$$

bestimmen [125–127]. Dabei ist $\theta \neq 1$ ein positiver Parameter und $\boldsymbol{P}$ eine positiv definite Matrix. Der Parameter θ ist insoweit kritisch, als dass seine Wahl die Konvergenzeigenschaften des Verfahrens bestimmt. Etwas genauer formuliert, muss θ relativ klein sein, um überhaupt Konvergenz zu sichern. Dieses kleine θ führt dann aber nur zu einer langsamen Konvergenz. Es wurde beobachtet, dass es Werte für θ geben kann, welche die theoretischen Konvergenzbedingungen nicht erfüllen, aber dennoch zu sehr guten Konvergenzeigenschaften für spezielle Testprobleme führen [125–127].

Bemerkung 3.6. *Der Gradient für die in diesem Abschnitt betrachtete Funktion (3.64) hat die Gestalt*

$$\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}) = \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws} + \boldsymbol{q}\ . \tag{3.85}$$

Definition 3.7. *Wir führen jetzt eine spezielle Norm ein, die Norm bezüglich einer positiv definiten Matrix $\boldsymbol{A} \in \mathbb{R}^{n,n}$. Sei $\boldsymbol{x} \in \mathbb{R}^{n}$, dann definieren wir die Norm $\|\cdot\|_{\boldsymbol{A}}$ als*

$$\|\boldsymbol{x}\|_{\boldsymbol{A}} = \sqrt{\boldsymbol{x}^{T}\boldsymbol{A}\boldsymbol{x}}. \tag{3.86}$$

Satz 3.4. *In der Vorschrift (3.84) entspricht der Vektor $\boldsymbol{\alpha}_{ws}^{k'+1}$ der Projektion des Vektors*

$$\boldsymbol{\alpha}_{ws}^{k'} - \theta\boldsymbol{P}^{-1}\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})$$

auf Ω bezüglich der Norm $\|\cdot\|_{\boldsymbol{P}}$.

Beweis: Sei

$$\tilde{\boldsymbol{\alpha}}_{ws}^{k'} := \boldsymbol{\alpha}_{ws}^{k'} - \theta\boldsymbol{P}^{-1}\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\ .$$

Dann gilt

[20] Man beachte, dass wir diese Voraussetzung nicht erfüllen. Weitere Details dazu werden später beleuchtet.

$$
\begin{aligned}
& \boldsymbol{\alpha}_{ws}^{k'+1} = \underset{\boldsymbol{\alpha}_{ws}\in\Omega}{\arg\min} \quad \|\boldsymbol{\alpha}_{ws} - \tilde{\boldsymbol{\alpha}}_{ws}^{k'}\|_{\boldsymbol{P}} \\
\overset{(3.86)}{\Leftrightarrow} \quad & \boldsymbol{\alpha}_{ws}^{k'+1} = \underset{\boldsymbol{\alpha}_{ws}\in\Omega}{\arg\min} \quad \sqrt{\left(\boldsymbol{\alpha}_{ws} - \tilde{\boldsymbol{\alpha}}_{ws}^{k'}\right)^T \boldsymbol{P}\left(\boldsymbol{\alpha}_{ws} - \tilde{\boldsymbol{\alpha}}_{ws}^{k'}\right)} \\
\overset{(3.85)}{\Leftrightarrow} \quad & \boldsymbol{\alpha}_{ws}^{k'+1} = \underset{\boldsymbol{\alpha}_{ws}\in\Omega}{\arg\min} \quad \sqrt{\boldsymbol{\alpha}_{ws}^T\boldsymbol{P}\boldsymbol{\alpha}_{ws} - 2\boldsymbol{\alpha}_{ws}\boldsymbol{P}\boldsymbol{\alpha}_{ws}^{k'} + 2\boldsymbol{\alpha}_{ws}\theta\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} + 2\boldsymbol{\alpha}_{ws}\theta\boldsymbol{q}} \\
\Leftrightarrow \quad & \boldsymbol{\alpha}_{ws}^{k'+1} = \underset{\boldsymbol{\alpha}_{ws}\in\Omega}{\arg\min} \quad \sqrt{\frac{1}{2}\boldsymbol{\alpha}_{ws}^T\frac{\boldsymbol{P}}{\theta}\boldsymbol{\alpha}_{ws} + \left(\boldsymbol{q} + \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} - \frac{\boldsymbol{P}}{\theta}\boldsymbol{\alpha}_{ws}^{k'}\right)^T\boldsymbol{\alpha}_{ws}} \,.
\end{aligned}
$$

□

Mittels einer einfachen Rechnung kann die Formulierung (3.84) also umgewandelt werden in

$$
\boldsymbol{\alpha}_{ws}^{k'+1} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta\boldsymbol{P}^{-1}\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] \,. \tag{3.87}
$$

Diese Darstellung verdeutlicht, dass es sich um Projektionsalgorithmen handelt. Die Ähnlichkeit zur Methode der Fixpunktiterationen ist offensichtlich. Der einzige Unterschied zwischen (3.77) und (3.87) besteht in der Matrix $\boldsymbol{P}^{-1}$.

3.3.1.6 Verallgemeinertes Projektionsverfahren

Die Überlegungen, welche wir zum Verallgemeinerten Verfahren der Projizierten Gradienten führten, kann man auch hier anwenden. Wir definieren einen Korrekturschritt, der zu einem Verfahren zulässiger Richtungen führt. Die einzelnen Schritte verlaufen völlig analog, man lese dazu auf Seite 60 nach. Das Verallgemeinerte Projektionsverfahren ist demnach von der Form

$$
\boldsymbol{\alpha}_{ws}^{k'+1} = \theta_{k'}\mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \rho\boldsymbol{P}^{-1}\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] + (1-\theta_{k'})\boldsymbol{\alpha}_{ws}^{k'} \,. \tag{3.88}
$$

Wir definieren $\boldsymbol{x}_{k'}^{*} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \rho\boldsymbol{P}^{-1}\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right]$ und sehen, dass wir es erneut mit einem Abstiegsverfahren zu tun haben, denn

$$
\begin{aligned}
\boldsymbol{\alpha}_{ws}^{k'+1} &= \theta_{k'}\boldsymbol{x}_{k'}^{*} + (1-\theta_{k'})\boldsymbol{\alpha}_{ws}^{k'} \\
&= \boldsymbol{\alpha}_{ws}^{k'} + \theta_{k'}\left(\boldsymbol{x}_{k'}^{*} - \boldsymbol{\alpha}_{ws}^{k'}\right) \\
&= \boldsymbol{\alpha}_{ws}^{k'} + \theta_{k'}\boldsymbol{s}^{k'} \,.
\end{aligned}
$$

Dabei wird die zulässige Richtung $\boldsymbol{s}^{k'}$ analog zu (3.79) definiert.

3.3.1.7 Extragradientenverfahren

Das sogenannte Extragradientenverfahren stellt eine Modifikation der einfachen Fixpunktiteration dar. Im Wesentlichen wird dabei anstelle einer einzigen, auf zwei Projektionen pro Iterationsschritt zurückgegriffen (vgl. Seite 432 in [53]). Kurz dargestellt ergibt sich $\boldsymbol{\alpha}_{ws}^{k'+1}$ wie folgt:

$$\boldsymbol{\alpha}_{\text{hilf}} = \text{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta\nabla W_{\text{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] \quad \text{und} \tag{3.89}$$

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \text{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta\nabla W_{\text{d}}^{-}(\boldsymbol{\alpha}_{\text{hilf}})\right] . \tag{3.90}$$

Der Vektor $\boldsymbol{\alpha}_{\text{hilf}}$ stellt in jedem Iterationsschritt einen Hilfsvektor dar. Hintergrund dieser Methode ist die Tatsache, dass die Konvergenz dieses Verfahrens im Vergleich zur einfachen Fixpunktmethode unter abgeschwächten Bedingungen nachgewiesen werden kann, dazu siehe Lemma 7.33 in [53].

3.3.1.8 Modifiziertes Extragradientenverfahren

Ebenfalls in [53] wird eine Modifikation des Extragradientenverfahrens vorgestellt. Sowohl die einfache Fixpunktiteration als auch das Extragradientenverfahren operieren mit einer einzigen konstanten Schrittweite, wohingegen beim Gradientenverfahren und beim Projizierten Gradientenverfahren die Schrittweite in jeder Iteration neu angepasst wird. Das Modifizierte Extragradientenverfahren verbindet diese Methoden durch Einführung einer variablen Schrittweite $\theta_{k'}$ innerhalb des Extragradientenverfahrens. Dadurch kann unter noch schwächeren Voraussetzungen eine globale Konvergenzaussage getroffen werden.

Wir werden an dieser Stelle keine genauen Angaben dazu machen, wie die Schrittweite $\theta_{k'}$ im Allgemeinen berechnet werden soll, denn die Anzahl der Modifikationen ist sehr groß. Wir geben daher nur die globale Idee des Verfahrens wieder.

$$\boldsymbol{\alpha}_{\text{hilf}}^{n} = t_{k'}\text{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta\nabla W_{\text{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] + (1 - t_{k'})\boldsymbol{\alpha}_{ws}^{k'} \quad \text{und} \tag{3.91}$$

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \text{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta_{k'}\nabla W_{\text{d}}^{-}(\boldsymbol{\alpha}_{\text{hilf}}^{n})\right] . \tag{3.92}$$

Die Iterierte $\boldsymbol{\alpha}_{ws}^{k'+1}$ ist im Gegensatz zum Extragradientenverfahren von einem veränderlichen Parameter $\theta_{k'}$ sowie einem modifizierten Hilfsvektor abhängig. Dieser Hilfsvektor $\boldsymbol{\alpha}_{\text{hilf}}^{n}$ ergibt sich in diesem Fall aus einer Konvexkombination von $\boldsymbol{\alpha}_{ws}^{k'}$ und dem schon bekannten $\boldsymbol{\alpha}_{\text{hilf}}$. Ein Modifiziertes Extragradientenverfahren kann in [53] nachgelesen werden (Algorithmus 7.36). Dort wird auch gezeigt, wie sich die Parameter $t_{k'}$ und $\theta_{k'}$ bestimmen lassen.

3.3.1.9 Zusammenfassung

Tabelle 3.2 stellt alle in diesem Abschnitt wiederholten Verfahren dar. Dabei wird nur auf die Form der Lösungsupdates eingegangen, um zu zeigen, wie verwandt diese Methoden sind. Zur Anwendung der einzelnen Algorithmen müssen die jeweiligen Quellen genauer studiert werden, um die notwendigen Parameter und Schrittweiten so zu wählen, dass eine ausreichend schnelle Konvergenz gesichert ist. Bei Betrachtung der Tabelle gelangt man

Name	Form des Updates
Gradientenverfahren (unrestringiert)	$\boldsymbol{\alpha}_{ws}^{k'+1} = \boldsymbol{\alpha}_{ws}^{k'} - \theta_{k'} \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})$
Methode der Fixpunktiterationen	$\boldsymbol{\alpha}_{ws}^{k'+1} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right]$
Verfahren der Projizierten Gradienten	$\boldsymbol{\alpha}_{ws}^{k'+1} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta_{k'} \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right]$
Verallgemeinertes Verfahren der Projizierten Gradienten	$\boldsymbol{\alpha}_{ws}^{k'+1} = \theta_{k'} \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'} \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] + (1-\theta_{k'})\boldsymbol{\alpha}_{ws}^{k'}$
Projektionsverfahren	$\boldsymbol{\alpha}_{ws}^{k'+1} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta \boldsymbol{P}^{-1} \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right]$
Verallgemeinertes Projektionsverfahren	$\boldsymbol{\alpha}_{ws}^{k'+1} = \theta_{k'} \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \rho \boldsymbol{P}^{-1} \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] + (1-\theta_{k'})\boldsymbol{\alpha}_{ws}^{k'}$
Extragradientenverfahren	$\boldsymbol{\alpha}_{\mathrm{hilf}} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right]$ $\boldsymbol{\alpha}_{ws}^{k'+1} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{\mathrm{hilf}})\right]$
Modifiziertes Extragradientenverfahren	$\boldsymbol{\alpha}_{\mathrm{hilf}}^{n} = t_{k'} \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] + (1-t_{k'})\boldsymbol{\alpha}_{ws}^{k'}$ $\boldsymbol{\alpha}_{ws}^{k'+1} = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \theta_{k'} \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{\mathrm{hilf}}^{n})\right]$

Tabelle 3.2: Zusammenfassung der wiederholten Optimierungsalgorithmen als Grundlage für den implementierten QP-Löser.

zu dem Eindruck, dass der feste Parameter ρ im Verallgemeinerten Projektionsverfahren ebenfalls flexibel sein sollte, wie bei den Methoden der Projizierten Gradienten. Diesen Ansatz verfolgt nun die in dieser Arbeit umgesetzte Methode. Sie wird im nächsten Abschnitt vorgestellt.

3.3.2 Verallgemeinerte Variablen-Projektionsmethode

In diesem Abschnitt beschreiben wir die spezielle Projektionsmethode aus [138], die dort unter dem Namen *generalized variable projection method* (GVPM) vorgestellt wird. Wir werden sie Verallgemeinerte Variablen-Projektionsmethode nennen und die Abkürzung VVPM benutzen. VVPM basiert auf der Methode der Verallgemeinerten Projektionsverfahren, hat jedoch flexible Projektionsparameter. Diese Kombination haben wir noch nicht vorgestellt, da sie in der Standardliteratur nicht enthalten ist. VVPM in [138] ist eine Weiterentwicklung und Verbesserung verschiedener, sehr ähnlicher Variablen-Projektionsmethoden, wie sie beispielsweise in [125–128] zu finden sind. Es ist wichtig zu erwähnen, dass das Verfahren in [138] mit $\boldsymbol{P} = \boldsymbol{I}$ arbeitet und damit offensichtlich ein Spezialfall des Verallgemeinerten Verfahrens der Projizierten Gradienten ist. Darauf wird in der Originalarbeit jedoch nicht eingegangen. Wir werden die genaue Umsetzung dieser Methode diskutieren. Wir wollen darauf hinweisen, dass es sich hierbei nicht um ein neues Verfahren handelt, auch wenn man bei der Lektüre diesen Eindruck bekommt. Das kann bei Konvergenzfragen durchaus hilfreich sein. Wir wollen diesen Abschnitt deshalb auch nutzen, um auf die Unterschiede zu anderen Variablen-Projektionsmethoden aufmerksam zu machen. Das betrifft die Form der zu lösenden Aufgaben, die Wahl bestimmter Parameter und Startwerte sowie die Form der Update-Regeln.

Im Folgenden werden wir θ bzw. $\theta_{k'}$ immer als Schrittweite bezeichnen. Der Parameter ρ bzw. $\rho_{k'}$, der, wie schon festgestellt wurde, ebenfalls als Schrittweite interpretiert werden kann, wird von nun an Projektionsparameter genannt, um Verwechslungen zu vermeiden. Die Matrix $\boldsymbol{Q}_{\mathrm{dd}}$ ist positiv semidefinit. In den Arbeiten [125, 126] werden QP-Aufgaben mit positiv definiten Matrizen betrachtet (streng konvexe Zielfunktionen). Verwendet man die dort vorgestellten Algorithmen, muss man zusätzlich Prüfschritte einbauen für den Fall, dass $\boldsymbol{Q}_{\mathrm{dd}}$ nicht positiv definit ist. Teilweise wurde überhaupt nicht näher auf die Eigenschaften der betrachteten Zielfunktion eingegangen; sofern es sich jedoch um Arbeiten im Zusammenhang mit Support-Vektor-Maschinen handelt, kann man davon ausgehen, dass es sich um Probleme mit konvexen, quadratischen Zielfunktionen und linearen Nebenbedingungen handelt [26], da sowohl in der SVM-Literatur, als auch in Arbeiten zu Kernmethoden vorausgesetzt wird, dass die Grammatrix positiv semidefinit ist [90].

Bemerkung 3.7. *Die Matrix $\boldsymbol{Q}$ besteht aus den Einträgen*

$$Q_{ij} = y_i y_j k(\boldsymbol{x}^i, \boldsymbol{x}^j) \quad (1 \leq i, j \leq rd), \tag{3.93}$$

wobei k die sogenannte Kernfunktion ist, welche jeweils zwei Vektoren des Datenraumes einen Wert zuordnet, der die zwischen den Punkten vorhandene Ähnlichkeit widerspiegelt. Diese Kernfunktion wird in der Literatur zu Support-Vektor-Maschinen und Kernalgorithmen im Allgemeinen als positiv definit bezeichnet, falls für beliebige $a_1, .., a_{rd} \in \mathbb{R}$

$$\sum_{i=1}^{rd} \sum_{j=1}^{rd} a_i a_j k(\boldsymbol{x}^i, \boldsymbol{x}^j) \geq 0 \tag{3.94}$$

gilt [135]. Man beachte, dass diese Definition eines positiv definiten Kerns nicht garantiert, dass $\boldsymbol{Q}$ positiv definit ist. Wir bleiben bei der üblichen Bezeichnung der Semidefinitheit. $\boldsymbol{Q}_{\mathrm{dd}}$ ist auch immer positiv semidefinit.

Die von uns vorgestellte Methode [138] stellt keine Konvexitätsforderungen an die Zielfunktion, das Verfahren ist also auch anwendbar auf alle symmetrischen Matrizen. Abbildung 3.5 stellt die VVP-Methode grob dar. Innerhalb einer Schleife wird mehrmals ein separables quadratisches Optimierungsproblem bearbeitet. Dessen Lösungen werden jeweils zur Aktualisierung des Lösungsvektors der ursprünglichen Aufgabe genutzt. Die einzelnen Schritte werden im Folgenden näher vorgestellt.

Initialisierung (Schritt 1)
Lösung eines vereinfachten quadratischen Optimierungsproblems (Schritt 2)
Liniensuche (Schritt 3)
Aktualisierung des Lösungsvektors
Berechnung des neuen Projektionsparameters (Schritt 4)
Abbruchbedingung (Schritt 5) nicht erfüllt

Abbildung 3.5: Grobes Schema zur Verallgemeinerten Variablen-Projektionsmethode.

Bemerkung 3.8. *Die VVP-Methode ähnelt dem Verallgemeinerten Projektionsverfahren, wurde jedoch in [126] als Modifikation eines einfachen Projektionsverfahrens, siehe Seite 61, vorgestellt. Es hat sich gezeigt, dass die klassischen Verfahren sehr langsam konvergieren, falls der Parameter θ alle notwendigen Konvergenzbedingungen erfüllt [125, 126]. Der Vorteil der dort vorgestellten Verfahren liegt in den schwachen Bedingungen an die Projektionsparameter. Es sei zu bemerken, dass in den Arbeiten [125–128] ausschließlich Optimierungsprobleme mit dünnbesetzten Matrizen betrachtet werden, wohingegen [138, 163] sich den speziellen Aufgaben, wie sie von Support-Vektor-Maschinen zu lösen sind, zuwenden. Die Matrizen sind dort weitgehend dicht besetzt.*

Die nun folgenden Abschnitte beschreiben die iterative VVP-Methode zur Lösung der Aufgabe (3.64)

3.3.2.1 Datenübergabe und Initialisierungen (Schritt 1)

Der Zerlegungsalgorithmus übergibt der Routine die Matrix $\boldsymbol{Q}_{\mathrm{dd}}$, den Vektor $\boldsymbol{q}$, den zu $\boldsymbol{\alpha}_{ws}$ gehörigen Klassenvektor $\boldsymbol{y}_{\mathrm{d}}$, die rechte Seite $e \in \mathbb{R}$ der Gleichheitsnebenbedingung (3.65) sowie den Parameter C. Für die Definition des Optimierungsproblems (Schritt 2) benötigen wir einen zulässigen Startvektor $\boldsymbol{\alpha}_{ws}^1 \in \mathbb{R}^{ws}$, siehe dazu Bemerkung 3.10 auf Seite 72. Weiterhin werden ein Intervall $[\rho_{min}, \rho_{\max}]$ mit $\rho_{min} > 0$ für die möglichen Projektionsparameterwerte sowie ein positiver Startwert $\rho_1 \in [\rho_{min}, \rho_{\max}]$ für den Projektionsparameter benötigt [138]. Wir werden später darauf eingehen, wie man die Initialisierungen sinnvoll treffen kann. Zunächst setzen wir die Kenntnis dieser Werte voraus. In anderen Variablen-Projektionsmethoden wird kein Intervall $[\rho_{min}, \rho_{\max}]$ vorgegeben [128], jedoch wird die Update-Formel für den Projektionsparameter dann so gestaltet, dass die Folge $\{\rho_{k'}\}_{k'>1}$ sowohl nach unten als auch nach oben beschränkt ist. Diese Eigenschaft ist eine der Voraussetzungen, die gelten müssen, um zeigen zu können, dass die VVP-Methode konvergiert. Das VVP-Verfahren verwendet noch weitere Parameter und Konstanten, wir werden sie an den passenden Stellen definieren. Während der Initialisierungen gilt $k' = 1$. Die Schritte 2 bis 5 werden im Anschluss in einer Schleife solange durchlaufen, bis die Abbruchbedingung in Schritt 5 erfüllt ist. Dabei wird k' jeweils in Schritt 5 um 1 inkrementiert. Schritt 1 wird nicht nocheinmal aufgerufen.

3.3.2.2 Lösen eines vereinfachten Optimierungsproblems (Schritt 2)

Ausgehend von den Daten, die von der Zerlegungsroutine übermittelt werden sowie den Initialisierungen aus Schritt 1 ($k' = 1$) bzw. den internen Aktualisierungen $\boldsymbol{\alpha}_{ws}^{k'}$ und $\rho_{k'}$ ($k' > 1$) wird nun folgendes Problem betrachet:

$$\begin{aligned} \boldsymbol{x}_{k'}^* &= \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'} \nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}_{ws}^{k'})\right] \\ &= \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'}\left(\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{q}\right)\right] . \end{aligned} \tag{3.95}$$

Andere Variablen-Projektionsmethoden [125] lösen die allgemeinere Aufgabe

$$\boldsymbol{x}_{k'}^* = \mathrm{Proj}_{\Omega}\left[\boldsymbol{\alpha}_{ws}^{k'} - \rho_{k'}\boldsymbol{P}^{-1}\left(\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{q}\right)\right] . \tag{3.96}$$

Sicherlich fragt man sich an dieser Stelle, warum das neue Verfahren als Spezialfall den Namen *generalized variable projection method* erhalten hat. Das hat andere Gründe, speziell geht es dabei um die Berechnung des Projektionsparameters. Dass die Matrix $\boldsymbol{P}$ keine Rolle mehr spielt, wird in [138] auch gar nicht erwähnt. Die Projektion muss in jeder Iteration neu berechnet werden. Dazu rufen wir Definition 3.6 in Erinnerung. Die Aufgabe (3.95) besteht demnach in der Bestimmung von

$$\boldsymbol{x}^*_{k'} = \underset{\boldsymbol{x}\in\Omega}{\arg\min} \quad \left\| \boldsymbol{x} - \left(\boldsymbol{\alpha}^{k'}_{ws} - \rho_{k'} \left(\boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}^{k'}_{ws} + \boldsymbol{q} \right) \right) \right\| \ . \tag{3.97}$$

Wir betrachten hier die einfache euklidische Norm, d.h. $\boldsymbol{P} = \boldsymbol{I}_{ws}$.
Sei

$$\tilde{\boldsymbol{\alpha}}^{k'}_{ws} := \boldsymbol{\alpha}^{k'}_{ws} - \rho_{k'} \left(\boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}^{k'}_{ws} + \boldsymbol{q} \right) \ ,$$

dann folgt

$$\begin{aligned}
\boldsymbol{x}^*_{k'} &= \underset{\boldsymbol{x}\in\Omega}{\arg\min} \quad \sqrt{\left(\boldsymbol{x} - \tilde{\boldsymbol{\alpha}}^{k'}_{ws}\right)^T \left(\boldsymbol{x} - \tilde{\boldsymbol{\alpha}}^{k'}_{ws}\right)} \\
&= \underset{\boldsymbol{x}\in\Omega}{\arg\min} \quad \boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{\alpha}^{k'}_{ws} + 2\boldsymbol{x}^T\rho_{k'} \left(\boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}^{k'}_{ws} + \boldsymbol{q} \right) \\
&= \underset{\boldsymbol{x}\in\Omega}{\arg\min} \quad \boldsymbol{x}^T \frac{\boldsymbol{I}_{ws}}{\rho_{k'}} \boldsymbol{x} + \left(\boldsymbol{q} + \left(\boldsymbol{Q}_{\mathrm{dd}} - \frac{\boldsymbol{I}_{ws}}{\rho_{k'}} \right) \boldsymbol{\alpha}^{k'}_{ws} \right)^T \boldsymbol{x} \ .
\end{aligned} \tag{3.98}$$

Damit erhalten wir erneut eine quadratische Optimierungsaufgabe. Sie hat die Form

$$\min_{\boldsymbol{x}\in\mathbb{R}^{ws}} \quad \frac{1}{2}\boldsymbol{x}^T \boldsymbol{I}^{\rho_{k'}}_{ws} \boldsymbol{x} + \left(\boldsymbol{q} + (\boldsymbol{Q}_{\mathrm{dd}} - \boldsymbol{I}^{\rho_{k'}}_{ws}) \, \boldsymbol{\alpha}^{k'}_{ws} \right)^T \boldsymbol{x} \tag{3.99}$$

unter den Nebenbedingungen

$$\boldsymbol{y}_d^T \boldsymbol{x} = e \ , \tag{3.100}$$

$$\boldsymbol{0} \leq \boldsymbol{x} \ , \tag{3.101}$$

$$\boldsymbol{x} \leq \boldsymbol{C} \ . \tag{3.102}$$

Dabei ist $\boldsymbol{I}^{\rho_{k'}}_{ws} = \frac{\boldsymbol{I}_{ws}}{\rho_{k'}}$ die Diagonalmatrix, die dadurch entsteht, dass alle Einträge in $\boldsymbol{I}_{ws}$ durch den Wert von $\rho_{k'}$ geteilt werden. Die sich in den Durchläufen des Verfahrens verändernden Daten sind daher $\boldsymbol{I}^{\rho_{k'}}_{ws}$ sowie der Lösungsvektor $\boldsymbol{\alpha}^{k'}_{ws}$. Auf die Rolle von $\boldsymbol{x}^{k'}$ kommen wir im Anschluss zu sprechen. Die VVP-Methode verändert die Struktur des Problems also dahingehend, dass im quadratischen Teil der Zielfunktion keine vollbesetzte Matrix $\boldsymbol{Q}_{\mathrm{dd}}$, sondern eine streng positive Diagonalmatrix $\boldsymbol{I}^{\rho_{k'}}_{ws}$ betrachtet wird. Es entsteht ein separables, streng konvexes Optimierungsproblem, welches wir mit einem Algorithmus lösen können, der diese besondere Struktur ausnutzt und damit auch für größere Teilprobleme eingesetzt werden kann. An dieser Stelle gehen wir nicht näher darauf ein und verweisen auf Abschnitt 3.4. Wir bezeichnen die Lösung von (3.99) in der k'-ten

Iteration weiterhin als $\boldsymbol{x}^*_{k'}$. Üblicherweise hatten wir bisher die jeweils neu erzeugten Iterierten mit $\boldsymbol{\alpha}_{ws}^{k'+1}$ bezeichnet, das VVP-Verfahren zählt jedoch zu den Verfahren mit einem Korrekturschritt

$$\boldsymbol{\alpha}_{ws}^{k'+1} = \theta_{k'}\boldsymbol{x}^*_{k'} + (1-\theta_{k'})\boldsymbol{\alpha}_{ws}^{k'} \; . \tag{3.103}$$

Zu den weiteren Details sei auf Abschnitt 3.3.2.3 verwiesen. Abgesehen von den offenen Details zum Korrekturschritt ist noch zu hinterfragen, ob wir es mit einem Abstiegsverfahren zu tun haben, d.h. ob ausgehend von einem nichtoptimalen $\boldsymbol{\alpha}_{ws}^{k'}$ ein $\boldsymbol{\alpha}_{ws}^{k'+1}$ generiert wird mit

$$W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'+1}) < W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'}) \; ? \tag{3.104}$$

Dazu zunächst der folgende Satz aus der unrestringierten Optimierung.

Satz 3.5 ([57], Lemma 3.1. S.66)**.** *Ein Vektor $\boldsymbol{s}^{k'}$ aus dem zulässigen Bereich von (3.99) ist Abstiegsrichtung im Punkt $\boldsymbol{\alpha}_{ws}^{k'}$, wenn gilt*

$$\left(\nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'})\right)^T \boldsymbol{s}^{k'} < 0 \; . \tag{3.105}$$

Bemerkung 3.9. *Der Satz ist nicht bewiesen, auf die hier notwendigen Schritte gehen wir an dieser Stelle kurz ein:*
Sei $\theta > 0$ und $\left(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}\right)$ zulässig. Dann gilt

$$W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}) = W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'}) + \theta\nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'})^T\boldsymbol{s} + \theta\|\boldsymbol{s}\|\zeta \; , \tag{3.106}$$

mit $\lim\limits_{\theta\to 0}\zeta = 0$. Das führt auf

$$\frac{W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}) - W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'})}{\theta} = \nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'})^T\boldsymbol{s} + \|\boldsymbol{s}\|\zeta \; . \tag{3.107}$$

Daraus folgt, es existiert ein $\delta > 0$, sodass für alle $\theta \in (0,\delta)$ gilt:

$$W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}) < W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'}) \; . \tag{3.108}$$

Wir verwenden $\boldsymbol{s}^{k'} = \boldsymbol{x}^*_{k'} - \boldsymbol{\alpha}_{ws}^{k'}$, siehe (3.103). Das führt zu der Frage, ob es sich dann wirklich um ein Abstiegsverfahren nach Definition (3.104) handelt? Diese werden wir auf Seite 71 beantworten.

Prinzipiell ist es nicht vorgeschrieben, mit der Matrix $\boldsymbol{I}_{ws}^{\rho_{k'}} = \frac{\boldsymbol{I}_{ws}}{\rho_{k'}}$ zu arbeiten. Man kann unter anderem in [126, 127] nachlesen, dass $\boldsymbol{P}^{\rho_{k'}} = \frac{\boldsymbol{P}}{\rho_{k'}}$ zulässig ist, sofern $\boldsymbol{P} \in \mathbb{R}^{ws,ws}$ eine positiv definite Matrix ist. Die zugehörige Norm ist dann analog zum allgemeinen

Projektionsverfahren zu verwenden und ist abhängig von der speziellen Matrix $\boldsymbol{P}$. Da Aufgaben der Form (3.99) innerhalb der VVP-Methode mehrmalig betrachtet werden, ist es natürlich wichtig, dass $\boldsymbol{P}^{\rho_{k'}}$ im Gegensatz zu $\boldsymbol{Q}_{\mathrm{dd}}$ eine günstige Struktur hat. Gewünscht sind dabei Diagonal- oder Blockdiagonalmatrizen [125, 128]; das Problem soll so gut wie möglich vereinfacht werden. Der von uns umgesetzte Algorithmus [116] zur Lösung von (3.99), welchen wir im Abschnitt 3.4 vorstellen werden, ist nur anwendbar für positive Diagonalmatrizen im quadratischen Teil der Zielfunktion. Bei Arbeit mit Matrizen $\boldsymbol{P}$, welche diese Voraussetzung nicht erfüllen, muss auf andere Methoden zurückgegriffen werden.

3.3.2.3 Korrektur (Schritt 3)

Im klassischen Verfahren der Projizierten Gradienten werden jeweils abwechselnd eine Projektion und eine neue Schrittweite berechnet. Im VVP-Verfahren schließt sich jedoch an die Berechnung der Projektion $\boldsymbol{x}^*_{k'}$ die Bestimmung der neuen Iterierten für die Lösung an. Wie schon erwähnt, setzen wir

$$\begin{aligned} \boldsymbol{\alpha}_{ws}^{k'+1} &= \theta_{k'}\boldsymbol{x}^*_{k'} + (1-\theta_{k'})\boldsymbol{\alpha}_{ws}^{k'} \\ &= \boldsymbol{\alpha}_{ws}^{k'} + \theta_{k'}(\boldsymbol{x}^*_{k'} - \boldsymbol{\alpha}_{ws}^{k'}) \\ &= \boldsymbol{\alpha}_{ws}^{k'} + \theta_{k'}\boldsymbol{s}^{k'} \,, \end{aligned} \tag{3.109}$$

wobei wir gleichzeitig

$$\boldsymbol{s}^{k'} = \boldsymbol{x}^*_{k'} - \boldsymbol{\alpha}_{ws}^{k'} \tag{3.110}$$

definieren. Wir bezeichnen $\theta_{k'} \in \mathbb{R}_+$ als den k'-ten Korrekturparameter des Verfahrens bzw. den k'-ten Schrittweitenparameter, je nachdem, wie das Verfahren interpretiert wird. Dieser wird über eine beschränkte Strahlminimierung bestimmt. $\theta_{k'}$ löst die Aufgabe [138]

$$\min_{\theta\in[0,1]} \quad S(\theta) := W_{\mathrm{d}}^{-}\left(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}^{k'}\right) \,. \tag{3.111}$$

Die Funktion S hat nach (3.64) die Form

$$S(\theta) = \frac{1}{2}\left(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}}\left(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}^{k'}\right) + \boldsymbol{q}^T\left(\boldsymbol{\alpha}_{ws}^{k'} + \theta\boldsymbol{s}^{k'}\right) \,. \tag{3.112}$$

Wegen Symmetrie von $\boldsymbol{Q}_{\mathrm{dd}}$ gilt

$$\left(\boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{s}^{k'} = \left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} \,.$$

Um das Minimum θ^* der Funktion S zu bestimmen, setzen wir die Ableitung nach θ gleich Null:

$$0 \stackrel{!}{=} \theta \left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} + \left(\boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} + \boldsymbol{q}^T \boldsymbol{s}^{k'} \tag{3.113}$$

und erhalten zunächst

$$\theta^* = \frac{-\left(\left(\boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} + \boldsymbol{q}^T \boldsymbol{s}^{k'}\right)}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} = \frac{-\left(\left(\boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} + \boldsymbol{q}^T\right) \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} , \tag{3.114}$$

also

$$\theta^* = \frac{-\left(\nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'})\right)^T \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} . \tag{3.115}$$

Wir betrachten zunächst den Fall $\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} > 0$. Dann sollte der Zähler von (3.115) nichtnegativ sein, damit θ^* positiv ist. Das gilt genau dann, wenn $\boldsymbol{s}^{k'}$ eine Abstiegsrichtung ist. Das soll jetzt gezeigt werden. Für zulässige Punkte $\boldsymbol{x}$ gilt [126]

$$\left(\boldsymbol{I}_{ws}^{\rho_{k'}} \left(\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'}\right) + \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{q}\right)^T (\boldsymbol{x} - \boldsymbol{x}_{k'}^*) \geq 0 . \tag{3.116}$$

Für den speziellen Fall $\boldsymbol{x} = \boldsymbol{\alpha}_{ws}^{k'}$ ergibt sich

$$\begin{aligned}
& \left(\boldsymbol{I}_{ws}^{\rho_{k'}} \left(\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'}\right) + \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{q}\right)^T \left(\boldsymbol{\alpha}_{ws}^{k'} - \boldsymbol{x}_{k'}^*\right) \geq 0 \\
\Rightarrow \quad & \left(\boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{q}\right)^T \left(\boldsymbol{\alpha}_{ws}^{k'} - \boldsymbol{x}_{k'}^*\right) \geq -\left(\boldsymbol{I}_{ws}^{\rho_{k'}} (\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'})\right)^T (\boldsymbol{\alpha}_{ws}^{k'} - \boldsymbol{x}_{k'}^*) \\
\stackrel{\boldsymbol{I}_{ws}^{\rho_{k'}} \text{ symmetrisch}}{\Rightarrow} \quad & \left(\boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{q}\right)^T \left(\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'}\right) \leq -(\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'})^T \boldsymbol{I}_{ws}^{\rho_{k'}} (\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'}) \\
\Rightarrow \quad & \nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'}) \boldsymbol{s}^{k'} \leq -\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{I}_{ws}^{\rho_{k'}} \boldsymbol{s}^{k'} .
\end{aligned}$$

Da $\boldsymbol{I}_{ws}^{\rho_{k'}}$ für alle k' eine positiv definite Matrix ist, gilt $\left(\nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^{k'})\right)^T \boldsymbol{s}^{k'} \leq 0$, d.h. wir haben es trotz der Korrekturformel (3.109) mit einem Abstiegsverfahren zu tun. Es bleibt noch zu zeigen, dass wir wirklich ein Minimum bestimmt haben. Dazu betrachten wir die rechte Seite der Gleichung (3.113) - die Ableitung von S. Für diese lineare Funktion in θ bilden wir erneut die Ableitung und sehen, dass

$$\nabla^2 S(\theta^*) = \left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}$$

gilt. Da wir angenommen hatten, dass $\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} > 0$ gilt, handelt es sich bei θ^* tatsächlich um ein Minimum. Das Verfahren setzt nicht voraus, dass die Matrix $\boldsymbol{Q}_{\mathrm{dd}}$ positiv definit ist. Deshalb kann $\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} \leq 0$ auftreten.[21] In diesem Fall lässt sich mittels (3.115) überhaupt kein positives θ^* berechnen, sodass dieser Fall ausgeschlossen werden muss [138]. Wir fassen zusammen:

$$\theta_{k'} = \begin{cases} 1 & \text{falls } \left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} \leq 0 \\ \min\left\{1; \theta^*\right\} & \text{sonst} \end{cases}. \tag{3.117}$$

Bemerkung 3.10 (Initialisierung)**.** *In einigen VP-Verfahren muss der Startvektor $\boldsymbol{\alpha}_{ws}^1$ nicht aus dem zulässigen Bereich von (3.58) stammen [125]. In dem von uns betrachteten Verfahren VVPM wird jedoch ein zulässiger Startpunkt verlangt. Um zu sichern, dass $\boldsymbol{\alpha}_{ws}^1$ zulässig ist, wird zunächst für $k' = 0$ die Aufgabe (3.95) mit $\boldsymbol{\alpha}_{ws}^0 = \boldsymbol{\alpha}_{ws}$ und $\rho_0 = 0$ gelöst. Im Anschluss daran wird jedoch kein θ_0 berechnet, sondern es gilt $\theta_0 = 1$. Setzt man dann für ein Update der Lösung die entsprechenden Werte in Gleichung (3.109) ein, erhält man*

$$\boldsymbol{\alpha}_{ws}^1 = \boldsymbol{\alpha}_{ws}^0 + \theta_0 \left(\boldsymbol{x}_0^* - \boldsymbol{\alpha}_{ws}^0\right) \tag{3.118}$$

und erkennt, dass die Wahl von $\theta_0 = 1$ zu $\boldsymbol{\alpha}_{ws}^1 = \boldsymbol{x}_0^*$ führt. $\boldsymbol{x}_0^*$ ist aber ein zulässiger Punkt von (3.58); das ergibt sich direkt aus dem Vergleich der Nebenbedingungen. Wir erwähnen das an dieser Stelle bewusst, denn der Zerlegungsalgorithmus übergibt der VVP-Routine keineswegs einen zulässigen Startvektor. Die Voraussetzung zur Anwendung von VVPM ist nicht erfüllt. Wir nutzen daher obige Überlegungen, um einen zulässigen Startvektor zu generieren.

3.3.2.4 Aktualisierung des Projektionsparameters (Schritt 4)

Ein wichtiger Parameter der Aufgabe (3.99) im zweiten Schritt der k'-ten Iteration ist $\rho_{k'}$. In den bekannten Verallgemeinerten Projektionsverfahren ist der positive Parameter ρ konstant [126]. Problematisch bei der Wahl von ρ ist die Tatsache, dass ein ρ, welches die Konvergenzkriterien dieses Verfahrens erfüllt, im Allgemeinen so klein ist, dass das Verfahren sehr langsam konvergiert. Die VVP-Methode aktualisiert diesen Projektionsparameter jeweils in Abhängigkeit der neu berechneten Abstiegsrichtung. Diese Aktualisierung erfolgt nach der Regel [138]

$$\rho_{k'+1} = \begin{cases} \rho_{\max} & \text{falls } \left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} \leq 0 \\ \min\left\{\rho_{\max}; \max\left\{\rho_{\min}; \rho_{k'+1}^{(r_{k'})}\right\}\right\} & \text{sonst} \end{cases}. \tag{3.119}$$

[21] Dieses Problem entfällt bei Verfahren mit streng konvexer Zielfunktion, z.B. in [125].

Der Index $r_{k'}$ hat die Aufgabe, festzulegen, nach welcher Vorschrift der Projektionsparameter berechnet werden soll. Wir betrachten zwei verschiedene Varianten, sodass $r_{k'} \in \{1, 2\}$ gilt. Die Vorschriften haben die Form

$$\rho_{k'+1}^{(1)} := \frac{\left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)^T \left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)}{\left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)} \tag{3.120}$$

bzw.[22]

$$\rho_{k'+1}^{(2)} := \frac{\left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)}{\left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}}^2 \left(\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\right)} \,. \tag{3.121}$$

Diese beiden Projektionsparameter sind im Jahr 1988 für die Lösung unrestringierter Optimierungsaufgaben mittels Gradientenverfahren vorgestellt worden [7] und sind als Konkurrenten des klassischen Cauchy-Prinzips (3.72) zu sehen. Die wichtige Aussage jener Arbeit ist, dass die Schrittweite nicht weniger Einfluss auf die Performance des Gradientenverfahrens hat, als die jeweilige Abstiegsrichtung und deshalb deren Auswahl genau zu überdenken sei. Zu diesem Zeitpunkt wurden die Regeln (3.120) und (3.121) nicht gemeinsam verwendet, die Ergebnisse in [7] sind parallel mit beiden Regeln produziert worden. Für das dortige Beispiel ist die Verbesserung erstaunlich. Während das Abbruchkriterium im klassischen Fall nach 183 Iterationen erfüllt war, benötigten die Verfahren mit den neuen Schrittweiten nur 27 bzw. 26 Iterationen. Ein kritischer Überblick zu diesen Schrittweiten ist im Jahr 2001 erschienen [50].

Bemerkung 3.11. *Dass das klassische Gradientenverfahren ineffizient arbeitet, ist schon seit langem bekannt. In den 50er Jahren wurde von Hestenes und Stiefel die Methode der konjugierten Gradienten (CG-Verfahren) entwickelt. Mittlerweile gibt es verschiedene Methoden, die in die Gruppe der CG-Verfahren eingeordnet werden. Zu den modifizierten Verfahren siehe beispielsweise [8], wo insbesondere auf Möglichkeiten der Vorkonditionierung eingegangen wird. Vorkonditionierung bewirkt bei guter Wahl des Vorkonditionierers Einsparung von Rechenzeit und Erhöhung der Stabilität des Verfahrens.*

Das VVP-Varfahren startet bei $k' = 1$ entweder mit $r_1 = 1$ oder mit $r_1 = 2$. Wir erklären später, warum es sinnvoll ist, die beiden Regeln im Wechsel zu verwenden, anstatt sich auf eine Möglichkeit festzulegen. Wir stellen dann auch ein Verfahren vor, welches definiert, wann jeweils der Wechsel zwischen den beiden Vorschriften zu erfolgen hat. Anwendung von (3.109) führt auf

[22] Für eine quadratische Matrix $\boldsymbol{A}$ bezeichne $\boldsymbol{A}^2$ im Folgenden das Matrixprodukt $\boldsymbol{A}\boldsymbol{A}$.

$$\rho_{k'+1}^{(r_{k'})} = \begin{cases} \frac{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} & \text{falls } r_{k'} = 1 \\ \frac{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}}^2 \boldsymbol{s}^{k'}} & \text{sonst} \end{cases} . \tag{3.122}$$

Obige Update-Regeln verändern sich, wenn die Matrix $\boldsymbol{P}$ nicht die Einheitsmatrix, sondern eine andere positive Diagonalmatrix ist, wie folgt:

$$\rho_{k'+1}^{(r_{k'})} = \begin{cases} \frac{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{P} \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} & \text{falls } r_{k'} = 1 \\ \frac{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{P}^{-1} \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} & \text{sonst} \end{cases} . \tag{3.123}$$

Man lese dazu beispielsweise in [136, 162] nach, wobei in diesen Arbeiten die Aktualisierung des Projektionsparameters allgemein in einer anderen Variante beschrieben wird. Sie hat die Form

$$\rho_{k'+1} = \begin{cases} \rho_{k'} & \text{falls } \|\boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}\|^2 \leq \epsilon \|\boldsymbol{s}^{k'}\|^2 \\ \begin{cases} \frac{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{P}^{-1} \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} & \text{falls} \quad \mod(k', \iota) \geq \frac{\iota}{2} \\ \frac{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{P} \boldsymbol{s}^{k'}}{\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}} & \text{sonst} \end{cases} & \text{sonst .} \end{cases} \tag{3.124}$$

Die Bedingung $\|\boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'}\|^2 \leq \epsilon \|\boldsymbol{s}^{k'}\|^2$ ist für hinreichend kleine ϵ äquivalent dazu, dass

$$\left(\boldsymbol{s}^{k'}\right)^T \boldsymbol{Q}_{\mathrm{dd}} \boldsymbol{s}^{k'} = 0$$

gilt. Sie ist ausreichend, da in diesen Fällen positiv semidefinite Matrizen vorausgesetzt werden. $\iota \in \mathbb{N}_+$ ist ein Wechselindikator. Der Ausdruck

$$\mod(k', \iota) \geq \frac{\iota}{2}$$

sorgt dafür, dass jeweils abwechselnd eine der beiden Regeln $\frac{\iota}{2}$-halbe mal angewendet wird. Diese Idee wurde in [163] entwickelt. Über ungerade Werte für ι werden keine Aussagen gemacht, da ohnehin vorgeschlagen wird, $\iota = 6$ zu verwenden, d.h. nach jeweils 3 Iterationen die Update-Regel zu wechseln. Bevor die Idee aufkam, die beiden Regeln (3.120) und (3.121) zu kombinieren, wurde $\rho_{k'+1}^{(2)}$ favorisiert [125–128]. Eine Anwendung

von $\rho^{(1)}_{k'+1}$ findet sich in [128] als Spezialfall des Projektionsparameters für die sogenannte AVP-Methode (*adaptive variable projection method*), wenn man den dortigen Parameter η auf den Wert 1 setzt. Es zeigte sich, dass die Kombination dieser Regeln im Vergleich zu der alleinigen Verwendung einer Regel bessere Ergebnisse liefert [163]. Die Strategie, die Regel jeweils nach dreimaligem Gebrauch zu wechseln, führte zu guten Ergebnissen. Ausgehend davon wurde in [138] ein dynamisches System entwickelt, welches beide Projektionsparameter berechnet und den besseren auswählt. Diese Neuentwicklung führte auch zu dem neuen Namen des Verfahrens (VVPM), denn hier handelt es sich wirklich um eine Verallgemeinerung – die Verallgemeinerung der Wahl des Projektionsparameters. Im Anschluss beschreiben wir die Wechselstrategie.

Zunächst werden natürliche Zahlen $n_{\min}$ und $n_{\max}$ als die minimale und maximale Anzahl der durchgängigen Anwendung einer Regel gewählt. Dabei muss natürlich $0 < n_{\min} \leq n_{\max}$ gelten. n_{akt} gebe an, wie oft die zuletzt verwendete Regel nach dem letzten Wechsel schon benutzt wurde. Dann wechseln wir zu der anderen Regel falls

- $n_{\text{akt}} = n_{\max}$ oder
- $n_{\min} \leq n_{\text{akt}} < n_{\max}$ und ein Wechselkriterium erfüllt ist.

Wir nutzen zwei Wechselkriterien aus [138], die jetzt definiert werden.

Definition 3.8 (trennende Schrittweite)**.** *Wir betrachten im Schritt $k'+1$ die zuletzt berechnete Schrittweite $\rho_{k'}$. Wir berechnen $\rho^{(1)}_{k'+1}$ und $\rho^{(2)}_{k'+1}$ nach den Regeln (3.120) und (3.121). Gilt dann*

$$\rho^{(2)}_{k'+1} < \rho_{k'} < \rho^{(1)}_{k'+1} \,, \tag{3.125}$$

so erklären wir $\rho_{k'}$ zu einer trennenden Schrittweite und wechseln die Berechnungsvorschrift im $(k'+1)$-ten Schritt.

Definition 3.9 (Schrittweite schlechten Abstiegs)**.** *Wir berechnen θ^* nach der Regel (3.114). θ_u und θ_o seien zwei relle Konstanten, welche die Bedingungen $0 < \theta_u \leq 1 \leq \theta_o$ erfüllen. Dann nennen wir $\rho_{k'}$ eine Schrittweite schlechten Abstiegs, falls*

- $\theta^* < \theta_u$ *und* $r_{k'} = 1$ *oder*
- $\theta^* > \theta_o$ *und* $r_{k'} = 2$.

Auch in diesem Fall wird im $(k'+1)$-ten Schritt die alternative Berechnungsvorschrift gewählt. θ_u und θ_o sind während der Laufzeit konstant, können aber zu Beginn gewählt werden. Hinweise zur sinnvollen Wahl dieser Parameter kann man in [138] finden.

Obige Wechselkriterien und deren Anwendung sind erstmals in [138] vorgestellt worden und zeichnen sich dadurch aus, dass sie die bis dahin existierenden Formen zur Generierung der Projektionsparameter als Spezialfälle enthalten. Setzt man beispielsweise $n_{\min} = n_{\max}$ erreicht man einen garantierten Wechsel nach genau $n_{\min}$ Iterationen. Weist man $n_{\min}$ einen sehr hohen Wert zu, kommt es dazu, dass während der Laufzeit überhaupt nicht gewechselt wird.

Definition 3.10 (Initialisierung)**.** *Bisher ist nicht festgelegt worden, mit welchem Wert die Folge $\{\rho_{k'}\}_{k'>0}$ initialisiert wird. Die Angaben in [138] besagen, dass zur Bestimmung von ρ_1 die folgende Aufgabe gelöst werden sollte:*

$$\rho_1 = \frac{1}{\left\| Proj_{\Omega}\left[\boldsymbol{\alpha}_{ws}^1 - \nabla W_{\mathrm{d}}^-(\boldsymbol{\alpha}_{ws}^1)\right] - \boldsymbol{\alpha}_{ws}^1 \right\|_\infty} \tag{3.126}$$

Die Projektion im Nenner entspricht genau der Aufgabe (3.95) für $\rho = 1$. Wir wählen $[\rho_{\min}, \rho_{\max}] = [10^{-30}, 10^{30}]$, wie auch in [138] (vermutlich übernommen aus [11]).

3.3.2.5 Abbruchbedingung (Schritt 5)

Im fünften Schritt des VVP-Verfahrens ist zu überprüfen, ob der neu berechnete Punkt $\boldsymbol{\alpha}_{ws}^{k'+1}$ die Aufgabe (3.64) ausreichend genau löst, oder ob das Verfahren weiterlaufen soll. Eine Abbruchbedingung der Variablen-Projektionsmethode wird beispielsweise in [125] vorgeschlagen. Der Algorithmus stoppt, sobald

$$\frac{\|\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'}\|}{\|\boldsymbol{\alpha}_{ws}^{k'}\|} \leq \varepsilon \tag{3.127}$$

auftritt. Dabei ist ε ein konstanter Parameter und sollte etwa zwischen 10^{-12} und 10^{-6} liegen.

Bemerkung 3.12. *Wir übernehmen die Abbruchbedingung* (3.127) *für das VVP-Varfahren mit $\varepsilon = 10^{-10}$. Unsere Tests haben gezeigt, dass für kleinere Werte der innere Löser zu häufig gerufen wird und für größere Werte die Ergebnisse des VVP-Verfahrens zu schlecht sind, um den Zerlegungsalgorithmus schnell voranzutreiben. Je nach Datensatz reagieren die Verfahren natürlich immer unterschiedlich. Um die Konvergenz des Gesamtverfahrens zu verbessern, wäre es auch möglich, zu Beginn der Optimierung einen größeren Wert zu wählen und diesen dann zum Ende hin zu verkleinern. Der Hintergrund dieser Überlegung ist, dass der innere Löser zu Beginn der Optimierung keine genauen Ergebnisse liefern muss, da sich der Lösungsvektor noch oft ändern wird. Ein grobes Abbruchkriterium des inneren Lösers ist ausreichend, um Fortschritte der Zerlegungsmethode zu erzielen und spart Zeit. Bei fortgeschrittener Optimierung der umgebenden Schleife muss ε dann verkleinert werden.*

Die Update-Formel (3.109) sowie die Definition von $\boldsymbol{s}^{k'}$ auf Seite 70 führen auf

$$\boldsymbol{\alpha}_{ws}^{k'+1} - \boldsymbol{\alpha}_{ws}^{k'} = \theta_{k'}\left(\boldsymbol{x}_{k'}^* - \boldsymbol{\alpha}_{ws}^{k'}\right) = \theta_{k'}\boldsymbol{s}^{k'} \ .$$

Das ermöglicht uns, die Abbruchbedingung über

$$\theta_{k'}\|\boldsymbol{s}^{k'}\| \leq \varepsilon\|\boldsymbol{\alpha}_{ws}^{k'}\| \tag{3.128}$$

zu prüfen.

3.3.2.6 Effizienz

Abgesehen von Aufgabe (3.99) liegt die Hauptlast des VVP-Verfahrens in der Berechnung des Matrix-Vektor-Produktes $\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{s}^{k'}$ in den Schritten 3 und 4. An dieser Stelle werden wir einige Techniken vorstellen, welche die Rechenlast des Verfahrens reduzieren.

Die Aufgabe (3.99) wird in jedem Iterationsschritt an den von uns gewählten inneren Löser übergeben. Wie später zu sehen sein wird, ist es wichtig, dass im quadratischen Teil der Aufgabe eine positive Diagonalmatrix steht. Für alle $i = 1, .., ws$ gilt

$$\{I_{ws}^{\rho_{k'}}\}_{ii} = \frac{1}{\rho_{k'}} .$$

Auf Seite 67 wurde ρ_1 als positiv festgelegt. Diese Initialisierung, die positive Semidefinitheit von $\boldsymbol{Q}_{\mathrm{dd}}$, die Update-Regel (3.119) und die Abbruchbedingung (3.128) sorgen dafür, dass auch alle weiteren $\rho_{k'}$ positiv sind. Wir vereinfachen die Darstellung zu

$$\begin{aligned}
\boldsymbol{x}_{k'}^{*} &= \underset{\boldsymbol{x}\in\Omega}{\arg\min} \left\{ \frac{1}{2}\boldsymbol{x}^T \frac{\boldsymbol{I}_{ws}}{\rho_{k'}}\boldsymbol{x} + \left(\boldsymbol{q} + \left(\boldsymbol{Q}_{\mathrm{dd}} - \frac{\boldsymbol{I}_{ws}}{\rho_{k'}} \right) \boldsymbol{\alpha}_{ws}^{k'} \right)^T \boldsymbol{x} \right\} \quad \Leftrightarrow \\
\boldsymbol{x}_{k'}^{*} &= \underset{\boldsymbol{x}\in\Omega}{\arg\min} \left\{ \frac{1}{2}\boldsymbol{x}^T \boldsymbol{x} + \left(\rho_{k'}\boldsymbol{q} + (\rho_{k'}\boldsymbol{Q}_{\mathrm{dd}} - \boldsymbol{I}_{ws}) \boldsymbol{\alpha}_{ws}^{k'} \right)^T \boldsymbol{x} \right\} \quad \Leftrightarrow \\
\boldsymbol{x}_{k'}^{*} &= \underset{\boldsymbol{x}\in\Omega}{\arg\min} \left\{ \frac{1}{2}\boldsymbol{x}^T \boldsymbol{x} + \left(\rho_{k'} \left(\boldsymbol{q} + \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} \right) - \boldsymbol{\alpha}_{ws}^{k'} \right)^T \boldsymbol{x} \right\} .
\end{aligned}$$

Abschnitt 3.4 geht dann von der Aufgabe

$$\min_{\boldsymbol{x}\in\mathbb{R}^{ws}} \quad \frac{1}{2}\boldsymbol{x}^T \boldsymbol{x} + \left(\rho_{k'} \left(\boldsymbol{q} + \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} \right) - \boldsymbol{\alpha}_{ws}^{k'} \right)^T \boldsymbol{x} \tag{3.129}$$

mit den Nebenbedingungen (3.100) bis (3.102) aus.

Bemerkung 3.13 (Erinnerung)**.** *Wie schon erwähnt wurde, wird zu Beginn des VVP-Verfahrens die Aufgabe (3.129) mit $\rho_0 = 0$ gelöst. Der Lösungsvektor $\boldsymbol{x}_1^*$ definiert dann den Startvektor $\boldsymbol{\alpha}_{ws}^1$. Im Anschluss daran wird $\boldsymbol{\alpha}_{ws}^1$ verwendet, um (3.129) für $\rho = 1$ zu lösen. Der Lösungsvektor wird benötigt, um das noch unbekannte ρ_1 zu berechnen. Eine weitere Verwendung gibt es nicht. Diese Aufgaben sollten erledigt werden, bevor die große Optimierungsschleife startet.*

Wir speichern jeweils in der k'-ten Iteration $\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'+1}$ im Vektor $\boldsymbol{t}^{k'+1}$ ab. $\boldsymbol{t}^1$ muss zunächst über $\boldsymbol{t}^1 = \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^1$ initialisiert werden. Die Aktualisierung erfolgt dann im vierten Schritt über die Formel

$$\begin{aligned} \boldsymbol{t}^{k'+1} = \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'+1} &\overset{(3.109)}{=} \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'} + \boldsymbol{Q}_{\mathrm{dd}}\theta_{k'}\left(\boldsymbol{x}_{k'}^{*} - \boldsymbol{\alpha}_{ws}^{k'}\right) \\ &\overset{(3.110)}{=} \boldsymbol{t}^{k'} + \theta_{k'}\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{s}^{k'} \ . \end{aligned}$$

Gehen wir zusätzlich davon aus, dass $\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{s}^{k'}$ im Vektor $\tilde{\boldsymbol{s}}^{k'}$ gespeichert wird, reduziert sich der Aufwand zu

$$\boldsymbol{t}^{k'+1} = \boldsymbol{t}^{k'} + \theta_{k'}\tilde{\boldsymbol{s}}^{k'} \ .$$

Der Vektor $\tilde{\boldsymbol{s}}^{k'}$ kann jeweils in Schritt 2 berechnet werden, nachdem das Optimierungsproblem gelöst wurde und $\boldsymbol{x}_{k'}^{*}$ zur Verfügung steht.

Ein wichtiger Vorteil einer Zielfunktion der Form (3.129) innerhalb des VVP-Verfahrens ist zunächst, dass im quadratischen Teil nichts angepasst werden muss. Außerdem ist

$$\rho_{k'}\left(\boldsymbol{q} + \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws}^{k'}\right) - \boldsymbol{\alpha}_{ws}^{k'}$$

schnell zu berechnen. Bei Verwendung von $\boldsymbol{t}^{k'}$ reduziert sich der Aufwand auf

$$\rho_{k'}\left(\boldsymbol{q} + \boldsymbol{t}^{k'}\right) - \boldsymbol{\alpha}_{ws}^{k'} \ . \tag{3.130}$$

Für die Berechnungen von $\theta_{k'}$ in Schritt 3 und $\rho_{k'+1}$ in Schritt 4 benötigen wir mehrmalig den Wert $\boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{s}^{k'}$ und profitieren dort von $\tilde{\boldsymbol{s}}^{k'}$. Weiterhin sollten $\left(\boldsymbol{\alpha}_{ws}^{k'}\right)^{T}\boldsymbol{Q}_{\mathrm{dd}}$ für (3.114) und $\left(\boldsymbol{s}^{k'}\right)^{T}\boldsymbol{Q}_{\mathrm{dd}}$ für (3.122) bekannt sein. Da $\boldsymbol{Q}_{\mathrm{dd}}$ symmetrisch ist, gelten

$$\left(\boldsymbol{\alpha}^{k'}\right)^{T}\boldsymbol{Q}_{\mathrm{dd}} \ = \ \left(\boldsymbol{t}^{k'}\right)^{T} \ ,$$

und

$$\left(\boldsymbol{s}^{k'}\right)^{T}\boldsymbol{Q}_{\mathrm{dd}} \ = \ \left(\tilde{\boldsymbol{s}}^{k'}\right)^{T} \ ,$$

und wir können erneut $\boldsymbol{t}^{k'}$ und $\tilde{\boldsymbol{s}}^{k'}$ nutzen.

Um eine bessere Übersicht zu schaffen, geben wir in Abbildung 3.6 das VVP-Verfahren in der Form an, wie wir es benutzen. Diese Abbildung stellt eine Ergänzung zu den sehr allgemeinen Schritten in Abbildung 3.5 dar. Der Iterationsindex k' ist in der Abbildung der Einfachheit halber weggelassen worden.

Eingabe: $\boldsymbol{\alpha}_{ws},\ \boldsymbol{Q}_{\mathrm{dd}},\ \boldsymbol{q},\ \boldsymbol{y}_{\mathrm{d}},\ e$ Konstanten: $C,\ \rho_{\min},\ \rho_{\max},\ n_{\min},\ n_{\max}$

berechne $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Omega} \left\{ \frac{1}{2}\boldsymbol{x}^T\boldsymbol{x} - \boldsymbol{\alpha}_{ws}^T\boldsymbol{x} \right\},\ \boldsymbol{\alpha}_{ws} = \boldsymbol{x}^*,\ \boldsymbol{t} = \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{\alpha}_{ws},\ \boldsymbol{p} = \boldsymbol{q} + \boldsymbol{t}$

$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Omega} \left\{ \frac{1}{2}\boldsymbol{x}^T\boldsymbol{x} + \left(\boldsymbol{p} - \boldsymbol{\alpha}_{ws}\right)^T\boldsymbol{x} \right\},\ \rho = \frac{1}{\|\boldsymbol{x}^* - \boldsymbol{\alpha}_{ws}\|_\infty},\ n_{\mathrm{akt}} = 1,\ r = 1$

berechne $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Omega} \left\{ \frac{1}{2}\boldsymbol{x}^T\boldsymbol{x} + \left(\rho\boldsymbol{p} - \boldsymbol{\alpha}_{ws}\right)^T\boldsymbol{x} \right\},\ \boldsymbol{s} = \boldsymbol{x}^* - \boldsymbol{\alpha}_{ws},\ \tilde{\boldsymbol{s}} = \boldsymbol{Q}_{\mathrm{dd}}\boldsymbol{s}$

$\boldsymbol{s}^T\tilde{\boldsymbol{s}} \leq 0$? (j / n)

- j: $\theta = 1$
- n: $\theta = \frac{-\boldsymbol{p}^T\boldsymbol{s}}{\boldsymbol{s}^T\tilde{\boldsymbol{s}}}$; $\theta > 1$? (j / n)
  - j: $\theta = 1$
  - n:

$\boldsymbol{\alpha}_{ws} = \boldsymbol{\alpha}_{ws} + \theta\boldsymbol{s},\ \boldsymbol{t} = \boldsymbol{t} + \theta\tilde{\boldsymbol{s}},\ \boldsymbol{p} = \boldsymbol{q} + \boldsymbol{t}$

$\boldsymbol{s}^T\tilde{\boldsymbol{s}} \leq 0$? (j / n)

- j: $\rho = \rho_{\max}$
- n: Berechnung von $\rho^{(1)}$, $\rho^{(2)}$; $n_{\mathrm{akt}} > n_{\min}$? (j / n)
  - j: $n_{\mathrm{akt}} > n_{\max}$ oder ρ erfüllt ein Wechselkriterium ? (j / n)
    - j: $r = \mathrm{mod}(r, 2) + 1,\ n_{\mathrm{akt}} = 1$
    - n:
  - n:

$\rho = \min\left\{\rho_{\max}, \max\left\{\rho_{\min}, \rho^{(r)}\right\}\right\}$

$n_{\mathrm{akt}} = n_{\mathrm{akt}} + 1$

solange $\theta\|\boldsymbol{s}\| > \varepsilon\|\boldsymbol{\alpha}_{ws}\|$

Abbildung 3.6: VVP-Algorithmus in effizienter Form.

3.4 Innerer Löser im Zerlegungsalgorithmus

Wir haben gesehen, dass innerhalb der Variablen-Projektionsmethode mehrmalig die quadratische Optimierungsaufgabe

$$\min_{\boldsymbol{x}\in\Omega} \quad \frac{1}{2}\boldsymbol{x}^T\boldsymbol{x} + \left(\rho_{k'}\left(\boldsymbol{q}+\boldsymbol{t}^{k'}\right) - \boldsymbol{\alpha}_{ws}^{k'}\right)^T \boldsymbol{x} \tag{3.131}$$

gelöst werden muss. Sie unterscheidet sich von (3.58) in der Struktur des quadratischen Terms, denn jetzt arbeiten wir mit der Einheitsmatrix anstelle der vollbesetzten Matrix $\boldsymbol{Q}_{\mathrm{dd}}$. Diese Eigenschaft macht sich der Algorithmus von Pardalos und Kovoor [116] zu Nutze. Er wird in diesem Abschnitt vorgestellt.

3.4.1 Transformation

Bevor wir damit beginnen, wird die Aufgabe (3.131) zunächst so dargestellt, wie sie der Algorithmus übergeben bekommt. Man beachte, dass bis auf den gesuchten Vektor $\boldsymbol{x}^*$ alle Daten konstant sind, d.h. der Ausdruck

$$\rho_{k'}\left(\boldsymbol{q}+\boldsymbol{t}^{k'}\right) - \boldsymbol{\alpha}_{ws}^{k'}$$

beinhaltet keinerlei variable Werte; auch die einzelnen Indizes der VVP-Methode interessieren uns hier nicht. Wir vermeiden überladene Notation indem wir definieren:

$$\begin{aligned} \hat{\boldsymbol{u}} &:= \rho_{k'}\left(\boldsymbol{q}+\boldsymbol{t}^{k'}\right) - \boldsymbol{\alpha}_{ws}^{k'}\,, \\ \hat{\boldsymbol{y}} &:= \boldsymbol{y}_{\mathrm{d}}\,. \end{aligned}$$

Wir erhalten dann die Aufgabe

$$\min_{\boldsymbol{x}} \quad \frac{1}{2}\boldsymbol{x}^T\boldsymbol{x} + \hat{\boldsymbol{u}}^T\boldsymbol{x}\,. \tag{3.132}$$

mit den Nebenbedingungen

$$\begin{aligned} \hat{\boldsymbol{y}}^T\boldsymbol{x} &= e\,, \\ \boldsymbol{0} &\leq \boldsymbol{x}\,, \\ \boldsymbol{x} &\leq \boldsymbol{C}\,. \end{aligned}$$

Der Algorithmus von Pardalos und Kovoor, den wir im Folgenden kurz als PK-Algortihmus bezeichnen werden, wird in [138] empfohlen. Er ist geeignet zur Lösung quadratischer Optimierungsaufgaben der Form (3.132), wobei im quadratischen Teil der Zielfunktion eine Diagonalmatrix $\boldsymbol{D}$ verwendet werden darf, d.h. der Algorithmus löst Aufgaben, bei denen der quadratische Term die Form $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{D}\boldsymbol{x}$ hat. Während an $\hat{\boldsymbol{u}}$ keine Bedingungen geknüpft werden, muss $\boldsymbol{D}$ jedoch eine streng positive Diagonalmatrix sein. Da wir ohnehin nur die Einheitsmatrix betrachten, erfüllen wir diese Voraussetzung. Die Methoden aus [15] und [109] sind ebenfalls geeignet und werden neben [116] in den Arbeiten [136, 162, 163] empfohlen.

Bemerkung 3.14. *In [116] wird eine negative Diagonalmatrix gefordert. Davon sollte man sich nicht stören lassen, da dort eine entsprechende Aufgabe maximiert wird.*

Bemerkung 3.15. *Technisch gesehen, setzt sich die Methode in [116] mit einem anders strukturierten Problem auseinander. Es hat die Form*

$$\min_{\boldsymbol{z}} \quad \sum_{i=1}^{ws} c_i z_i^2 \tag{3.133}$$

mit

$$\begin{aligned} \sum_{i=1}^{ws} c_i z_i &= d\,, \\ \boldsymbol{a} &\leq \boldsymbol{z}\,, \\ \boldsymbol{z} &\leq \boldsymbol{b}\,, \end{aligned}$$

und muss zunächst durch Datentransformation aufgestellt werden.

Eine geeignete Transformation für unsere Aufgabe lässt sich leicht bestimmen. Sie hat die Form

$$z_i = \frac{x_i D_{ii} + \hat{u}_i}{2\hat{y}_i} \quad (i = 1, .., ws)\,. \tag{3.134}$$

Obwohl wir mit $\boldsymbol{D} = \boldsymbol{I}_{ws}$ arbeiten, achten wir darauf, den Algorithmus in der allgemeinen Form darzustellen. Aus der Transformationsvorschrift (3.134) berechnen wir nun die Parameter $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ und d. Für die Zielfunktion gilt

$$\begin{aligned}
\frac{1}{2}\boldsymbol{x}^T\boldsymbol{D}\boldsymbol{x} + \hat{\boldsymbol{u}}^T\boldsymbol{x} &= \frac{1}{2}\sum_{i=1}^{ws} x_i^2 D_{ii} + \sum_{i=1}^{ws} \hat{u}_i x_i \\
&\overset{(3.134)}{=} \frac{1}{2}\sum_{i=1}^{ws} D_{ii}\left(\frac{4z_i^2\hat{y}_i^2 - 4z_i\hat{y}_i\hat{u}_i + \hat{u}_i^2}{D_{ii}^2}\right) + \sum_{i=1}^{ws}\hat{u}_i\left(\frac{2z_i\hat{y}_i - \hat{u}_i}{D_{ii}}\right) \\
&= 2\sum_{i=1}^{ws}\frac{\hat{u}_i z_i \hat{y}_i}{D_{ii}} - \sum_{i=1}^{ws}\frac{\hat{u}_i^2}{D_{ii}} + 2\sum_{i=1}^{ws}\frac{\hat{y}_i^2 z_i^2}{D_{ii}} - 2\sum_{i=1}^{ws}\frac{\hat{u}_i z_i \hat{y}_i}{D_{ii}} + \frac{1}{2}\sum_{i=1}^{ws}\frac{\hat{u}_i^2}{D_{ii}} .
\end{aligned}$$

Zusammenfassen und Streichen von Konstanten, die für die Minimierungsaufgabe nicht relevant sind, führen zu

$$\min_{\boldsymbol{z}} \quad \sum_{i=1}^{ws}\frac{\hat{y}_i^2 z_i^2}{D_{ii}} .$$

Für binäre Klassifikationsprobleme gilt $\hat{y}_i \in \{-1, 1\}$ und somit $\hat{y}_i^2 = 1$, sodass

$$\min_{\boldsymbol{z}} \quad \sum_{i=1}^{ws}\frac{z_i^2}{D_{ii}}$$

übrig bleibt. Wir halten daher fest:

$$c_i = \frac{1}{D_{ii}} \quad (i = 1, .., ws) . \tag{3.135}$$

c_i $(i = 1, .., ws)$ existieren immer, da die Matrix $\boldsymbol{D}$ nach Voraussetzung keine Nullelemente auf der Diagonalen besitzen darf.

Betrachten wir die Transformation der Nebenbedingungungen:

$$\begin{aligned}
\hat{\boldsymbol{y}}^T\boldsymbol{x} = e \quad &\overset{(3.134)}{\Leftrightarrow} \quad \sum_{i=1}^{ws}\hat{y}_i\left(\frac{2z_i\hat{y}_i - \hat{u}_i}{D_{ii}}\right) = e \\
&\Leftrightarrow \quad 2\sum_{i=1}^{ws}\frac{z_i\hat{y}_i^2}{D_{ii}} = \sum_{i=1}^{ws}\frac{\hat{y}_i\hat{u}_i}{D_{ii}} + e \\
&\Leftrightarrow \quad \sum_{i=1}^{ws}\frac{z_i}{D_{ii}} = \frac{1}{2}\left(\sum_{i=1}^{ws}\frac{\hat{y}_i\hat{u}_i}{D_{ii}} + e\right) .
\end{aligned}$$

Daraus folgt direkt

$$d = \frac{1}{2}\left(\sum_{i=1}^{ws} \frac{\hat{y}_i \hat{u}_i}{D_{ii}} + e\right) .$$

Für $\mathbf{0} \leq \boldsymbol{x}$ bzw. $\boldsymbol{x} \leq \boldsymbol{C}$ liefert die Transformation:

$$0 \leq \frac{2z_i\hat{y}_i - \hat{u}_i}{D_{ii}} \quad \Rightarrow \quad \hat{u}_i \leq 2z_i\hat{y}_i$$

und

$$\frac{2z_i\hat{y}_i - \hat{u}_i}{D_{ii}} \leq C \quad \Rightarrow \quad 2z_i\hat{y}_i \leq D_{ii}C + \hat{u}_i .$$

Für z_i ergeben sich in Abhängigkeit von $\hat{y}_i$ unterschiedliche Vorzeichen. Wir betrachten zunächst den Fall $\hat{y}_i = 1$:

$$\frac{\hat{u}_i}{2} \leq z_i \leq \frac{D_{ii}C + \hat{u}_i}{2} \quad \Rightarrow \quad a_i = \frac{\hat{u}_i}{2},\ b_i = \frac{D_{ii}C + \hat{u}_i}{2} .$$

Für $\hat{y}_i = -1$ gilt:

$$\frac{D_{ii}C + \hat{u}_i}{-2} \leq z_i \leq \frac{\hat{u}_i}{-2} \quad \Rightarrow \quad a_i = \frac{D_{ii}C + \hat{u}_i}{-2},\ b_i = \frac{\hat{u}_i}{-2} .$$

Damit sind die notwendigen Vorbetrachtungen abgeschlossen.

3.4.2 Eigenschaften der Lösung

Wir versuchen ein quadratisches Problem mit linearen Nebenbedingungen zu lösen. Es ist bekannt [36], dass solche Aufgaben keine lokalen Minima aufweisen. Weiterhin können wir wiederum ausnutzen, dass ein Punkt, welcher die Karush-Kuhn-Tucker-Bedingungen erfüllt, automatisch der gesuchte Lösungspunkt ist. Auf dieser Grundlage arbeitet auch der PK-Algorithmus.

Bemerkung 3.16. *Das System der KKT-Bedingungen zur Aufgabe (3.133) hat folgende Form:*

$$
\begin{aligned}
2\boldsymbol{c}\boldsymbol{z} + \nu\boldsymbol{c} + \boldsymbol{\mu} - \boldsymbol{\lambda} &= \mathbf{0}\,, && (3.136)\\
\sum_{i=1}^{ws} c_i z_i - d &= 0\,, && (3.137)\\
\boldsymbol{a} - \boldsymbol{z} &\leq \mathbf{0}\,, && (3.138)\\
\boldsymbol{z} - \boldsymbol{b} &\leq \mathbf{0}\,, && (3.139)\\
\boldsymbol{\lambda}(\boldsymbol{a} - \boldsymbol{z}) &= \mathbf{0}\,, && (3.140)\\
\boldsymbol{\mu}(\boldsymbol{z} - \boldsymbol{b}) &= \mathbf{0}\,, && (3.141)\\
\boldsymbol{\lambda} &\geq \mathbf{0}\,, && (3.142)\\
\boldsymbol{\mu} &\geq \mathbf{0}\,. && (3.143)
\end{aligned}
$$

Zur Aufstellung von (3.136) bis (3.143) siehe Definition 3.1. und Satz 3.4 in [36].

Die zentrale Aussage des PK-Algorithmus geben wir im folgenden Satz wider.

Satz 3.6 ([116])**.** *Sei $\boldsymbol{z} \in \mathbb{R}^{ws}$ ein zulässiger Punkt von (3.133). Dann sind folgende Aussagen äquivalent:*

1. *Es gibt ein $\kappa^* \in \mathbb{R}$, sodass für alle z_i $(i = 1, .., ws)$*

$$
z_i = \begin{cases} a_i & \textit{falls } \kappa^* < a_i \\ \kappa^* & \textit{falls } a_i \leq \kappa^* \leq b_i \\ b_i & \textit{falls } b_i < \kappa^* \end{cases} \qquad (3.144)
$$

   gilt.

2. *$\boldsymbol{z}$ ist globale Lösung der Aufgabe (3.133).*

Beweis: 1. $\Rightarrow$ 2.: Wir zeigen die Gültigkeit der KKT-Bedingungen.

Da $\boldsymbol{z}$ nach Voraussetzung ein zulässiger Punkt ist, erfüllt er die Nebenbedingungen von (3.133). Die Bedingungen (3.137), (3.138) und (3.139) sind gesichert.

Seien

$$
\begin{aligned}
\nu &= -2\kappa^*\,,\\
\lambda_i &= 2c_i \max\{0, a_i - \kappa^*\} \quad \forall i = 1, .., ws\,,\\
\mu_i &= 2c_i \max\{0, \kappa^* - b_i\} \quad \forall i = 1, .., ws\,,
\end{aligned}
$$

dann sind (3.142) und (3.143) erfüllt, denn nach (3.135) gilt $\boldsymbol{c} \geq \mathbf{0}$.

Die restlichen Bedingungen zeigen wir über Fallunterscheidung.

I. Falls $b_i < \kappa^*$, dann gilt wegen (3.144) $z_i = b_i$. Wegen $a_i \leq b_i$ folgt außerdem $a_i - \kappa^* < 0$. Wir erhalten $\lambda_i = 0$ und $\mu_i = 2c_i(\kappa^* - b_i)$ und es gelten (3.136), (3.140) und (3.141).

II. Falls $a_i > \kappa^*$, dann gilt wegen (3.144) $z_i = a_i$. Wegen $a_i \leq b_i$ folgt außerdem $b_i - \kappa^* > 0$. Wir erhalten $\lambda_i = 2c_i(a_i - \kappa^*)$ und $\mu_i = 0$ und es gelten (3.136), (3.140) und (3.141).

III. Falls $a_i \leq \kappa^* \leq b_i$, dann gilt wegen (3.144) $z_i = \kappa^*$. Es folgt außerdem $a_i - \kappa^* \leq 0$ und $\kappa^* - b_i \leq 0$. Wir erhalten $\lambda_i = 0$ und $\mu_i = 0$ und es gelten (3.136), (3.140) und (3.141).

Da alle KKT-Bedingungen erfüllt sind, ist $\boldsymbol{z}$ globale Lösung der Aufgabe (3.133).

2. $\Rightarrow$ 1.: Wir zeigen die Existenz von $\kappa^* \in \mathbb{R}$ mit den angegebenen Eigenschaften.

Sei $\boldsymbol{z}$ globale Lösung der Aufgabe (3.133), dann existieren $\nu \in \mathbb{R}, \boldsymbol{\lambda} \in \mathbb{R}^{ws}$ sowie $\boldsymbol{\mu} \in \mathbb{R}^{ws}$, welche die KKT-Bedingungen erfüllen. Aus der ersten Bedingung (3.136) erhalten wir

$$z_i + \frac{\nu}{2} = \frac{\lambda_i - \mu_i}{2c_i} \,. \tag{3.145}$$

I. Falls $z_i + \frac{\nu}{2} < 0$, gilt wegen (3.142) und (3.143) $\mu_i \neq 0$. Daraus und aus (3.141) folgt $z_i = b_i$.

II. Falls $z_i + \frac{\nu}{2} > 0$ gilt wegen (3.142) und (3.143) $\lambda_i \neq 0$. Daraus und aus (3.140) folgt $z_i = a_i$.

Wir prüfen jetzt, wo sich $\frac{\nu}{2}$ bezüglich des Intervalls $[a_i, b_i]$ befinden kann und welche Konsequenzen das für die Existenz eines geeigneten κ^* hat:

$$\begin{aligned}
b_i + \frac{\nu}{2} < 0 \quad &\Rightarrow \quad z_i + \frac{\nu}{2} < 0 \quad \overset{I.}{\Rightarrow} \quad z_i = b_i \,, \\
a_i + \frac{\nu}{2} > 0 \quad &\Rightarrow \quad z_i + \frac{\nu}{2} > 0 \quad \overset{II.}{\Rightarrow} \quad z_i = a_i \,, \\
&\qquad a_i \leq -\frac{\nu}{2} \leq b_i \quad \overset{I.,II.}{\Rightarrow} \quad z_i = -\frac{\nu}{2} \,.
\end{aligned}$$

Letzterer Fall mag nicht gleich zu sehen sein. Nehmen wir an, dass $z_i < -\frac{\nu}{2}$ gilt, dann entspricht das Fall I. und wir haben $z_i = b_i$, was aber wegen $z_i < -\frac{\nu}{2} \leq b_i$ unmöglich ist. Behaupten wir $z_i > -\frac{\nu}{2}$ führt das analog zum Widerspruch. Es bleibt nur $z_i = -\frac{\nu}{2}$ übrig.

Wir fassen zusammen, die Wahl von $\kappa^* = -\frac{\nu}{2}$ sichert die Eigenschaft (3.144). $\square$

Bemerkung 3.17. *Satz 3.6 macht keine Aussagen darüber, ob es überhaupt zulässige Punkte gibt. Untersuchung der drei Nebenbedingungen ergibt, dass die Aufgabe (3.133) keine zulässigen Punkte besitzt, falls*

$$\sum_{i=1}^{ws} c_i a_i > d \quad \textit{oder} \quad \sum_{i=1}^{ws} c_i b_i < d \; . \tag{3.146}$$

Diese Fälle sollten ausgeschlossen werden. Es muss also insbesondere gelten

$$\sum_{i=1}^{ws} c_i a_i \leq d \leq \sum_{i=1}^{ws} c_i b_i \; . \tag{3.147}$$

Wir haben in Satz 3.6 gesehen, dass für alle i z_i Funktionen von κ sind. Wir stellen diese Abhängigkeit in Abbildung 3.7 dar. Alle $z_i(\kappa)$ sind stückweise linear und monoton

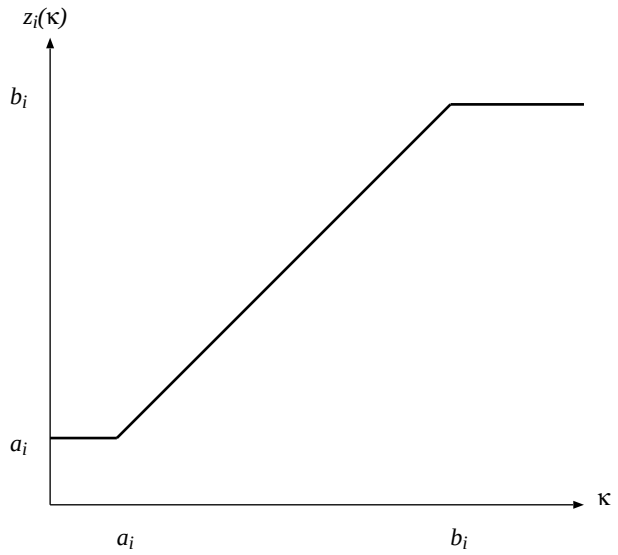

Abbildung 3.7: Beispiel für z_i als Funktion von κ.

wachsend.

Definition 3.11. *Im Folgenden bezeichnen wir die Stellen, an denen eine stückweise lineare Funktion ihren Anstieg ändert als Bruchstellen.*

Definition 3.12 ([116])**.** *Die Funktion $\psi : \mathbb{R} \to \mathbb{R}$ sei definiert als*

$$\psi(\kappa) = \sum_{i=1}^{ws} c_i z_i(\kappa) \; . \tag{3.148}$$

Folgerung 3.3. *Offensichtlich ist ψ ebenfalls stückweise linear und monoton wachsend. Seine Bruchstellen liegen bei a_i, b_i $(i = 1, .., ws)$, Dopplungen sind nicht ausgeschlossen.*

Satz 3.7. *Findet man ein $\kappa^* \in \mathbb{R}$ mit*

$$\psi(\kappa^*) = \sum_{i=1}^{ws} c_i z_i(\kappa^*) = d \,, \tag{3.149}$$

wobei $z_i(\kappa^*)$ für alle $i = 1, .., ws$ der Vorschrift (3.144) entsprechen, hat man die Lösung $\boldsymbol{z}^*$ von (3.133) lokalisiert.

Beweis: $\boldsymbol{z}(\kappa^*)$ ist ein zulässiger Punkt von (3.133) und erfüllt deshalb die Voraussetzung des Satzes 3.6. Dessen Anwendung liefert die Behauptung. □

3.4.3 Lokalisierung eines KKT-Punktes

Nachdem wir bisher gezeigt haben, von welcher Struktur die Lösung ist und welche Bedingungen an κ^* gestellt werden, erklären wir jetzt, nach welcher Idee κ^* lokalisiert wird. Sei $[L, R]$ das kleinste Intervall, welches κ^* enthält, wobei L und R Bruchstellen der Funktion ψ sein sollen. Dann befinden sich zwischen L und R keine weiteren Bruchstellen. Wir veranschaulichen das anhand der Abbildung 3.8. Sie zeigt beispielhaft, wie die Funktion ψ

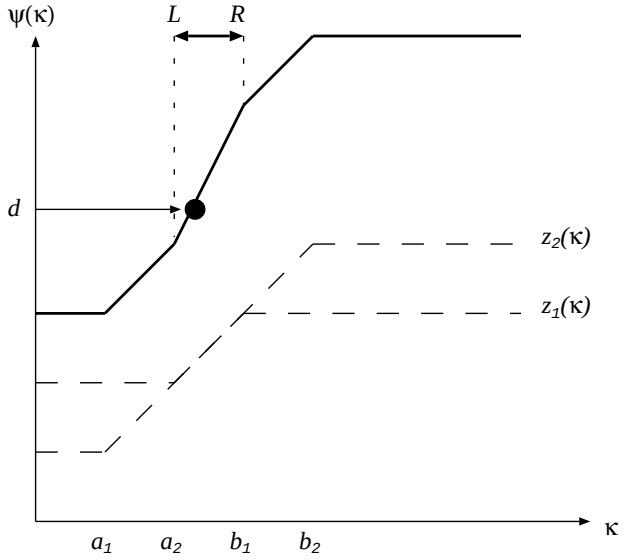

Abbildung 3.8: Beispiel für die Bildung der Funktion ψ.

aus zwei Funktionen $z_1(\kappa)$ und $z_2(\kappa)$ entsteht. In diesem Fall haben wir für eine einfache Darstellung angenommen, dass $c_1 = c_2 = 1$ gilt. Das entspricht auch unserer Wahl von

$\boldsymbol{D} = \boldsymbol{I}_{ws}$. Nach Satz 3.6 gilt zunächst für alle i mit $[a_i, b_i] \leq [L, R] : z_i = b_i$ und analog für $[L, R] \leq [a_i, b_i] : z_i = a_i$. In [116] sind diese Vorschriften als strikte Ungleichungen angegeben. Das ist jedoch nicht korrekt[23] und kann zu Fällen führen, die unbestimmt sind. Jetzt bleiben nur noch Fälle übrig mit $a_i \leq L$ und $b_i \geq R$, denn Bruchstellen in $[L, R]$ hatten wir ausgeschlossen. Das sind diejenigen Variablen, für die nach Regel (3.144) $z_i = \kappa^*$ gelten soll. Wir betrachten nun die Bedingung (3.149). Diese spalten wir auf in

$$d = \sum_{b_i \leq L} c_i b_i + \sum_{a_i \geq R} c_i a_i + \sum_{a_i \leq L, R \leq b_i} c_i \kappa^* \tag{3.150}$$

und erhalten durch umstellen eine explizite Formel für die Berechnung von κ^*:

$$\kappa^* = \frac{d - \left(\sum_{b_i \leq L} c_i b_i + \sum_{a_i \geq R} c_i a_i\right)}{\sum_{a_i \leq L, R \leq b_i} c_i} \,. \tag{3.151}$$

Bemerkung 3.18 (Existenz von κ^*)**.** *Man könnte kritisch bemerken, dass in (3.151) der Fall*

$$\sum_{a_i \leq L, R \leq b_i} c_i = 0$$

auftreten könnte. Das passiert dann, wenn es kein $i \in \{1, .., ws\}$ mit $[L, R] \subseteq [a_i, b_i]$ gibt, denn $c_i = 0$ kann nicht auftreten. Das bedeutet jedoch, dass die Notwendigkeit nach Vorschrift (3.144) $z_i = \kappa^*$ zu setzen, für kein i gegeben ist. Angenommen, es gäbe ein i mit $a_i \leq \kappa^* \leq b_i$, dann bedeutet das entweder $[L, R] = [a_i, b_i]$ oder $[L, R] \subset [a_i, b_i]$, denn L und R müssen Bruchstellen der Funktion ψ sein. In beiden Fällen folgt $\sum_{a_i \leq L, R \leq b_i} c_i \neq 0$. Wir berechnen also zunächst den Nenner der Vorschrift (3.151) und im Fall $\sum_{a_i \leq L, R \leq b_i} c_i = 0$ berechnen wir kein κ^*.

Unsere weitere Aufgabe besteht nun also darin, das minimale Intervall

$$[L, R] \quad (L, R \in \{-\infty, \infty, a_1, b_1, \ldots, a_{ws}, b_{ws}\})$$

zu finden, welches κ^* enthält, denn im Anschluss daran können wir κ^* bestimmen, den Vektor $\boldsymbol{z}^*$ ausrechnen und durch Rücktransformation gemäß (3.134) den gesuchten Lösungsvektor $\boldsymbol{x}^*$ generieren.

3.4.4 Minimales Lösungsintervall

Der PK-Algorithmus definiert eine Menge IP, in der zu Beginn alle Bruchstellen a_i, b_i $(i = 1, .., ws)$ sowie die Werte $-\infty$ und ∞ enthalten sind. Wir definieren L_{IP} und R_{IP} als

[23] Das wurde auch mit Luca Zanni abgestimmt.

das jeweils kleinste bzw. größte Element in IP. Das Intervall $[L_{IP}, R_{IP}]$ wird iterativ verkleinert. Die Menge $\{1, .., ws\}$ wird in jeder Iteration aufgeteilt in Indizes i mit $[L_{IP}, R_{IP}] \subseteq [a_i, b_i]$ und damit auch $[L, R] \subseteq [a_i, b_i]$ (Indexmenge $\mathcal{A}$), Indizes i für die (L_{IP}, R_{IP}) und $[a_i, b_i]$ disjunkt sind (Indexmenge $\mathcal{B}$) sowie Indizes i, für welche wir noch keine Aussage über z_i treffen können (Indexmenge $\mathcal{C}$), da entweder a_i oder b_i noch im Inneren des aktuellen Intervalls $[L_{IP}, R_{IP}]$ liegen, wir jedoch für das gesuchte Intervall $[L, R]$ ausgeschlossen hatten, dass es Bruchstellen enthält. Sobald die betreffenden Bruchstellen das Intervall (L_{IP}, R_{IP}) verlassen (Verkleinerung des Intervalls), kann jedes i dann einer der beiden Indexmengen $\mathcal{A}$ oder $\mathcal{B}$ zugeordnet werden. Abbildung 3.9 stellt die Situation dar. Im ersten Fall führt die Veränderung von L_{IP} dazu, dass b_i weiterhin im Intervall $[L_{IP}, R_{IP}]$ liegt und i die Indexmenge nicht verlässt. Im zweiten Fall erkennt man, dass $[a_i, b_i]$ nach der Änderung das Intervall $[L_{IP}, R_{IP}]$ beinhaltet. Damit bleibt es auch für alle weiteren Iterationen größer und i wechselt in die Indexmenge $\mathcal{A}$. Im dritten Fall entstehen durch die Änderung disjunkte Intervalle $[a_i, b_i]$ und (L_{IP}, R_{IP}), i wechselt in die Indexmenge $\mathcal{B}$.

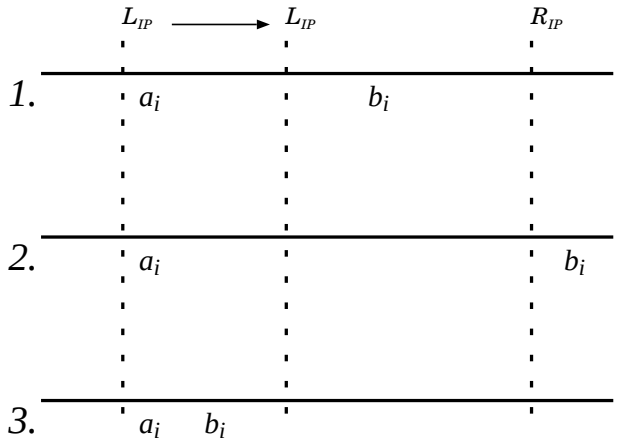

Abbildung 3.9: Beispiele zur Auswirkung einer Intervallverkleinerung.

Es stellt sich die Frage, wie die Verkleinerung von $[L_{IP}, R_{IP}]$ ablaufen soll. Es muss garantiert sein, dass der gesuchte Wert κ^* das Intervall nie verlässt. Wir legen dazu in jeder Iteration einen Wert $M \in (L_{IP}, R_{IP})$ fest, anhand dessen wir links oder rechts abschneiden. Wir berechnen dazu

$$\psi(M) = \sum_{i=1}^{ws} c_i z_i(M). \tag{3.152}$$

Sollte $\psi(M) \leq d$ gelten, bedeutet das, wir können das Intervall $[L_{IP}, M)$ außer Acht lassen und setzen $L_{IP} = M$; im Fall $\psi(M) \geq d$ gilt dann analog $R_{IP} = M$. Den Wert für M legen wir, wie es auch in [116] vorgeschlagen wird, als den Median von den in

IP enthaltenen Werten fest. Die Bestimmung des Medians ist nicht zeitaufwändig, da die Menge IP nur zu Beginn sortiert werden muss. Bei der Streichung von Elementen aus IP geht diese Ordnung nicht verloren und der Median ist jeweils das mittlere Element des verkleinerten Vektors.

Bemerkung 3.19. *Problematisch ist die Berechnung des Medians bei gerader Anzahl von Elementen. Angenommen, es gäbe nur noch zwei Bruchstellen innerhalb von $[L_{IP}, R_{IP}]$. Diese müssen das Intervall, und damit die Menge $\mathcal{C}$ noch verlassen. Der Median dieser beiden Werte ist ihr Mittelwert und liegt zwischen ihnen, sofern die Werte nicht gleich sind. Das bedeutet aber, der Median ist keine Bruchstelle der Funktion ψ. Das Szenario kann analog bei mehr als zwei Bruchstellen auftreten, sofern es sich um ein Vielfaches von zwei handelt, und führt im Extremfall zu einem falschen Intervall $[L, R]$. Wir umgehen das Problem, indem wir jeweils die größere der beiden Bruchstellen als neuen Median wählen. Dabei gibt es keinen Konflikt, denn dieses Vorgehen führt ausschließlich für den Fall, dass L_{IP} und R_{IP} benachbarte Bruchstellen sind, zum Stillstand. Dieser Fall entspricht jedoch der gesuchten Lösung, d.h. einem Intervall, welches κ^*, aber keine Bruchstellen im Inneren enthält. Weiterhin ist die Bestimmung des Medians kritisch, falls es eine große Anzahl von Werten in $[L_{IP}, R_{IP}]$ gibt, die einer der Grenzen entsprechen. Sobald solch ein Wert als Median gewählt wird, kommt es zum Stillstand des Verfahrens, denn die Menge IP verkleinert sich nicht und in allen folgenden Iterationen wird immer wieder der gleiche Median gewählt. Man könnte das vermeiden, indem der Median in solchen Fällen durch eine andere Bruchstelle ersetzt wird. Dieses Vorgehen ist jedoch nicht konsequent. Es könnte passieren, dass alle Werte im Inneren von $[L_{IP}, R_{IP}]$ einer dieser Grenzen entsprechen. Als Ausweg legen wir fest, dass das Intervall IP in jedem Schritt über $IP := (L_{IP}, R_{IP})$ definiert wird. Da dieses Intervall ausschließlich dazu verwendet wird, den Median zu finden, der dann zur Verkleinerung des Suchraumes benutzt wird, hat diese Änderung keinen Einfluss auf den Algorithmus und die Lösung.*

Die beschriebenen Schritte werden solange durchgeführt, bis keine Elemente mehr in $\mathcal{C}$ vorhanden sind und somit das gesuchte Intervall $[L, R]$ gefunden wurde. Es ist garantiert, dass κ^* in diesem Intervall liegt. Somit kann mittels Satz 3.6 und der Vorschrift (3.151) der vorläufige Lösungsvektor berechnet werden. Der von uns gesuchte Lösungsvektor $\boldsymbol{x}^*$ der Aufgabe (3.131) berechnet sich dann gemäß (3.134) über die Rücktransformation

$$x_i^* = \frac{2\hat{y}_i z_i^* - \hat{u}_i}{D_{ii}} \quad (i = 1, .., ws) \; . \tag{3.153}$$

Auch in diesem Abschnitt geben wir den Algorithmus in der Form an, wie wir ihn umgesetzt haben (Abbildung 3.10).

Eingabe: $\hat{\boldsymbol{u}},\ \hat{\boldsymbol{y}},\ e,\ C$

$i = 1, .., \text{ws}$

$$a_i = \begin{cases} \frac{\hat{u}_i}{2} & \text{falls } \hat{y}_i = 1 \\ \frac{C+\hat{u}_i}{-2} & \text{sonst} \end{cases} \qquad b_i = \begin{cases} \frac{C+\hat{u}_i}{2} & \text{falls } \hat{y}_i = 1 \\ \frac{\hat{u}_i}{-2} & \text{sonst} \end{cases}$$

$d = \frac{1}{2}\left(\sum_{i=1}^{\text{ws}} \hat{y}_i \hat{u}_i + e\right),\ L = -\infty,\ R = \infty,\ ts = 0,\ sw = 0$

$\mathcal{C} = \{1, .., \text{ws}\},\ IP = \{a_1, .., a_{\text{ws}}, b_1, .., b_{\text{ws}}, -\infty, \infty\}$

$\mathcal{C} \neq \emptyset$

$$M = \text{ median}(IP),\ test = ts + \sum_{\substack{i \in \mathcal{C} \\ b_i < M}} b_i + \sum_{\substack{i \in \mathcal{C} \\ a_i > M}} a_i + M \cdot sw + M \sum_{\substack{i \in \mathcal{C} \\ a_i \leq M \leq b_i}} 1$$

$test < d$ — j / n

L=M	R=M

$$IP = \{t \in IP : L < t < R\},\ ts = ts + \sum_{\substack{i \in \mathcal{C} \\ b_i \leq L}} b_i + \sum_{\substack{i \in \mathcal{C} \\ R \leq a_i}} a_i,\ sw = sw + \sum_{\substack{i \in \mathcal{C} \\ a_i \leq L, R \leq b_i}} 1$$

$\mathcal{C} = \{i \in \mathcal{C} : L < a_i < R \text{ oder } L < b_i < R\}$

$sw = 0$ — j / n

	$\kappa^* = \frac{d-ts}{sw}$

$i = 1, .., \text{ws}$

$b_i \leq L$ — j / n

$z_i = b_i$	$a_i \geq R$ — j / n	
	$z_i = a_i$	$z_i = \kappa^*$

$x_i = 2 z_i \hat{y}_i - \hat{u}_i$

Abbildung 3.10: Algorithmus von Pardalos und Kovoor.

3.4.5 Weitere Ansätze

Als innere Löser für die Aufgabe (3.131) kommen neben [116] auch andere Algorithmen in Frage. Weitere, in [136] vorgeschlagene, Methoden sind der Algorithmus von Brucker aus [15] sowie die von Nielsen und Zenios in [109] vorgestellten parallelisierbaren Algorithmen. Der Algorithmus von Pardalos und Kovoor ist dem Algorithmus von Brucker überlegen, da er eine Laufzeitverbesserung mit sich bringt. Der Algorithmus ist schon im seriellen Fall so schnell, dass hierbei der Aspekt der Parallelisierung in den Hintergrund tritt. Die Tests in Kapitel 7 werden die Laufzeitverhältnisse beleuchten. Dennoch wurde auch dieser Algorithmus parallelisiert, sodass das gesamte SVM-Training mit einem hohen Anteil paralleler Routinen ausgestattet ist. Wir verweisen auf Kapitel 6.

3.5 Zusammenfassung

In diesem Kapitel haben wir einen seriellen Algorithmus für das Training von Support-Vektor-Maschinen vorgestellt. Es basiert auf der bekannten Idee der Zerlegung und beinhaltet ein Projektionsverfahren zur Lösung der Subprobleme. Das Projektionsverfahren wiederum fußt auf einem schnellen inneren Löser für Diagonalmatrizen, wie sie im Projektionsverfahren iterativ entstehen. Dieser Algorithmus wird in den folgenden Kapiteln weiterentwickelt. In Kapitel 4 zeigen wir, welche Möglichkeiten es gibt, den Zerlegungsalgorithmus so zu erweitern, dass er mehrere SVM-Modelle umsetzt. Zusätzlich stellen wir die Möglichkeit der Klassengewichtung und andere kostensensitive Manipulationsmethoden vor. In Kapitel 6 beschreiben wir unseren Ansatz zur Parallelisierung des Verfahrens. Es sei aber hier schon erwähnt, dass die Parallelisierung einer einzelnen SVM-Trainingsphase auf der Ebene der Zerlegung, als auch auf der Ebene des Lösers für die Teilprobleme stattfinden wird. Der iterative Aufbau des Zerlegungsalgorithmus soll erhalten bleiben. Wir werden auf Vor- und Nachteile dieser Parallelisierung eingehen und analysieren, auf welcher Ebene andere Arbeiten den Parallelisierungsaspekt betrachten.

Kapitel 4

Kostensensitive Erweiterungen für Support-Vektor-Maschinen

In diesem Kapitel beschreiben wir Modifikationen und Erweiterungen der konventionellen Support-Vektor-Maschine und des Zerlegungsalgorithmus. Das Ziel dabei ist, die SVM-Software flexibel zu gestalten, um anspruchsvolle Datensätze nicht nur schnell, sondern vor allem auch genau zu klassifizieren. Dazu werden im Folgenden interessante Modellerweiterungen vorgeschlagen und diskutiert. Neben der Einbettung von drei SVM-Algorithmen zu einem Gesamtverfahren stellen wir verschiedene Techniken des kostensensitiven Lernens vor. Diese sind immer dann interessant, wenn eine einzelne Klasse von stärkerem Interesse ist, als die andere. Das kann zum Einen an einer unausgeglichenen Datenverteilung zwischen den Klassen, aber zum Anderen auch an den Eigenschaften der zugrunde liegenden Anwendung liegen. Zusätzlich erweitern wir die Standard-Kernfunktion, um auch im Innersten der SVM flexibel sein zu können. Die Modifikationen betreffen drei Ebenen:

- Manipulation der Trainingsdaten,
- Manipulation des SVM-Trainings und seiner Komponenten,
- Manipulation der finalen Klassifikationsfunktion.

Bevor wir uns in Kapitel 6 dem Problem der Rechenzeit und der Parallelisierung zuwenden, spielen Effizienzaspekte in diesem Kapitel noch keine Rolle. Vielmehr wird hier zunächst komplizierterer Code beschrieben, der es jedoch ermöglicht, auf spezielle Wünsche hinsichtlich der Modelle über einfache Programmoptionen einzugehen.

4.1 Unausgeglichene und kostensensitive Datensätze

Der Begriff der Unausgeglichenheit taucht sehr häufig bei der Analyse realer Daten auf [21]. Im Zusammenhang mit Klassifikationsverfahren ist dieser Begriff relativ klar eingegrenzt.

Wie man anhand von Abbildung 4.1 erkennen kann, adressieren wir Datensätze, bei denen

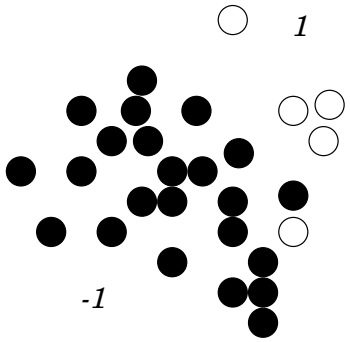

Abbildung 4.1: Beispiel für einen unausgeglichenen Datensatz.

die Anzahl der Punkte mit dem Klassenlabel 1 signifikant kleiner ist, als die Anzahl der Daten mit Label -1.[24] Es gibt keine Regel, ab welchem Verhältnis eine Unausgeglichenheit vorliegt, Erfahrungen zeigen jedoch, dass schon ab etwa $1:2$ im Sinne der Sensitivität schlechte Klassifikatoren erlernt werden und Möglichkeiten zur Verbesserung der Modelle gesucht werden sollten [5]. Derartige Datensätze treten bei realen Anwendungen viel häufiger auf, als man allgemein annimmt [21, 75, 99, 120, 153, 157]. Zusätzlich zu unausgeglichenen Problemen gibt es viele Anwendungen, die sehr sensitiv sind. Das bedeutet, dass eine Klasse von großem Interesse ist und somit besonders gut erkannt werden muss. Wir sprechen dabei von Kostensensitivität. Typischerweise treten diese beiden Effekte gekoppelt auf. Wir verwenden daher die Begriffe Unausgeglichenheit und Kostensensitivität oft im gleichen Kontext. Im Folgenden verstehen wir unter einem unausgeglichenen bzw. kostensensitiven Klassifikationsproblem eines, bei dem mindestens eines der folgenden Kriterien zutrifft.

- Die positive Klasse ist von größerem Interesse.
- Die positive Klasse ist der Anzahl nach unterbesetzt.
- Die positive Klasse ist sehr schwer zu erkennen.
- Fehler in der positiven Klasse sind mit deutlich höheren Kosten verbunden und sollen vermieden werden.

In der Literatur werden die Begriffe auch sehr oft synonym verwendet. Mischformen der Probleme, z.B. Daten die sensitiv sind und gleichzeitig die positive Klasse dominiert, gibt es auch, diese sind aber seltener.

Beispiel 4.1. *Viele Krankheiten und Mangelerscheinungen lassen sich anhand der Analyse von Blutproben erkennen. Für die aus dem Blut gewonnenen Merkmale lassen sich Klassifikatoren für bestimmte Fragestellungen erlernen. Krankheiten, die sehr selten auftreten*

[24] oder andersherum

($< 1\%$)*, führen zu unausgeglichenen Daten, falls die Stichprobe in etwa der allgemeinen Bevölkerungsstruktur entspricht. Entnimmt man ausschließlich Patienten mit Verdacht auf diese Krankheit Blut, kann die entstehende Stichprobe auch in die andere Richtung unausgeglichen (überrepräsentiert) sein.*

Das kritische an einem Klassifikator, der aus sehr unausgeglichenen Daten entstanden ist, kann die völlige Unfähigkeit, positive Punkte zu erkennen, sein. Das passiert immer dann, wenn während der Parameteroptimierung einfache Kriterien, wie die Gesamtfehleranzahl angesetzt werden. Dann wählen die meisten Algorithmen einen Klassifikator, der immer die negative Klasse vorhersagt. Durch die Unausgeglichenheit fallen die wenigen falsch negativen Testpunkte während der Validierung nicht stark ins Gewicht und der Eindruck eines guten Ergebnisses entsteht. Weitere Details dazu folgen in Kapitel 5. Es sei bemerkt, dass klassische Algorithmen des Data-Mining prinzipiell dazu tendieren, die größere Klasse besser zu erkennen, als die kleine [3, 120]. Durch das Ziel, Überanpassung zu vermeiden, setzt sich diese Eigenschaft auch bei modernen Methoden weiter fort. Schwierig ist es auch, einen Klassifikator zu beurteilen, der auf einem „guten“ (einfachen) Datensatz trainiert und später auf realen (schwierigen) Daten eingesetzt werden soll.

Unausgeglichenheit von Daten führt nicht immer zu einem schlechten SVM-Modell. Die Hyperebene wird ohnehin nur von wenigen Punkten bestimmt (vgl. Seite 23), sodass die Menge der Support-Vektoren oft sogar sehr ausgeglichen ist. Die Problematik betrifft vielmehr den subjektiven Wunsch nach der Erkennung der Klasse 1. Wenn wir es mit derartigen Daten zu tun haben, ist in den meisten Fällen nur die Klasse 1 interessant und soll erkannt werden. Die Sensitivität spielt daher eine viel größere Rolle, als die Genauigkeit allgemein. Für das Beispiel der Erkennung von Krankheiten bedeutet das: es ist nicht schlimm, wenn für eine Blutprobe fälschlicherweise die Klasse 1 (krank) festgestellt wird, denn in diesem Fall werden weitere Untersuchungen folgen. Es kommt jedoch einer Katastrophe gleich, wenn ein Patient nach Hause geschickt wird, obwohl er eine anderweitig nicht gleich zu erkennende Krankheit hat. Diese Problematik kann man sich anhand des Beispiels HIV-Test klar machen [123]. Die Tatsache, dass unausgeglichene Daten und die dabei auftretenden Probleme bei der Klassifikation innerhalb der Theorie des SVM's selten betrachtet werden, bedeutet nicht, dass die einfachen SVM-Algorithmen diese Fälle optimal behandeln.[25] Vielmehr ist das darauf zurückzuführen, dass die häufig verwendeten Standarddatensätze relativ ausgeglichen sind [63] und dass im Allgemeinen die vorhandenen Verfahren einfach kopiert bzw. benutzt werden. Die Unterscheidung der beiden Fehlerarten bereitet zusätzliche Mühe, die oft gescheut wird, bzw. bei vorhandener Software nicht möglich ist.

[25] persönliche Kommunikation (*Machine Learning Summer School, 12.-25. September 2004, Berder Island, Frankreich*)

4.2 Manipulation des SVM-Trainings und seiner Komponenten

Wir stellen drei Methoden zur flexiblen SVM-Klassifikation dar. Die Ansätze adressieren die SVM-Modelle, die Wahl des Strafparameters für die Modelle sowie die Wahl der Kernfunktion.

4.2.1 Einbettung anderer SVM-Modelle

In Kapitel 3 haben wir uns mit der Implementierung einer Support-Vektor-Maschine nach dem L_1-Norm-Modell beschäftigt. Üblicherweise wird das L_1-Norm-Modell implementiert [18, 19, 74, 118, 130, 138], da die am häufigsten verwendeten SVM-Trainingsmethoden - die Working-Set-Algorithmen - immer anhand dieses Modells vorgestellt werden. Der Grund für die bevorzugte einfache Aufsummierung der Schlupfvariablen könnte auch in der Tatsache liegen, dass das L_2-Norm-Modell zu viele Support-Vektoren produziert.[26] Wie schon im Abschnitt 2.4.2 gezeigt wurde, unterscheiden sich die weichen Modelle untereinander durch unterschiedliche Behandlung von Verletzungen der Marge während des Trainings. In beiden Modellen werden diese Fehler toleriert und über eine Kostenfunktion im Optimierungsproblem abgerechnet. Dazu werden die Werte der sogenannten Schlupfvariablen, die positiven Differenzen zwischen den vorgegebenen und tatsächlichen Zielfunktionswerten, entweder einfach (L_1) oder quadratisch (L_2) aufsummiert. Der Effekt der quadratischen Aufsummierung von Schlupfvariablen ist in Abbildung 4.2 dargestellt. Es

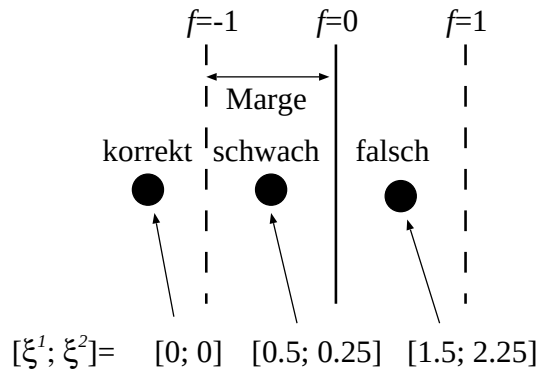

Abbildung 4.2: Einfluß quadratischer Aufsummierung.

handelt sich dort um einen Punkt der Klasse -1 und wir haben drei mögliche Klassifikationsergebnisse dargestellt. Der erste Wert in jeder Klammer gibt den Wert der Schlupfvariable für das L_1-Norm- und der zweite für das L_2-Norm-Modell an. Man erkennt, dass

[26] "It has the disadvantage of producing too many support vectors." S. Sathiya Keerthi, persönliche Kommunikation, 2003

korrekte Klassifikationen, die nur an der Marge scheitern, vom L_2-Norm-Modell weniger bestraft, wohingegen wahre Klassifikationsfehler durch die Quadrierung eines Wertes, der mindestens 1 beträgt, stärker geahndet werden. Wir werden uns mit der Frage, welches der beiden Modelle zu bevorzugen ist, in Kapitel 7 auseinandersetzen. In diesem Abschnitt soll zunächst gezeigt werden, dass ausgehend von unserer L_1-Norm-Implementierung andere SVM-Modelle durch einfache Modifikationen problemlos verwendet werden können.

4.2.1.1 L_2-Norm

Für den Fall der quadratischen Fehlersummierung (L_2-Norm) hat die Zielfunktion der dualen Optimierungsaufgabe bis auf einen modifizierten Kern

$$\tilde{k}(\boldsymbol{x}^i, \boldsymbol{x}^j) := k(\boldsymbol{x}^i, \boldsymbol{x}^j) + \frac{1}{2C}\delta_{i,j} \tag{4.1}$$

die Form (2.32). Bei den Nebenbedingungen entfällt die obere Schranke, d.h. $\boldsymbol{\alpha} \leq \boldsymbol{C}$ ist nicht mehr zwingend nötig.

```
function new_kernel(i,j)
    new_kernel = kernel(i,j)
    if (i==j) and (model==2) then
        new_kernel=new_kernel+1/2C
    end if
end function new_kernel
```

Abbildung 4.3: Anpassung des Kerns an das L_2-Norm-Modell.

Der Parameter C schlägt sich bei der dualen Formulierung der L_2-Aufgabe also ausschließlich in der Berechnung des Kerns, und auch dort nur für Diagonalelemente der Grammatrix nieder. Deshalb kann man ein vorhandenes Programm durch eine Modifikation in der Funktion zur Kernauswertung so verändern, dass es das L_2-Norm-Modell implementiert. Der Pseudocode dazu könnte dann so aussehen, wie in Abbildung 4.3 dargestellt. Neben der Modifikation des Kerns müssen die neuen Nebenbedingungen beachtet werden. In einer L_1-Implementierung sind obere Schranken für $\boldsymbol{\alpha}$ bedacht. Diese werden inaktiviert über die Wahl von $C = \infty$. Dabei ist zu beachten, dass für den Kern ein neuer Parameter $\tilde{C} = C$ benutzt werden muss, damit nicht mit ∞ im Kern gerechnet wird. Durch Nachrechnen kann man sehen, dass sich die KKT-Bedingungen nicht ändern, d.h. bei $C = \infty$ kann das im Abschnitt 3.2.3 beschriebene Verfahren zur Aktualisierung der Arbeitsmenge weiter verwendet werden.

4.2.1.2 Harte Trennung

Das Erlernen einer hart trennenden Hyperebene ist für reale Datensätze zwar selten möglich, kann jedoch im Einzelfall zur perfekten Trennung von Daten herangezogen werden. Das Modell ist sehr einfach einzubinden. Um von (2.29) zu (2.19) zu wechseln, muss lediglich $C = \infty$ gesetzt werden. Da das Modell der harten Trennung keinen C-Parameter kennt, muss die Zielfunktion nicht modifiziert werden. Es sei erwähnt, dass eine scharfe Trennung im Allgemeinen nicht erwünscht ist, auch wenn sie über eine günstige Parameteranpassung möglich wäre. Der Grund dafür ist die Gefahr der Überanpassung der Support-Vektor-Maschine an die Trainingsdaten. Wir verweisen dazu auf Definition 2.19.

Nachdem wir gezeigt haben, dass wir eine flexible SVM-Implementierung anbieten, die über wenige Parameter so gesteuert werden kann, dass alle 3 bekannten SVM-Modelle getestet werden können, sollen in den folgenden Abschnitten spezielle Methoden zur kostensensitiven SVM-Klassifikation vorgestellt werden. Das Problem unausgeglichener und kostensensitiver Datensätze wurde im Abschnitt 4.1 bereits erläutert. Die Techniken zum Erlernen sensitiver und gleichzeitig genauer Modelle werden in Kapitel 7 getestet.

4.2.2 Modifizierte Fehlergewichtung

Generalisierungseigenschaften eines SVM-Modells werden neben der Wahl einer Kernfunktion mit entsprechendem Kernparameter auch stark durch die Wahl des Parameters C beeinflusst. Im Fall von $C = \infty$ handelt es sich um das Modell der harten Trennung, bei dem keine Fehler während des Trainings zugelassen werden. Das führt bei schlecht trennbaren Daten zu einer zu stark angepassten Hyperebene oder gar zu einem Scheitern des Trainings allgemein. Der Parameter C sollte deshalb bei realen Datensätzen einen endlichen Wert haben. Mit dessen Hilfe gelangt man dann zu den schon vorgestellten Modellen weicher Trennung. Je kleiner der Wert des Parameters wird, umso mehr Fehler darf der Algorithmus tolerieren, um die Marge so groß wie möglich zu machen. Das führt zu einer guten Hyperebene, solange C nicht zu klein gewählt wird. Der Parameter C ist dafür verantwortlich, die gegensätzlichen, oder besser gesagt in Konkurrenz stehenden, Ziele der Fehlerminimierung und Abstandsmaximierung in Ausgleich zu bringen. Diese einfach gewichteten Zielsetzungen sind immer dann sinnvoll, wenn die Anzahl von Punkten in beiden Klassen ausgeglichen ist, Fehler gleich teuer sind und die Trainingsdaten die Klasseneigenschaften gut repräsentieren.

Das allgemeine SVM-Modell unterscheidet grundsätzlich keine Fehlerarten. Klassifikationsfehler in den Trainingsdaten werden einfach oder quadratisch aufsummiert und mit dem Parameter C gewichtet. Damit gehen die Klasseninformationen komplett verloren. Ein Ausweg aus dieser Situation wird beispielweise in [106] vorgestellt und ist auch in [101] und [108] wiederzufinden. Die Idee ist, Fehler erster Art härter zu bestrafen, als Fehler

zweiter Art.[27] Das kann sehr einfach realisiert werden, indem der Parameter C auf zwei Parameter C^+ und C^- gesplittet wird. Bei einem Fehler erster Art wird mit C^+ gewichtet und bei einem Fehler zweiter Art mit C^-. Die Feststellung der Fehlerart ist trivial, da der Vektor $\boldsymbol{y}$ die Klassen jederzeit bereit hält. Die Wahl dieser Parameter kann willkürlich oder nach einem festen Schema erfolgen. Bekannt ist die Methode [101]

$$C^+ = \Lambda \frac{rd^+ + rd^-}{rd^+} \qquad C^- = \Lambda \frac{rd^+ + rd^-}{rd^-} \tag{4.2}$$

mit rd^+ bzw. rd^- als der Anzahl der Trainingspunkte in Klasse 1 und -1. Der Parameter $\Lambda > 0$ kann frei gewählt werden. Je höher dieser ist, umso stärker werden Fehler allgemein gewichtet, wobei das Verhältnis $C^+ : C^-$ konstant bleibt und durch die Mächtigkeiten der einzelnen Klassen gegeben ist. Im Gegensatz dazu kann man die beiden Parameter auch als unabhängig einstufen. Wir werden in Kapitel 7 zeigen, dass eine Optimierung der beiden Parameter nicht immer auf das konstante Verhältnis führt und dass zudem das bei der Optimierung verwendete Gütemaß und die darin enthaltenen Prioritäten darüber entscheiden, wie das tatsächliche Verhältnis zwischen den Parameterwerten ist. Wir verweisen an dieser Stelle auch auf Kapitel 5, in welchem wir uns mit Möglichkeiten der Parameteroptimierung auseinandersetzen. Die Wahl (4.2) ist immer dann geeignet, wenn aus Zeitgründen nur ein Parameter optimiert werden kann. Interessante Details zu dieser Problematik sind in [108] zu finden. In dieser Arbeit wird untersucht, wie sich das gesamte SVM-Optimierungsproblem ändert, wenn eine Minimierung des sogenannten F-Maßes schon in einem einzelnen Training angestrebt wird. Hintergrund der Überlegungen war dort ebenfalls der Aspekt unausgeglichener Datensätze. Es wird gezeigt, dass die Maximierung dieses Maßes für die Trainingsdaten auch über eine geeignete Wahl von C^+ und C^- im Standard-Modell erreicht werden kann. Auf das F-Maß und seine exakte Definition werden wir in Kapitel 5 noch genauer eingehen. Die Nutzung von C^+ und C^- führt im erfolgreichen Fall dazu, dass die optimale Hyperebene etwas näher an der Klasse -1 liegt (Abbildung 4.4). Anders ausgedrückt kann man sagen, dass sich die Gesamtmarge nicht mehr als Summe der beiden gleichen Einzelmargen mg, sondern als $mg_1 + mg_{-1}$ berechnet, wobei $mg_1 > mg_{-1}$ gelten wird, falls $C^+ > C^-$. Der mit x gekennzeichnete Punkt in Abbildung 4.4 ist besonders interessant. Man sollte sich bewußt sein, dass bei sehr wenigen Punkten einer Klasse, die Gefahr der Ausreißer, beispielsweise durch falsche Angabe des Labels, viel höher ist, als bei ausgeglichenen Daten, insbesondere wenn man mit einem großen Wert für C^+ arbeitet. Dieser führt dazu, dass um den Punkt x herum die Klasse 1 definiert wird. Gehört dieser jedoch zur Klasse -1 oder als positiver Punkt in die Region der anderen Punkte der Klasse 1, ist die ursprüngliche Hyperebene die bessere. Die Konsequenzen daraus sind:

[27] Als Quelle für diese Idee wird [78] genannt. In dieser Arbeit wird eine gewichtete Klassifikation mittels Boosting [131] betrachtet.

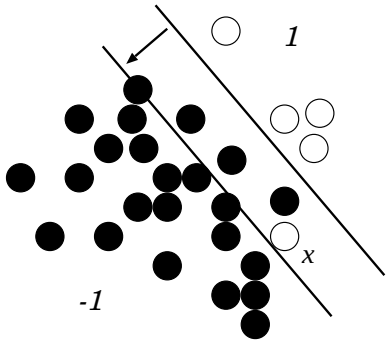

Abbildung 4.4: Verschiebung der Hyperebene durch C^-.

- Elemente der Klasse 1 werden verstärkt korrekt klassifiziert, auch wenn sie im Bereich der ursprünglichen Trennlinie liegen. Die Fehler erster Art nehmen ab.
- Elemente der Klasse -1 werden zunehmend falsch klassifiziert, wenn sie im Bereich der ursprünglichen Trennlinie liegen. Die Fehler zweiter Art nehmen zu.

Diesen Effekt kann man sich anhand der Abbildung 4.5 verdeutlichen, wobei sich die Trennfunktion von rechts in Richtung negativer Klasse nach links verschoben hat.

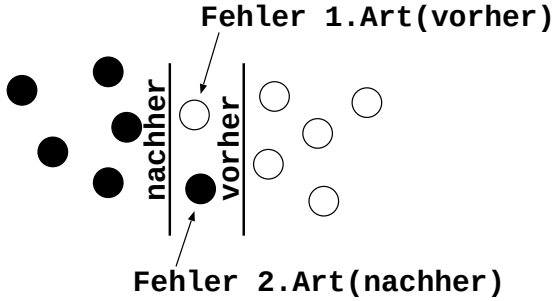

Abbildung 4.5: Fehler erster und zweiter Art in Konkurrenz.

Fazit: die Erhöhung der Sensitivität mittels gewichteter Fehlermaße entspricht nicht einem Free Lunch[28], ist aber für viele praktische Fragestellungen von großer Wichtigkeit. Leider werden Performance-Vergleiche zwischen gewichteten und ungewichteten Support-Vektor-Maschinen selten veröffentlicht, sodass der Einfluss von Gewichten auf die Zielfunktion und Konvergenz von Lernalgorithmen nicht klar ist. Insbesondere sollte untersucht werden, welchen Preis man für eine bessere Erkennung der Klasse 1 zu zahlen hat. Als Kriterien kommen Trainingszeit, Fehler in Klasse -1 sowie Robustheit bei geringen Änderungen der Parameter und Daten in Frage. In Kapitel 7 werden wir Tests zur modifizierten Fehlergewichtung präsentieren.

[28]Gewinn ohne Einsatz von Mitteln oder Erleidung von Verlusten

```
function new_kernel(i,j)
    new_kernel = kernel(i,j)
    if (i==j) and (model==2) then
        if (y_i==1)  new_kernel=new_kernel+1/2C+
        if (y_i==-1) new_kernel=new_kernel+1/2C-
    end if
end function new_kernel
```

Abbildung 4.6: Anpassung eines Kerns für das L_2-Norm-Modell mit Fehlergewichtung.

Implementierung

Da sich die modifizierte Fehlergewichtung nur auf den Strafparameter C auswirkt, kann diese in die SVM-Software eingebaut werden. Dazu sollte man die sich ergebenden Änderungen an den Modellen der weichen Trennung und die daraus resultierenden Folgen für die primalen und dualen Otimierungsaufgaben untersuchen. Das Modell der harten Trennung bleibt unverändert, da es den Parameter C nicht nutzt. Die Zielfunktion der primalen Optimierungsaufgabe einer L_1-Norm- bzw. L_2-Norm-Support-Vektor-Maschine ($k = 1$ bzw. $k = 2$) ändert sich wie folgt:

$$\min_{\substack{\boldsymbol{w}\in\mathcal{M}\\ b\in\mathbb{R}\\ \boldsymbol{\xi}\in\mathbb{R}^{rd}}} \quad \frac{1}{2}\|\boldsymbol{w}\|_{\mathcal{M}}^2 + C^+ \sum_{i:y_i=1} \xi_i^k + C^- \sum_{j:y_j=-1} \xi_j^k \,. \tag{4.3}$$

Die Nebenbedingungen bleiben identisch. Bei der Summierung der Schlupfvariablen wird also unterschieden, ob ein negativer (erste Summe) oder positiver Punkt (zweite Summe) betroffen ist.

Für den Fall $k = 2$ ist die Berücksichtigung eines zweiten Fehlergewichtes sehr einfach. In der ursprünglichen dualen Optimierungsaufgabe (2.37) wird der Parameter C ausschließlich für den modifizierten Kern $\tilde{k}$, siehe (4.1), verwendet. Wir implementieren daher den Kern nach dem bekannten Schema, vergleiche Abbildung 4.6. Die neue Zielfunktion der dualen Aufgabe hat dann wegen (2.23), (4.1) und den hier betrachteten Gewichten die Form

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^{rd}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{rd} \alpha_i - \frac{1}{2}\sum_{i=1}^{rd}\sum_{j=1}^{rd} y_i y_j \alpha_i \alpha_j \left[k(\boldsymbol{x}^i, \boldsymbol{x}^j) + \delta_{i,j}\left(\frac{\delta_{y_i,1}}{2C^+} + \frac{\delta_{y_i,-1}}{2C^-}\right)\right] \tag{4.4}$$

unter den Nebenbedingungen

$$\boldsymbol{\alpha}^T \boldsymbol{y} = 0 \qquad \text{und} \qquad \boldsymbol{\alpha} \geq \mathbf{0}\,.$$

Für den Fall $k = 1$ muss die Berücksichtigung eines zweiten Fehlergewichtes an mehreren Stellen des Algorithmus vorgenommen werden, denn die duale Aufgabe hat durch C beeinflusste Nebenbedingungen, siehe (2.32). Diese Nebenbedingungen werden zu

$$\boldsymbol{\alpha}^T \boldsymbol{y} = 0 \qquad \text{und} \qquad \forall i: \ 0 \leq \alpha_i \leq \begin{cases} C^+ & \text{falls } y_i = 1 \\ C^- & \text{sonst} \end{cases} .$$

Die Modifikation ist also stark vom verwendeten Algorithmus abhängig.

Wir beschreiben nun kurz, wie wir die neuen Gewichte C^+ und C^- in unsere Implementierung eingebettet haben. Wie schon erwähnt, werden das Modell harter Trennung und auch das L_2-Norm-Modell daraus über $C = \infty$ bzw. einen zusätzlich angepassten Kern erzeugt. Das gleiche trifft dann natürlich auch auf C^+ und C^- zu. Es gibt mehrere Stellen, an denen wir den Parameter C benutzen. In der äußeren Routine wird er für die Methode von Zoutendijk gebraucht, siehe dazu Abbildung 3.2, und weiterhin für das Auffüllen der Arbeitsmenge nach dem Schema aus [139], welches wir auf Seite 52 vorgestellt haben. Die erste innere Routine, die VVP-Methode, benutzt den Parameter C nicht direkt. In Abbildung 3.6 sieht man, dass für die jeweilige Definition des zu lösenden Minimierungsproblems die zulässige Menge Ω anzugeben ist. Diese Menge wird definiert über die Nebenbedingungen, welche C als obere Schranke für $\boldsymbol{x}$ enthalten. Weitere Verwendung findet C in dieser Routine nicht. Die Nebenbedingungen haben Einfluß auf den Algorithmus von Pardalos und Kovoor, die zweite innere Routine. Dieser ist jedoch beschränkt auf die Initialisierung der Vektoren $\boldsymbol{a}$ und $\boldsymbol{b}$, siehe z.B. in Abbildung 3.10. Alles in allem ist die Zahl der zu ändernden Stellen überschaubar.

Die neuen Abfragen im Verfahren von Zoutendijk lauten dann:

$$\texttt{if } \left\{ y_i = -1 \texttt{ and } \alpha_i^k < C^- \right\} \texttt{ or } \left\{ y_i = 1 \texttt{ and } \alpha_i^k > 0 \right\}$$

und

$$\texttt{if } \left\{ y_i = 1 \texttt{ and } \alpha_i^k < C^+ \right\} \texttt{ or } \left\{ y_i = -1 \texttt{ and } \alpha_i^k > 0 \right\}.$$

Das Auffüllen der Arbeitsmenge bei nicht ausreichender Anzahl von Pärchen, welche die KKT-Bedingungen verletzen, erfolgt nach dem Schema aus Abschnitt 3.2.3.4 mit den Änderungen:

- in Schritt 2:
$$0 < \alpha_i < \begin{cases} C^+ & \texttt{if } \ y_i = 1 \\ C^- & \texttt{else} \end{cases} ,$$
- in Schritt 4:
$$\alpha_i = \begin{cases} C^+ & \texttt{if } \ y_i = 1 \\ C^- & \texttt{else} \end{cases} .$$

Die VVP-Methode wird nicht geändert, da der C-Parameter dort nicht explizit benötigt wird. Im PK-Algorithmus passen wir die Initialisierungen folgendermaßen an:

$$a_i = \begin{cases} \frac{\hat{u}_i}{2} & \texttt{if } \hat{y}_i = 1 \\ \frac{C^- + \hat{u}_i}{-2} & \texttt{else} \end{cases} \quad \text{und} \quad b_i = \begin{cases} \frac{C^+ + \hat{u}_i}{2} & \texttt{if } \hat{y}_i = 1 \\ \frac{\hat{u}_i}{-2} & \texttt{else} \end{cases} .$$

Diese lassen sich sehr einfach einbauen, da durch die Unterscheidung der y_i-Werte jeweils schon das passende C eingesetzt werden kann, ohne eine Abfrage einbauen zu müssen.

Zur Umsetzung einer gewichteten Support-Vektor-Maschine muss neben den Änderungen am Optimierungsverfahren auch eine angepasste Berechnung des Schwellwertes b implementiert werden. Wir berechnen diesen im Standardmodell über den robusten Durchschnitt aller unbeschränketen Support-Vektoren als

$$b_r^* = \frac{\sum_{i \in \mathcal{FSV}} \left(y_i - \sum_{j \in \mathcal{SV}} y_j \alpha_j^* k(\boldsymbol{x}^i, \boldsymbol{x}^j) \right)}{|i \in \mathcal{FSV}|} , \tag{4.5}$$

siehe dazu Abschnitt 2.5. An dieser Formel ändert sich nichts, jedoch muss die Menge FSV neu definiert werden. Die Fehlergewichtung wird wie üblich über eine Fallunterscheidung für C realisiert, d.h.

$$i \in \mathcal{FSV} \Longleftrightarrow 0 < \alpha_i < C = \begin{cases} C^+ & \texttt{if } y_i = 1 \\ C^- & \texttt{else} \end{cases} .$$

Für die Nutzung des L_2-Norm-Modells bauen wir einen modifizierten Kern mit den Parametern $\tilde{C}^+ = C^+$ und $\tilde{C}^- = C^-$ ein und setzen dann die Originalparameter C^+ und C^- auf ∞. Bei der Berechnung von b in der Formel (2.46) wird dann ebenfalls auf $\tilde{C}^+$ und $\tilde{C}^-$ zurückgegriffen. Die Wahl von Modell und Gewichtung wird über Optionen gesteuert. Alles in allem ist die Erweiterung der Decomposition-Methode auf unterschiedliche Fehlergewichte und Modelle nicht schwierig. Die Anzahl der zu optimierenden Parameter einer Support-Vektor-Maschine erhöht sich dadurch jedoch. Das sind beispielsweise

- 3 bei Nutzung des einfachen Gauß-Kerns und
- 4 bei Nutzung des einfachen Polynomialkerns.

Die Schwierigkeit wird also vielmehr darin liegen, die flexiblen Möglichkeiten überhaupt zu nutzen. Wir werden in Kapitel 7 zeigen, welche Verbesserungen uns die Nutzung der Fehlergewichtung bringt.

4.2.3 Kernmodifikationen

Eine weitere Flexibilitätskomponente innerhalb unserer SVM-Software stellen die Kerne zur Verfügung. Die Funktion und die Eigenschaften von Kernen sind im Abschnitt 2.3

diskutiert worden. Dort haben wir auch einige spezielle Kerne vorgestellt. In diesem Abschnitt soll es nun darum gehen, die Standardform der Kerne zu verändern. Die Motivation ist auch hier wieder die Möglichkeit der Verbesserung von Klassifikationsergebnissen. Im Folgenden stellen wir zwei Methoden vor, um Kerne flexibel zu gestalten. Da diese Methoden bisher kaum eingesetzt werden, versuchen wir in Kapitel 7 konkrete Anwendungsmöglichkeiten aufzuzeigen.

4.2.3.1 Multiparameter-Kerne

Die meisten Kerne verfügen über einen Parameter, welcher das SVM-Modell steuert. Bei einigen Kernen ist es möglich, diesen Parameter den einzelnen Dimensionen des Datensatzes zuzuordnen, sodass jede Variable ein eigenes Gewicht bei der Ähnlichkeitsmessung bekommt [42]. Die Kerne unserer Arbeit, die dafür in Frage kommen, sind der Gauß-, der Slater- und der Polynomialkern. Die verallgemeinerten Formen werden im Folgenden definiert.

Definition 4.1 (verallgemeinerter Gauß-Kern)**.** *Über ein einfaches Umschreiben des Gauß-Kerns mittels*

$$\begin{aligned} k^G(\boldsymbol{x}^i, \boldsymbol{x}^j) &= \exp\left(-\frac{\|\boldsymbol{x}^i - \boldsymbol{x}^j\|^2}{2\sigma^2}\right) \\ &= \exp\left(-\sum_{k=1}^{n} \frac{\left(x_k^i - x_k^j\right)^2}{2\sigma^2}\right) \end{aligned}$$

erkennt man, dass σ in jeder Dimension benötigt wird. Der neue verallgemeinerte Kern hat die Form

$$k_*^G(\boldsymbol{x}^i, \boldsymbol{x}^j) := \exp\left(-\sum_{k=1}^{n} \frac{(x_k^i - x_k^j)^2}{2\sigma_k^2}\right) \qquad (\boldsymbol{\sigma} \geq \mathbf{0})\,. \tag{4.6}$$

Definition 4.2 (verallgemeinerter Slater-Kern)**.** *Auch der Slater-Kern kann umgeformt werden*

$$\begin{aligned} k^S\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) &= \exp\left(-\frac{\|\boldsymbol{x}^i - \boldsymbol{x}^j\|}{2\sigma^2}\right) \\ &= \exp\left(-\sqrt{\sum_{k=1}^{n} \frac{\left(x_k^i - x_k^j\right)^2}{4\sigma^4}}\right) \end{aligned}$$

und erklärt so die Form des neuen Kerns, den wir definieren als

$$k_*^S\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) := \exp\left(-\sqrt{\sum_{k=1}^{n} \frac{(x_k^i - x_k^j)^2}{4\sigma_k^4}}\right) \qquad (\boldsymbol{\sigma} \geq \mathbf{0})\,. \tag{4.7}$$

Definition 4.3 (verallgemeinerter Polynomialkern [20])**.** *Beim Polynomialkern wird zur Verallgemeinerung ein neuer Parameter eingeführt. Ausgehend vom Skalarprodukt innerhalb des Kerns, vgl. Definition 2.24, wird analog zu den bisherigen Überlegungen folgender neuer Polynomialkern definiert:*

$$k_*^P\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) := \left(c + \sum_{k=1}^{n} \frac{x_k^i \cdot x_k^j}{\sigma_k^2}\right)^d \qquad (\boldsymbol{\sigma} \geq \mathbf{0})\,. \tag{4.8}$$

Bei diesen Kernen erhält jede Dimension k des Datenraumes ein eigenes Gewicht σ_k. Die Anzahl der zu optimierenden Kernparameter erhöht sich dadurch von 1 auf n. Bei hochdimensionalen Datensätzen mit mehreren hundert Variablen führt das zu einem beinahe unlösbaren Problem. Das ist auch der Grund für die spärliche Anzahl an Veröffentlichungen zu diesem Thema. Wahrscheinlich existiert auch keine SVM-Software[29], die verallgemeinerte Kerne nutzen und optimieren kann. In der Literatur wurde ein Hinweis auf den verallgemeinerten Gauß-Kern in [20] gefunden. Es wird gezeigt, dass die Verwendung verschiedener Kernparameter nicht zu einem übertrainierten System führt. Wir werden bei unserer Arbeit versuchen, diese Art der Kerne zu nutzen und erhoffen uns davon verlässliche und interpretierbare Klassifikatoren. Für Daten mit einer zu großen Anzahl an Merkmalen kann man entweder zum Standardmodell umschalten, d.h.

$$\sigma_1 = \ \ldots\ = \sigma_n = \sigma$$

wählen oder statistische Verfahren zur Reduktion des Datenraumes, wie zum Beispiel die klassische Hauptkomponentenanalyse [31] nutzen. Einen Vektor $\boldsymbol{\sigma}$ zu optimieren hat unter anderem auch den positiven Nebeneffekt einer impliziten Variablenselektion. Wenn jede Raumdimension einen eigenen Wert σ_k hat, dann wird eine Variable mit wenig Einfluss auf die Klassifikation einen kleineren Wert aufweisen, als ein sehr einflußreiches Merkmal. Man kann daher eine Variablenselektion durchführen, indem

- ein fester Prozentsatz von Variablen mit kleinen Werten σ_k entfernt wird, oder
- alle Spalten k mit $\sigma_k < \sigma_{\min}$ entfernt werden.

Nach langer Suche in der Literatur, haben wir diese Idee auch in [20] gefunden. Für große Datensätze bietet die Kombination statistischer Verfahren und dieser impliziten Methode der Variablenselektion eine optimale Grundlage zur Identifizierung der wichtigsten Merkmale. Statistische Verfahren identifizieren hochkorrelierte Variablengruppen und säubern Merkmale ohne signifikante Streuung mühelos und ohne komplizierte Parameterwahl aus. Die übrigen Variablen werden dann direkt von der Support-Vektor-Maschine mit verallgemeinertem Kern untersucht. Neben der Variablenselektion ermöglichen wir außerdem eine stark verbesserte Interpretierbarkeit der Hypothesenfunktion. Die unterschiedlich großen

[29] Alle Pakete zu überprüfen ist nicht möglich, viele sind nicht frei zugänglich. Bisher gibt es jedoch keine dokumentierte Software dazu.

Werte für alle σ_k^* im finalen Modell zeigen deutlich, welche Merkmale großen Einfluß auf die Klassentrennung haben. Bei einem einfachen Kern gibt es nur ein σ^*, welches im Mittel die besten Ergebnisse geliefert hat. Genau dadurch entsteht die so oft bemängelte Schwachstelle der Support-Vektor-Klassifikation. SVM's werden seit langem als *black-box*-Verfahren bezeichnet [71], weil es keine Möglichkeit der Interpretation der Hypothesenfunktion gibt. Lediglich der Einfluss einzelner Trainingspunkte ist anhand des erlernten Vektors $\boldsymbol{\alpha}^*$ und den dadurch gegebenen Support-Vektoren zu erklären. Das ist aber für die Praxis nur teilweise von Bedeutung, denn hohe Kosten entstehen auch durch die Suche und Messung von Merkmalen. Dieser Problematik wirken wir mit verallgemeinerten Kernen entgegen. Sollte eine gründliche Suche im Merkmalsraum nicht möglich sein, kann man mittels eines Multiparameter-Kerns auch etwas weniger aufwändige Tests durchführen. Man könnte beispielsweise die folgenden Einschränkungen treffen:

- Zunächst wird ein optimales einzelnes σ bestimmt und ausgehend davon werden alle σ_k im Intervall $[\sigma - \epsilon, \sigma + \epsilon]$ für ein $\epsilon > 0$ optimiert.
- Die Variablen werden in eine bestimmte Anzahl $\tilde{n} < n$ von Gruppen zusammengefasst und es gilt $\sigma_k = \sigma_{\tilde{k}}$, falls k zur Gruppe $\tilde{k}$ $(1 \leq \tilde{k} \leq \tilde{n})$ gehört.

4.2.3.2 Ensemble-Kerne

Ensemble-Methoden sind ein modernes Teilgebiet des maschinellen Lernens [103]. Es gibt sehr viele Ausprägungen dieser Methoden, aber die Grundidee dahinter ist immer die gleiche. Wenn ein einzelner Klassifikator Daten nur schlecht klassifiziert oder instabil ist und auf kleinste Daten- und Parameteränderungen extrem reagiert, kann es sinnvoll sein, mehrere Klassifikatoren zu trainieren und als Ausgabewert immer ein gewichtetes Mittel der Ausgabewerte aller Funktionen zu wählen [60]. Bekannte Algorithmen zur Umsetzung von Ensemble-Klassifikation sind zum Beispiel Boosting [131] und Bagging [14].

Ensemble-Methoden könnte man auch im Zusammenhang mit SVM-Klassifikation verwenden. Zwar ist allgemein bekannt, dass die Methoden besonders gut für sehr schwache Klassifikationsalgorithmen funktionieren, eine Nutzung für starke Algorithmen wie die SVM's ist aber nicht ausgeschlossen. Erste Tests dazu wurden in [33] vorgestellt. Diese waren von relativ schlechter Performance. Kürzlich veröffentlichte Studien [158] zeigen jedoch, dass es sich lohnt, Ensemble-Methoden auch für Support-Vektor-Maschinen zu entwickeln. Dabei sind mehrere SVM-Klassifikationssysteme basierend auf unterschiedlichen Deskriptorsets erzeugt worden. Alle Systeme wurden trainiert und der Klassifikator arbeitete mit der einfachen Mehrheitsentscheidung. Man kann also verschiedene SVM's trainieren und dann kombinieren, wir gehen jedoch einen etwas anderen Weg. Wir haben uns überlegt, die Ensemble-Idee direkt im Inneren der SVM zu implementieren – in der Kernfunktion. Es gibt sehr unterschiedliche Kernfunktionen, jedoch gibt es einige Eigenschaften, die für alle positiv semidefiniten Kerne gelten. Dazu gehört auch die Eigenschaft der Kernkombination.

Bemerkung 4.1. *Seien k_1 und k_2 positiv definite Kerne und $\kappa_1, \kappa_2 \in [0, 1]$. Dann gelten [36]:*

- $\tilde{k}_1 := \kappa_1 \cdot k_1$ *und* $\tilde{k}_2 := \kappa_2 \cdot k_2$ *sind Kerne,*
- $\tilde{k} := \tilde{k}_1 + \tilde{k}_2$ *ist ein Kern.*

Man kann also einen neuen Kern als Kombination anderer Kerne definieren. Den grundlegenden Ideen der Ensemble-Methoden folgend, haben wir eine gewichtete Kernkombination für Support-Vektor-Maschinen implementiert. Für zwei Kerne k_1 und k_2 und einen Parameter $\kappa \in [0, 1]$ berechnet der neue Kern Ähnlichkeiten im Merkmalsraum als

$$k\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) := \kappa \cdot k_1\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) + (1 - \kappa) \cdot k_2\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) \ . \tag{4.9}$$

Dieser Kern kann für alle SVM-Modelle verwendet werden. Der Parameter κ muss zusätzlich zu den anderen SVM-Parametern optimiert werden. Sollte sich κ während der Optimierung stark in Richtung 0 oder 1 bewegen, spricht das dafür, dass einer der beiden Kerne die besten Ergebnisse liefert. Der Nachteil von gewichteten Kernen ist der hohe Rechenaufwand bei der Berechnung der Grammatrix. Für sehr große Datensätze, bei denen die Kernberechnungen einen signifikanten Teil der Trainingszeit ausmachen, kommt es damit zu starken Zunahmen der Rechenzeit.

4.3 A priori und a posteriori Manipulation des SVM-Trainings

Bisher haben wir Methoden diskutiert, die direkt am Lernverfahren ansetzen. Auf dem Gebiet des Data-Mining gibt es aber auch Ansätze, welche die Güte eines Modells durch äußere Einflüsse, die unabhängig vom eigentlichen Lernverfahren sind, verbessern sollen. Bekannte Beispiele dafür sind Datensäuberung und Variablenselektion [31]. Beide Methoden sind hinreichend getestet und dokumentiert. In diesem Kontext wollen wir weitere Ansätze zu Handhabung unausgeglichener Daten vorstellen, die weniger bekannt sind, aber in der Praxis sehr effektiv sein können. Da sie im Zusammenhang mit Support-Vektor-Maschinen bisher praktisch kaum zum Einsatz gekommen sind, lohnt sich eine Untersuchung um so mehr.

4.3.1 Manipulation der Trainingsdaten – Sampling-Methoden

Man kann schon bei der Auswahl der Trainingsdaten Einfluss auf die Klassifikationsfunktion nehmen. Dazu gibt es die beiden Methoden Undersampling und Oversampling. Beim

Undersampling werden aus der großen Klasse Punkte entfernt, wohingegen beim Oversampling die Punkte der seltenen Klasse mehrfach verwendet werden [98]. Gelegentlich werden die Kopien der positiven Daten verrauscht, um bessere Effekte zu erzielen.[30] Im Allgemeinen werden die Klassengrößen einander angepasst. Das ist in der Historie der Lernverfahren begründet, denn die klassischen Lernalgorithmen sind stark auf ausgeglichene Klassen ausgelegt.[31] Auch viele der heutigen Algorithmen des maschinellen Lernens können das Problem des kostensensitiven Lernens nicht optimal handhaben.[32]

Über die Frage, welche der beiden Methoden, die sinnvollere ist, wird noch aktiv diskutiert [3]. Für SVM's gibt es bisher so gut wie keine Arbeiten dazu. Wir halten an dieser Stelle die nach unserer Meinung für das SVM-Lernen wichtigen Fakten fest:

- Oversampling vergrößert die Anzahl der Trainingsdaten und führt im Allgemeinen zu einer längeren Trainingsphase.
- Das Auftreten doppelter Punkte kann sich negativ auf die Eigenschaften der Grammatrix auswirken[33] und damit auch die Konvergenz von SVM-Algorithmen beeinflussen [135].
- Oversampling führt nicht zu Informationsverlust in den Trainingsdaten.
- Wieviele Kopien der Daten sinnvoll sind, ist unklar.
- Das Verrauschen von Daten ist schwierig, insbesondere für dünnbesetzte binäre Datensätze.
- Undersampling birgt die Gefahr, wichtige Punkte zu eliminieren.
- Es gibt keine verläßlichen Algorithmen zur automatischen Elimination, von Ausreißerselektion [31] abgesehen.
- Da die SVM ohnehin nur die wichtigsten Punkte in die Klassifikationsfunktion einbettet, vgl. Abbildung 2.11, stellt sich die Frage, inwieweit man in diesen Prozess eingreifen sollte.

Wir werden in Kapitel 7 Tests zum Sampling durchführen, wobei wir uns auf das Oversampling beschränken werden. Undersampling sollte von Experten der jeweiligen Anwendung durchgeführt werden.

[30] Florian Markowetz, persönliche Kommunikation, 2003

[31] Als Beispiel sei hier die Maximum-Entropie-Methode genannt, welche wir in [38, 83] vergleichend zu SVM's getestet haben.

[32] "*the curse of learning in imbalanced data*" [148]

[33] Sind zwei Trainingspunkte identisch, hat die Grammatrix keinen Vollrang mehr und ist somit auch nicht positiv definit.

4.3.2 Manipulation des Klassifikationsfunktion – Schwellwertverschiebung

Eine weitere interessante Idee zur Handhabung unausgeglichener Datensätze beruht auf der Modifikation der finalen Zielfunktion. Eine interessante Idee dazu wurde in [5] skizziert. Grundlage der dortigen Überlegungen ist ebenfalls die Einsicht, dass eine höhere Sensitivität bei Duldung eines Anstiegs der falsch positiven Punkte durchaus von Interesse sein kann. Es wird vorgeschlagen, sich diesem Ziel mittels einer Wahl von b^* zu nähern, welches nicht zwingend die KKT-Bedingungen erfüllt. Das macht immer dann Sinn, wenn Kosten für Fehler unterschiedlich sind und sich Punkte beider Klassen innerhalb des Gebietes der Marge befinden. Dieses Szenario entspricht realen unausgeglichenen Datensätzen, die groß und teilweise verrauscht sind.

Die aus diesem Ansatz entstehenden Fragen sind:

- Gibt es ausreichend viele (positive) Testdaten, um den tatsächlichen Effekt einer Änderung von b^* erkennen zu können?
- Welches Kriterium sollte für die Optimierung des Schwellwertes angesetzt werden?

Diese beiden Fragen stellen sich jedoch auch bei der modifizierten Fehlergewichtung. Die Problematik der Datenverfügbarkeit liegt in der Verantwortung des Nutzers. Mit Optimierungskriterien werden wir uns im nächsten Kapitel ausführlich auseinandersetzen. In Kapitel 7 untersuchen wir, inwieweit die Idee der Schwellwertverschiebung mit der Fehlergewichtung kombiniert werden kann.

Geometrisch interpretiert, wird auch bei der Schwellwertverschiebung der in Abbildung 4.4 dargestellte Effekt erzielt, wobei die Änderung erst nach der eigentlichen Modellierung erfolgt. Eine Gesamtfehlerminimierung wird sich damit zwar selten erzielen lassen, jedoch steigen die Möglichkeiten, einen sensitiven Klassifikator zu generieren.

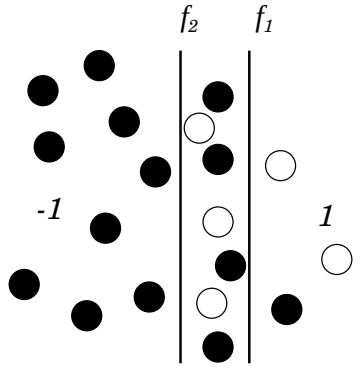

Abbildung 4.7: Effekt einer Änderung des Schwellwertes für Testdaten.

In Abbildung 4.7 bezeichnet f_1 die erlernte Klassifikationsfunktion. Dabei gibt es drei falsch negative und einen falsch positiven Punkt. Bei einer Verschiebung der Hyperebene (f_2) ergeben sich fünf falsch positive, aber keine falsch negativen Punkte. Bei angenommenen Kosten von 2 für falsch negative und 1 für falsch positive Punkte ergibt sich eine Kostenersparnis von 2. In [5] wird empfohlen, die Sensitivität zu maximieren und gleichzeitig eine untere Schranke für die Spezifität zu fordern. Die Verschiebung der Hyperebene sollte nicht nur im Hinblick auf die positiven Punkte erfolgen, denn sonst wird ein konstant positiver Klassifikator generiert. Auf die Problematik der Einseitigkeit einiger Modelloptimierungen wird in Kapitel 5 noch ausführlich eingegangen. In [165] wurde gezeigt, dass die Methode der Schwellwertverschiebung beim Einsatz neuronaler Netze zu sehr guten Ergebnissen führen kann. Wir werden diese Untersuchungen in unserer Arbeit mit Support-Vektor-Maschinen durchführen und zeigen, dass sich auch hier positive Effekte erzielen lassen. Die Tests werden in Kapitel 7 vorgestellt. Es sei noch erwähnt, dass diese Methode zwar sehr einfach ist, jedoch in ihrer Wichtigkeit nicht unterschätzt werden sollte. Auf dem Gebiet der Klassifikation wird ein enormer Aufwand betrieben, um sensitive Ergebnisse zu erhalten. Sollte die Schwellwertverschiebung im Zusammenhang mit SVM's einsetzbar sein, kann man von einem Erfolg sprechen.

4.4 Zusammenfassung

In diesem Kapitel haben wir uns mit Erweiterungen der Standard-SVM's beschäftigt. Dazu wurden zahlreiche Modifikationen vorgeschlagen, die alle darauf abzielen, die Klassifikation flexibel und so genau wie möglich zu gestalten. Neben der Einbettung von drei SVM-Algorithmen zu einem Gesamtverfahren haben wir verschiedene Techniken des kostensensitiven Lernens vorgestellt und implementiert. Diese sollten immer dann eingesetzt werden, wenn eine einzelne Klasse von stärkerem Interesse ist, als die andere. Wir haben die Standard-Kernfunktion erweitert, sodass nun gleichzeitig verschiedene Kerne mit mehreren Parametern verwendet werden können. Testergebnisse präsentieren wir später in Kapitel 7. Neben den beschriebenen positiven Effekten, fällt auf, dass bei allen Methoden neue Parameter ins Spiel gekommen sind. Die Optimierung dieser Parameter und die Auswahl passender Modelle kann sich schnell zu einem nicht zu unterschätzenden Problem entwickeln. Im folgenden Kapitel setzen wir uns mit dieser Aufgabe auseinander und entwickeln Vorschläge zur SVM-Parameteroptimierung.

Kapitel 5

Parameteroptimierung

Für das Training einer Support-Vektor-Maschine müssen Parameterwerte an die jeweiligen Daten angepaßt werden. Verschiedenste Parameter und ihre Funktionen sind in den bisherigen Kapiteln vorgestellt worden. Tatsache ist, dass die Generalisierungsfähigkeit einer SVM, d.h. die Qualität der Klassifikation, von den gewählten Parameterwerten abhängt [105]. Ungünstige Parameterwerte führen zum Scheitern einer Support-Vektor-Maschine und je besser man sie festlegt, umso verlässlicher ist auch die erlernte Klassifikationsfunktion. Das Problem der Parameteroptimierung für Support-Vektor-Maschinen ist ein in den letzten Jahren stark diskutiertes Thema [20, 79], wird aber dennoch zu oft unterschätzt oder ignoriert, beispielsweise wenn Ergebnisse neu entwickelter Methoden mit Ergebnissen von SVM-Software ohne Parametertuning verglichen werden [5]. Dieses Kapitel beschäftigt sich mit der Thematik der Parameteroptimierung für Support-Vektor-Maschinen. Wir diskutieren verschiedene Qualitätsmaße und stellen eine von uns getestete Software zur gradientenfreien Parameteroptimierung vor.

Jede SVM-Software beinhaltet *default*-Werte für alle Modellparameter, um zu sichern, dass jeder Nutzer ein Ergebnis generieren kann. Diese Standardwerte sind erstaunlicherweise fast immer gleich:

$$C = 1 \qquad \sigma = \frac{1}{n} \qquad (\text{z.B. [19]}) \,. \tag{5.1}$$

Sie sollten nur dann benutzt werden, wenn keine Möglichkeit zur Parameteroptimierung gegeben ist und auch kein Vorwissen zu den Daten vorhanden ist. Veröffentlichungen zeigen deutlich, dass diese Parameter so gut wie nie brauchbar sind. Beispielsweise wurden für die sogenannten Splice-Daten

$$C = 1348 \qquad \sigma = \sqrt{21.6} \qquad (\text{in [79]}) \,. \tag{5.2}$$

als günstige Parameterwerte bestimmt.

5.1 Qualitätsmessung und Gütemaße

Für die Parameteroptimierung ist es notwendig, interne Tests bewerten zu können. Dazu muss eine Risikofunktion $\Phi : \mathbb{R}^m \rightarrow \mathbb{R}$ existieren, die als Eingabe eine m-elementige Parameterkombination $\boldsymbol{\pi}$ akzeptiert und nach Validierung oder Recall (vgl. Abschnitt 2.1) eine reelle Zahl zurückliefert, welche das diesen Parametern zugeordnete Risiko widerspiegelt. Für ein solches Qualitätsmaß sollten gelten:

$$\begin{aligned} \Phi(\boldsymbol{\pi}) &\geq 0 \; \forall \boldsymbol{\pi} \in \mathbb{R}^m \, , \\ \Phi(\boldsymbol{\pi}) &= 0 \; \Leftrightarrow \text{ keine Verbesserung möglich,} \\ \Phi(\boldsymbol{\pi}) &> 0 \; \Leftrightarrow \text{ Verbesserung möglich.} \end{aligned}$$

Für die Definition eines Qualitätsmaßes Φ ist es nicht notwendig, das letztendlich verwendete Optimierungsverfahren zu kennen. Auch die genaue Anzahl der relevanten Parameter m ist nicht wichtig. Das Maß muss jedoch unser Empfinden für ein optimales Modell widerspiegeln.

Wir wollen die Parameteroptimierung so gestalten, dass ausschließlich die Werte des Qualitätsmaßes als Informationsquelle verwendet werden dürfen, also insbesondere keine weiteren Interaktionen während der Laufzeit notwendig sind. Liefert Φ für zwei Parameterkombinationen, von denen wir eine bevorzugen, den gleichen Funktionswert, fehlt diese Information und beide Kombinationen werden als gleich gut eingestuft. Deshalb sollte Φ so definiert werden, dass unsere Präferenzrelation $\succ$ gilt:

$$\begin{aligned} \Phi(\boldsymbol{\pi}^i) < \Phi(\boldsymbol{\pi}^j) &\iff \boldsymbol{\pi}^i \succ \boldsymbol{\pi}^j \text{ (wir präferieren Kombination } i\text{) und} \\ \Phi(\boldsymbol{\pi}^i) = \Phi(\boldsymbol{\pi}^j) &\iff \boldsymbol{\pi}^i \sim \boldsymbol{\pi}^j \text{ (wir sind indifferent) .} \end{aligned} \tag{5.3}$$

Eine Parameterkombination $\boldsymbol{\pi}^i$ sollte immer dann gegenüber $\boldsymbol{\pi}^j$ bevorzugt werden, wenn sie einen geringeren Funktionswert Φ zur Folge hat. Dieses Modell gilt nur für Risikomaße, die auf einer Art Fehlersummierung aufbauen, wie es für Support-Vektor-Maschinen durchaus üblich ist. Der Begriff des Gütemaßes ist in diesem Zusammenhang verwirrend, wir werden in diesem Kapitel jedoch auch monoton wachsende Funktionen der Güte vorstellen, sodass wir hier nicht verallgemeinern können. Primäres Ziel ist daher immer die Minimierung des Risikos bzw. die Maximierung der Güte. Ob es sich bei der Funktion Φ um Risiko oder Güte handelt, sollte aus der jeweiligen Definition hervorgehen.

Parameteroptimierung soll das Risiko, den sogenannten Generalisierungsfehler, minimieren. Dabei handelt es sich jedoch um eine nicht berechenbare Größe. Es muss ein Performancemaß definiert werden, welches zur Abschätzung des wahren Risikos geeignet ist.

Als Bewertungsmaß bei v-facher Kreuzvalidierung kann beispielsweise eine Risikofunktion Φ der Form

$$\Phi(\boldsymbol{\pi}) = td^{-1} \sum_{j=1}^{v} \sum_{i=(j-1)\frac{td}{v}+1}^{j\cdot\frac{td}{v}} \max\left\{-y_i \cdot h_{\boldsymbol{\pi}}^j(\boldsymbol{x}^i), 0\right\} \tag{5.4}$$

eingesetzt werden. Dabei ist $\boldsymbol{\pi}$ ein Parametervektor. $h_{\boldsymbol{\pi}}^j : \mathbb{R}^n \to \{-1, 1\}$ ist die während der j-ten Iteration der Validierung erlernete Hypothesenfunktion. Die Fehler während der Validierung werden in den v Durchgängen aufsummiert und zur Normierung mittels der Anzahl aller Punkte gewichtet. Es wird diejenige Parameterkombination $\boldsymbol{\pi}^*$ ausgesucht, welche das kleinste Risiko $\Phi(\boldsymbol{\pi}^*)$ aufweist. (5.4) basiert auf der einfachen Genauigkeit.

Die in den Definitionen 2.9 bis 2.12 angegebenen Kennzahlen könnten zur Parameteroptimierung herangezogen werden, sind jedoch sehr einseitig. Wird nur auf eine Klasse geachtet, besteht die Gefahr, dass der Klassifikator bei der anderen Klasse versagt. Die Optimierung der Genauigkeit ist bei unausgeglichenen Datensätzen ebenfalls kein sinnvolles Optimierungskriterium. Wir müssen uns mit dieser Problematik auseinandersetzen und Kennzahlen definieren, welche über Parameterwerte flexibel die Priorisierung einzelner Ziele ermöglichen.

Wir entwickeln im Laufe dieses Abschnittes Qualitätsmaße, welche nicht auf groben Hypothesen $h_{\boldsymbol{\pi}}$, sondern auf den linearen Zielfunktionen $f_{\boldsymbol{\pi}}$ arbeiten und zusätzlich sensitiv sind. Damit ist eine bessere Fehlerbewertung möglich, als die grobe Einteilung in richtige und falsche Klassifikationen liefert. Die Motivation für die Suche solcher Qualitätsmaße liegt darin, dass in Anlehnung an Abschnitt 4.2.2 Interesse an hoher Sensitivität für unausgeglichene Datensätze besteht. Dieses Ziel kann mit einem einfachen Performancemaß nicht erreicht werden, denn die Optimierung von Gewichten C^+ und C^- erfordert auch ein entsprechendes Qualitätsmaß. Am Ende dieses Abschnittes sollen geeignete Gütemaße für die SVM-Parameteroptimierung vorliegen. Sie dienen als Grundlage für den Optimierungsalgorithmus, den wir im Abschnitt 5.2 besprechen werden, sowie für unsere Tests in Kapitel 7.

Die Genauigkeit (2.1) ist klasseninvariant. Sie ändert ihren Wert nicht, wenn sich Sensitivität und Spezifizität des Modells entgegengesetzt gleichmäßig ändern. Eine für Regressionsmodelle mit rd-facher Kreuzvalidierung entwickelte Q^2-Statistik [105] ist von der Form

$$\Phi(\boldsymbol{\pi}) = \frac{\sum_{i=1}^{td}(f_{\boldsymbol{\pi}}^i(\boldsymbol{x}^i) - y_i)^2}{\sum_{i=1}^{td}(\bar{y} - y_i)^2} \,.$$

In diesem Fall gibt es für jeden Punkt $\boldsymbol{x}^i$ eine neue Zielfunktion $f_{\boldsymbol{\pi}}^i$, weil die *leave-one-out* Methode verwendet wurde. Man kann leicht sehen, dass zum Einen der Ausdruck im

Nenner eine parameterunabhängige Konstante ist und zum Anderen der Zähler so einfach ist, dass zwei verschiedene Fehler zum gleichen Zuwachs führen können. Für $y_i = 1$ und $f^i_{\boldsymbol{\pi}}(\boldsymbol{x}^i) = 0.0$ bzw. $f^i_{\boldsymbol{\pi}}(\boldsymbol{x}^i) = 2.0$ wird im Zähler jeweils 1.0 addiert. Das Beispiel der Vorhersage des Aufwachzeitpunktes nach Narkotisierung [110] verdeutlicht auch für Regressionsaufgaben die Problematik einer derart allgemeinen Fehlersummierung. Es gibt Maße, die aufgrund ihrer Struktur besser geeignet sind, um Ergebnisse sensitiv zu bewerten. Im Folgenden werden einige interessante Maße vorgestellt.

5.1.1 Kostenmatrizen

Eine besonders für mehrklassige Probleme übliche Art der Testbewertung besteht in der Aufsummierung der Fehler mittels Kostenmatrix. Eine Kostenmatrix $\boldsymbol{KSM}$ gibt an, mit welchem Faktor ω jede Klassifikationsausgabe belegt wird. Die Kostenmatrix für den binären Fall hat die Form

	Klasse 1	Klasse -1
Hypothese 1	ω_{rp}	ω_{fp}
Hypothese -1	ω_{fn}	ω_{rn}

Tabelle 5.1: Kostenmatrix für ein binäres Klassifikationsproblem.

Zur Modellbewertung werden die Matrizen $\boldsymbol{KFM}$ (Tabelle 2.1) und $\boldsymbol{KSM}$ elementweise multipliziert und anschließend müssen alle Produkte aufsummiert werden. Die Summe wird durch die Anzahl der Punkte geteilt. Das Risikomaß ist dann definiert als

$$\Phi(\boldsymbol{\pi}) = td^{-1} \sum_{i=1}^{2} \sum_{j=1}^{2} KFM_{i,j} \cdot KSM_{i,j} \; . \tag{5.5}$$

Angepasst an die Validierungsdarstellung (5.4) besagt diese Vorgehensweise

$$\begin{aligned} \Phi(\boldsymbol{\pi}) \;\; = \;\; & td^{-1} \sum_{j=1}^{v} \sum_{i=(j-1)\frac{td}{v}+1}^{j\cdot\frac{td}{v}} \Big(\delta_{y_i,1} \left(\delta_{h^j_{\boldsymbol{\pi}}(\boldsymbol{x}^i),1} \omega_{rp} + \delta_{h^j_{\boldsymbol{\pi}}(\boldsymbol{x}^i),-1} \omega_{fn} \right) \\ + \;\; & \delta_{y_i,-1} \left(\delta_{h^j_{\boldsymbol{\pi}}(\boldsymbol{x}^i),-1} \omega_{rn} + \delta_{h^j_{\boldsymbol{\pi}}(\boldsymbol{x}^i),1} \omega_{fp} \right) \Big) \; . \end{aligned} \tag{5.6}$$

Für das Standardverfahren (5.4) zur einfachen Summierung von Fehlern gelten

$$\omega_{rp} = 0, \; \omega_{rn} = 0 \qquad \text{und} \qquad \omega_{fn} = 1, \; \omega_{fp} = 1 \; .$$

Die Änderung der Faktoren kann beliebig erfolgen [2]. Auch negative Werte können verwendet werden.

5.1.2 Anreicherungsfaktor

Dieser Faktor berechnet das Verhältnis zwischen dem Anteil tatsächlich positiver Punkte in den als positiv klassifizierten Testpunkten, und dem Anteil positiver Punkte in den gesamten Testdaten. Er besagt demnach, um welchen Faktor die Wahrscheinlichkeit einen positiven Punkt „zu ziehen“ ansteigt, wenn man anstelle des Gesamtdatenbestandes nur aus den Punkten mit Hypothese 1 auswählt. Ein sensitiver Klassifikator sorgt dafür, dass die positiven Punkte verdichtet vorliegen. In der Pharmaindustrie spielt der Anreicherungsfaktor eine wichtige Rolle [82], denn er hilft dabei, aus einer großen Datenbank eine Teilmenge von Substanzen herauszufiltern, für die dann eine gewünschte Eigenschaft mit höherer Wahrscheinlichkeit vorliegt, als im Gesamtbestand. Ausgehend von dieser Menge können dann weitere Untersuchungen stattfinden.

Definition 5.1 (Anreicherung, Enrichment)**.** *Der Anreicherungsfaktor basiert auf dem Maß der Präzision und ist definiert als*

$$\begin{aligned} ef &:= \frac{rp}{rp+fp}\Big/\frac{pp}{td} \\ &= \frac{pr \cdot td}{pp}\,. \end{aligned} \tag{5.7}$$

Folgerung 5.1. *Es gilt* $ef \geq 0$ *sowie*

$$ef \begin{cases} > 1 & \textit{Anreicherung (positiver Effekt)} \\ = 1 & \textit{kein Effekt der Anreicherung} \\ < 1 & \textit{Konzentrationsabnahme von Klasse } 1 \textit{ (schlechte Klassifikation)} \end{cases}.$$

Der Anreicherungsfaktor ist nach oben beschränkt durch die Anzahl der Daten. Es gilt

$$ef \leq \frac{td}{pp}\,,$$

wobei Gleichheit auftreten kann, wenn es keine falsch positiven Punkte gibt. Der Anreicherungsfaktor ist ein gutes Kriterium zur Hervorhebung der Klasse 1, es gibt jedoch durch die starke Fixierung auf die Konzentration positiver Punkte in den als positiv klassifizierten Daten Kritikpunkte an dieser Kennzahl, auf die wir im Folgenden kurz eingehen.

Beispiel 5.1. *Seien* $pp = 10$ *und* $np = 100$.

1. *Für* $rp = 5,\ rn = 100$ *gilt*

$$ef^{(1)} = \frac{5}{5+0}\Big/\frac{10}{10+100} = 11\,.$$

2. *Für* $rp = 1,\ rn = 100$ *gilt*

$$ef^{(2)} = \frac{1}{1+0} \Big/ \frac{10}{10+100} = 11\ .$$

3. *Für* $rp = 9,\ rn = 99$ *gilt*

$$ef^{(3)} = \frac{9}{9+1} \Big/ \frac{10}{10+100} = 9,9\ .$$

Die Gleichheit von $ef^{(1)}$ und $ef^{(2)}$ entspricht nicht unseren Vorstellungen einer guten Klassifikationsbewertung. Weiterhin ist es bedenklich, dass $ef^{(3)} < ef^{(1)}$ und $ef^{(3)} < ef^{(2)}$ gelten, denn im letzten Fall wurden beinahe alle positiven Punkte erkannt, wobei es nur einen falsch positiven Punkt gab. Diese Ergebnisse zeigen, dass das Konzept der Anreicherung, welches stark auf Präzision abzielt, zu unerwünschten Effekten führen kann und damit als alleiniges Gütemaß zur Parameteroptimierung nicht geeignet ist. Ein anderes Problem dieses Maßes ist die fehlende Normierung. Ergebnisse für verschiedene Datensätze lassen sich nicht vergleichen. Wir nutzen den Anreicherungsfaktor deshalb nur zur Auswertung von Testergebnissen, nicht aber zur Auswertung interner Tests während der Parameteroptimierung [82].

5.1.3 ROC-Maß

Die Receiver-Operating-Characteristik (ROC) diente ursprünglich in der Signalerkennung zur graphischen Darstellung und zum Vergleich von Klassifikationsergebnissen [49, 147] und wird mittlerweile auch in anderen Gebieten eingesetzt, z.B. in der Medizin. Im ROC-Raum werden Sensitivität und Spezifität dargestellt. Typischerweise werden ganze Graphen eingetragen. Bei der Anwendung von SVM's entstehen jeweils nur einzelne Punkte, die aber ebenfalls eingetragen werden können. In Abbildung 5.1, die Punkte im ROC-Raum zeigt, ist a der beste und d der schlechteste Punkt. Zusätzlich sind b und c besser als d. Die Punkte b und c lassen sich ohne weitere Prioritäten nicht vergleichen. Neben „guten" und „schlechten" unterscheidet man dabei auch zwischen „konservativen" und „liberalen" Klassifikatoren [49]. Ein konservatives Verfahren neigt dazu, positive Klassifikationen nur dann zuzulassen, wenn eine hohe Sicherheit besteht. Das resultiert in geringer Sensitivität und hoher Spezifität (Punkt b). Liberale Methoden tendieren im Gegensatz dazu, Punkte auch bei geringem Verdacht als positiv zu klassifizieren. Das wirkt sich gut auf die Sensitivität aus, lässt jedoch die Spezifität sinken (Punkt c).

Ausgehend von dem Einheitsquadrat der ROC-Visualisierung wurde in [133] ein neues Gütemaß zur Bewertung von Testergebnissen entwickelt. Anschaulich soll das neue Maß den Abstand zum schlechtesten Punkt im Raum darstellen, siehe Abbildung 5.1. Ausgehend vom euklidischen Abstand ist das normierte Gütemaß, im Folgenden als ROC-Maß bezeichnet, definiert als

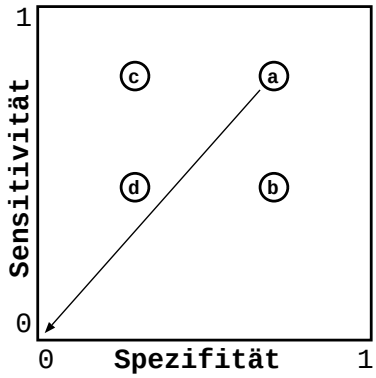

Abbildung 5.1: ROC-Visualisierungsbox.

$$roc := \sqrt{\frac{se^2 + sp^2}{2}} \, . \tag{5.8}$$

Beispiel 5.2. *Für die in Beispiel 5.1 angeführten Fälle liefert das ROC-Maß*

$$roc^{(1)} = \sqrt{0.5 \cdot (0.5 + 1)} \approx 0.87 \, ,$$

$$roc^{(2)} = \sqrt{0.5 \cdot (0.1 + 1)} \approx 0.74 \, ,$$

und

$$roc^{(3)} = \sqrt{0.5 \cdot (0.9 + 0.99)} \approx 0.97 \, .$$

5.1.4 F-Maß

Das sogenannte F-Maß ist eine Abbildung von Präzision und Sensitivität auf ein Einzelmaß. Es entspricht dem harmonischen Mittel beider Maße [108] und eignet sich gut zur Bewertung von Klassifikationen auf unausgeglichenen Daten, da die beiden Einzelmaße stark zur Klasse 1 hingewandt sind [43]. Die Sensitivität allein eignet sich nicht als Gütemaß, da sie die falsch positiven Punkte nicht kontrolliert. Als Gegenstück dazu eignet sich die Spezifität, aber auch, wie im Falle des F-Maßes, die Präzision.

Definition 5.2 (F-Maß)**.**

$$\begin{aligned} fm &:= \frac{1}{\frac{1}{2}\left(\frac{1}{pr}+\frac{1}{se}\right)} \\ &= 2\,\frac{pr\cdot se}{pr+se} \\ &= 2\,\frac{rp}{rp+fp}\cdot\frac{rp}{pp}\Big/\left(\frac{rp}{rp+fp}+\frac{rp}{pp}\right) \\ &= \frac{2rp}{pp+rp+fp}\,. \end{aligned} \tag{5.9}$$

Die Fälle $pr = 0$ *und* $se = 0$ *sind nicht vorgesehen, sodass wir für* $pr = 0$ *oder* $se = 0$ *dem Maß den Wert* 0 *geben.*

Folgerung 5.2. *Wegen* $pr \in [0,1]$ *und* $se \in [0,1]$ *gilt*

$$fm \in [0,1]\,.$$

Beispiel 5.3. *Für die in Beispiel 5.1 angeführten Fälle liefert das F-Maß*

$$fm^{(1)} = \frac{2\cdot 5}{10+5+0} \approx 0.67\,,$$

$$fm^{(2)} = \frac{2\cdot 1}{10+1+0} \approx 0.18\,,$$

und

$$fm^{(3)} = \frac{2\cdot 9}{10+9+1} \approx 0.90\,.$$

Die Bewertung der Beispiele mittels F-Maß entspricht unseren Vorstellungen einer sensitiven Qualitätsmessung. Auf den Gebieten des Information Retrieval und der maschinellen Spracherkennung ist das F-Maß relativ häufig anzutreffen [124]. Interessant ist die Verwendung der Präzision, die nicht den Anteil falscher Klassifikationen in den negativen Punkten misst, sondern den Anteil falsch positiver Punkte in den als positiv klassifizierten Punkten berechnet. Hier gibt es eine enge Beziehung zum Anreicherungsfaktor (5.7). Im Vergleich zum ROC-Maß ist die deutlich unterschiedliche Bewertung von Fall 2 und 3 positiv zu sehen.

In unserer Software sind die Gütemaße (2.1), (2.2), (2.3), (2.4), (5.7), (5.8) und (5.9) implementiert und können optional gewählt werden (Default: F-Maß).

5.1.5 Parametrisierte Maße

Bei den Maßen (5.8) und (5.9) liegt jeweils eine Gleichgewichtung von Sensitivität und Spezifität bzw. Präzision vor. Für kostensensitive Anwendungen entspricht das nicht immer den Präferenzen. Wir erweitern die Maße nun um einen Parameter zur Gewichtsverlagerung.

Das ROC-Maß kann man erweitern durch eine flexible Formulierung, die Gewichte zuläßt. Sie hat die Form [133]

$$roc_{\gamma,\delta} := \sqrt{\frac{\gamma \cdot se^2 + \delta \cdot sp^2}{\gamma + \delta}} \quad (\gamma, \delta \in \mathbb{R}_+) \; . \tag{5.10}$$

Geometrisch kann man Gewichte als Streckung oder Stauchung des Quadrates in der jeweiligen Dimension interpretieren, sodass auch eine Reduzierung auf einen Parameter möglich ist. Sei $\beta := \gamma/(\gamma + \delta)$, dann hat das zu (5.10) äquivalente Maß die Form

$$roc_\beta := \sqrt{\beta \cdot se^2 + (1-\beta) sp^2} \quad (\beta \in [0,1]) \; . \tag{5.11}$$

Der Vorteil in dieser Formulierung liegt darin, dass nur ein Parameter gewählt werden muss.

Beispiel 5.4. *Für Fall* 2 *in Beispiel 5.1, also für* $se = 0.1$ *und* $sp = 1.0$*, liefert das variable ROC-Maß (5.11) je nach Faktor* β *sehr unterschiedliche Ergebnisse:*

$$\begin{aligned} roc_{0.1} &= \sqrt{0.001 + 0.9} \approx 0.95 \; , \\ roc_{0.5} &= \sqrt{0.005 + 0.5} \approx 0.71 \; , \\ roc_{0.9} &= \sqrt{0.009 + 0.1} \approx 0.33 \; . \end{aligned}$$

Über den Faktor β *im variablen ROC-Maß können Ergebnisse also je nach Fragestellung und Kosten höchst unterschiedlich bewertet werden.*

Eine Gewichtung kann auch zum F-Maß (5.9) hinzugefügt werden [124]. Mittels eines Gewichtes $\alpha \in [0,1]$ kann man abweichend vom Standardwert 0.5 ein flexibles Maß definieren [94].

Definition 5.3 (variables F-Maß)**.**

$$fm_\alpha := \frac{1}{\alpha \frac{1}{pr} + (1-\alpha)\frac{1}{se}} \; . \tag{5.12}$$

α ermöglicht ein Abschwächen oder Verstärken des Sensitivitäts-Einflusses. Es gilt $fm_\alpha \in [0,1]$. Die allgemein bekannte Darstellung des variablen F-Maßes hat die Form

$$fm_\beta := \frac{(1+\beta^2)\, pr \cdot se}{\beta^2 \cdot pr + se} \quad (\beta \in [0,\infty]) \tag{5.13}$$

und entsteht aus (5.12) durch einfaches Umformen mit $\alpha := \frac{1}{\beta^2+1}$. Es gilt $fm_\beta \in [0,1]$ und es sind

- $fm_0 = pr$,
- $fm_\infty = se$.

Beispiel 5.5. *Für Fall 2 in Beispiel 5.1, also für $se = 0.1$ und $pr = 1.0$, liefert das variable F-Maß (5.13) je nach Faktor β sehr unterschiedliche Ergebnisse:*

$$fm_{0.5} = \frac{(1+0.25)\, 1.0 \cdot 0.1}{0.25 \cdot 1.0 + 0.1} \approx 0.36$$
$$fm_{1.0} = \frac{(1+1.0)\, 1.0 \cdot 0.1}{1.0 \cdot 1.0 + 0.1} \approx 0.18$$
$$fm_{1.5} = \frac{(1+2.25)\, 1.0 \cdot 0.1}{2.25 \cdot 1.0 + 0.1} \approx 0.14$$

Über den Faktor β im variablen F-Maß können Ergebnisse also je nach Fragestellung und Kosten höchst unterschiedlich bewertet werden.

Aus den Beispielen 5.4 und 5.5 ziehen wir folgende Konsequenzen:

1. Variable Maße stellen flexible und interpretierbare Größen dar, welche durchaus Verwendung bei der SVM-Parameteroptimierung finden sollten. In konkreten Anwendungen wurde mit der Wahl von $\beta = 1$ [166] die Variabilität bisher selten genutzt. Es wird zu prüfen sein, ob eine andere Gewichtung für spezielle Datensätze individuelle Verbesserungen liefert.

2. Über den Faktor β können wichtige Ziele gesteuert werden, β selbst ist jedoch keine im engeren Sinne zu optimierende Variable, sondern muss a priori gewählt werden, sodass alle Parametertests mit dem gleichen Maß bewertet werden.

Die Parametrisierung ist in unserer Software als Option für F-Maß und ROC-Maß wählbar.

5.1.6 Maße mit weichen Zählern

In den letzten Abschnitten ist viel über die Unterscheidung von Fehlerarten berichtet worden. Kritisch an einer Fehlerbewertung ist jedoch zusätzlich die Frage: was genau ist ein Fehler? Üblicherweise werden Klassifikationsfehler über die Bedingung

$$y \cdot h(\boldsymbol{x}) = -1 \tag{5.14}$$

definiert. Das bedeutet, es liegt eine scharfe Trennung zwischen korrektem und falschem Ergebnis vor. In Kapitel 2 haben wir gesehen, dass während des Trainings Klassifikationen, die zwar korrekt, aber zu schwach sind (unscharfe Klassifikationen mit Verletzung der Marge), einen positiven Schlupfvariablenwert zugeordnet bekommen. In Anlehnung an die Idee der Marge stellen wir im Folgenden neue weiche Maße vor [43].

Zunächst rufen wir die Nebenbedingungen der primalen Aufgaben weicher SVM-Modelle in Erinnerung. Es müssen

$$y_i(\langle \boldsymbol{w}, \boldsymbol{x}^i \rangle + b) \geq 1 - \xi_i \;\; (i = 1, \ldots, rd)$$

gelten, siehe (2.30) und (2.35). Es werden nicht nur falsche, sondern auch unscharfe Klassifikationen dokumentiert. Für $y_i = 1$ bedeutet das

$$f(\boldsymbol{x}^i) = \begin{cases} \geq 1 & \Rightarrow \quad \xi_i = 0 \\ \in [0,1) & \Rightarrow \quad \xi_i \in (0,1] \\ \in (-\infty, 0) & \Rightarrow \quad \xi_i \in (1, \infty) \end{cases} .$$

Der Fall $y_i = -1$ wird analog hergeleitet. Dieses Schema übernehmen wir für die Auswertung von Validierungsergebnissen. Dazu werden die Funktionswerte anstelle der Hypothesen betrachtet. Wir schränken diese flexible Bewertung jedoch auf das Intervall $[-1, 1]$ ein, damit Punkte mit großem Betrag der Zielfunktion das Modell nicht zu stark beeinflussen.

Definition 5.4 (Kennzahl richtig positiver Klassifikationen bei Beachtung der Marge)**.**

$$\tilde{rp} := \sum_{i \in \mathcal{TD}: y_i = 1, 1 \leq f(\boldsymbol{x}^i)} 1 + \sum_{i \in \mathcal{TD}: y_i = 1, 0 \leq f(\boldsymbol{x}^i) < 1} f(\boldsymbol{x}^i) \; .$$

Definition 5.5 (Kennzahl richtig negativer Klassifikationen bei Beachtung der Marge)**.**

$$\tilde{rn} := \sum_{i \in \mathcal{TD}: y_i = -1, f(\boldsymbol{x}^i) \leq -1} 1 + \sum_{i \in \mathcal{TD}: y_i = -1, -1 < f(\boldsymbol{x}^i) < 0} -f(\boldsymbol{x}^i) \; .$$

Definition 5.6 (Kennzahl falsch positiver Klassifikationen bei Beachtung der Marge)**.**

$$\tilde{fp} := \sum_{i \in \mathcal{TD}: y_i = -1, 1 \leq f(\boldsymbol{x}^i)} 1 + \sum_{i \in \mathcal{TD}: y_i = -1, 0 \leq f(\boldsymbol{x}^i) < 1} f(\boldsymbol{x}^i) \; .$$

Definition 5.7 (Kennzahl falsch negativer Klassifikationen bei Beachtung der Marge)**.**

$$\tilde{fn} := \sum_{i \in \mathcal{TD}: y_i = 1, f(\boldsymbol{x}^i) \leq -1} 1 + \sum_{i \in \mathcal{TD}: y_i = 1, -1 < f(\boldsymbol{x}^i) < 0} -f(\boldsymbol{x}^i) \; .$$

Weiche Zähler nach den Definitionen 5.4 bis 5.6 wirken sich direkt auf Sensitivität (2.2), Spezifität (2.3) und Präzision (2.4) aus. Wir bezeichnen sie als $\tilde{se}$, $\tilde{sp}$ und $\tilde{pr}$.

Bemerkung 5.1. *Die Wertebereiche von $\tilde{se}$, $\tilde{sp}$ und $\tilde{pr}$ ändern sich nicht. Es gelten weiterhin $\tilde{se},\ \tilde{sp},\ \tilde{pr} \in [0,1]$. Dennoch werden aufgrund der Beachtung der Marge die erreichten Werte deutlich kleiner sein, als im Standardmodell.*

Definition 5.8. *Das variable ROC-Maß mit Beachtung der Marge ist analog zu (5.11) definiert als*

$$\tilde{roc}_\beta := \sqrt{\beta \cdot \tilde{se}^2 + (1-\beta)\tilde{sp}^2}\ . \tag{5.15}$$

Definition 5.9. *Das variable F-Maß mit Beachtung der Marge ist analog zu (5.13) definiert als*

$$\tilde{fm}_\beta = \frac{(\beta^2+1)\tilde{pr} \cdot \tilde{se}}{\beta^2 \cdot \tilde{pr} + \tilde{se}}\ . \tag{5.16}$$

Es gelten $\tilde{roc}_\beta \in [0,1]$ und $\tilde{fm}_\beta \in [0,1]$, sodass die neuen flexiblen parametrisierten Maße (5.15) und (5.16) normiert sind. Weiche Zähler können auch für die einfachen Gütemaße wie die Genauigkeit verwendet werden.

Die weiche Fehlermessung haben wir mittels einer Option für ROC- und F-Maß sowie alle einfachen Gütemaße implementiert.

5.2 Numerische Parameteroptimierung mit *APPSPACK*

Im letzten Abschnitt haben wir Qualitätsmaße zur Bewertung von Klassifikationsergebnissen während der Kreuzvalidierung oder anderer Testmethoden vorgestellt. Jedes Maß liefert eine Zahl, welche die Güte der verwendeten Parameter widerspiegelt. Im Laufe der Arbeit haben wir viele Parameter vorgestellt, sodass sich die Anpassung für größere Datensätze zu einem Problem entwickelt. Es stellt sich die Frage, welche Parameterkombinationen überhaupt getestet werden sollen. Nach einem kurzen Überblick zu einfachen Methoden stellen wir im Anschluss ein Softwarepaket zur gradientenfreien Parameteroptimierung vor.

Es gibt viele Formen der SVM-Parameteroptimierung, die sehr einfach sind. Sie beinhalten keine Optimierungsverfahren. Unter einer iterativen Suche versteht man das Durchprobieren einiger Parameterkombinationen, wobei nach jedem Durchlauf abhängig von der Performance entweder gestoppt oder wieder mit neu bestimmten Parameterwerten validiert wird. Diese Methode führt oft zu scheinbar guten Ergebnissen, ist jedoch auf die Ideen des Nutzers beschränkt. Auch birgt sie die Gefahr, sich in scheinbar vielversprechenden Gegenden des Parameterraumes zu verfangen. Für schon bekannte Datensätze ist diese Methode interessant, um zu untersuchen, wie Ergebnisse abhängig von Änderungen optimaler

Parameterwerte variieren. Daraus lassen sich interessante Aussagen zur Robustheit von SVM-Klassifikatoren herleiten. Untersuchungen zur Robustheit von SVM-Klassifikatoren gibt es hinsichtlich verrauschter Daten, jedoch nicht bezüglich Parameterschwankungen.

Die Rastersuche, auch *grid search* oder Gittersuche genannt, stellt eine sehr einfache und weit verbreitete Methode der SVM-Parameteroptimierung dar. Dabei wird a priori für jeden Parameter eine endliche Anzahl von Werten angegeben, die getestet werden sollen. Es werden alle möglichen Parameterkombinationen verwendet. Diese Art der Parametersuche wird beispielsweise in [67] vorgeschlagen. Für den einfachen Fall der beiden Parameter C und σ wird dort speziell das Gitter

$$C = 2^{-5},\ 2^{-3},\ \ldots,\ 2^{15} \qquad \text{und} \qquad \sigma = 2^{-15},\ 2^{-13},\ \ldots,\ 2^{3}$$

empfohlen, welches unabhängig von den jeweiligen Daten ist. Bei mehr als zwei Parametern dauert diese Gittersuche jedoch zu lange oder das verwendete Gitter ist sehr grob und führt dadurch zu schlechten Resultaten. Letzterer Fall tritt in der Praxis sehr oft auf, insbesondere wenn die Datensätze sehr groß sind und ein einzelnes Training schon sehr lange dauert. Ein leicht verbessertes Verfahren ist, Validierungen auf einem groben Gitter mit anschließender Validierung auf einem feinen Gitter, welches in dem Teil des ersten Gitters mit den besten Ergebnissen liegt, durchzuführen [67]. Die Gefahr liegt dann jedoch darin, das erste Gitter so grob zu wählen, dass interessante Bereiche verloren gehen. Angenommen, wir optimieren 3 Parameter mittels einer zehnfachen Kreuzvalidierung. Für jeden Parameter stellen wir 5 Werte zur Verfügung. Dann haben wir insgesamt $10 \cdot 5 \cdot 5 \cdot 5 = 1250$ Support-Vektor-Maschinen zu trainieren, bis die beste Parameterkombination feststeht. Bei nur 5 Werten pro Parameter kann man jedoch nicht von einer Optimierung sprechen. Die Rastersuche ist also für mehr als zwei Parameter kaum einsetzbar.

Wir haben bisher verdeutlicht, dass triviale Arten der Parameteroptimierung nicht geeignet sind, um eine größere Anzahl von Parametern anzupassen. Beim einfachsten SVM-Modell mit 2 Parametern kann man die Validierungen noch gut überblicken; bei mehr als 2 Parametern sollte der Prozess automatisiert werden. Der Vorteil von algorithmisch gesteuerten Optimierungsmethoden ist die verminderte Anzahl an zu testenden Kombinationen durch gezielte Suche nach vielversprechenden Richtungen. Das ermöglicht die Nutzung komplexer Lernmethoden mit zahlreichen Parametern.

Prinzipiell unterscheidet man zwei Vorgehensweisen zur Parameteroptimierung. Gradientenbasierte Verfahren, wie zum Beispiel konjugierte Gradienten und Newton-Verfahren [57], erfordern die Bestimmung des Gradienten der Zielfunktion. Die im Abschnitt 5.1 vorgestellten Gütemaße sind nicht differenzierbar, sodass diese Methoden ausscheiden. Gradientenfreie Verfahren werden immer dann eingesetzt, wenn die Bestimmung des Gradienten nicht möglich ist, denn sie kommen einzig mit der Auswertung der Zielfunktion aus. Im Zusammenhang mit Support-Vektor-Maschinen können sie für jedes Gütemaß verwendet werden, sofern dieses einem beliebigen Punkt des Parameterraumes eine reelle Zahl zuordnen kann. Gradientenfreie Optimierung ähnelt einer iterativen Suche. Ausgehend von

einer Parameterkombination werden iterativ neue Kombinationen getestet, wobei die Suchrichtungen jedoch über ein Optimierungsverfahren gesteuert werden. Für Support-Vektor-Maschinen angewandte Verfahren sind beispielsweise die Mustersuche aus [28] in [105] und die Schwarmoptimierung aus [13, 81] in [115]. Zur Nutzung gradientenfreier Verfahren ist eine zu minimierende Auswertungsfunktion zur Verfügung zu stellen. Typischerweise ist das die Genauigkeit, was wir aber ändern wollen. Wir werden im Allgemeinen mit dem Maß (5.16) arbeiten. Für die Minimierung nutzen wir dann das zugehörige E-Maß [94].

Definition 5.10 (E-Maß)**.**

$$e\tilde{m}_\beta := 1 - f\tilde{m}_\beta \,. \tag{5.17}$$

Alle anderen implementierten Maße stehen natürlich auch zur Verfügung und können mittels einer Option mit weichen Zählern versehen werden. Im Folgenden stellen wir eine frei verfügbare Software zur gradientenfreien Parameteroptimierung vor. Diese Software wird für die Tests in Kapitel 7 verwendet.

APPSPACK

APPSPACK-4.0 (*asynchronous parallel pattern search for derivative-free optimization*) [55, 86] ist ein C++ Softwarepaket zur Lösung von beschränkten und unbeschränkten Optimierungsproblemen. Es implementiert ein gradientenfreies Suchverfahren, welches für Probleme mit kompliziert auszuwertenden Zielfunktionen konzipiert wurde. Wir werden es für die SVM-Parameteroptimierung verwenden. Die Software sowie zahlreiche Publikationen sind verfügbar unter

http://software.sandia.gov/appspack [34] .

Anhand der aktuellen Dokumentationen [55] und [86] stellen wir auf den folgenden Seiten das in *APPSPACK* umgesetzte Verfahren vor, um zu zeigen, wie die Suche nach optimalen Punkten im Parameterraum funktioniert.

Betrachtet wird die Optimierungsaufgabe

$$\min_{\boldsymbol{\pi}\in\mathbb{R}^m} \quad \Phi(\boldsymbol{\pi}) \tag{5.18}$$

unter den Nebenbedingungen

$$\boldsymbol{u} \leq \boldsymbol{\pi} \leq \boldsymbol{o} \tag{5.19}$$

[34] Dank an Tamara Kolda für die Aufnahme unserer Arbeiten [40, 43, 48] in die Publikationsliste.

mit $\boldsymbol{u}, \boldsymbol{o} \in \mathbb{R}^m$. Die unteren und oberen Schranken sind optional. Sie werden für jeden Parameter einzeln angegeben. Die Behandlung von Nebenbedingungen war für *APPSPACK* ursprünglich nicht angedacht ([66], *APPSPACK*-1.0), in den Anwendungen zeigte sich jedoch die Notwendigkeit, Nebenbedingungen beachten zu können, sodass ab Version *2.0* untere und obere Schranken definiert werden konnten. Da die Funktion Φ minimiert wird, muss das gewählte Gütemaß ebenfalls eine zu minimierende Funktion sein. Da das F-Maß (5.16) diese Voraussetzung nicht erfüllt, haben wir das E-Maß (5.17) eingeführt. Analog kann man bei normierten Maßen vorgehen.[35]

Der *APPSPACK*-Algorithmus zeichnet sich dadurch aus, dass er im parallelen Modus asynchron optimiert. Iterativ werden Mengen von Punkten für den Test berechnet und verschickt. Die Ergebnisse werden später wieder eingesammelt, jedoch in einem flexiblen Modus. Es wird nicht gewartet, bis alle Ergebnisse zur Verfügung stehen. Dauert eine Berechnung etwas länger, wird das entsprechende Ergebnis einfach später betrachtet, falls es dann noch von Interesse ist. Zu weiteren Aspekten der asynchronen parallelen Optimierung verweisen wir auf Kapitel 6. In diesem Abschnitt besprechen wir den zugrunde liegenden Optimierungsalgorithmus.

Das grobe Schema des *APPSPACK*-Algorithmus ist zunächst in Abbildung 5.2 dargestellt, eine genauere Darstellung folgt. Ein von *APPSPACK* betrachteter Punkt $\boldsymbol{\pi}^{\text{neu}}$ ist immer ein Sohn eines anderen Punktes $\boldsymbol{\pi}^{\text{V}}$, seines Vaters, d.h.

$$\boldsymbol{\pi}^{\text{neu}} = \boldsymbol{\pi}^{\text{V}} + \Delta_{\text{neu}} \boldsymbol{r}_{\text{neu}} \ . \tag{5.20}$$

Er wird mittels folgender Informationen als Objekt abgespeichert:

- Vektor $\boldsymbol{\pi}^{\text{neu}} \in \mathbb{R}^m$,
- Nummer su_{neu} der realisierten Suchrichtung $\boldsymbol{r}_{\text{neu}}$ bei Übergang zu $\boldsymbol{\pi}^{\text{neu}}$,
- Marke τ_{neu},
- Marke τ^V_{neu} des Vaters,
- Funktionswert Φ_{neu}, falls vorhanden,
- Funktionswert Φ^V_{neu} des Vaters,
- Schrittweite Δ_{neu} bei Übergang zum Sohn,
- Zustandsinformationen:

$$\text{Punkt } \boldsymbol{\pi}^{\text{neu}} \text{ wurde } \begin{cases} \text{nicht ausgewertet} \\ \text{ausgewertet} \end{cases} ,$$

[35] Der Anreicherungsfaktor (5.7) hat eine datenabhängige obere Schranke und eignet sich nicht für die Optimierung. Er ist jedoch zur Darstellung von Ergebnissen für unausgeglichene Datensätze sehr hilfreich. Wir verweisen dazu auf unsere Ergebnisse in [82].

1. Generiere eine Menge von Testpunkten

$$\boldsymbol{T}^{\text{neu}} := \{\boldsymbol{\pi}^{\text{opt}} + \Delta_i \boldsymbol{r}_i : i \in \mathcal{R}\}\,.$$

   Dabei sind $\boldsymbol{\pi}^{\text{opt}}$ der beste bis dahin gefundene Parametervektor, $\boldsymbol{r}_i$ die i-te Suchrichtung und Δ_i die zugehörige Schrittweite. $\mathcal{R}$ ist die Indexmenge von Suchrichtungen, die getestet werden sollen.

2. Übergib die Menge $\boldsymbol{T}^{\text{neu}}$ an ein Auswertungssystem und betrachte die Menge $\boldsymbol{T}^{\text{alt}}$ der bereits ausgewerteten Punkte.

3. Prüfe, ob es einen Punkt $\boldsymbol{\pi}^{\text{alt}}$ in $\boldsymbol{T}^{\text{alt}}$ gibt, der besser als $\boldsymbol{\pi}^{\text{opt}}$ ist. Als Notation dafür wählen wir $\boldsymbol{\pi}^{\text{alt}} \succ_{APPS} \boldsymbol{\pi}^{\text{opt}}$. *APPSPACK* prüft dazu zwei Bedingungen, die noch beschrieben werden. Gibt es ihn, dann war die Iteration erfolgreich, sonst war sie erfolglos.

4. Bei erfolgreicher Iteration, ersetze $\boldsymbol{\pi}^{\text{opt}}$ durch $\boldsymbol{\pi}^{\text{alt}}$. Aktualisiere die Menge der Suchrichtungen und lösche die überflüssigen Punkte in der Auswertungswarteschleife.

5. Bei erfolgloser Iteration, reduziere einige Schrittweiten in geeignetem Maße und starte erneut.

Abbildung 5.2: *APPSPACK*-Schema.

$$\text{Zielfunktionswert ist } \begin{cases} \Phi_{\text{neu}} \\ \text{nicht bekannt} \end{cases},$$

$$\text{Abstiegsbedingung ist } \begin{cases} \text{erfüllt} \\ \text{nicht erfüllt} \end{cases}.$$

Zur Definition der Abstiegsbedingung sei auf (5.22) verwiesen. Die folgenden Abschnitte werden sich mit wichtigen Aspekten der Implementierung gradientenfreier Optimierung innerhalb des *APPSPACK*-Pakets beschäftigen. Zusammenfassend wird auf Seite 131 der Algorithmus schematisch dargestellt.

5.2.1 Vergleich von Kandidaten

Im dritten Schritt des *APPSPACK*-Schemas muss geprüft werden, ob ein Punkt $\boldsymbol{\pi}^{\text{alt}}$ besser als $\boldsymbol{\pi}^{\text{opt}}$ ist ($\boldsymbol{\pi}^{\text{alt}} \succ_{APPS} \boldsymbol{\pi}^{\text{opt}}$). Dazu werden zwei Bedingungen [55] definiert, die beide erfüllt sein müssen.

- Die erste Bedingung besagt, dass der Zielfunktionswert der Gütefunktion für $\boldsymbol{\pi}^{\text{alt}}$ dem bisherigen optimalen Wert vorgezogen wird. Das tritt ein, falls

$$\Phi(\boldsymbol{\pi}^{\text{alt}}) < \Phi(\boldsymbol{\pi}^{\text{opt}}) \tag{5.21}$$

  gilt. Für $\boldsymbol{\pi}^{\text{alt}} = \boldsymbol{\pi}^{\text{opt}}$ wird einheitlich jeweils der Parametervektor mit der kleineren Kennnummer gewählt.

- Die zweite Bedingung soll sichern, dass beim Übergang vom Vater zum Sohn eine adäquate Abnahme des Zielfunktionswertes vorliegt, d.h.

$$\Phi_{\text{alt}} < \Phi_{\text{alt}}^{V} - \varsigma\Delta_{\text{alt}}^{2} \, . \tag{5.22}$$

  $\varsigma \geq 0$ kann gewählt werden. Im Fall $\varsigma = 0$ spricht man von einer einfachen, sonst von einer hinreichenden Abnahme.

5.2.2 Nebenbedingungen und Skalierung

APPSPACK unterstützt einfache obere und untere Schranken. Für unsere Anwendung der Parameteroptimierung ist das ausreichend. Alle Schranken müssen vor der Benutzung des Programms in einer Datei abgelegt werden. Sie werden dann zur Generierung von Suchrichtungen und Testpunkten verwendet.

Zu allen Nebenbedingungen muss eine Skalierung definiert werden. Variablenskalierung ist für gradientenfreie Verfahren von großer Bedeutung. *APPSPACK* generiert den Skalierungsvektor $\boldsymbol{l} \in \mathbb{R}^m$ standardmäßig über die Schranken als

$$\boldsymbol{l} := \boldsymbol{o} - \boldsymbol{u} \, . \tag{5.23}$$

Sollte es keine endlichen Schranken geben, muss $\boldsymbol{l}$ vom Nutzer zur Verfügung gestellt werden.

5.2.3 Suchrichtungen und Schrittweiten

Nach jeder erfolgreichen Iteration von *APPSPACK* muss eine neue Menge von Suchrichtungen $\boldsymbol{R}$ generiert werden. Dazu werden die skalierten Koordinatenrichtungen herangezogen, wobei nur Punkte innerhalb des durch die Nebenbedingungen gegebenen Tangentialkegels betrachtet werden, also

$$\boldsymbol{R} := \left\{\boldsymbol{r}^1, .., \boldsymbol{r}^p\right\} = \left\{-l_i\boldsymbol{e}^i : \pi_i > u_i\right\} \cup \left\{l_i\boldsymbol{e}^i : \pi_i < o_i\right\} \, . \tag{5.24}$$

Mit $\boldsymbol{e}^i$ wird wie üblich der i-te Einheitsvektor (der Dimension m) bezeichnet. Jede Suchrichtung i hat eine Nummer τ_i und eine ihr zugeordnete Schrittweite Δ_i. Die Kennnummer

ist notwendig, um feststellen zu können, ob in der Auswertungswarteschleife Punkte mit dieser Suchrichtung warten. Für $\tau_i = -1$ existieren keine unausgewerteten Punkte mehr, die über die Richtung $\boldsymbol{r}^i$ generiert wurden und Δ_i wird benutzt, um einen neuen Testpunkt zu definieren. Für $\tau_i \neq 1$ gilt: τ_i ist die Nummer des Punktes, der aus Δ_i und $\boldsymbol{r}^i$ entstanden ist.

5.2.4 Generierung neuer Testpunkte

Die Menge $\mathcal{I} := \{i : \Delta_i \geq \Delta_{\text{tol}},\ \tau_i = -1\}$ enthält Suchrichtungen, die noch nicht konvergiert haben und auch keine Testpunkte im Auswertungssystem liegen haben. Für jedes $i \in \mathcal{I}$ wird ein zulässiger Testpunkt generiert. Sollte ein Punkt

$$\boldsymbol{\pi}^{\text{neu}} = \boldsymbol{\pi}^{\text{akt}} + \Delta_i \boldsymbol{r}_i$$

nicht zulässig sein, wird eine sogenannte Pseudoschrittweite $\tilde{\Delta}_i$ ermittelt. Sie ist definiert als die größtmögliche Schrittweite, die zu einem zulässigen Punkt führt, d.h.

$$\tilde{\Delta}_i = \max\left\{\Delta \in [0, \Delta_i] : \boldsymbol{u} \leq \boldsymbol{\pi}^{\text{akt}} + \Delta \boldsymbol{r}^i \leq \boldsymbol{o}\right\} . \tag{5.25}$$

5.2.5 Abbruchkriterien

Das wichtigste Abbruchkriterium in *APPSPACK* fußt auf den Schrittweiten. Dahinter steht die Tatsache, dass die Schrittweiten mit abnehmender Norm des Gradienten hinreichend glatter Funktionen kleiner werden und dadurch ein natürliches Abbruchkriterium liefern, auch wenn keine Gradienteninformationen zur Verfügung stehen. *APPSPACK* stoppt die Optimierung, sobald

$$\Delta_i < \Delta_{\text{tol}} \quad \forall i = 1, .., p\ .$$

Dabei ist Δ_{tol} eine a priori festzulegende minimale Schrittweite. Der *APPSPACK*-Algorithmus ist konvergent, falls die Zielfunktion stetig ist. Zusätzlich hat sich in der Praxis gezeigt, dass der Algorithmus auch für nicht-glatte Zielfunktionen, die auf komplexen Simulationen beruhen, verwendet werden kann [86]. Zu weiteren Details verweisen wir auf [6].

Es gibt zwei weitere Abbruchkriterien, die zur Verfügung stehen. Sie müssen vom Nutzer explizit gewählt werden. Sie basieren nicht auf Methoden zur Minimierung der Norm des Gradienten, vielmehr handelt es sich um einfache Kriterien ohne Glattheitsvoraussetzungen an die Zielfunktion. Die erste Methode besagt, die Optimierung zu beenden, sobald ein Punkt $\boldsymbol{\pi}^*$ lokalisiert wurde, für den

$$\Phi(\boldsymbol{\pi}^*) \leq \Phi_{\text{tol}}$$

gilt. Φ_{tol} ist ein a priori festgelegter Zielfunktionswert, der den Benutzer zufriedenstellt. Dieses Kriterium ist interessant, denn oft herrscht der Wunsch, einen bestimmten Schwellwert zu unterschreiten, ohne die Rechenzeit exzessiv auszudehnen. Kritisch an diesem Kriterium ist, dass eine Wahl von Φ_{tol} nicht garantiert, dass dieser Wert überhaupt durch die zugrunde liegende Simulation realisierbar ist. Ein anderes Verfahren ist jedoch immer anwendbar. Es besagt, die Optimierungsschritte zu stoppen, sobald die Anzahl der Funktionsaufrufe einen festgelegten Wert erreicht hat. Dieses Kriterium ist von Vorteil, falls nur begrenzte Rechenzeit bei sehr aufwändigen Modellen zur Verfügung steht. Über die Güte des damit angepassten Parametervektors läßt sich dann eine Aussage treffen, indem das letzte erreichte Minimum des Gütemaßes betrachtet wird. Ein guter Startwert kann auch helfen, zu einer akzeptablen Lösung zu finden. Alle Abbruchkriterien können kombiniert werden.

5.2.6 Auswertungssystem

Das Auswertungssystem von *APPSPACK* hat einen internen und einen externen Bereich. Der externe Modus muss eine Menge $\boldsymbol{T}^{\mathrm{neu}}$ von neuen Testpunkten einsammeln und eine Menge $\boldsymbol{T}^{\mathrm{alt}}$ von bereits ausgewerteten Punkten abgeben. Im internen Level erfolgt eine dreistufige Verarbeitung. Jeder Punkt wartet zunächst in der Warteschleife, bis die nötigen Ressourcen für seine Auswertung frei sind. In dieser Phase werden Punkte wieder entfernt, falls *APPSPACK* schon einen erfolgreichen Schritt in der entsprechenden Richtung getätigt hat. Im zweiten Level folgt die Berechnung des Zielfunktionswertes. Im Anschluss daran hat der Punkt zu warten, bis sein Ergebnis das Auswertungssystem verlassen kann.

5.2.7 Fazit

An dieser Stelle fassen wir alle für uns relevanten Eigenschaften von *APPSPACK* zusammen:

- Die Software ist frei verfügbar und wird regelmäßig von einem Entwicklerteam verbessert.
- Die Prozedur zur Auswertung der Zielfunktion muss nicht in die C++ Software eingebettet werden, sondern kann als separates Programm vorliegen. Andere Programmiersprachen können problemlos genutzt werden, da ausschließlich über Input-Output-Dateien kommuniziert wird.
- Es ist sowohl serielles als auch paralleles[36] Rechnen möglich.
- Es werden keine Gradienteninformationen benötigt.

[36] Siehe dazu Kapitel 6.

- Die Form der Zielfunktion ist nicht vorgegeben. Es kann sich um eine komplexe Simulation handeln. Ausschließlich aussagekräftige Zielfunktionswerte werden benötigt.
- Es können viele Parameter gleichzeitig optimiert werden (bis zu 100) [55].
- Es liegen gute Konvergenzeigenschaften vor.
- Das Scheitern einzelner Simulationen führt nicht zum Abbruch der Optimierung. Punkte, die keine sinnvolle Auswertung liefern, bekommen den Funktionswert ∞ zugewiesen. Der Nutzer kann dafür sogenannte *error strings* nutzen.

Der *APPSPACK*-Algorithmus ist in Abbildung 5.3 zusammenfassend dargestellt. Details zur parallelen Version der Software werden in Kapitel 6 erläutert.

Wir verweisen an dieser Stelle auf Anhang A, der Informationen zum Bug Fixing von *APPSPACK* sowie zur Struktur der `.apps` Datei (Kommunikation mit der SVM-Software) enthält.

5.3 Zusammenfassung

In diesem Kapitel haben wir Qualitätsmaße zur SVM-Parameteroptimierung vorgestellt und weiterentwickelt. Ausgehend von einfachen Maßen haben wir die Notwendigkeit flexibler und sensitiver Maße motiviert. Mittels dieser Maße können Ziele, wie beispielsweise kostensensitive Klassifikation, umgesetzt werden. Aspekte kostensensitiver Klassifikation mit Support-Vektor-Maschinen wurden bereits in Kapitel 4 behandelt. Im Anschluss haben wir Methoden zur SVM-Parameteroptimierung betrachtet. Dabei haben wir hauptsächlich die Software *APPSPACK* zur gradientenfreien Optimierung vorgestellt. Diese Methode der Parameteroptimierung eignet sich für unser Anliegen, wie wir im Abschnitt 5.2.7 zusammenfassend aufgelistet haben. In Kapitel 7 präsentieren wir Ergebnisse, die wir bei der Kopplung von *APPSPACK* mit unserer SVM-Software erreicht haben. Im folgenden Kapitel wenden wir uns zunächst Aspekten der Parallelisierung der SVM-Software und der Nutzung der Parallelität von *APPSPACK* zu.

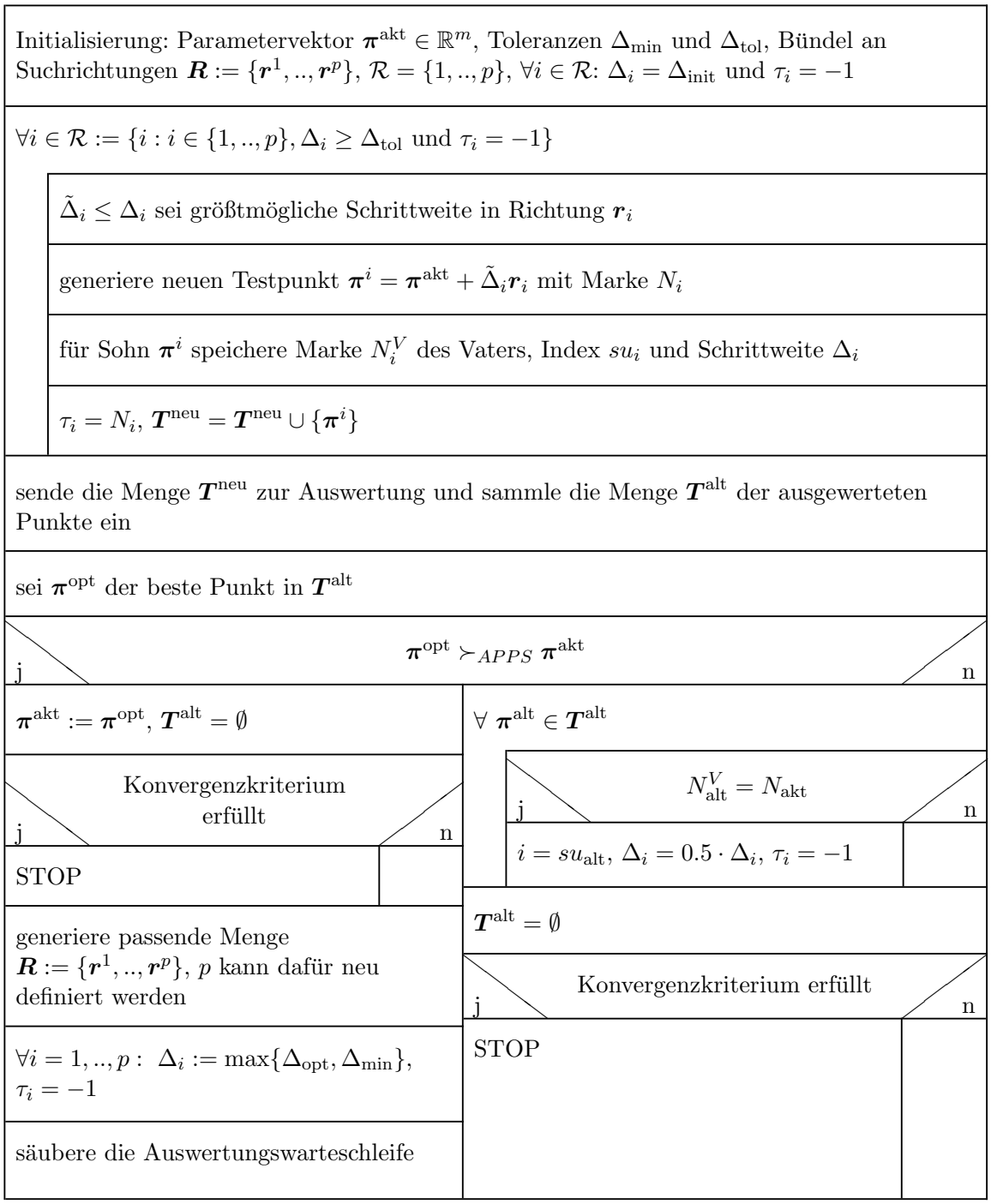

Abbildung 5.3: *APPSPACK*-Algorithmus.

Kapitel 6

Parallelisierung

Datensätze, die in Zukunft zur Klassifikation verwendet werden sollen, werden nicht mehr nur aus einigen hundert Punkten mit wenigen Variablen bestehen. In der Pharmaforschung gibt es heute schon Datenbanken, die mit hunderttausenden von Strukturen mit hunderten Attributen gefüllt sind [85]. Daraus entsteht der Wunsch, möglichst viele Daten zur Untersuchung von Abhängigkeiten zu verwenden. Auch bei anderen Anwendungen steigt die Anzahl verfügbarer Daten dramatisch an. Als Beispiel sei die Textklassifikation genannt, welche in der Zeit von Online-Recherchen immer wichtiger wird. Die Entwicklung skalierbarer paralleler Algorithmen des Data-Mining ist von großem Interesse.

Ein großer Nachteil der heutigen seriellen Support-Vektor-Maschinen ist der hohe Zeitaufwand bei der Berechnung großer Datensätze [22]. Dadurch wird die Anwendbarkeit des Verfahrens eingeschränkt auf kleine und mittelgroße Probleme. Die Entwicklung paralleler SVM's ist ein neues und wichtiges Forschungsthema. Es existieren einige junge Arbeiten auf diesem Gebiet, wobei sich die meisten auf Heuristiken für das Training auf reduzierten und verteilten Daten beschränken. Das bedeutet, große Datensätze werden entweder vor dem Training mittels Selektionsverfahren auf die gewünschte Größe verkleinert oder man teilt die Daten in mehrere Gruppen gewünschter Größe ein und für jede Gruppe wird das Lernverfahren unabhängig eingesetzt, bevor im Anschluss die Ergebnisse mittels komplizierter Heuristiken kombiniert werden. Teilweise werden auch nur lineare SVM's verwendet, da sich dafür vorhandene parallele Algorithmen der Optimierung nutzen lassen. Als Pakete für das Trainieren von SVM's auf großen Daten sind sie nicht geeignet.

In diesem Kapitel beschäftigen wir uns mit der Entwicklung einer parallelen SVM-Software sowie mit deren Kopplung an die parallele Version der *APPSPACK*-Software zur Parameteroptimierung. Die SVM-Software basiert auf dem in Kapitel 3 ausführlich beschriebenen Algorithmus sowie den in den Kapiteln 4 und 5 vorgestellten Modifikationen. Als Zwischenebene zur Realisierung der notwendigen Güteabfragen während der Parameteroptimierung dient eine parallele Kreuzvalidierungsroutine. Die Struktur des Gesamtsystems ist in Abbildung 6.1 dargestellt.

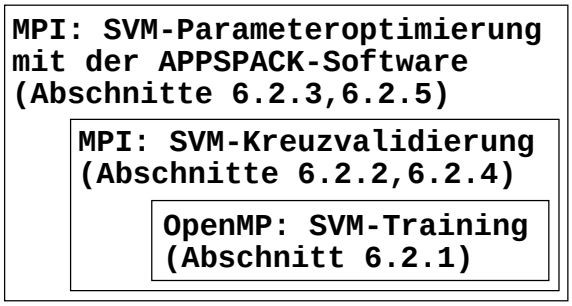

Abbildung 6.1: Schema zum Parallelisierungskonzept mit Zuordnung zu den Abschnitten.

Nach einem Überblick zum aktuellen Stand der Entwicklungen auf dem Gebiet des parallelen Data-Minings werden wir zunächst darstellen, welche Arbeiten zu parallelen Support-Vektor-Maschinen schon dokumentiert sind. Im Anschluss beschreiben wir unser dreistufiges flexibles Parallelisierungskonzept. Testergebnisse folgen in Kapitel 7.

6.1 Stand der Forschung

Die meisten Data-Mining-Algorithmen haben lange Laufzeiten für große Datensätze, während die Anzahl zu verarbeitender Daten in beiden Dimensionen, d.h. die Anzahl der Beispiele und der Variablen, stetig steigt. In Ergänzung zur Verbesserung serieller Algorithmen kann die Entwicklung und Nutzung paralleler Methoden die Rechenzeit verkürzen. Dieser Abschnitt gibt einen Überblick zu Aktivitäten des maschinellen Lernens auf großen Datensätzen, insbesondere auf dem Gebiet paralleler Klassifikationsverfahren, speziell SVM's.

6.1.1 Paralleles Data-Mining

Die Entwicklung paralleler Data-Mining-Methoden ist nicht neu. Schon in [140] wurden Strategien zur Implementierung paralleler Data-Mining-Algorithmen vorgestellt und bewertet. Die meisten parallelen Data-Mining-Methoden wurden für Distributed-Memory-Systeme entwickelt [72]. Dazu gehören

- Assoziationsregeln:

  Assoziationsregel-Algorithmen (ARA's) [64] nehmen eine wichtige Stellung innerhalb des Data-Mining ein. Die häufigsten Anwendungen beschäftigen sich mit Markt- und Konsumanalyse. ARA's müssen eine Menge von Teilmengen der Daten generieren, die häufig in den Datenbanken vorkommen. Zusätzlich müssen Regeln erlernt werden, die erklären, wie eine Teilmenge von Daten die Anwesenheit bestimmer anderer Daten in einer Datenbank beeinflusst. Die Algorithmen sind teuer bei Berechnung und sind gekennzeichnet durch viel Input/Output. Es existieren Daten- und

Taskparallele Implementierungen paralleler ARA's. Kritisch sind Synchronisation, Kommunikation, Workload, Datenverteilung und I/O-Minimierung. Eine interessante hybride Implementierung, welche die Vorteile beider Methoden ausnutzt, wird in [160] erklärt.

- k-Cluster-Verfahren:

  Clusterung [70] ist ebenfalls eine sehr wichtige Aufgabe bei der Datenanalyse. Objekte mit ähnlicher Struktur sollen erkannt und in eine Gruppe eingeordnet werden. Die Anzahl k der Gruppen kann gewählt werden. Während der Berechnung müssen k Clustermittelpunkte so gefunden werden, dass der mittlere Abstand eines Punktes zu seinem Clustermittelpunkt so klein wie möglich ist. Das Problem ist NP-vollständig [104]. Typischerweise wird eine iterative Prozedur implementiert, welche ausgehend von k Startpunkten die Mittelpunkte ändert, wobei als Kriterium eine Kostenfunktion minimiert wird. Anwendungen reichen von Vektorquantisierung über Zeitreihenanalyse, Visualisierung und Textanalyse, bis hin zu Navigation [30]. Vorgeschlagene Parallelisierungskonzepte addressieren die Abstandsberechnungen, welche die Rechenzeit für große Datensätze sehr stark dominieren. Diese sind unabhängig voneinander und können datenparallel gerechnet werden. Probleme gibt es mit den Kommunikationskosten [30]. Ein anderer Ansatz implementiert ein Master-Slave-Schema, in dem der Master die Daten aufteilt und verschickt [77]. Die Slaves arbeiten untereinander mittels Broadcast-Funktionen.

- Entscheidungsbäume:

  Klassifikationsaufgaben werden häufig über Entscheidungsbaumalgorithmen [122] gelöst. Ihre Popularität ist darin begründet, dass ein Baum schnell zu erlernen sowie einfach zu interpretieren ist, bei gleichzeitig hoher Genauigkeit. Die zunehmenden Daten bei der Klassifikation haben auch hier zu Engpässen und daher zu parallelen Entwicklungen geführt. Die Datenreduktion führte nicht mehr zu guten Lösungen. Die Baummethode ist nicht trivial parallelisierbar, da die Struktur des Baums irregulär ist und erst während der Laufzeit bestimmt wird. Die Arbeistlast an jedem Knoten ist unterschiedlich und datenabhängig. In [143] werden synchrone und unterteilte Baumberechnungen für die Parallelisierung vorgestellt. Beide Methoden haben Nachteile, wie z.B. Kommunikationskosten und Workloadprobleme (synchron) sowie Datenumverteilung (unterteilt). Eine hybride Formulierung führte zu einer Verbesserung. Weitere Ansätze sind in [76] zu finden.

- k-nächste-Nachbarn:

  Nächste-Nachbarn-Klassifikation [61] ist eine parameterfreie Lernmethode, bei der die Klassifikation eines Punktes anhand der k nächsten Nachbarn in der Umgebung, im Allemeinen über eine Mehrheitsentscheidung, vorgenommen wird. Der Speicher- und Rechenaufwand dieses oft verwendeten Verfahrens ist für große Daten wie auch bei anderen Data-Mining-Algorithmen sehr hoch. In [16] wird ein paralleler Al-

gorithmus zur Zerlegung einer Punktmenge vorgestellt, welcher sich auch für k-nächste-Nachbarn-Klassifikation einsetzen läßt. Bei der Parallelisierung wurden sogenannte "rake" und "compress" Routinen eingesetzt. Es werden Abschätzungen zur Zeitkomplexität gemacht, jedoch leider keine Testergebnisse angegeben.

- Neuronale Netze:

  Neuronalen Netzen [12] wird nachgesagt, sie seien den SVM's am ähnlichsten. Die Ideen sind vergleichbar, jedoch variiert die Struktur neuronaler Netze sehr stark. Es gibt einfache Netze, wie das Perceptron, und sehr komplizierte Netze mit vielen Zwischenschichten, bei denen viele Modellgewichte anzupassen sind. Eine Arbeit zu paralleler Klassifikation mit neuronalen Netzen ist [35], wobei man dort auf ein paralleles Training verzichtet hat mit der Begründung, dass dieses bei realen Anwendungen nur einmalig stattfindet. Man hat sich daher darauf konzentriert, die Klassifikationsphase zu parallelisieren, um Vorhersagen in Echtzeit machen zu können. Ein echtes paralleles Training neuronaler Netze ist uns bisher nicht bekannt.

- Boosting:

  Die Methode des Boosting [131] lernt einen hochgenauen Klassifikator dadurch, dass diverse schwache Klassifikatoren vereint werden. Für eine parallele Version wird der Gesamtdatensatz aufgeteilt und jeder Teildatensatz wird für einen Lernvorgang verwendet. Im Anschluss werden Ergebnisse ausgetauscht und Klassifikatoren kombiniert [93]. Am Ende exisitert ein Modell aus gewichteten Klassifikationsfunktionen für jeden Teildatensatz. Dieses kann zur Klassifikation neuer Daten verwendet werden.

In letzter Zeit werden auch parallele Data-Mining-Algorithmen für Shared-Memory-Systeme entwickelt und getestet. Dafür gibt es verschiedene Gründe:

- Shared-Memory-Maschinen haben seit einigen Jahren immer bessere Skalierungseigenschaften [72].
- Die neuen Maschinen bieten große Hauptspeicher und hohe Bus-Bandbreiten, was für Data-Mining-Anwendungen von enormer Wichtigkeit ist [140].
- SMP-Cluster sind mittlerweile – auch in der Industrie – verbreitet.

In [117] wurde ein Algorithmus zur parallelen Assoziationsregelentdeckung entwickelt und in [161] beschreiben die Autoren parallele Entscheidungsbaumverfahren für SMP-Systeme. Insgesamt gibt es zur Zeit aber noch wenige Entwicklungen auf diesem Gebiet, obwohl die Hersteller großer SMP-Systeme Data-Warehousing und Data-Mining als Zielanwendungen sehen. Ein Grund dafür ist, dass sich serielle Data-Mining-Algorithmen nicht trivial im Shared-Memory-Modus parallelisieren lassen. Oft ist es notwendig, die seriellen Algorithmen zu ändern, um überhaupt Parallelisierbarkeit zu ermöglichen. Insgesamt kann man tatsächlichen Bedarf, Nutzen und Kosten noch nicht abschätzen.

Alternativ zur Parallelisierung von Lernalgorithmen selbst, kann man auch die oftmals notwendige Kreuzvalidierung parallelisieren. Kreuzvalidierung wurde auf Seite 8 vorgestellt. Die Machine-Learning-Software *WEKA* [156], eine frei verfügbare Implementation von Methoden des maschinellen Lernens, stellt eine parallele Kreuzvalidierung für seine Klassifikatoren zur Verfügung [18].

Ganz aktuell gibt es Bemühungen, paralleles Data-Mining und maschinelles Lernen weiter voran zu treiben[37], wovon auch die SVM's profitieren werden.

Fazit: Data-Mining-Algorithmen verarbeiten eine große Anzahl von Daten, wobei das zu lernende Modell von allen zur Verfügung stehenden Daten abhängig sein sollte, denn das Entfernen von Punkten[38] zugunsten der Rechenzeit führt im Allgemeinen nicht zum Erfolg. Datenparallele Modelle führen meist nicht zum gewünschten Ergebnis oder erfordern spezielle Abschätzungen und Techniken, die eine gute Lösung garantieren. Die Parallelisierung wird dadurch erschwert [140].

6.1.2 Parallelität mit Support-Vektor-Maschinen

Support-Vektor-Maschinen haben in den letzten Jahren großen Zuspruch erfahren. Immer mehr Anwendungsgebiete mit großen Datenmengen werden aufgetan. Dazu gehören die Bioinformatik, die Medizininformatik, wissenschaftliche Datenanalyse, Finanzdatenanalyse, Telekommunikationsanwendungen und Marketing [72]. Große Datenmengen führen zu enormer Rechenintensität bei SVM's und so war abzusehen, dass bald erste Ideen zur Nutzung von SVM's auf Parallelrechnern entstehen würden. Effiziente und parallele Nutzung von SVM's stellt ein junges und dynamisches Forschungsgebiet dar. Zur Zeit ist die Entwicklung paralleler SVM's noch in ihren Anfängen, Implementierungen existieren kaum. Möglichkeiten des effizienten und parallelen Rechnens im Zusammenhang mit SVM's ergeben sich durch Methoden für:

1. verkürztes Training

   Algorithmen werden effizienter gestaltet, vereinfacht oder die Verkürzung der Rechenzeit ergibt sich durch geeignete Datenverkleinerung, z.B. mittels Undersampling, vergleiche Seite 107.

2. paralleles Training

   Das SVM-Training, d.h. die Lösung des quadratischen Optimierungsproblems findet parallel statt. Das kann mittels paralleler numerischer Algorithmen, aber auch datenparallel mittels einer Heuristik passieren, welche Daten auf die Prozessoren aufteilt, auf denen dann unabhängig seriell trainiert wird. Im Anschluss muss ein Schema zur Erstellung des finalen Modells existieren.

[37] ECML PKDD 2006 Workshop on Parallel Data Mining

[38] von der Ausreißereliminierung abgesehen

3. paralleles Mehrklassen-Training

   Ein Mehrklassenproblem läßt sich relativ leicht über die Definition mehrerer binärer SVM's behandeln [154]. Diese binären SVM's sind dann völlig unabhängig voneinander und können gleichzeitig trainiert werden. Die finale Klassifikation wird dann über eine Mehrheitsentscheidung vorgenommen.

4. parallele Validierung

   Bei der Kreuzvalidierung sind die anfallenden Aufgaben ebenfalls unabhängig voneinander, vergleiche Abbildung 2.1. Parallelisierung ist möglich, wobei maximal v CPU's in Frage kommen. Kommunikation ist notwendig, um ein Gesamtergebnis der Validierung zu berechnen.

5. parallele Parameteroptimierung

   Bei der SVM-Parameteroptimierung werden mehrere Parameterkombinationen getestet. Dabei liegt jedem Test mindestens ein SVM-Training zugrunde. Nichtiterative Prozeduren, wie beispielsweise die Gittersuche, eignen sich besonders gut für eine Parallelisierung.

Diese Methoden könnten auch gekoppelt eingesetzt werden, siehe dazu Abbildung 6.2. Für unsere Zwecke kommen die Ansätze 2.,4. und 5. in Frage.

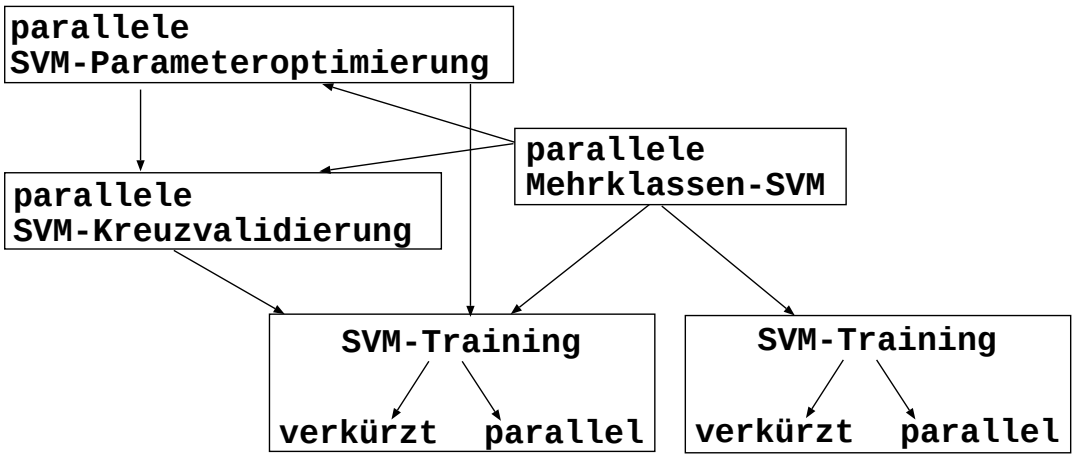

Abbildung 6.2: Schema zu möglichen Anknüpfungspunkten bei der SVM-Parallelisierung.

Viele der kürzlich publizierten parallelen SVM-Methoden erscheinen auf den ersten Blick neu und interessant, jedoch fällt bei näherer Prüfung auf, dass oft auf mehr oder weniger einfache Methoden zur Datenaufteilung zurückgegriffen wird. Die bekanntesten Methoden werden im Folgenden dargestellt.

- In [159] wird eine Methode zur Handhabung großer Datensätze beschrieben. Dabei wird zunächst ein Cluster-Verfahren benutzt, um interessante Trainingsdaten zu identifizieren, die den Lerneffekt der finalen SVM maximieren. Im Allgemeinen sollten

das Punkte mit kleiner Marge sein. Diese werden im Anschluss für das tatsächliche SVM-Training herangezogen. Die Methode kann leider nur für lineare Kerne verwendet werden und ist somit wenig interessant.

- [121] stellt eine Methode zur parallelen Berechnung der Grammatrix auf Distributed-Memory-Systemen vor. Leider zeigt sich, dass der erreichbare Speedup wegen der hohen Kommunikationskosten nicht vielversprechend ist. Als Ausweg wird in dieser Arbeit eine dünnbesetzte Matrix vorgestellt, welche die Kernmatrix für den Fall des Gauß-Kerns approximiert. Bei Nutzung der Approximation verringern sich die Kommunikationskosten enorm und führen zu guten Speedup-Werten. Leider geht die Approximation wie zu erwarten zu Lasten der Klassifikationsgüte und ist zudem zeitaufwändig.

- In [32] wird ein Trainingsverfahren für mehrklassige SVM's vorgestellt. Dabei wird ebenfalls ein Vorverarbeitungsschritt durchgeführt. Die Grammatrix für das Gesamtproblem wird mittels einer Blockdiagonalmatrix approximiert und in einem parallelen Optimierungsschritt wird für jeden Einzelblock ein Training durchgeführt. Dabei werden jeweils die Support-Vektoren gesammelt und stehen am Ende für das eigentliche Training, welches aber seriell durchgeführt wird, zur Verfügung. Die Approximation wird nicht beschrieben und auch die Behandlung der linearen Nebenbedingung bleibt offen.

- Ein parallelisierbarer SVM-Algorithmus für große Datensätze wird in [24] vorgestellt. Die Grundidee ist, mehrere SVM's zu trainieren, wobei jede SVM einen kleinen Teil der Daten zugeordnet bekommt. Nach den unabhängigen Trainingsphasen wird eine Kostenfunktion auf dem Gesamtdatensatz minimiert und die verschiedenen SVM-Modelle agieren als eine Art Expertensystem. Die unabhängigen Trainingsphasen könnten dann parallel durchgeführt werden.

- Die *Cascade SVM*, die in [54] als parallele Support-Vektor-Maschine vorgestellt wird, ist den drei schon erwähnten Methoden [24, 32, 159] grundsätzlich sehr ähnlich. Die Trainingsdaten werden in mehrere Gruppen aufgeteilt und für jede Gruppe wird eine SVM trainiert. Darauf folgen dann mehrere Schichten, in denen die Ergebnisse ausgewertet werden, siehe Abbildung 6.3. Das Ziel dabei ist auch hier, möglichst alle Support-Vektoren für das eigentliche Training zur Verfügung zu stellen und alle anderen Punkte herauszufiltern. Da die Anzahl der Support-Vektoren im Allgemeinen viel kleiner ist, als die Gesamtpunktzahl, erscheint die Idee sinnvoll. Allerdings ist das Verfahren unbrauchbar für große Datensätze mit vielen Support-Vektoren, da der eigentliche Trainingsalgorithmus wie so oft nicht parallel ist. Viele Methoden versuchen, unwichtige Punkte zu entfernen, und führen dann doch ein serielles Training durch. Die meisten Veröffentlichungen, die zur Zeit zu parallelen SVM's existieren, stellen derartige Systeme vor. Die *Cascade SVM* hat einen weiteren Nachteil. Wie man in Abbildung 6.3 sehen kann, nimmt die Zahl an CPU's

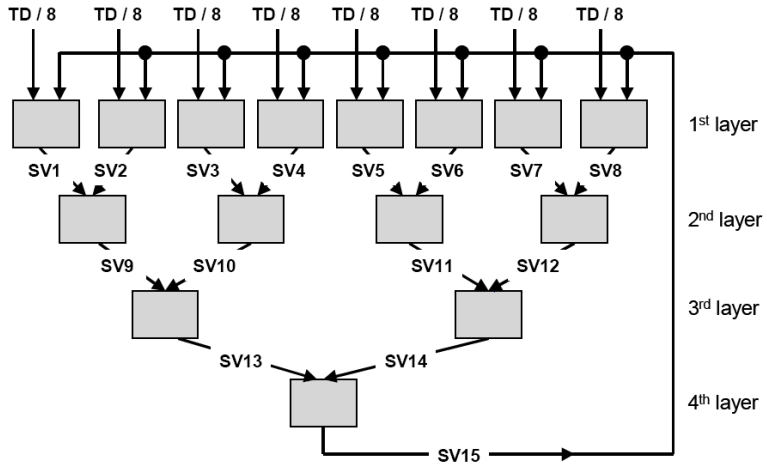

Abbildung 6.3: Struktur der *Cascade SVM* (Bild kopiert aus [54]).

ab (von 8 auf 1). Die Layer sind abhängig voneinander, sodass in jedem Schritt tatsächlich nur die Hälfte der CPU's Arbeit bekommt.

- In [119] werden zwei parallelisierbare SVM-Algorithmen vorgestellt, die sich gut für Datensätze eignen, die in genau einer Dimension sehr groß sind. Das kann entweder die Anzahl von Trainingspunkten oder die Anzahl von Attributen sein.[39] Die Parallelisierbarkeit wird jeweils durch die Umformulierung des SVM-Optimierungsproblems erreicht. Es entsteht ein lineares Gleichungssystem, für dessen Lösung existierende parallele Algorithmen eingesetzt werden können. Der Nachteil beider Methoden ist, dass sie ausschließlich für lineare SVM's geeignet sind. Die in der Arbeit angekündigte Version für nichtlineare SVM's ist bisher nicht erschienen. Das vorgeschlagene Schema arbeitet mit dem primalen Optimierungsproblem im Merkmalsraum und läßt sich offensichtlich nicht trivial auf das duale Problem übertragen. Sollte die Merkmalsabbildung bekannt sein, könnte man einen der Algorithmen verwenden. Im Allgemeinen kennt man die Daten im Merkmalsraum aber nicht. Genau aus dem Grund sind wir auf nichtlineare kern-basierte SVM's angewiesen.

- Eine MPI-basierte Software für echtes paralleles SVM-Training mit guten Skalierungseigenschaften wurde kürzlich in [164] vorgestellt. Sie basiert auf dem Zerlegungsalgorithmus und dem in [27] vorgestellten QP-Löser von Dai und Fletcher. Die Parallelität entsteht unter anderem durch verteilte Berechnung des Gradienten der Zielfunktion im Zerlegungsverfahren. Wir werden im Abschnitt 6.2 nochmals auf diese Software zu sprechen kommen.

- Eine Methode zur parallelen Parameteroptimierung von SVM's wurde in [129] vorgestellt. Basierend auf der Erkenntnis, dass das Parameterfittingproblem bei SVM's zu enormen Rechenzeiten führt, wurde eine parallele Methode dafür entwickelt. Sie

[39] Letzteres zum Beispiel bei Microarray-Daten mit mehreren tausend Variablen

basiert auf einem evolutionären Algorithmus (EA). Evolutionäre Algorithmen [4] und andere gradientenfreie Optimierungsalgorithmen sind interessante Methoden im Hinblick auf SVM-Parameteroptimierung, da sie heuristische Maße optimieren können, die keinerlei Gradienteninformationen liefern. Wir haben uns mit diesem Aspekt in Kapitel 5 beschäftigt.

6.2 Dreistufig parallele SVM-Software

In diesem Abschnitt stellen wir unsere Implementierung einer parallelen Support-Vektor-Maschine vor. Dabei werden wir drei parallele Ebenen behandeln, die zusammen, aber auch in beliebigen Kombinationen verwendet werden können. Dieser Abschnitt basiert auf unseren Arbeiten [37, 39–42, 45–48].

6.2.1 JUMP-Cluster

Wir arbeiten auf dem IBM p690 Cluster JUMP (Juelich Multi Processor), welcher sich im Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich befindet. JUMP ist ein SMP-Cluster-System mit 41 Knoten zu je 32 Power4+ CPU's (1.7 GHz) und 128 GB Hauptspeicher. Die Peak-Performance der 1312 CPU's liegt bei etwa 9 TFlop/s [29]. Das in 2003 und 2004 eigens errichtete Gebäude ist in Abbildung 6.4 dargestellt. Insgesamt stehen $1000\mathrm{m}^2$ Fläche zur Verfügung. Eine frei tragende Deckenkonstruktion, $6500\mathrm{m}^3$ Rauminhalt sowie $80\mathrm{cm}$ Doppelboden sind weitere Merkmale des neuen Gebäudes. Pro Stunde kommt es in der Halle zu insgesamt 38 Luftwechseln. Eine Brandfrüherkennung sowie eine Argon-Löschanlage sind nur zwei der Vorkehrungen, die helfen sollen, den Rechner und das Gebäude vor Schaden zu schützen. Der Rechner ist in Abbildung 6.5 zu sehen.

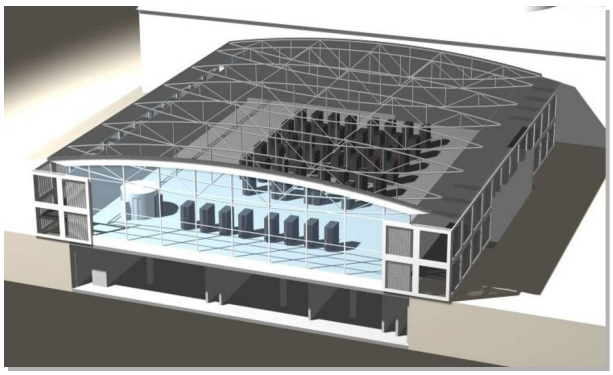

Abbildung 6.4: Neues Gebäude für den Rechner JUMP aus dem Jahr 2004.

Abbildung 6.5: Rechnerhalle am Zentralinstitut für Angewandte Mathematik.

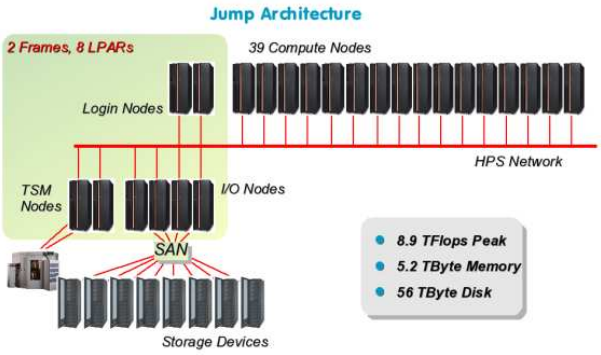

Abbildung 6.6: JUMP-Architektur.

6.2.2 Innere parallele Ebene: einzelne SVM-Trainingsphase

Das einzelne SVM-Training ist für große Datensätze sehr zeitintensiv [39]. In den letzten beiden Dekaden ist ein enormes Wachstum der elektronisch verfügbaren Daten zu verzeichnen – mit einer Verdopplung etwa alle 20 Monate, sodass effiziente und schnelle Algorithmen von enormer Wichtigkeit sind. Es bietet sich daher an, das SVM-Training zu parallelisieren.

Heutzutage gibt es immer mehr Parallelrechner mit globalem oder Multiprozessor-Shared-Memory. Für diese Maschinen existieren einige parallele Data-Mining-Algorithmen, wie z.B. Entscheidungsbäume [72]; parallele SVM-Trainingsverfahren existieren dafür bisher nicht. Einer der Gründe dafür könnte die enorme Popularität des SMO-Verfahrens [118] in der SVM-Community sein. Dieser Algorithmus arbeitet mit zwei aktiven Punkten pro Iteration, siehe Abschnitt 3.1.3. Ein einzelner Schritt ist dadurch sehr schnell, aber nicht parallelisierbar. Eine Parallelisierung kann demnach nur dadurch erfolgen, dass mehrere Iterationen gleichzeitig durchgeführt werden. Das verändert die Lösung und führt zu unerwünschten Effekten [89]. Die Parallelisierung beruht dabei auf einer Datenaufteilung, sodass mindestens zwei SVM's trainiert und deren Ergebnisse am Ende kombiniert werden. Für mehr als zwei Prozessoren gestaltet sich die Datenaufteilung extrem kompliziert,

wie in [89] dargestellt wurde. Es gab schon mehrere Bemühungen, SMO zu parallelisieren.[40] Bisher gibt es zwar keine Publikationen dazu, aber wir glauben, dass diese bald erscheinen werden. Wahrscheinlich wird die Parallelisierung darauf abzielen, Daten zu verteilen, mehrere SMO-Verfahren anzustoßen und zwischendurch Ergebnisse zu sammeln und auszuwerten.

Der in Kapitel 3 vorgestellte Zerlegungsalgorithmus mit den Modifikationen aus Kapitel 4 ist von uns optimiert und parallelisiert worden. Dafür sind zunächst alle Matrix- und Vektoroperationen durch Aufrufe von effizienten ESSL-Routinen ersetzt worden. Die ESSL-Bibliothek (Engineering Scientific Subroutine Library) von IBM [69] enthält BLAS-Routinen (Basic Linear Algebra Subroutines) [34, 92] und andere wichtige mathematische Operationen, die für Power4-Prozessoren optimiert sind. Die Bibliothek wird in den Routinen

- Zerlegung (siehe Abschnitt 3.2),
- Projektion (siehe Abschnitt 3.3) und
- innerer Löser (siehe Abschnitt 3.4)

verwendet. Dabei kommen hauptsächlich die Funktionen

- *DAXPY* ($\boldsymbol{y} = \alpha\boldsymbol{x} + \boldsymbol{y}$),
- *DCOPY* (kopiere $\boldsymbol{x}$ nach $\boldsymbol{y}$),
- *DDOT* (Skalarprodukt von $\boldsymbol{x}$ und $\boldsymbol{y}$),
- *DGEMV* ($\boldsymbol{y} = \beta\boldsymbol{y} + \alpha\boldsymbol{A}\boldsymbol{x}$),
- *DNORM2* (y ist euklidische Norm des Vektors $\boldsymbol{x}$),
- *DVEA* ($\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$),
- *DVES* ($\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y}$) und
- *DYAX* ($\boldsymbol{y} = \alpha\boldsymbol{x}$)

zum Einsatz [59].

In Abbildung 6.7 haben wir die grobe Struktur der Arbeitslast dargestellt. Die diversen Sortierverfahren arbeiten sehr effizient und verbrauchen nur wenig Zeit, wohingegen die Berechnung der Kernmatrizen und die ESSL-Routinen typischerweise sehr viel Zeit benötigen [41], insbesondere bei großen Datensätzen ist der prozentuale Anteil dieser Routinen an der Gesamtrechenzeit sehr hoch. Diese beiden Klassen werden primär für die Parallelisierung anvisiert. In Kapitel 7 werden wir zeigen, wie die Verteilung der Rechenzeit konkret aussieht.

Die ESSL-Routinen können durch Nutzung der parallelen ESSLSMP-Bibliothek, die auf dem Supercomputer JUMP [29] (vgl. Abschnitt 6.2.1) installiert ist, parallel ausgeführt

[40] persönliche Kommunikation (Fudan University, Cornell University)

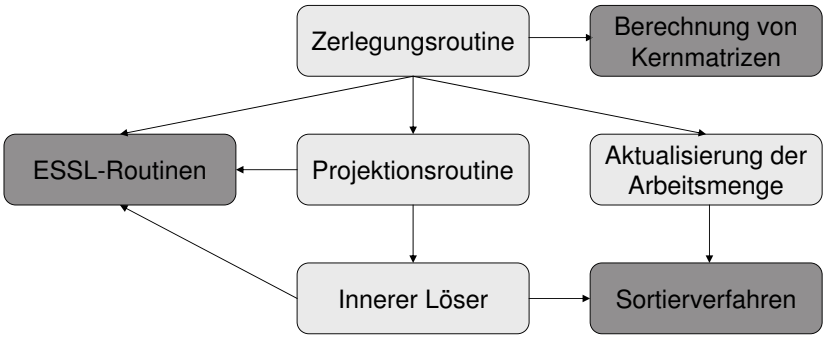

Abbildung 6.7: Grober Aufbau des SVM-Trainings (besonders rechenintensive Routinen grau markiert).

werden. Dazu muss der Quellcode nicht geändert, sondern lediglich das makefile angepasst werden mittels:

```
-qsmp=omp   -lesslsmp  .
```

Kernmatrizen werden in der Zerlegungsroutine berechnet. Wie auf Seite 36 in (3.11) dargestellt, werden für die Definition des Teilproblems innerhalb des Verfahrens der Zerlegung zwei Matrizen benötigt: $\boldsymbol{Q}_{\mathrm{dd}} \in \mathbb{R}^{\mathrm{ws,ws}}$ und $\boldsymbol{Q}_{\mathrm{fd}} \in \mathbb{R}^{\mathrm{rd-ws,ws}}$. Diese werden auch für das Update des Gradienten der Zielfunktion verwendet, siehe (3.27). Die Matrix $\boldsymbol{Q}_{\mathrm{dd}}$ berechnen wir in einer parallelen OpenMP[41]-Schleife. Die Berechnung und Speicherung der Matrix $\boldsymbol{Q}_{\mathrm{fd}}$ wird für große Datensätze zu einem Problem. Es gibt drei Möglichkeiten, dieses zu beheben.

- Es wird ausreichend Speicher allokiert und die Berechnung der Matrix erfolgt parallel. Die Matrix-Vektor-Produkte $\boldsymbol{Q}_{\mathrm{fd}}^{k} \cdot \left(\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} - \boldsymbol{\alpha}_{\mathrm{d}}^{k}\right)$ in (3.27) und $\left(\boldsymbol{Q}_{\mathrm{fd}}^{k}\right)^{T} \cdot \boldsymbol{\alpha}_{\mathrm{f}}^{k}$ können parallel mittels der Routine *DGEMV* berechnet werden.
- Mittels der Stripmining-Technik [58] wird iterativ jeweils nur ein Block der Matrix in passender Größe berechnet und verwendet.
- Die Berechnung und Speicherung der Matrix wird umgangen. Damit wird Speicherkapazität und Rechenzeit gespart. Notwendige Werte werden nach Bedarf parallel berechnet. Im Folgenden beschreiben wir die dafür notwendigen Änderungen an der Routine der Zerlegung. Die Ideen wurden inhaltlich übernommen aus [164], wo sie für den MPI-parallelen Modus beschrieben worden sind.

Bei der Berechnung des Gradienten mittels der Update-Formel (3.27) nutzen wir folgendes

[41] OpenMP ist eine plattformunabhängige Programmierschnittstelle für Shared-Memory-Parallelisierung.

Schema:

$$\begin{pmatrix} \boldsymbol{Q}_{\mathrm{dd}} \left(\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} - \boldsymbol{\alpha}_{\mathrm{d}}^{k}\right) \\ \boldsymbol{Q}_{\mathrm{fd}} \left(\boldsymbol{\alpha}_{\mathrm{d}}^{k+1} - \boldsymbol{\alpha}_{\mathrm{d}}^{k}\right) \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{ws} \mathbb{1}_{\alpha_{\mathrm{d}}^{k+1}(j) \neq \alpha_{\mathrm{d}}^{k}(j)} y_1 y_j k(\boldsymbol{x}^1, \boldsymbol{x}^j) \\ \cdots \\ \sum_{j=1}^{ws} \mathbb{1}_{\alpha_{\mathrm{d}}^{k+1}(j) \neq \alpha_{\mathrm{d}}^{k}(j)} y_{rd} y_j k(\boldsymbol{x}^{rd}, \boldsymbol{x}^j) \end{pmatrix} . \tag{6.1}$$

Hierbei wird die Tatsache ausgenutzt, dass sich bei einem Optimierungsschritt im Allgemeinen nur ein Teil der Einträge im Vektor $\boldsymbol{\alpha}_{\mathrm{d}}$ verändert. Mittels einer Schleife werden die Differenzen in jeder Dimension geprüft und nur diejenigen Summenglieder berechnet, für welche sich die Lösung verändert hat. Dafür definieren wir die Toleranzvariable ϵ_{∇}, welche wir typischerweise mit 0.0001 annehmen. Dabei kann man viele Kernberechnungen sparen. In Kapitel 7 werden wir zeigen, welche Zeiteinsparungen durch dieses abgewandelte Gradientenupdate möglich sind. Die Schleife wird im parallelen Modus ausgeführt.

Das zu berechnende Matrix-Vektor-Produkt $\left(\boldsymbol{Q}_{\mathrm{fd}}^{k}\right)^{T} \cdot \boldsymbol{\alpha}_{\mathrm{f}}^{k}$ kann ebenfalls umgangen werden, denn es gilt

$$\boldsymbol{Q}_{\mathrm{df}}^{k} \cdot \boldsymbol{\alpha}_{\mathrm{f}}^{k} - \mathbf{1} = \underbrace{\boldsymbol{Q}_{\mathrm{dd}}^{k} \cdot \boldsymbol{\alpha}_{\mathrm{d}}^{k} + \boldsymbol{Q}_{\mathrm{df}}^{k} \cdot \boldsymbol{\alpha}_{\mathrm{f}}^{k} - \mathbf{1}}_{\nabla W_{\mathrm{d}}^{-}(\boldsymbol{\alpha}^{k})} - \boldsymbol{Q}_{\mathrm{dd}}^{k} \cdot \boldsymbol{\alpha}_{\mathrm{d}}^{k} . \tag{6.2}$$

Das heißt, der Vektor $\boldsymbol{q}^{k}$ für die Definition des Teilproblems, siehe (3.62), wird aus dem schon berechneten Gradienten sowie einem relativ kleinen Matrix-Vektor-Produkt, welches zudem auch parallel mittels *DGEMV* berechnet werden kann, erzeugt. In Kapitel 7 werden wir die parallele Skalierbarkeit dieser Methode dokumentieren. Es sei noch erwähnt, dass unser paralleles Training ausschließlich auf einem Knoten des JUMP-Systems und mit maximal 32 CPU's durchgeführt werden kann. Knotengrenzen werden erst durch die anderen Ebenen überbrückt. Sie werden im Folgenden vorgestellt.

6.2.3 Mittlere parallele Ebene: Kreuzvalidierung

Die sich um das SVM-Training herum befindende Kreuzvalidierungsroutine wurde ebenfalls parallelisiert. Da die einzelnen Schritte bis auf die Berechnung des Endergebnisses in Form eines Gütemaßes vollständig unabhängig sind, haben wir uns für eine verteilte v-fache Kreuzvalidierung entschieden [37]. Die Trainings- und Testphasen werden nacheinander den zur Verfügung stehenden Prozessoren zugewiesen.

Mittels grobkörniger Parallelisierung mit MPI hat jede der c CPU's v/c Validierungsschritte auszuführen. Der Wurzelprozess liest, bearbeitet und verteilt die notwendigen Daten und Parameter mittels kollektiver Funktionen. Jeder Prozess speichert die Ergebnisse der lokalen Tests und stellt sie am Ende der Validierung der Wurzel zur Verfügung. Diese berechnet aus dem Gesamtergebnis das Gütemaß. Jeder Validierungsschritt besteht aus einem SVM-Training auf einer Datenmatrix mit n Variablen und $rd(1 - 1/v)$ Beispielen.

Die Trainingszeit ist abhängig von der Größe der Matrix und den Parameterwerten. Da sich die Parameter während einer einzelnen Validierung nicht ändern, können diese einzelnen Schritte mit etwa identischer Arbeitslast sehr einfach aufgeteilt werden.

Es ist zu beachten, dass die Anzahl c verwendeter CPU's sinnvollerweise höchstens v und c möglichst ein Teiler von v sein sollte. Andere Kombinationen führen zu schlechter Effizienz, sodass bei der Wahl der Validierungsmethode und der Anzahl von CPU's darauf geachtet werden sollte.

6.2.4 Äußere parallele Ebene: Parameteroptimierung

Im Abschnitt 5.2 haben wir die *APPSPACK*-Software vorgestellt, welche wir zur SVM-Parameteroptimierung einsetzen. Eine parallele Version dieser Software steht ebenfalls frei zur Verfügung [56]. In der seriellen Version werden alle Testpunkte hintereinander ausgewertet. Je nach Güte des letzten Parametertupels wird eine neue Parameterkombination bestimmt und getestet. In der MPI-parallelen Version werden die Prozesse aufgespalten in einen sogenannten Master und w Worker [55]. Der Master definiert in einem asynchronen Modus neue Testpunkte und gibt sie an freie Worker weiter. Jeder Worker nutzt seine eigenen Ressourcen, um die Zielfunktion für seine Testpunkte auszuwerten. Die Ergebnisse teilt er dem Master mittels MPI-Kommunikation mit, siehe dazu Abbildung 6.8. In der seriellen Version übernimmt ein Prozess die Aufgabe des Masters und eines Workers, d.h. die Testpunkte werden direkt nach Ihrer Generierung ausgewertet.

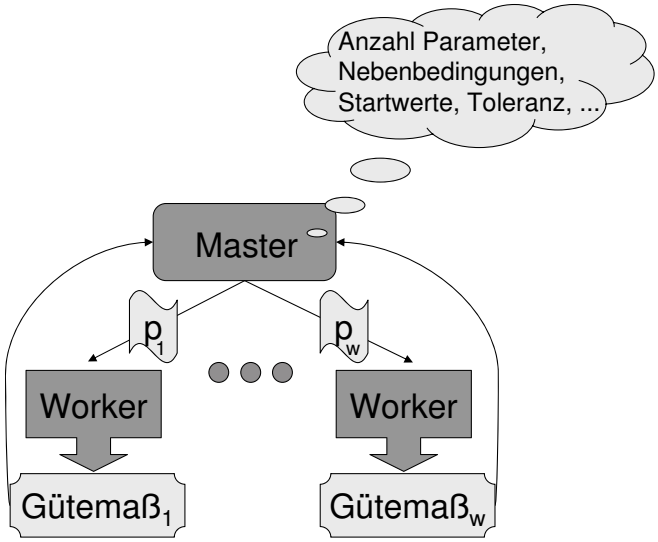

Abbildung 6.8: Paralleles Master-Worker-Schema der *APPSPACK*-Software.

Die Benutzung der Software erfolgt analog zum seriellen Fall. Eine `.apps` Datei mit den notwendigen Informationen wird angelegt und das gewünschte ausführbare Programm

muss verfügbar sein. Die MPI-parallele Version von *APPSPACK* wird beim Compilieren des Quellcodes zusammen mit der seriellen Version erstellt. In [40] haben wir die parallele *APPSPACK*-Software erstmals erfolgreich für das SVM-Parameterfitting eingesetzt. Dort wurde zunächst die serielle Support-Vektor-Maschine verwendet.

6.2.5 Kopplung der inneren und mittleren parallelen Ebenen

Die innere und mittlere parallele Ebene sind zu einer hybrid-parallelen Software zusammengefügt worden, um die Möglichkeiten der parallelen Nutzung zu verbessern [37]. Die äußere parallele Routine stellt die verteilte Validierung zur Verfügung und kann auch über einzelne Knoten hinweg eingesetzt werden. Der maximal erreichbare Speedup ist begrenzt auf v. Die restlichen CPU's können für das innere parallele Training verwendet werden, siehe Abbildung 6.9. Jeder Prozess nutzt die gleiche Anzahl an Threads für das Training, sodass jeder Validierungsschritt ähnliche Speedup-Werte aufweist. Uns steht nun eine flexible parallele SVM-Software für High-End-Maschinen mit SMP-Architektur zur Verfügung, mit der große Klassifikationsanwendungen aus der Bioinformatik oder anderen Gebieten in kurzer Zeit durchlaufen werden können.

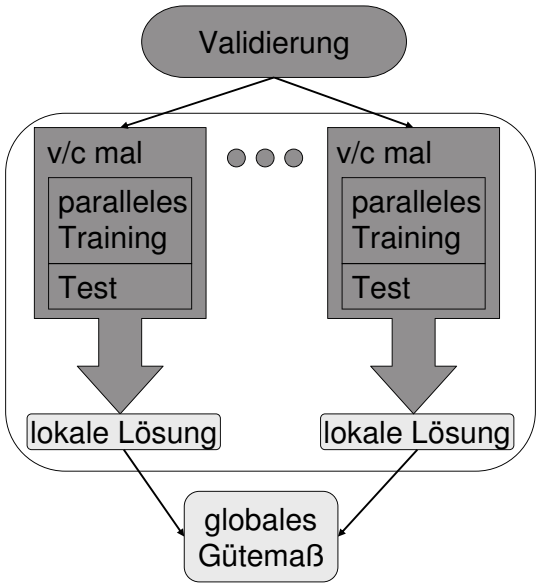

Abbildung 6.9: Hybrid-paralleles Validierungsschema der SVM-Software.

Der zusätzliche Speedup ist besonders interessant bei der Parametersuche, bei der sehr viele Validierungen stattfinden. Abgesehen von der einfachen Gittersuche wird Parametersuche meist iterativ ausgeführt, sodass sich die Parallelisierung eines Optimierungsschrittes anbietet. In unseren Tests in Kapitel 7 werden wir darauf eingehen.

6.2.6 Kopplung aller parallelen Ebenen

Die hybrid-parallele SVM-Software zeigt gute Performance auf dem SMP-Cluster JUMP (vgl. Kapitel 7). Der erreichbare Speedup ist jedoch aufgrund folgender Eigenschaften beschränkt:

- Die seriellen Anteile des Trainings beschränken den erreichbaren Speedup auf etwa 8 [37]. Größere Daten und Arbeitsmengen verbessern die Skalierbarkeit, da die parallelen Anteile größer werden, allerdings skalieren OpenMP-Anwendungen auf dem JUMP-System erfahrungsgemäß nur für wenige Threads zufriedenstellend, sodass wir ein einzelnes Training nicht mit mehr als 8 Threads durchführen.
- Die Anzahl der CPU's für die verteilte Validierung ist beschränkt auf v. Wir arbeiten oft mit $v = 8$.

Wir schlussfolgern, dass es auf Clustern von SMP-Knoten mit hunderten CPU's Freiheitsgrade für eine weitere parallele Ebene gibt. In diesem Zusammenhang streben wir eine dritte parallele Ebene für die SVM-Software an. Diese soll die parallele *APPSPACK*-Software enthalten. Unserer Meinung nach gibt es noch einen weiteren positiven Effekt der Nutzung von *APPSPACK* mit unserer parallelen Software. Wenn man die Gesamtzahl an CPU's, die zur Verfügung stehen, nicht ausschließlich für *APPSPACK*, sondern gemischt für beide parallele Pakete verwendet, kommt es zu folgendem Szenario. Jede SVM-Validierung ist schneller beendet als im seriellen Fall. Dadurch bekommt der *APPSPACK*-Master die Ergebnisse in kürzeren Intervallen gemeldet und kann geeignete Suchrichtungen zur Generierung neuer Testpunkte besser zuordnen. Die Anzahl der Worker ist zwar deutlich kleiner, führt aber keinesfalls zu Nachteilen, da die Worker besser ausgelastet sein werden als zuvor. Wir werden unsere Behauptung in Kapitel 7 belegen.

Die Zusammenführung der parallelen Ebenen ist nicht trivial. *APPSPACK* ruft ein externes Programm auf. Der Aufruf eines parallelen Programms, bei dem neue Ressourcen benötigt werden, würde durch den Scheduler der Maschine geblockt, bis neue Ressourcen frei werden. Dieses Szenario ist zumindest auf dem von uns verwendeten JUMP-System nicht realisierbar. Beide Pakete müssen modifiziert werden. In diesem Abschnitt beschreiben wir die durchgeführten Anpassungen zur Zusammenführung der drei parallelen Ebenen.

Die *APPSPACK*-Software muss in zweierlei Hinsicht modifiziert werden.

- Der externe Programmaufruf muss abgestellt werden. Die Zielfunktion (das Gütemaß der Support-Vektor-Maschine) muss direkt im Programm enthalten sein. Dazu muss ein neuer *APPSPACK evaluator* geschrieben werden.
- Um das parallele Schema in *APPSPACK* zu ändern, muss zunächst ein neuer *executor* implementiert werden, da das *APPSPACK GCI Interface* zu MPI nicht mehr eingesetzt werden kann.

Wie diese Änderungen durchgeführt werden müssen, wurde kürzlich in [56] beschrieben. In der Version 4.0.2. von *APPSPACK* (Dezember 2005) wird dafür auch ein spezielles

custom example zur Verfügung gestellt, welches einen neuen *evaluator* und einen neuen *executor* mit den gewünschten Eigenschaften enthält [56]. Es wird eine interne Funktion ausgewertet und die MPI-Funktionen sind direkt in den Routinen ohne Umweg über ein Interface implementiert.[42] Folgende Arbeiten standen noch an:

- Die interne C++ Beispielfunktion muss ersetzt werden durch den Aufruf unserer Fortran-Routine.
- Das vorhandene MPI-Konstrukt musste so geändert werden, dass MPI-Prozesse auf zwei Ebenen – in *APPSPACK* und in SVM – zusammenarbeiten und dass beide Ebenen kommunizieren können. Dazu wurden die CPU's in zwei Klassen eingeteilt: echte Worker und jeweils zugeordnete Slaves, die an der SVM-Parallelität beteiligt sind und keine eigenen *APPSPACK*-Testpunkte erhalten dürfen. Das neue Schema ist in Abbildung 6.10 dargestellt. Es ist zu beachten, dass während der Laufzeit der SVM jeder Worker mit als Slave agiert und somit immer beschäftigt ist, im Gegensatz zum Master, der nicht beteiligt ist.

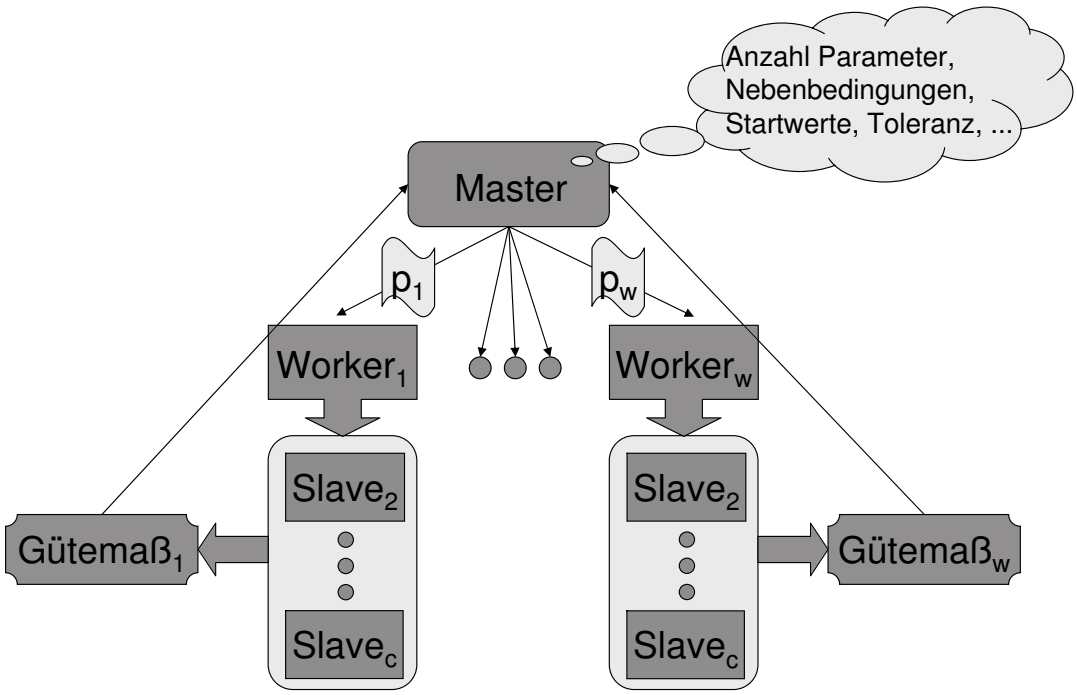

Abbildung 6.10: Neues Schema für das Zusammenspiel von *APPSPACK* und SVM.

Im Anhang B beschreiben wir die von uns durchgeführten Änderungen am Source-Code der beiden Pakete, welche eine gekoppelt parallele Nutzung ermöglichen. Zusätzlich er-

[42] Möglicherweise haben unsere Nachfragen dazu beigetragen.

klären wir anhand eines Loadleveler-Scripts die Nutzung des Supercomputers und gehen auf eine Schwachstelle unserer Implementierung ein.

6.3 Zusammenfassung

In diesem Kapitel haben wir unsere dreistufig parallele SVM-Software vorgestellt. Mittels eines Überblicks zu parallelen Data-Mining- und speziell SVM-Methoden haben wir gezeigt, dass es auf dem Gebiet paralleler SVM's noch Handlungsbedarf gibt. Im Anschluss wurden die drei parallelen Ebenen vorgestellt. Sie beinhalten paralleles SVM-Training, verteilte Kreuzvalidierung und parallele Parameteroptimierung. Letztere Funktionalität wird durch das Programm *APPSPACK* zur Verfügung gestellt. Wir haben die Kopplung der Ebenen erklärt, wobei die Hauptarbeit darin bestand, das MPI-Schema von *APPSPACK* zu verändern, vgl. Anhang B. Unsere parallele SVM-*APPSPACK*-Software stellt einen weiteren Schritt bei der Entwicklung paralleler Data-Mining-Methoden dar. Im nächsten Kapitel werden wir Testergebnisse für unser Paket präsentieren.

Kapitel 7

Tests

In den Kapiteln 3 bis 6 wurden die Grundlagen und Erweiterungen unserer SVM-Implementierung vorgestellt. Ausgehend von einem Zerlegungsalgorithmus für ein SVM-Optimierungsproblem haben wir Modellerweiterungen zur Verbesserung der Ergebnisse und der Flexibilität vorgeschlagen. Im Anschluss daran wurden die Parameteroptimierung sowie unsere neue dreistufige Parallelisierung besprochen. Dieses Kapitel soll nun dazu diesen, exemplarisch Testergebnisse vorzustellen, welche die Flexibilität, Genauigkeit und Geschwindigkeit der Software dokumentieren. Die Tests sollen nicht dazu diesen, zu zeigen, dass SVM's zur Datenklassifikation geeignet sind. Es soll auch nicht mit anderen Methoden verglichen werden. Derartige Arbeiten sind schon so zahlreich erschienen, dass wir diese Aspekte bewußt ausblenden.

7.1 Datensätze

In diesem Kapitel stellen wir Testergebnisse für ausgewählte Datensätze vor. Diese Daten sind frei verfügbar, die Quellen sind im Folgenden mit angegeben. Stellvertretend für das dieser Arbeit nah stehende Industrieprojekt [85], wurden auch zwei projektrelevante Datensätze ausgewählt. Das war möglich, da zu diesen Datensätzen freigegebene Publikationen existieren [38, 82, 83]. Andere Projektdaten können in dieser Arbeit nicht verwendet werden. Wir beschreiben zunächst alle von uns verwendeten Datensätze.

7.1.1 Kleine Datensätze

Unter einem kleinen Datensatz verstehen wir eine Menge von bis zu 1000 Punkten.

7.1.1.1 Kreditkarten – *australian*

Der Datensatz einer Bank behandelt Kreditkarten-Anträge. Insgesamt stehen 690 Fälle zur Verfügung, von denen 307 positiv entschieden wurden [122]. Die Anzahl der Variablen ist 14, wobei eine Mischung von 4 binären, 4 diskreten und 6 reellwertigen Variablen vorliegt. Der Datensatz steht unter [19] zur Verfügung.

7.1.1.2 Brustkrebs - *cancer*

Der Datensatz stammt von der University of Wisconsin Hospitals (Madison) und wurde in [100] vorgestellt. Er ist verfügbar unter [63]. Die Anzahl der Punkte beträgt 699 bei einer Anzahl von 10 Variablen. Es handelt sich um Untersuchungen von Gewebeproben. Die Punkte gehören zu den Klassen bösartig oder gutartig. Die bösartige Klasse, von uns als positive Klasse deklariert, ist leicht unterbesetzt (34%). Kostensensitivität liegt vor, da falsch negative Punkte für die Patientinnen schwere Konsequenzen mit sich bringen. Die Variablen beschreiben Gewebemerkmale wie Größe, Gleichmäßigkeit der Zellgrößen und der Zellstrukturen oder Art der Randverwachsungen. Die Variable zur Speicherung der Datensatznummer haben wir entfernt.[43] Einige Missing Values in den Daten müssen in Kauf genommen werden. 16 Punkte sind davon betroffen. Wir haben diese Daten durch Mittelwerte ersetzt.[44]

7.1.1.3 Zytochrom-P450 2D6-Inhibitoren mit Ensemble-Eigenschaften – *ensemble*

Eine für die Medikamentenentwicklung äußerst wichtige Gruppe von Enzymen ist das Zytochrom-P450-System (CYP450). In der Leber des Menschen katalysieren diese Enzyme den Metabolismus von etwa 90% aller Medikamente [152] und sind für Wirkungen und Nebenwirkungen verantwortlich. Eine aktuelle Anwendung betrifft die CYP450-Inhibitor Klassifikation [87, 88, 145, 158]. Die interessierende Klasse besteht aus Wirkstoffen, welche hemmend wirken. Diese sollten möglichst nicht übersehen werden, ihr Anteil in den Daten ist jedoch klein. Der Datensatz, der in [38] ausführlich beschrieben ist, entstammt dem GALA-Projekt und ist sehr kostensensitiv. Er besteht aus 263 Strukturen mit je 557 Variablen (*ensemble features*). Die positive Klasse der 2D6-Inhibitoren ist mit etwa 18% unterbesetzt.

7.1.1.4 Diabetes – *diabetes*

Der Pima Indian Diabetes Datensatz aus dem Jahr 1990 stammt von der John Hopkins University und wurde von Vincent Sigilito (Applied Physics Laboratory) zur Verfügung

[43] Diese wird gelegentlich mit als Variable verwendet, wovon dringend abzuraten ist.

[44] Eine andere Version des Datensatzes streicht die Beobachtungen, sodass 683 Punkte übrig bleiben.

gestellt [17]. Die Daten sind unter [19] verfügbar. Sie enthalten 768 Beispiele mit 8 Variablen. Die Klassifikation unterscheidet zwischen Zeichen für eine Diabetes und gesunden Patienten. Die untersuchten Personen waren alle weiblich und älter als 21 Jahre. Die Anzahl der Schwangerschaften ist eine Variable in dem Datenmaterial. Die Klasse der positiven Diabetestestung macht etwa ein Drittel der Daten aus.

7.1.1.5 Künstliche Daten – *fourclass*

Der Datensatz wurde in [65] beschrieben. Er wurde künstlich erzeugt zum Testen von Klassifikatoren. In einem zweidimensionalen Raum liegen 862 Punkte aus vier Klassen. Die Klassen sind irregulär verteilt und weisen isolierte Regionen auf, die eine Klassifikation erschweren. Der Datensatz wurde transformiert zu einem binären Problem und ist in dieser Form unter [19] verfügbar.

7.1.2 Große Datensätze

Unter einem großen Datensatz verstehen wir eine Menge mit mehr als 1000 Punkten.

7.1.2.1 Einkommensvorhersage – *a9a*

Der sogenannte Adult-Datensatz, der unter [63] verfügbar ist, beinhaltet das Problem einer Einkommensvorhersage. Basierend auf 14 Merkmalen, von denen 8 binär und 6 reellwertig sind, soll entschieden werden, ob ein bestimmtes Einkommen (50000USD) überschritten wird oder nicht (binäres Klassifikationsproblem). Dabei wird auf Merkmale wie Alter, Geschlecht, Ausbildung, Familienstand und einige mehr zurückgegriffen. Der Datensatz besteht aus 48842 Beispielen mit etwa 25% positiven Punkten. Wir verwenden den Datensatz in der Form mit 32561 Trainings- und 16281 Testpunkten und 123 binären Variablen, die dadurch entstanden sind, dass die 6 reellwertigen Variablen in Quantile diskretisiert wurden [118]. Unter [19] kann man diese Form des Datensatzes unter dem Namen *a9a* herunterladen.

7.1.2.2 Astrophysik – *astro*

Der Datensatz, welcher unter [19] verfügbar ist stammt ursprünglich von Jan Conrad (Uppsala University, Schweden). Es handelt sich dabei um eine Astropartikel-Anwendung, wobei es keine weiteren dokumentierten Informationen gibt. Der Datensatz wurde als Beispiel in die Dokumentation [67] aufgenommen, da die Klassifikation mittels der LIBSVM-Software verbessert worden ist. Bei 4 Variablen beträgt die Anzahl der Trainingsdaten 3089. Zusätzlich steht ein Testdatensatz mit 4000 Punkten zur Verfügung. Die positiven Punkte machen bei diesem Datensatz etwa ein Drittel der Daten aus. Über die Kostenverteilung wissen wir aber nichts.

7.1.2.3 Zytochrom-P450 1A2-Inhibitoren mit Bindungseigenschaften – *bonds*

Wir verwenden einen weiteren Datensatz der CYP450-Klasse, der ebenfalls dem GALA-Projekt entstammt. In diesem Fall wird das Isoform 1A2 der P450-Superfamilie betrachtet. Der Datensatz ist viel größer als der *ensemble* Datensatz, da er für jede Struktur mehrere Feature-Vektoren enthält. Dazu wurden für alle Atome jeder Substanz eigene Feature-Vektoren (*atom features*) berechnet. Zusätzlich wurden Bindungseigenschaften (*bond features*) berechnet, die ebenfalls zu neuen Daten führten. Insgesamt beträgt die Anzahl der Daten 27478. Details zu dem Datensatz sind in [82] zu finden, wobei wir eine erweiterte Version[45] mit 40000 Trainingspunkten verwenden. In [82] ist auch die Variablenselektion beschrieben, die wir eingesetzt haben, um die Anzahl der Variablen auf 10 bzw. 50 zu reduzieren.

7.1.2.4 Schilddrüsenfunktion – *thyroid*

Der Datensatz besteht aus 7200 Punkten mit 21 Variablen. 15 Variablen sind binär, die restlichen 6 sind reellwertig. Der Datensatz ist verfügbar unter [63]. Er wurde erstmalig in [122] erwähnt. Die Klassifikationsaufgabe besteht darin, zu entscheiden, ob eine Schilddrüsenüberfunktion vorliegt (*hypothyroid*) oder nicht. Der Datensatz ist stark unausgeglichen, da 93% der Punkte zur Klasse *nicht hypothyroid* gehören. Bei diesem Datensatz wurden die unklaren Fälle (*subnormale Funktion*) mit in die kleine Klasse eingeordnet. Streng genommen eignet sich der Datensatz also auch für eine mehrklassige Analyse. Wir beschäftigen uns in dieser Arbeit mit dem uns vorliegenden binären Fall, was einer Klassifikation von *one versus all* für den Mehrklassenfall entspricht [154]. Das Klassifikationsproblem ist nicht nur unausgeglichen, sondern auch kostensensitiv, da falsch negative Vorhersagen kritischer sind als falsch positive. Das ist bei Anwendungen aus der Medizin häufig der Fall. Der Datensatz ist aufgeteilt in 3772 Trainings- und 3428 Testpunkte.

7.1.2.5 Autoindustrie – *vehicle*

Der Datensatz ist unter [19] verfügbar. Er stammt von einem LIBSVM-Nutzer aus Deutschland [67], der nicht genannt wird. Es handelt sich dabei um eine Anwendung der Autoindustrie, wobei es keine weiteren dokumentierten Informationen gibt.[46] Der Datensatz wurde als Beispiel in die Dokumentation [67] aufgenommen, da die Klassifikation mittels der LIBSVM-Software verbessert worden ist. Bei 22 Variablen beträgt die Anzahl der Trainingsdaten 1243. Zusätzlich steht ein kleiner Testdatensatz mit 41 Punkten zur Verfügung. Über die Kostenverteilung ist nichts bekannt.

[45] durch Sampling

[46] Bei Daten aus der Industrie ist diese Geheimhaltung notwendig.

7.1.2.6 Webseitenklassifikation – *w8a*

Wir untersuchen einen Datensatz zur Textklassifikation, der in [118] verwendet wurde und bei [19] unter dem Namen *w8a* verfügbar ist. Insgesamt 300 Variablen werden verwendet, um Inhalte von Webseiten zu beschreiben. Diese Seiten sollen dann einer von zwei Kategorien zugeordnet werden. Der Datensatz enthält 49749 Daten mit 1479 Treffern (3%). Zusätzlich gibt es 14951 Testdaten. Das Lernproblem ist stark unausgeglichen.

In Tabelle 7.1 sind alle relevanten Datensätze nochmals kurz aufgelistet.

Bemerkung 7.1. *Typischerweise werden wir in diesem Kapitel kaum Ergebnisse anderer Verfahren zum Vergleich heranziehen. Die gängige Praxis, die eigenen hochoptimierten Ergebnisse, mit unoptimierten Ergebnissen anderer Methoden zu vergleichen, erscheint sehr fragwürdig. Andererseits können wir auch nicht auf publizierte optimierte Ergebnisse zurückgreifen, da die Aufbereitung der Daten und die konkrete Parameteroptimierung in den seltensten Fällen beschrieben ist und Nachfragen oft ergeben, dass auch auf Testdaten optimiert worden ist, beispielsweise wenn optimierte Validierungsergebnisse präsentiert werden.*[47]

7.2 Zerlegungsalgorithmus

In diesem Abschnitt sollen zunächst Eigenschaften des Zerlegungsalgorithmus untersucht werden. Wir werden einzelne Optimierungen des Verfahrens erklären. Diese liegen dann den folgenden Abschnitten zugrunde.

7.2.1 Kernfunktion

Die Kernberechnungen nehmen bekanntlich einen großen Teil der SVM-Rechenzeit in Anspruch. Im Abschnitt 4.2.3.1 haben wir neue Multiparameter-Kerne vorgestellt. Im Allgemeinen führen diese zu erhöhten Rechenzeiten. In diesem Testabschnitt soll untersucht werden, welche Kosten dadurch entstehen. Es stellt sich die Frage, wie man solche Kerne effizient implementieren kann. Reduziert man die Dauer einer einzelnen Kernberechnung, verringert man dadurch auch die Gesamtdauer des Trainings. Mit diesem Thema haben wir uns in [42, 45] beschäftigt. Am Beispiel des Gauß-Kerns werden wir in diesem Abschnitt eine Möglichkeit der Beschleunigung der Kernberechnung erklären. Testergebnisse werden zeigen, wie die konkreten Verbesserungen aussehen.

Zunächst untersuchen wir die Laufzeiten für den *a9a* Datensatz mit 15000 Punkten, jeweils für den einfachen und den Multiparameter-Gauß-Kern. Es wird eine Arbeitsmenge mit 650 Punkten verwendet. Die folgenden Laufzeiten wurden gemessen ($C = 1$, $\sigma = 3$):

[47] Dazu existiert eine sehr große Anzahl an Publikationen

	Training	Test
a9a (123 Variablen)		
- positive Punkte	7841	3846
- negative Punkte	24720	12435
astro (4 Variablen)		
- positive Punkte	1089	2000
- negative Punkte	2000	2000
australian (14 Variablen)		
- positive Punkte	228	79
- negative Punkte	272	111
bonds (10, 50 Variablen)		
- positive Punkte	2560	478
- negative Punkte	37440	7000
cancer (9 Variablen)		
- positive Punkte	123	118
- negative Punkte	227	231
diabetes (8 Variablen)		
- positive Punkte	203	65
- negative Punkte	397	103
ensemble (557 Variablen)		
- positive Punkte	37	11
- negative Punkte	163	52
fourclass (2 Variablen)		
- positive Punkte	262	45
- negative Punkte	438	117
thyroid (21 Variablen)		
- positive Punkte	284	250
- negative Punkte	3488	3178
vehicle (22 Variablen)		
- positive Punkte	296	41
- negative Punkte	947	0
w8a (300 Variablen)		
- positive Punkte	1479	454
- negative Punkte	48270	14497

Tabelle 7.1: Charakteristik der Datensätze für Validierung, Training und Test.

ws	Größe der Arbeitsmenge im Training
ϵ_∇	Schranke für das Gradientenupdate im Training (siehe S. 145)
t_{valid}	Validierungszeit (in Sekunden)
s_{valid}	Speedup bei der Validierung
fk_{valid}	Fehlklassifikationen bei der Validierung
fn_{valid}	falsch negative Validierungspunkte
fp_{valid}	falsch positive Validierungspunkte
$\#Z$	Anzahl der Zerlegungsschritte beim Training
$\#K$	Anzahl der Kernberechnungen beim Training (in Millionen)
sv	Anzahl der Support-Vektoren in den Trainingsdaten
sv_p	Anzahl positiver Support-Vektoren in den Trainingsdaten
sv_n	Anzahl negativer Support-Vektoren in den Trainingsdaten
Δ_b	absolute Schwellwertverschiebung nach dem Training (siehe Abschnitt 4.3.2)
fk_{train}	Fehlklassifikationen beim Training
fn_{train}	falsch negative Trainingspunkte
fp_{train}	falsch positive Trainingspunkte
t_{train}	Trainingszeit (in Sekunden)
t^c_{train}	Trainingszeit mit c Threads (in Sekunden)
s^c_{train}	Speedup im Training mit c Threads
fk_{test}	Fehlklassifikationen im Test
fn_{test}	falsch negative Testpunkte
fp_{test}	falsch positive Testpunkte
ac	Genauigkeit
se	Sensitivität
sp	Spezifität
pr	Präzision
ef	Anreicherungsfaktor
roc	ROC-Maß
fm	F-Maß
em	E-Maß

Tabelle 7.2: Wichtige Notationen für die Auswertung der Tabellen in den folgenden Abschnitten.

- 112 Sekunden (einfacher Kern),
- 177 Sekunden (Multiparameter-Kern).

Der Multiparameter-Kern verbraucht in diesem Beispiel 70% mehr Rechenzeit als der Standardkern. Der Grund dafür sind die Divisionsoperationen, die sich bei diesem Kern nicht mehr außerhalb sondern innerhalb der Summe befinden, vgl. (4.6). Zur Effizienzsteigerung des Rechnens schlagen wir die folgenden Transformationen vor. Für den Gauß-Kern gilt

$$k^G(\boldsymbol{x}, \boldsymbol{z}) := \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2}\right) = \exp\left(-\sum_{k=1}^{n}\left(\frac{x_k - z_k}{\sqrt{2}\sigma}\right)^2\right) . \tag{7.1}$$

Mittels einer Datentransformation

$$t(\boldsymbol{x}) := \boldsymbol{x}/\sqrt{2}\sigma$$

reduziert sich die Darstellung auf

$$k^G(\boldsymbol{x}, \boldsymbol{z}) := \exp\left(-\|t(\boldsymbol{x}) - t(\boldsymbol{z})\|\right) . \tag{7.2}$$

Analog kann man für den Multiparameter-Kern vorgehen. Dieser war definiert als

$$k_*^G(\boldsymbol{x}, \boldsymbol{z}) := \exp\left(-\sum_{k=1}^{n}\frac{(x_k - z_k)^2}{2\sigma_k^2}\right) . \tag{7.3}$$

Die Datentransformation

$$t_*(\boldsymbol{x}) := \left(x_1/\sqrt{2}\sigma_1, \ldots, x_n/\sqrt{2}\sigma_n\right)$$

reduziert die Darstellung auf

$$k_*^G(\boldsymbol{x}, \boldsymbol{z}) := \exp\left(-\|t_*(\boldsymbol{x}) - t_*(\boldsymbol{z})\|\right) . \tag{7.4}$$

Diese a priori Transformationen können also für beide Kerne verwendet werden und führen dann zu einem einheitlichen parameterfreien Gauß-Kern. Die Transformationszeiten sind vernachlässigbar [45] und es werden keine weiteren Ressourcen verbraucht, da wir die Matrix der Trainingsdaten direkt überschreiben. Wir haben die neue Implementierung getestet. Als Beispiel für obige Werte ergibt sich eine Trainingszeit von

- 105 Sekunden.

Im Vergleich zu 177 Sekunden für den Multiparameter-Kern ohne Transformation wird eine beachtliche Zeitersparnis bei gleichem Ergebnis erzielt. Für weitere Beispiele verweisen wir auf [42]. Im Folgenden verwenden wir wenn möglich nur diese sparsame Version des Kerns. Eine Option regelt den Einsatz. Bei Ensemble-Kernen (vgl. Abschnitt 4.2.3.2) ist diese a priori Transformation nicht möglich, da die Transformation die Trainingsdaten fest überschreibt. Unsere Software fängt diese Kombination ab.

7.2.2 Einsparung von Kernauswertungen

Auf Seite 145 hatten wir eine im Vergleich zur ursprünglich vorgestellten Version effizientere Methode des Zerlegungsalgorithmus vorgestellt, welche die Berechnung der gemischten Kernmatrix umgeht und ausschließlich die aktive Kernmatrix berechnet und speichert. Für große Datensätze ist das sehr wichtig, um mit dem vorhandenen Speicher auszukommen und auch große Arbeitsmengen verwenden zu können. In diesem Abschnitt soll für kleine und große Daten gezeigt werden, wie sich die Summe der Kernberechnungen mit beiden Methoden verhält. Dabei werden verschiedene Arbeitsmengen getestet. Gleichzeitig untersuchen wir die Veränderung der Trainingszeit. Für große Datensätze sollte eine deutliche Reduzierung der Rechenzeit sichtbar werden.

	Originalversion					**abgewandelte Version**				
ws	10	50	100	200	500	10	50	100	200	500
$\#K$	10.4	2.2	0.6	0.6	0.3	10.0	1.9	0.5	0.5	0.3
t_{train}	3.22	1.19	0.59	1.25	3.22	3.07	1.00	0.55	1.22	3.13

Tabelle 7.3: Untersuchung der ursprünglichen und modifizierten Zerlegungsmethode für verschiedene Größen der Arbeitsmenge beim Training (*australian* Datensatz, $C = 10$, $\sigma = 2$).

	Originalversion					**abgewandelte Version**				
ws	10	50	100	200	500	10	50	100	200	500
$\#K$	2.7	0.5	0.7	0.6	1.1	2.6	0.3	0.3	0.3	0.9
t_{train}	0.97	0.37	2.04	2.89	13.94	0.90	0.37	1.89	2.78	13.73

Tabelle 7.4: Untersuchung der ursprünglichen und modifizierten Zerlegungsmethode für verschiedene Größen der Arbeitsmenge beim Training (*fourclass* Datensatz, $C = 10$, $\sigma = 0.5$).

In den Tabellen 7.3 und 7.4 kann man erkennen, dass mit der neuen Version schon für kleine Datensätze Kernberechnungen eingespart werden. Die Trainingszeiten verändern sich ebenfalls, jedoch zunächst nur geringfügig. In Tabelle 7.5 haben wir zunächst einen Teil der *a9a* Daten verwendet. Bei diesem Training mit 15000 Punkten kann man schon deutlichere Effekte erkennen. Die Anzahl der Kernberechnungen nimmt für $ws > 100$ stark ab. Die Trainingszeiten reduzieren sich um bis zu 40%. Gute Trainingszeiten liegen bei beiden Methoden übrigens grob im Bereich $ws \in [500, 1000]$ und die Einsparungen mittels der neuen Methode sind dort auch sehr positiv. Eine genauere Analyse optimaler Arbeitsmengen folgt im Abschnitt 7.2.4. Wir zeigen nun, wie sich die abgewandelte Version der Zerlegungsmethode für große Datensätze verhält. Dazu betrachten wir den

vollständigen *a9a* Datensatz in Tabelle 7.6. Auch hier konnten wie erwartet bis zu 40% Zeit und Kernberechnungen eingespart werden.

	Originalversion					**abgewandelte Version**				
ws	50	100	500	1000	2000	50	100	500	1000	2000
$\#K$	1083	882	345	345	390	1082	866	275	214	228
gespart	–	–	–	–	–	0%	2%	20%	38%	42%
t_{train}	442	355	156	225	463	431	345	126	132	326
gespart	–	–	–	–	–	2%	3%	19%	41%	30%

Tabelle 7.5: Untersuchung der ursprünglichen und modifizierten Zerlegungsmethode für verschiedene Größen der Arbeitsmenge beim Training (*a9a* Datensatz mit 15000 Trainingspunkten, $C = 1$, $\sigma = 3$).

	Originalversion					**abgewandelte Version**				
ws	50	100	500	1000	2000	50	100	500	1000	2000
$\#K$	7046	5656	2328	1693	1758	5965	5043	1979	1175	1073
gespart	–	–	–	–	–	15%	11%	15%	31%	39%
t_{train}	3017	2430	1103	1162	1676	2480	2105	897	688	1041
gespart	–	–	–	–	–	18%	13%	18%	41%	38%

Tabelle 7.6: Untersuchung der ursprünglichen und modifizierten Zerlegungsmethode für verschiedene Größen der Arbeitsmenge beim Training (*a9a* Datensatz, $C = 1$, $\sigma = 3$).

In diesem Abschnitt haben wir gezeigt, dass die von uns implementierte effiziente Methode der Zerlegung für kleine, mittlere und große Datensätze geeignet ist. Während sich für kleine Daten die Trainingszeiten wenig ändern, lassen sich für größere Daten enorme Einsparungen erzielen. Diese Zeiteinsparungen sind deutlich korrelliert mit der Anzahl eingesparter Kernberechnungen. Im Folgenden wird sowohl im seriellen, als auch im parallelen Modus ausschließlich mit der modifizierten Form gerechnet, ohne dass darauf erneut eingegangen wird.

7.2.3 Gradient

An dieser Stelle wollen wir nochmals kurz auf die Implementierung des Gradientenupdates im modifizierten Zerlegungsalgorithmus eingehen. Der von uns verwendete Parameter ϵ_∇ für das Update des Gradienten (vgl. Seite 145) kann im Prinzip beliebig gewählt werden. Es ist auch möglich, die Abfrage völlig zu entfernen. Dabei spart man sich die vielen Abfragen im Programm. Um zu zeigen, warum wir $\epsilon_\nabla = 1e - 4$ verwenden, seien die folgenden Testreihen gegeben. Es werden exemplarisch die Fälle

- keine ϵ_∇-Abfrage,
- $\epsilon_\nabla = 1e-4$ und
- $\epsilon_\nabla = 1e-10$

gezeigt.

ws	10	20	50	80	100	120	150	200	350	500
keine ϵ_∇-Abfrage										
$\#Z$	2078	1191	78	16	11	8	7	6	4	1
$\#K$	10.6	12.4	2.2	0.7	0.7	0.6	0.7	0.8	1.2	0.5
sv	167	167	167	167	167	167	167	167	167	167
t_{train}	3.2	4.3	1.1	0.5	0.6	0.5	1.4	1.3	2.6	3.2
fk_{train}	44	44	44	44	44	44	44	44	44	44
fk_{test}	26	26	26	26	26	26	26	26	26	26
$\epsilon_\nabla = 1e-10$										
$\#Z$	2078	1191	78	16	11	8	7	6	4	1
$\#K$	10.6	12.3	2.0	0.6	0.5	0.5	0.5	0.5	0.7	0.3
sv	167	167	167	167	167	167	167	167	167	167
t_{train}	3.2	4.4	1.0	0.5	0.6	0.5	0.8	1.3	2.5	3.2
fk_{train}	44	44	44	44	44	44	44	44	44	44
fk_{test}	26	26	26	26	26	26	26	26	26	26
$\epsilon_\nabla = 1e-4$										
$\#Z$	2057	1153	73	16	11	8	7	6	4	1
$\#K$	10.0	11.4	1.9	0.6	0.5	0.5	0.5	0.5	0.7	0.3
sv	167	167	167	167	167	167	167	167	167	167
t_{train}	3.1	4.1	1.0	0.5	0.6	0.5	0.8	1.2	2.4	3.1
fk_{train}	44	44	44	44	44	44	44	44	44	44
fk_{test}	26	26	26	26	26	26	26	26	26	26

Tabelle 7.7: Untersuchung der modifizierten Zerlegungsmethode für unterschiedliche Werte von ϵ_∇ bei verschiedenen Größen der Arbeitsmenge (*australian* Datensatz, $C = 10$, $\sigma = 2$).

Für den kleinen *australian* Datensatz (Tabelle 7.7) ist die Wahl von $\epsilon_\nabla = 1e-4$ sinnvoll bezüglich Trainingszeit und Kernberechnungen, wobei die Unterschiede noch gering sind. Zusätzlich haben wir die Anzahl der Support-Vektoren sowie die Trainings- und Testfehler bestimmt. Diese sind stets gleich. Die Wahl von $\epsilon_\nabla = 1e-4$ schränkt das Modell also nicht ein.

Für den weitaus größeren *a9a* Datensatz mit 15000 Trainingspunkten lassen sich jedoch Unterschiede erkennen. Dazu haben wir in Tabelle 7.8 jeweils die Ersparnis an Kernberechnungen und Rechenzeit in Prozent (verglichen mit dem teuersten Modell) angegeben.

ws	20	50	100	300	500	650	700	800	1000	2000
keine ϵ_∇-Abfrage										
$\#Z$	3206	1538	633	107	46	32	30	27	23	13
$\#K$	963	1157	956	491	357	326	330	341	368	442
t_{train}	385	462	385	204	159	158	169	185	233	481
$\epsilon_\nabla = 1e-10$										
$\#Z$	3206	1506	633	107	46	32	30	27	23	13
$\#K$	923	1117	910	416	258	216	217	211	218	228
t_{train}	369	445	366	175	118	107	112	113	135	319
$\epsilon_\nabla = 1e-4$										
$\#Z$	3204	1540	640	105	51	32	30	28	23	13
$\#K$	887	1082	882	392	275	214	215	211	214	228
gespart	8%	6%	9%	20%	23%	34%	35%	38%	42%	48%
t_{train}	354	431	355	165	126	105	110	114	132	326
gespart	8%	7%	10%	19%	21%	34%	35%	38%	43%	32%

Tabelle 7.8: Untersuchung der modifizierten Zerlegungsmethode für unterschiedliche Werte von ϵ_∇ bei verschiedenen Größen der Arbeitsmenge (*a9a* Datensatz mit 15000 Trainingspunkten, $C = 1$, $\sigma = 3$).

Es ist deutlich zu sehen, wie sich die Ersparnisse fast identisch entwickeln. Bei $ws = 2000$ ist die Zeitersparnis etwas geringer als die Ersparnis an Kernberechnungen, was daran liegt, dass bei dieser Größe der innere Löser offensichtlich sehr teuer wird. Diesen Effekt konnte man auch schon im letzten Abschnitt sehen.

7.2.4 Größe der Arbeitsmenge

In den vorangegangenen Abschnitten haben wir gesehen, dass verschiedene Größen ws zu unterschiedlichen Rechenzeiten führen. In diesem Abschnitt soll dieser Effekt in Anlehnung an unsere Arbeit [46] näher beleuchtet werden. Es ist bekannt, dass große Arbeitsmengen zu wenigen langsamen und kleine Arbeitsmengen zu vielen schnellen Zerlegungsschritten führen. Die optimale Größe für einen Datensatz zu definieren, ist jedoch nicht einfach. Die Frage ist, bei welcher Größe beide Effekte zu minimaler Trainingszeit führen, denn nur die ist letztlich entscheidend für den Anwender. Die folgenden Ergebnisse sollen dazu dienen, zu zeigen, wie variierende Werte für ws die Anzahl der Zerlegungsschritte und die Trainingszeit beeinflussen. Dabei werden wie bisher kleine und große Datensätze getestet.

Für den *australian* Datensatz in Tabelle 7.9 nimmt die Anzahl der Zerlegungsschritte mit wachsendem ws ab; besonders stark bei den kleineren Größen. Ein weiterer interessanter Aspekt ist die Anzahl der Kernberechnungen. Für kleine Größen ist die Zahl sehr hoch

ws	4	10	20	50	80	100	120	150	200	350	400	500
$\#Z$	3645	2057	1153	73	16	11	8	7	6	4	4	1
$\#K$	7.02	10.03	11.43	1.86	0.63	0.54	0.47	0.48	0.54	0.71	0.86	0.33
sv	167	167	167	167	167	167	167	167	167	167	167	167
t_{train}	2.28	3.07	4.13	1.00	0.48	0.55	0.47	0.80	1.22	2.41	4.48	3.13

Tabelle 7.9: Einfluß der Größe der Arbeitsmenge auf Zerlegungsschritte, Kernberechnungen und Trainingszeit (*australian* Datensatz, $C = 10$, $\sigma = 2$).

und nimmt dann etwa ab 50 drastisch ab. Für $ws = 120$ ist die Anzahl minimal. Genau bei diesem Wert wurde auch das Minimum der Trainingszeit erreicht. Die Trainingszeit war für kleine Arbeitsmengen hoch, wurde dann im Bereich $[50, 150]$ kleiner und stieg dann wieder an. Längste und kürzeste Trainingszeit liegen immerhin mit Faktor sechs auseinander.

ws	4	10	20	30	50	100	200	300	400	500	600	700
$\#Z$	2698	374	33	24	14	10	4	3	5	3	2	1
$\#K$	7.54	2.60	2.21	0.31	0.25	0.31	0.29	0.39	0.94	0.86	0.81	0.55
sv	89	89	89	89	89	89	89	89	89	89	89	89
t_{train}	2.26	0.90	0.22	0.27	0.37	1.89	2.78	3.97	12.20	13.73	10.04	9.09

Tabelle 7.10: Einfluß der Größe der Arbeitsmenge auf Zerlegungsschritte, Kernberechnungen und Trainingszeit (*fourclass* Datensatz, $C = 10$, $\sigma = 0.5$).

Für den etwas größeren *fourclass* Datensatz ergibt sich das folgende Bild (vgl. Tabelle 7.10). Auch hier ist die Anzahl der Kernberechnungen zu Beginn sehr hoch und nimmt dann drastisch ab. Bei $ws = 50$ wurde der minimale Wert der Testreihe angenommen. Die Trainingszeit entwickelte sich jedoch diesmal etwas anders. Das Minimum wird schon für $ws = 20$ angenommen bei einer gleichzeitig eher hohen Anzahl an Kernberechnungen. Diese Anzahl allein ist also nicht ausreichend, um die Trainingszeit zu erklären. Möglicherweise spielt hier die Anzahl an Zerlegungsschritten auch eine wichtige Rolle. Diese hat schnell abgenommen. Da die Berechnungen des inneren Lösers für $ws = 20$ sehr schnell sind, ist die Kombination in dem Fall optimal. Man kann auch gut erkennen, dass sich größere Arbeitsmengen sehr negativ auf die Rechenzeit auswirken. Wir haben bei den Tabellen 7.9 und 7.10 jeweils die Anzahl der Support-Vektoren mit angegeben, um zu zeigen, dass die Variable ws das erlernte Modell nicht stark beeinflusst. Bei großen Datensätzen gibt es das Phänomen, dass sich die Anzahl der Support-Vektoren bei verschiedenen Werten für ws leicht unterscheidet ([46] und unsere Tabellen 7.11 und 7.12). Dabei liegt jedoch kein Konflikt mit der globalen Optimierungsaufgabe vor. Da es bei dem iterativen Zerlegungsalgorithmus eine Abbruchbedingung gibt, wird das Ergebnis nie genau ausgerechnet. Durch unterschiedliche Arbeitsmengen entstehen damit auch unter-

schiedliche Zwischenergebnisse und sobald die Karush-Kuhn-Tucker-Bedingungen in unseren Tests mit der jeweiligen Toleranz erfüllt sind, stoppt das Verfahren. Im Allgemeinen wirkten sich die Unterschiede im Nachhinein kaum auf die Testgenauigkeiten aus. In der Praxis wird übrigens häufig versucht, bei sehr großen komplexen Daten die Trainingszeit durch Wahl einer groben Toleranz zu verringern. Die dabei entstehenden Ungenauigkeiten werden zugunsten der geringeren Rechenzeit in Kauf genommen.

ws	10	20	50	100	300	500	650	700	800	1000	1400
$\#Z$	6076	3204	1540	640	105	51	32	30	28	23	18
$\#K$	806	887	1082	866	392	275	214	215	211	214	222
sv	5527	5547	5552	5532	5547	5538	5542	5553	5547	5545	5539
t_{train}	334	354	431	345	165	126	105	110	114	132	199
ws	1800	2000	2400	2800	3000	3100	3300	3500	3700	3850	4000
$\#Z$	15	13	12	10	9	9	9	8	8	7	7
$\#K$	229	228	245	244	243	249	258	255	268	252	261
sv	5532	5538	5537	5540	5546	5553	5549	5547	5556	5548	5555
t_{train}	260	326	528	603	582	669	750	856	1090	960	1140

Tabelle 7.11: Einfluß der Größe der Arbeitsmenge auf Zerlegungsschritte, Kernberechnungen und Trainingszeit (*a9a* Datensatz mit 15000 Trainingspunkten, $C = 1$, $\sigma = 3$).

In Tabelle 7.11 sind zunächst wieder Ergebnisse für den reduzierten *a9a* Datensatz mit 15000 Punkten gegeben. Die Anzahl der Kernberechnungen pendelt sich ab etwa $ws = 500$ bei 200 bis 300 Millionen ein. Wie man auch in der graphischen Darstellung der Trainingszeit (Abbildung 7.1) erkennen kann, liegt eine günstige Arbeitsmenge bei etwa 650 Punkten.[48] Ein allgemein guter Bereich ist $[300, 1000]$. Für größere (oder kleinere) Werte von ws steigt die Zeit immer weiter an.

Ergebnisse für den kompletten *a9a* Datensatz sind in Tabelle 7.12 gegeben. Bei mehr als 3000 aktiven Punkten steigt die Rechenzeit drastisch an. Im Unterschied zum Datensatz mit 15000 Punkten, liegt die optimale Arbeitsmenge jedoch im Bereich $[600, 1400]$ (vgl. Abbildung 7.1), das Intervall hat sich also verschoben. Das Optimum wird bei $ws = 1200$ erreicht. Wir schließen daraus, dass eine Vergrößerung des Datensatzes zu mehr Zerlegungsschritten führt, die dann in ihrer Summe dafür sorgen, dass größere Schritte geringerer Anzahl plötzlich im Vorteil sind. Die allgemeinen Trends in Tabelle 7.12 sind:

- Im Bereich kleiner Arbeitsmengen werden tausende Zerlegungsschritte durchgeführt. Die Anzahl sinkt dann drastisch hin zu einer zweistelligen Zahl.
- Die Trainingszeit als Funktion von ws ist nicht linear. Es existiert ein Bereich von

[48] Allgemein sind in diesem Kapitel bei allen Testreihen deutlich mehr Werte untersucht worden. Aus diesen sind dann interessante Fälle und Bereiche ausgewählt worden, um die Tabellen nicht weiter aufzublähen.

ws	50	100	300	500	600	700	800	900	1000	1200
$\#Z$	3920	1646	305	158	111	87	71	71	52	39
$\#K$	5965	5043	2611	1979	1610	1441	1343	1393	1175	984
sv	11772	11774	11763	11745	11755	11779	11756	11742	11765	11760
t_{train}	2480	2105	1118	897	759	718	716	773	688	626
ws	1400	1600	2000	2200	2400	2600	2800	3000	3500	4000
$\#Z$	34	34	29	23	22	21	20	20	17	15
$\#K$	985	1055	1073	978	992	998	1035	1054	1050	1078
sv	11767	11753	11775	11781	11771	11767	11761	11784	11784	11758
t_{train}	700	828	1041	1043	1114	1130	1400	1557	2379	3400

Tabelle 7.12: Einfluß der Größe der Arbeitsmenge auf Zerlegungsschritte und Trainingszeit (*a9a* Datensatz, $C = 1$, $\sigma = 3$).

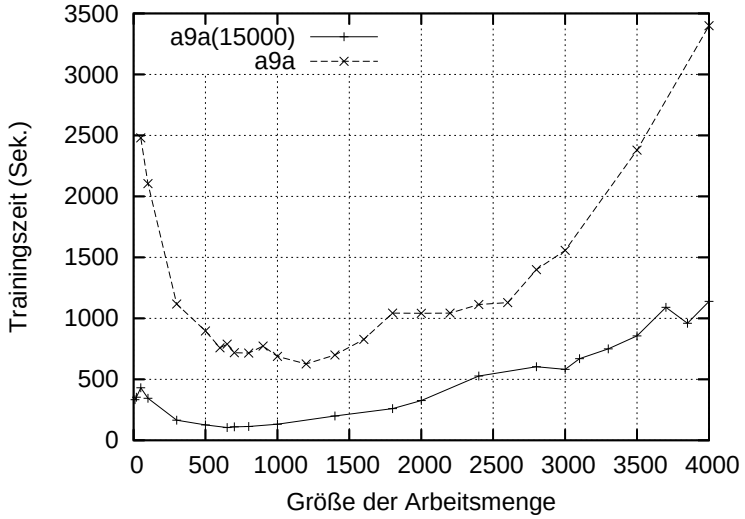

Abbildung 7.1: Graphische Darstellung der Trainingszeiten aus den Tabellen 7.11 und 7.12.

Arbeitsmengen, die zu besonders guten Ergebnissen führen. Dieser Bereich ist jedoch datenabhängig.

- Die Anzahl an Kernberechnungen pendelt sich auf einen Wert um etwa eine Milliarde ein. Die Anzahl der Kernberechnungen ist also im Bereich größerer Arbeitsmengen nicht für die steigende Rechenzeit verantwortlich. Diese wird demnach stark von den Kosten des QP-Lösers bestimmt.

Im Abschnitt 7.5.1 werden wir zeigen, wie auch das parallele Verhalten der Zerlegungsmethode von ws beeinflusst wird.

7.2.5 Profiling

In diesem Abschnitt wollen wir zusammenfassend zu den algorithmischen Tests zeigen, wieviel Zeit tatsächlich in den einzelnen Routinen verbraucht wird. Dazu führen wir, wie auch schon in [41], für verschiedene Datensätze ein Profiling mit dem GNU-Profiler *gprof* auf dem JUMP-System durch. Dabei wird die effiziente Form des Zerlegungsalgorithmus mit $\epsilon_\nabla = 1e-4$ und der a priori Datentransformation zur Kernberechnung eingesetzt.

Datensatz	*diabetes*	*astro*		*w8a*		
ws	10	10	100	100	500	1000
ESSL	0.01	0.35	0.76	1.91	49.93	76.28
Zerlegungsroutine	0.02	1.93	0.18	55.22	50.27	12.42
Kernberechnungen	0.13	10.14	0.52	1274.73	1310.33	344.18
Projektion	0.02	0.05	0.00	0.06	0.19	0.13
Innerer Löser	0.02	0.54	1.14	3.37	19.22	14.47
Sortieren	0.02	0.18	1.17	3.20	17.66	12.55
Aktualisierung der Arbeitsmenge	0.02	0.99	0.00	3.61	0.80	0.11
t_{train}	0.24	14.18	3.77	1342.10	1448.40	460.14

Tabelle 7.13: Profilingergebnisse (in Sekunden) für die Datensätze *diabetes*, *astro* und *w8a* (20000 Punkte), O3-Optimierung.

In Tabelle 7.13 zeigen wir beispielhaft Ergebnisse der Zeitmessungen für je einen kleinen, mittleren und großen Datensatz. Wir haben die Größe der Arbeitsmengen leicht variiert, um zu zeigen, wie sich die Verhältnisse der Rechenzeiten in den verschiedenen Routinen ändern. In der Tabelle sind jeweils nur ausschließlich die Zeiten angegeben, die netto durch diese Routinen verbraucht werden. Die Zeiten von inneren eigenständigen Routinen sind gesondert aufgeführt und wurden nicht kumuliert. Für eine bessere Übersicht stellen wir die Abhängigkeiten in Abbildung 7.2 nochmals dar.

Man kann erkennen, dass die Kernberechnungen in der Zerlegungsroutine immer einen signifikanten Teil der Rechenzeit verbrauchen. Für den *astro* Datensatz sieht man, dass eine Vergrößerung der aktiven Menge zu längeren Rechenzeiten des inneren Lösers und der darin enthaltenen Sortieralgorithmen führt. Gleichzeitig verringert sich die Rechenzeit bei den Kernauswertungen und führt dann insgesamt zur besseren Performance. Für den *w8a* Datensatz ist eine änliche Entwicklung zu erkennen. Für $ws = 1000$ kann viel Zeit bei den Kernen gespart werden. Da fallen die längeren Rechenzeiten bei den ESSL-Routinen kaum ins Gewicht.

Das Profiling hat nochmals gezeigt, wie unterschiedlich die Rechenzeiten für unterschiedlich große Arbeitsmengen sein können. Die Größe der Arbeitsmenge hat nicht nur großen Einfluß auf die Gesamtrechenzeit, sondern aber auch auf die prozentuale Verteilung der

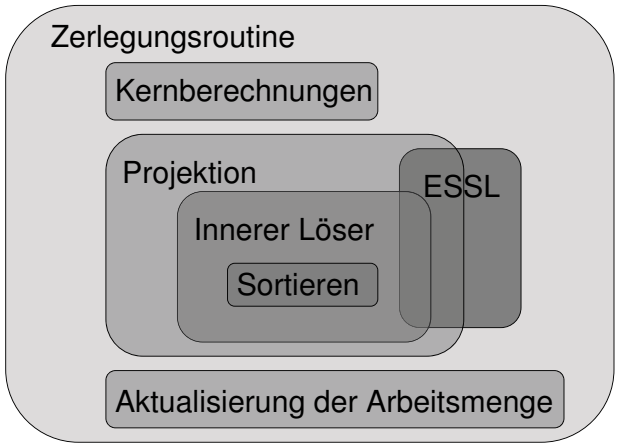

Abbildung 7.2: Grobe Struktur der Zerlegungsmethode zur Interpretation der Profiling-Ergebnisse in Tabelle 7.13.

Last. Durch den mehrschichtigen Aufbau der Zerlegungsmethode, siehe Abbildung 7.2, ist es schwer, genaue Prognosen zu den Rechenzeiten für neue Daten vorherzusagen, wenn die Größe der Arbeitsmenge noch unbekannt ist. Eine Idee für die Wahl der Arbeitsmenge ist, 10% der Trainingsdaten zu verwenden und dabei einen Maximalwert von beispielsweise 2000 Punkten nicht zu überschreiten. Wir verweisen an dieser Stelle auf Anhang C, wo wir zwei umfangreichere Testreihen zur Abhängigkeit der Trainingszeit von der Arbeitsmenge **und** der Datensatzgröße zeigen.

7.3 Kostensensitive Modellierung

In diesem Abschnitt werden Tests zu verbesserten Lernmethoden durchgeführt. Dabei werden Techniken, wie sie in Kapitel 4 vorgestellt wurden, eingesetzt, um Ergebnisse zu verbessern. Ein wichtiger Aspekt dieses Abschnitts ist kostensensitive Klassifikation. Thematisch wichtige Arbeiten dazu sind [40, 43, 44, 47] sowie die innerhalb des GALA-Projektes entstandenen Arbeiten [38, 82, 83, 132, 133].

7.3.1 Gewichtete SVM-Modelle

Wir studieren den Einfluss der Fehlergewichtung mittels C^+ und C^-, vergleiche dazu Abschnitt 4.2.2. Wir haben behauptet, dass sich durch ein Verhältnis $C^+/C^- \neq 1$ eine Veränderung der Fehlerquoten ergibt und die Fehlergewichtung somit ein Werkzeug zur kostensensitiven Klassifikation ist. Anhand einiger Tests werden wir zeigen, welchen Einfluss die Variierung von C^+ und C^- auf Trainings- und Testergebnisse hat. Bei den Ergebnissen handelt es sich nicht um hochoptimierte Werte, wir wollen an dieser Stelle

zunächst ein Gefühl dafür vermitteln, welche Möglichkeiten es gibt und wie stark Ergebnisse auseinanderdriften können. Mit der automatischen Parameteroptimierung befassen wir uns später im Abschnitt 7.4.

In Anlehnung an unsere Studien in [132] beginnen wir mit einer zweidimensionalen Darstellung von Ergebnissen für den *ensemble* Datensatz. Es wurden verschiedene Werte für C^+ und C^- verwendet. Alle anderen Parameter waren konstant (L_1-Norm-SVM, Gauß-Kern, $\sigma = 20$). Zunächst sind in Tabelle 7.14 die Ergebnisse des Trainings dargestellt, wobei die Ergebnisse für die ungewichteten Modelle fett eingetragen sind. An dieser Diagonale kann man wunderbar erkennen, wie mit wachsendem Strafparamter C die Trainingsfehler immer mehr abnehmen, bis es ab $C = 50$ keine Fehler mehr gibt. Durch eine höhere Bestrafung falsch negativer Punkte im gewichteten Modell (rechte obere Dreiecksmatrix) nimmt ihre Anzahl ab, gleichzeitig gibt es mehr falsch positive Punkte. Auffällig ist das starke Anwachsen dieser Fehler für kleine Werte von C^-.

		C^+								
		1	1.5	2	4	6	8	10	50	100
C^-	1	**29 : 0**	16 : 2	4 : 8	0 : 14	0 : 14	0 : 14	0 : 14	0 : 14	0 : 14
	1.5	31 : 0	**19 : 0**	10 : 1	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10	0 : 10
	2	31 : 0	25 : 0	**15 : 1**	0 : 7	0 : 9	0 : 9	0 : 9	0 : 9	0 : 9
	4	31 : 0	27 : 0	17 : 0	**4 : 1**	0 : 3	0 : 6	0 : 6	0 : 6	0 : 6
	6	31 : 0	27 : 0	19 : 0	5 : 0	**4 : 1**	0 : 2	0 : 4	0 : 5	0 : 5
	8	31 : 0	27 : 0	19 : 0	6 : 0	4 : 1	**2 : 1**	0 : 2	0 : 4	0 : 4
	10	31 : 0	27 : 0	19 : 0	6 : 0	4 : 0	3 : 0	**2 : 0**	0 : 3	0 : 3
	50	31 : 0	27 : 0	19 : 0	6 : 0	4 : 0	3 : 0	2 : 0	**0 : 0**	0 : 0
	100	31 : 0	27 : 0	19 : 0	6 : 0	4 : 0	3 : 0	2 : 0	0 : 0	**0 : 0**

Tabelle 7.14: Einfluß der Strafparameter auf (falsch negative : falsch positive) Punkte in den 200 Trainingsdaten (*ensemble* Datensatz).

Ein SVM-Modell sollte niemals nach der Anzahl der Trainingsfehler, sondern nach seiner Generalisierungsfähigkeit bewertet werden. Interessant sind demnach ausschließlich Validierungs- oder Testfehler auf unabhängigen Daten. In Tabelle 7.15 sind die korrespondierenden Ergebnisse dargestellt. Als sehr positiv bewerten wir die Entwicklung der Testergebnisse in der rechten oberen Dreiecksmatrix. Die Effekte der Fehlerverschiebung sind ähnlich denen im Training. Das beste Ergebnis mit höchster Sensitivität bei gleichzeitig höchster Spezifität ist ebenfalls fett dargestellt (1:7).

Die Testergebnisse aus Tabelle 7.15 haben wir in Abbildung 7.3 dargestellt. Die Bilder zeigen, wie sich die variierenden falsch negativen und falsch positiven Punkte über dem Parameterraum zusammen zu einer relativ einheitlichen Gesamtfehleranzahl summieren.

Es stellt sich die Frage, welchen Einfluss die Fehlergewichtung auf die Anzahl und Ver-

		C^+								
		1	1.5	2	4	6	8	10	50	100
	1	**11 : 0**	8 : 1	5 : 4	**1 : 7**	1 : 8	1 : 8	1 : 8	1 : 8	1 : 8
	1.5	11 : 0	**10 : 1**	8 : 2	4 : 6	3 : 6	3 : 6	3 : 6	3 : 6	3 : 6
	2	11 : 0	11 : 2	**10 : 2**	4 : 5	3 : 6	3 : 6	3 : 6	3 : 6	3 : 6
	4	11 : 0	11 : 0	10 : 2	**6 : 3**	4 : 5	4 : 4	4 : 3	4 : 3	4 : 3
C^-	6	11 : 0	11 : 0	10 : 2	8 : 3	**5 : 3**	4 : 4	4 : 3	4 : 3	4 : 3
	8	11 : 0	11 : 0	10 : 2	8 : 3	6 : 5	**4 : 4**	4 : 5	4 : 3	4 : 3
	10	11 : 0	11 : 0	10 : 2	8 : 3	6 : 5	4 : 4	**4 : 4**	4 : 3	4 : 3
	50	11 : 0	11 : 0	10 : 2	8 : 3	7 : 5	7 : 5	5 : 5	**5 : 3**	5 : 3
	100	11 : 0	11 : 0	10 : 2	8 : 3	7 : 5	7 : 5	5 : 5	5 : 3	**5 : 3**

Tabelle 7.15: Einfluß der Strafparameter auf (falsch negative : falsch positive) Punkte in den 63 Testdaten (*ensemble* Datensatz).

teilung der Support-Vektoren hat. Die im Folgenden präsentierten Ergebnisse für die Datensätze *cancer* und *thyroid* stützen sich auf [44], wobei die Tests für den *thyroid* Datensatz nochmals modifiziert worden sind. In [44] wurden nur die Trainingsdaten verwendet. Dazu wurden sie in Trainings- und Recallpunkte aufgeteilt. Wir haben nun den ganzen Trainingsdatensatz für das Training verwendet, denn für die Tests stand jetzt auch der Testdatensatz zur Verfügung. Bei der Wahl der Parameter in diesem Abschnitt haben wir den Kern und seinen Parameter pro Datensatz konstant gelassen. Zudem haben wir je zwei Werte für C definiert, die zweimal ungemischt und einmal gemischt verwendet worden sind. In Tabelle 7.16 sind die Ergebnisse von Validierung, Training und Test dargestellt. Die beiden Datensätze sind – genau wie der *ensemble* Datensatz – interessant, da sie kostensensitiv (*cancer*) bzw. stark unausgeglichen (*thyroid*) sind. Für die Kombination $C^+ = C^- = 30$ (*cancer*) bzw. $C^+ = C^- = 100000$ (*thyroid*), d.h. die Modelle mit den jeweils hohen Werten für den Strafparameter, wurden stets die wenigsten und für die gewichteten Verfahren die meisten Support-Vektoren erlernt. Bei den gewichteten Modellen ist zudem das Verhältnis positiver zu negativer Support-Vektoren deutlich verändert. Interessanterweise nimmt die Anzahl positiver Support-Vektoren ab (von 49 bzw. 19 auf 11 und von 149 bzw. 113 auf 46) und die Anzahl negativer zu (von 47 bzw. 19 auf 111 und von 156 bzw. 142 auf 384.). Die Generalisierungsfähigkeit der gewichteten Modelle ist sehr gut, d.h. die Ergebnisse der Validierung und des Trainings sind auch in den Testergebnissen wiederzufinden. Besonders positiv sind die erreichten Sensitivitäten bei den gewichteten Modellen. Die verbesserten Werte gehen zudem nicht auf Kosten der Genauigkeit.

Die Ergebnisse dieses Abschnitts und viele andere von uns durchgeführte Tests [38, 132] bestätigen, dass die Fehlergewichtung eine flexible und dennoch robuste Methode der kostensensitiven Klassifikation ist.

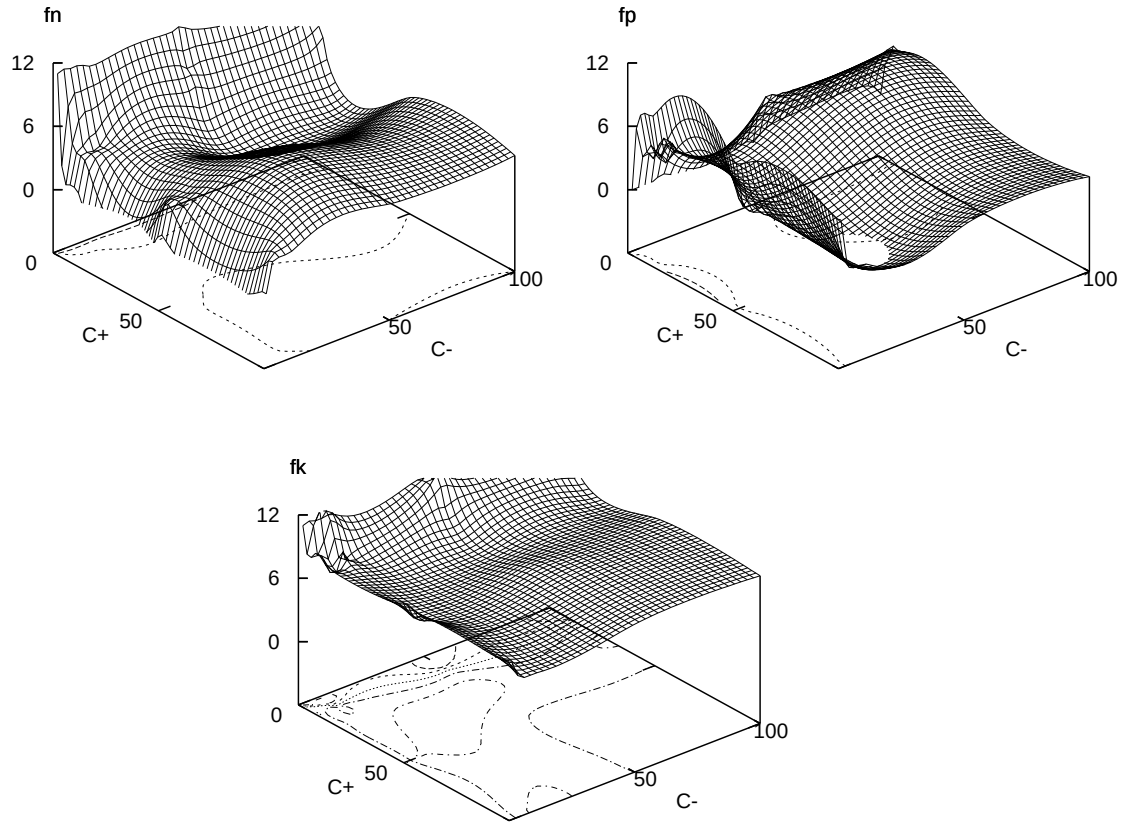

Abbildung 7.3: Anzahl falsch negativer (links oben) und falsch positiver (rechts oben) Testpunkte sowie Gesamtzahl falsch klassifizierter Testpunkte (unten) für variierende Werte von C^+ und C^- (*ensemble* Datensatz).

7.3.2 L_1-Norm und L_2-Norm

In diesem Abschnitt wollen wir zeigen, wie sich die SVM-Software für die beiden Modelle L_1-Norm und L_2-Norm verhält. Die Unterschiede zwischen den Optimierungsproblemen haben wir im Abschnitt 2.4.2 erklärt. Die Einbettung der L_2-Norm in die L_1-Norm-Software wurde im Abschnitt 4.2.1 beschrieben. Bei den Tests in diesem Abschnitt soll erneut auch die gewichtete Fehlergewichtung mit betrachtet werden. Wir versuchen zu zeigen, welches Modell vorzuziehen ist. Allgemein wird behauptet[49], dass das L_2-Norm-Modell zu viele Support-Vektoren definiert, eine Dokumentation dazu gibt es aber nicht. Unsere Tests mit zwei kostensensitiven Datensätzen sollen klären, ob das auch für unausgeglichene Probleme gilt. Zusätzlich ist zu klären, ob die L_2-Norm möglicherweise auch einen positiven Effekt hat, beispielsweise bei der Anzahl der Testfehler.

[49] "It has the disadvantage of producing too many support vectors." S. Sathiya Keerthi, persönliche Kommunikation, 2003

Datensatz	*cancer*			*thyroid*		
Parameter						
C^+	2	30	30	10000	100000	100000
C^-	2	30	2	10000	100000	10000
Validierung						
fk_{valid}	10	12	10	87	81	90
fn_{valid}	4	5	1	86	70	5
fp_{valid}	6	7	9	1	11	85
Training						
sv	96	38	122	305	255	430
sv_p	49	19	11	149	113	46
sv_n	47	19	111	156	142	384
fk_{train}	10	11	10	88	71	89
fn_{train}	4	4	1	78	62	1
fp_{train}	6	7	9	10	9	88
Test						
fk_{test}	15	11	8	113	102	111
fn_{test}	9	6	0	91	75	3
fp_{test}	6	5	8	22	27	108
ac	0.96	0.97	**0.98**	0.97	0.97	**0.97**
se	0.92	0.95	**1.00**	0.64	0.70	**0.99**
sp	0.97	0.98	0.97	0.99	0.99	0.97

Tabelle 7.16: Vergleich von Ergebnissen für L_1-Norm mit ungewichteten und gewichteten SVM's (*cancer* Datensatz ($\sigma = 20$) und *thyroid* Datensatz ($\sigma = 100$)).

Wir betrachten zunächst den *cancer* Datensatz, der auch schon im letzten Abschnitt untersucht worden ist. Die Ergebnisse für beide Modelle sind in Tabelle 7.17 dargestellt. Wir beobachten die folgenden Effekte:

- Validierung

  Die gewichteten Modelle schneiden unterschiedlich gut ab. Während für die ungewichteten Fälle, d.h. $C = 2$ und $C = 30$, die Validierungsergebnisse beider Modelle beinahe identisch sind, kommt es für $C^+ = 30$ und $C^- = 2$ im L_2-Norm-Modell zwar zu der gewünschten Erhöhung der Sensitivität analog zum L_1-Norm-Modell (1 falsch negativer Punkt), die Gesamtfehlerzahl steigt jedoch drastisch an. Es werden 20 Fehler gezählt, doppelt soviele wie im L_1-Norm-Modell.

- Training

  Das L_2-Norm-Modell hat für alle drei Parameterkombinationen die meisten Support-Vektoren berechnet; jeweils etwa doppelt soviele wie das L_1-Norm-Modell. Die Schwankungen der Anzahl der Support-Vektoren für verschiedene Parameter ent-

Modell	L_1-Norm			L_2-Norm		
Parameter						
C^+	2	30	30	2	30	30
C^-	2	30	2	2	30	2
Validierung						
fk_{valid}	10	12	10	10	13	20
fn_{valid}	4	5	1	4	6	1
fp_{valid}	6	7	9	6	7	19
Training						
sv	96	38	122	196	95	227
sv_p	49	19	11	67	45	15
sv_n	47	19	111	129	50	212
fk_{train}	10	11	10	10	11	19
fn_{train}	4	4	1	4	4	0
fp_{train}	6	7	9	6	5	19
Test						
fk_{test}	15	11	8	15	12	21
fn_{test}	9	6	0	10	7	0
fp_{test}	6	5	8	5	5	21
ac	0.96	0.97	**0.98**	0.96	0.97	0.94
se	0.92	0.95	**1.00**	0.92	0.94	1.00
sp	0.97	0.98	**0.97**	0.98	0.98	0.91

Tabelle 7.17: Vergleich von Ergebnissen für L_1-Norm- und L_2-Norm-Modelle mit ungewichteten und gewichteten SVM's für den *cancer* Datensatz ($\sigma = 20$).

wickelte sich bei den Modellen gleich. $C = 30$ führte zu wenigen (38 und 95) und $C = 2$ zu mindestens doppelt sovielen (96 und 196) Support-Vektoren. Bei den gewichteten Modellen gab es noch eine weitere Steigerung auf 122 und 227 Support-Vektoren. Die Trainingsfehler entwickeln sich wieder unterschiedlich. Während beim L_1-Norm-Modell die Gewichtung zu einer Verlagerung der unterschiedlichen Fehlerarten bei stabiler Gesamtfehlerzahl führt, reagiert das L_2-Norm-Modell sehr empfindlich auf die Gewichtung und es gibt etwa doppelt soviele Fehler, ausgelöst durch einen drastischen Sprung bei den falsch positiven Trainingspunkten.

- Test

  Das wichtige unabhängige Testergebnis bestätigt unsere Vermutungen. Die beschriebenen Tendenzen bleiben auch im Test erhalten. Dass das L_2-Norm-Modell mehr Support-Vektoren erzeugt, ist – wenn auch nicht gut dokumentiert, zumindest in der Community – bekannt, aber dass es so empfindlich auf die gewichtete Klassifikation reagiert, haben wir erst in den Tests herausgefunden. Unsere Ergebnisse sind in [44]

dokumentiert. Andere Studien dazu sind uns bisher nicht bekannt.

Für die Ergebnisse des *thyroid* Datensatzes in Tabelle 7.18 zeigen sich ähnliche Muster für Training, Validierung und Test. Für das gewichtete L_2-Norm-Modell sind die Fehlerzahlen jeweils etwas doppelt so hoch wie für das L_1-Norm-Modell. Für diesen Datensatz sind die Unterschiede bei den Support-Vektoren jedoch noch viel drastischer als beim *cancer* Datensatz.

Modell	L_1-Norm			L_2-Norm		
Parameter						
C^+	10000	100000	100000	10000	100000	100000
C^-	10000	100000	10000	10000	100000	10000
Validierung						
fk_{valid}	87	81	90	143	110	204
fn_{valid}	86	70	5	134	100	5
fp_{valid}	1	11	85	9	10	199
Training						
sv	305	255	430	1699	1150	2298
sv_p	149	113	46	188	173	138
sv_n	156	142	384	1511	977	2160
fk_{train}	88	71	89	145	99	196
fn_{train}	78	62	1	139	91	2
fp_{train}	10	9	88	6	8	194
Test						
fk_{test}	113	102	111	158	128	215
fn_{test}	91	75	3	139	106	2
fp_{test}	22	27	108	19	22	213
ac	0.97	0.97	**0.97**	0.95	0.96	0.94
se	0.64	0.70	**0.99**	0.44	0.58	0.99
sp	0.99	0.99	0.97	0.99	0.99	0.93

Tabelle 7.18: Vergleich von Ergebnissen für L_1-Norm- und L_2-Norm-Modelle mit ungewichteten und gewichteten SVM's für den *thyroid* Datensatz ($\sigma = 100$).

Eine weitere Studie zum Einfluß der Modelle und der Gewichtung haben wir in [132] für den *ensemble* Datensatz vorgestellt. Wir haben gezeigt, welchen Schwankungen Ergebnisse unterliegen, wenn einzelne Parameter geändert werden.

7.3.3 Schwellwertverschiebung

Die Schwellwertverschiebung war eine weitere von uns betrachtete Methode zur kostensensitiven Manipulation eines Klassifikators auf unausgeglichenen Datensätzen (Abschnitt 4.3.2). Dabei wird für jeden zu klassifizierenden Punkt $\boldsymbol{x}$ zu dem berechneten Zielfunktionswert $f(\boldsymbol{x})$ eine Konstante $\Delta_b \in \mathbb{R}$ hinzuaddiert, bevor schließlich die Signumfunktion über die Klassifikation entscheidet. Die Modifkation findet also erst nach dem Training statt. Das Ziel dieser Verschiebung der Hyperebene bei kostensensitiven Problemen ist, höhere Sensitivität zu erreichen. Wir wollen diese Technik exemplarisch für den *ensemble* Datensatz testen und zeigen, welche Kosten dabei entstehen.

In Tabelle 7.19 sind zunächst Ergebnisse für zwei ungewichtete L_1-Norm-Modelle ($C = 1,\ C = 100$) dargestellt. Die Ergebnisse ohne Verschiebung stammen aus Tabelle 7.15. Wir haben die Verschiebungen so gewählt, dass jeweils alle möglichen Werte für die Sensitivität erreicht werden bei gleichzeitiger Maximierung der Spezifität. Im unteren Teil der Tabelle haben wir die besten Ergebnisse zusammengefasst, um die Möglichkeiten und Kosten der Schwellwertverschiebung zu zeigen. Dabei haben wir festgestellt, dass sich mit diesem durchaus einfachen Werkzeug zwar eine bessere Sensitivität einstellen kann, die Kosten dafür aber sehr hoch sind. Ein Vergleich mit Tabelle 7.15 verdeutlicht, dass gerade im Bereich sehr hoher Sensitivität die Anzahl falsch positiver Punkte im Test viel zu hoch ist.

Schwierig bei der Schwellwertverschiebung ist die Wahl des Schwellwertes. Da der Wert erst nach dem Erlernen des Modells zu der Zielfunktion dazuaddiert wird, stellt sich die Optimierung schwierig dar. Man kann den Schwellwert bestimmen, indem man die Funktionswerte für beide Klassen betrachtet und den zusätzlichen Schwellwert in Prozent festlegt. Eine andere Methode, die wir getestet haben, ist, den Schwellwert mit in die Optimierung einfließen zu lassen. Dabei wird er als Parameter definiert und mit validiert. Es hat sich gezeigt [132], dass diese Methode gut generalisiert und eine geringere Streuung der Ergebnisse nach sich zieht, jedoch auch sehr zeitaufwändig ist.

Zusätzlich zu den einfachen Verschiebungen wollen wir testen, welchen Einfluß die Fehlergewichtung bei der Schwellwertverschiebung hat. Wir wählen $C^+ = 100$ und $C^- = 1$. Für diese Parameter gab es schon ohne Verschiebung ein gutes Ergebnis (vgl. Tabelle 7.15). Tabelle 7.20 zeigt erneut die gesamte Bandbreite der möglichen Ergebnisse. Im Vergleich zu den besten Ergebnissen in Tabelle 7.19 fällt auf, dass im Bereich $fn = 6,7$ und $fn < 4$ deutlich bessere Spezifitäten erreicht werden. Zur Veranschaulichung haben wie die Ergebnisse in Abbildung 7.4 dargestellt. Sowohl die Genauigkeit (links) als auch die Spezifität (rechts) sind im gewichteten Modell mit Schwellwertverschiebung bei gleicher Sensitivität besser als im ungewichteten Fall, besonders für Bereiche hoher Sensitivität ($se \in [0.7, 1.0]$). In Abbildung 7.5 haben wir – in Vorgriff auf Abschnitt 7.4 – die Werte des einfachen F-Maßes abgetragen. Das Bild zeigt die Reaktion des Gütemaßes auf diese Ergebnisse und verdeutlicht, dass das F-Maß gut auf die unterschiedlichen Szenarien reagieren kann.

	$C = 1$											
Δ_b	–	0.3	0.5	0.55	0.6	0.62	0.7	0.81	0.85	0.9	0.98	1.07
fn_{test}	11	10	9	8	7	6	5	4	3	2	1	0
fp_{test}	0	1	2	2	2	3	4	8	10	15	25	33
	$C = 100$											
Δ_b	-1.5	-1.35	-1.2	-1.0	-0.4	-0.33	–	0.1	0.4	0.7	1.4	1.6
fn_{test}	11	10	9	8	7	6	5	4	3	2	1	0
fp_{test}	0	0	0	1	3	3	3	4	9	18	35	37
	beste Ergebnisse											
fn_{test}	11	10	9	8	7	6	5	4	3	2	1	0
fp_{test}	0	0	0	1	2	3	3	4	9	15	25	33
ac	0.83	0.84	0.86	0.86	0.86	0.86	0.87	0.87	0.81	0.73	0.59	0.48
se	0.00	0.09	0.18	0.27	0.36	0.45	0.55	0.64	0.73	0.82	0.91	1.00
sp	1.00	1.00	1.00	0.98	0.96	0.94	0.94	0.92	0.83	0.71	0.52	0.37

Tabelle 7.19: Schwellwertverschiebung in ungewichteten L_1-Norm-Modellen (*ensemble* Datensatz).

	$C^+ = 100,\ C^- = 1$										
Δ_b	-1.4	-1.0	-0.77	-0.76	-0.75	-0.6	-0.33	-0.07	-0.05	-0.03	0.5
fn_{test}	11	10	8	7	6	5	4	3	2	1	**0**
fp_{test}	0	1	1	1	1	3	4	6	7	7	**16**
ac	0.83	0.83	0.86	0.87	0.89	0.87	0.87	0.86	0.86	0.87	0.75
se	0.00	0.09	0.27	0.36	0.45	0.55	0.64	0.73	0.82	0.91	1.00
sp	1.00	0.98	0.98	0.98	0.98	0.94	0.92	0.88	0.87	0.87	0.69

Tabelle 7.20: Schwellwertverschiebung im gewichteten L_1-Norm-Modell (*ensemble* Datensatz).

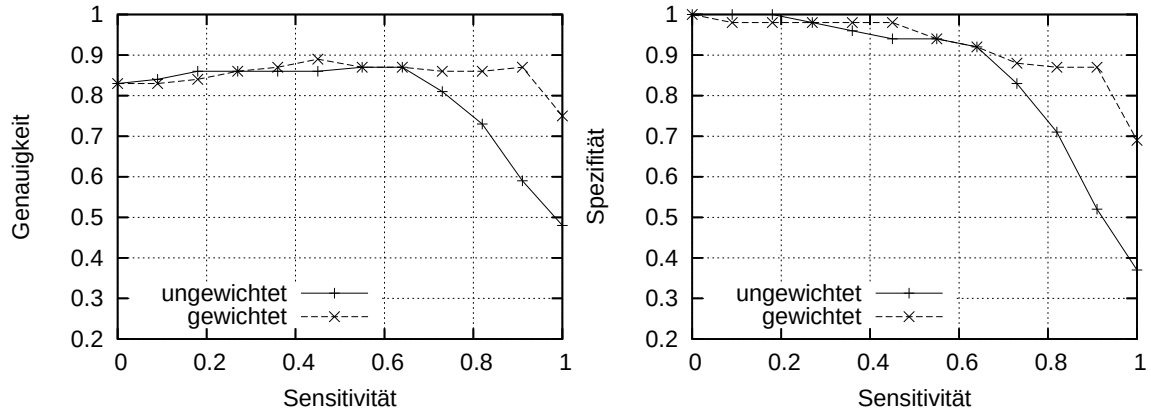

Abbildung 7.4: Graphische Darstellung der Ergebnisse für die Tabellen 7.19 und 7.20.

7.3.4 Sampling

Zu Techniken des kostensensitiven Lernens für den *ensemble* Datensatz haben wir Experimente in [38, 132] und [83] durchgeführt. Dabei ist auch der Aspekt des Samplings

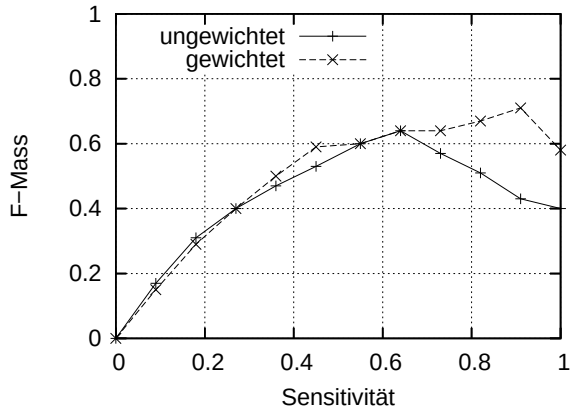

Abbildung 7.5: F-Maß ($\beta = 1$) zu den Ergebnissen der Tabellen 7.19 und 7.20.

betrachtet worden, vgl. Seite 107. Wir haben uns darauf beschränkt, die Methode des Oversamplings einzusetzen. Undersampling ist im Zusammenhang mit Support-Vektor-Maschinen nicht als sinnvoll anzusehen, da die SVM durch die Auswahl von Support-Vektoren selbst in der Lage ist, redundante Punkte in weiter Entfernung von der Hyperebene zu eliminieren. Durch Undersampling würden unter Umständen auch wichtige Information verloren gehen. Undersampling wird oft eingesetzt, wenn die Daten zu groß sind für eine Verarbeitung. Da wir große Daten behandeln wollen, benötigen wir diese Datenverkleinerung nicht.

Im Gegensatz zu den bisherigen Arbeiten [38, 83, 132], bei denen wir dreifaches Oversampling in den Trainingsdaten durchführten und damit die 35 positiven Punkte auf 105 aufgebläht haben, betrachten wir jetzt Oversampling-Faktoren von 1 bis 5. Bei einem Faktor von 5 gibt es mit 185 positiven Punkten mehr positive als negative Trainingsdaten. Sehr wichtig beim Oversampling ist, dass nur die Trainingsdaten aufgebläht werden und keine Kopien von positiven Punkten in die Recall- oder Testmege gelangen, wo sie künstlich das Testergebnis verbessern[50].

Unsere Ergebnisse sind in Tabelle 7.21 dargestellt. Es ist zu erkennen, dass Oversampling für kleine Werte von C eine Verbesserung der Sensitivität mit sich bringt. Gleichzeitig ist die Zunahme der Fehler zweiter Art vertretbar. Für den *ensemble* Datensatz ist die erzielbare Verbesserung enorm, da bei 13 positiven Testpunkten ein zusätzlicher korrekter positiver Punkt gleich 9% mehr Sensitivität bringt. Hohe Sensitivität bei vertretbaren Fehlern zweiter Art zu erreichen ist das Hauptziel dieser Anwendung [38]. Dennoch, einen besonderen Trend kann man in den Ergebnissen nicht erkennen. Die erzielten Sensitivitäten sind nicht so hoch, wie bei den anderen getesteten Methoden und für $C = 10$ passiert beispielsweise überhaupt nichts. Offensichtlich ist der Erfolg des Oversamplings stark von den anderen Parametern abhängig und wirkt nur bei kleinen Strafparameterwerten.

Wir untersuchen nun die Kombination von Sampling und Schwellwertverschiebung mit und ohne Fehlergewichtung. In Tabelle 7.22 kann man an den Zeilen erkennen, welchen

[50] Wiedererkennung von für die Modellierung verwendeten Punkten

Faktor	$C = 1$	$C = 2$	$C = 4$	$C = 6$	$C = 8$	$C = 10$
1	11 : 0	10 : 2	6 : 3	5 : 3	4 : 4	4 : 4
2	**5** : 3	**4** : 5	**4** : 4	**4** : 3	4 : 3	4 : 3
3	**4** : 7	**3** : 6	**4** : 4	**4** : 3	4 : 3	4 : 3
4	**3** : 8	**3** : 6	**4** : 4	**4** : 3	4 : 3	4 : 3
5	**3** : 8	**3** : 6	**4** : 4	**4** : 3	4 : 3	4 : 3

Tabelle 7.21: Einfluss von Oversampling auf (falsch negative : falsch positive) Testpunkte für ungewichtete L_1-Norm-Modelle (*ensemble* Datensatz).

Einfluß das Sampling hat. Im ungewichteten Modell nehmen falsch negative Punkte ab und falsch positive Punkte nehmen zu. Im gewichteten Modell passiert nicht viel, insbesondere lassen sich keine positiven Effekte finden. Betrachtet man die Spalten, so sieht man, dass die Schwellwertverschiebung bei allen Sampling-Arten gut greift. Ein besonders gutes Ergebnis ist fett dargestellt. Es scheint also, dass im Bereich hoher Sensitivität eine Kombination von Sampling und Schwellwertverschiebung für das ungewichtete Modell zu einem genauso guten Ergebnis (0:16) führt wie die Schwellwertverschiebung für das gewichtete Modell (vgl. Tabelle 7.20). Weitere Tests zu Schwellwertverschiebung in Kombination mit Oversampling und Gewichtung sind in [38] dokumentiert. Sie zeigen ähnliche Merkmale. Das Sampling kann immer dann von Nutzen sein, wenn ein gewichtetes Modell nicht verwendet wird, beispielsweise wenn die Parameteroptimierung zu teuer ist. Im Allgemeinen raten wir jedoch davon im Zusammenhang mit Support-Vektor-Maschinen ab, vgl. Abschnitt 4.3.1. Bei der Verwendung anderer Klassifikatoren kommt es vor, dass Oversampling notwendig ist, um die Voraussetzungen an die Methode zu erfüllen. Als Beispiel sei die Maximum-Entropie-Methode genannt, zu den Details verweisen wir auf [83].

Faktor	1	2	3	4	5
	$C = 1$				
–	11 : 0	5 : 3	4 : 7	3 : 8	3 : 8
$\Delta_b = 0.5$	9 : 2	2 : 10	**0 : 16**	0 : 17	0 : 18
$\Delta_b = 0.7$	5 : 4	1 : 17	0 : 20	0 : 21	0 : 22
	$C^+ = 100,\ C^- = 1$				
$\Delta_b = -0.75$	6 : 1	6 : 1	6 : 1	6 : 1	6 : 1
$\Delta_b = -0.03$	1 : 7	3 : 7	3 : 8	3 : 8	3 : 8
–	1 : 8	3 : 8	3 : 8	3 : 8	3 : 8

Tabelle 7.22: Einfluss von Oversampling auf ungewichtetes und gewichtetes L_1-Norm-Modell mit Schwellwertverschiebung (*ensemble* Datensatz).

7.4 Gütemaße und kostensensitive Parameteroptimierung

Wir haben in dieser Arbeit verschiedene Gütemaße vorgestellt (vgl. Kapitel 5). Für die Parameteroptimierung mit *APPSPACK* haben wir die Gütemaße mit zwei neuen Komponenten ausgestattet [43, 133]:

- Für die gewichteten Maße fm und roc regelt ein Parameter β, wie stark die Sensitivität das Maß bestimmt (Abschnitt 5.1.5).
- Diskrete Fehlermessung auf der Hypothese wurde durch ein weicheres Maß auf der Zielfunktion ersetzt und liefert feinere Auswertungen (Abschnitt 5.1.6).

Bei kostensensitiven Problemen nutzen wir typischerweise das F-Maß (5.13) zur Parameteroptimierung. In diesem Abschnitt wollen wir zeigen, warum das F-Maß gut geeignet erscheint. Stellvertretend für ein kostensensitives Anwendungsproblem betrachten wir zunächst den interessanten *ensemble* Projektdatensatz. Die bezüglich Sensitivität und Spezifität besten der in den Abschnitten 7.3.1 bis 7.3.4 erzielten Testergebnisse haben wir als ROC-Plot in Abbildung 7.6 dargestellt. ROC-Plots werden in der Medizin- und Pharmaforschung zur Bewertung von Klassifikatoren eingesetzt [147]. Verglichen mit der Diagonalen sind alle Ergebnisse gut. Anhand der ROC-Regel[51] läßt sich jedoch kein optimaler Punkt festlegen. Für eine automatisierte Parameteroptimierung muss die Auswertung mittels einer klar definierten Funktion erfolgen. Das Gütemaß ist also nicht nur wichtig, um gute und schlechte Ergebnisse zu trennen, sondern auch, um zwischen guten Ergebnissen mittels einer Präferenz auszuwählen.

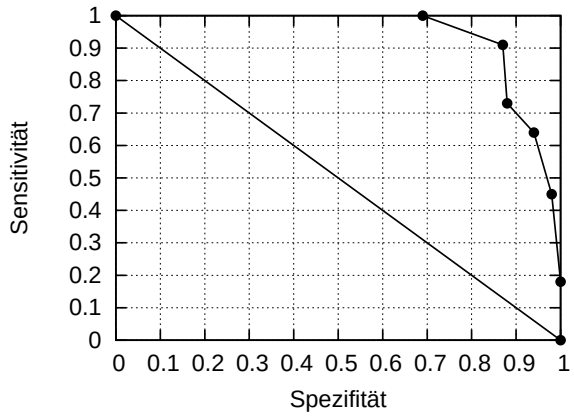

Abbildung 7.6: ROC-Graphik erzielter Klassifikationsergebnisse für den *ensemble* Datensatz.

Die Ergebnisse sind in Tabelle 7.23 zusammengefasst und mittels verschiedener Gütemaße bewertet worden. Man kann erkennen, dass der Anreicherungsfaktor nicht gut zur Optimierung geeignet ist, insbesondere da eine Normalisierung des Maßes fehlt. Die Genau-

[51] Liegt ein Punkt nordöstlicher als ein anderer, ist er streng vorzuziehen.

igkeit ist für kostensensitive Probleme zu allgemein; Sensitivität und Spezifität zu extrem, da bei der Optimierung nicht in der jeweils anderen Klasse geprüft werden kann, ob ein akzeptables Ergebnis erreicht wird. Die kombinierten Maße roc und fm spiegeln unsere Präferenzen gut wieder: das Ergebnis mit perfekter Sensitivität erscheint ein wenig zu teuer, sodass ein Fehler in Klasse 1 durchaus zu tolerieren ist. Die Maße bewerten jedoch die Nachbarpunkte unterschiedlich. Das ROC-Maß tendiert eher zum rechten Rand, während das F-Maß sich hin zur Genauigkeit bewegt. Wir werden beide Maße im Anschluss in einem echten Optimierungslauf für diesen Datensatz vergleichen.

Faktor	1	1	1	1	1	3
C^+	100	100	100	100	100	1
C^-	100	1	10	2	1	1
Δ_b	-1.2	-0.75	–	–	-0.03	0.5
fn_{test}	9	6	4	3	1	**0**
fp_{test}	**0**	1	3	6	7	16
ac	0.86	**0.89**	**0.89**	0.86	0.87	0.75
se	0.18	0.45	0.64	0.73	0.91	**1.00**
sp	**1.00**	0.98	0.94	0.88	0.87	0.69
pr	**1.00**	0.83	0.70	0.57	0.59	0.41
ef	**5.73**	4.77	4.01	3.27	3.37	2.33
roc	0.72	0.76	0.80	0.81	**0.89**	0.86
fm	0.31	0.59	0.67	0.64	**0.71**	0.58

Tabelle 7.23: Bisher erreichte Ergebnisse für den *ensemble* Datensatz und ihre Bewertung durch verschiedene Gütemaße (zeilenweise beste Werte fett).

7.4.1 Vergleich von ROC-Maß und F-Maß

Wir haben zwei Optimierungsläufe mit *APPSPACK* für den *ensemble* Datensatz mit dem F-Maß und dem ROC-Maß durchgeführt. Dabei ist zunächst die harte Fehlermessung verwendet worden. Tabelle 7.24 zeigt die Ergebnisse. In den Zeilen sind jeweils die von *APPSPACK* gefundenen neuen Minima für C^+, C^- und σ angegeben. Die letzte Zeile ist das Endergebnis. Es ist zu beachten, dass weit mehr Punkte getestet worden sind. Auch der erste Parameter, der hier konstant erscheint, wurde variiert. Obwohl die absoluten Werte der Gütemaße nicht vergleichbar sind, erkennt man, dass die vom ROC-Maß gefundene Lösung in der Validierung deutlich schwächer abschneidet, als die durch das F-Maß generierte Kombination (fett).

Wir wollen jetzt zeigen, welche Vorteile die weiche Fehlermessung für das F-Maß (5.16) gegenüber dem normalen F-Maß hat, wenn es zur kostensensitiven Parameteroptimierung

F-Maß

C^+	C^-	σ	rp_{valid}	fp_{valid}	em
10000	1.000	20.00	23	23	0.4458
10000	1.000	23.12	25	25	0.4243
10000	8.812	23.12	20	12	0.4203
10000	4.906	23.12	24	**13**	0.3514

ROC-Maß

C^+	C^-	σ	rp_{valid}	fp_{valid}	$1 - roc$
10000	1.000	20.00	23	23	0.2503
10000	1.000	23.12	25	25	0.2341
10000	1.000	22.34	25	23	0.2273
10000	1.976	22.34	24	**18**	0.2215

Tabelle 7.24: Vergleich von F-Maß und ROC-Maß bei der Optimierung (*ensemble* Datensatz).

eingesetzt wird. Wir fassen zunächst kurz die Unterschiede zwischen den Maßen zusammen. Das F-Maß ist eine Funktion von Sensitivität und Präzision. Diese einfachen Maße basieren auf den Anzahlen richtig positiver und falsch positiver Klassifikationen bei der Validierung. Das einfache F-Maß summiert diese Punkte einfach auf. Unser neues F-Maß summiert die Punkte nach folgendem Konzept: gilt für einen richtig oder falsch positiven Punkt $f < 1$, so wird nur der Funktionswert summiert. Das bedeutet, schwach korrekte Klassifikationen werden nur teilweise beachtet und leicht fehlerhafte Klassifikationen werden etwas weniger bestraft. Es stellt sich die Frage, ob diese Modifikation tatsächlich zu Änderungen bei der Optimierung mit *APPSPACK* führt? Dazu führen wir einen Test am *ensemble* Datensatz durch.

In Tabelle 7.25 sind – wie auch schon in Tabelle 7.24 – die Ergebnisse der Optimierung mit dem üblichen F-Maß ($\beta = 1$) dargestellt. Insgesamt wurden 66 der *APPSPACK*-Testpunkte durch die SVM mit 8-facher Validierung ausgewertet. Dafür wurden 231 Sekunden benötigt. Die dick dargestellten Werte gehören zu den tatsächlich bei der Optimierung verwendeten Ergebnissen. Angegeben ist jeweils der Wert des E-Maßes ($= 1-$F-Maß). Bei diesem Test wurde der Startpunkt $(100.0, 1.0, 20.0)$ definiert. Dieser war aus Tabelle 7.15 als guter Punkt bekannt. Die Ergebnisse zeigen, wie während der Optimierung der Wert des E-Maßes wie gewünscht kleiner wird. Ein Blick auf das weiche E-Maß, welches wir zusätzlich mit berechnet haben (nicht fett), zeigt jedoch ein anderes Bild. Das Maß ist im Optimum sehr hoch. Das letzendlich erzielte Ergebnis im unabhängigen Test ist zwar gut, aber bezüglich der Sensitivität wie erwartet schwach.

Betrachten wir nun das weiche F-Maß ($\beta = 1$). Wir haben den gleichen Test erneut durchgeführt, d.h. Startpunkt $(100.0, 1.0, 20.0)$. Die Ergebnisse (fett) in Tabelle 7.26 zeigen, wie *APPSPACK* nach und nach zum Optimum gelangte, wobei erneut nur die lokalen Minima

C^+	C^-	σ	rp_{valid}	fp_{valid}	$\tilde{tp}_{\text{valid}}$	$\tilde{fp}_{\text{valid}}$	em_1	$\tilde{em}_1$
10000	1.000	20.00	**23**	**23**	13.35	11.42	**0.4458**	0.5677
10000	1.000	23.12	**25**	**25**	15.52	13.82	**0.4243**	0.5322
10000	8.812	23.12	**20**	**12**	6.99	5.66	**0.4203**	0.7185
10000	4.906	23.12	**24**	**13**	7.98	6.45	**0.3514**	0.6896
Testergebnis: $fn = 4$, $fp = 3$								

Tabelle 7.25: *APPSPACK*-Optimierungsergebnisse für das einfache F-Maß (*ensemble* Datensatz).

(gerundet) angegeben sind.[52] Insgesamt wurden mit 87 Punkten mehr Läufe benötigt. Das spiegelt sich auch in der Zeit von 307 Sekunden wieder. Erfreulich ist das Testergebnis mit 100% Sensitivität bei 14 falsch positiven Punkten. Vergleichen wir dieses Ergebnis mit Tabelle 7.23, haben wir also eine Verbesserung erreicht. Die automatische Parameteroptimierung hat ohne Zutun von Wissen einzig über das Gütemaß ein Parametertupel gefunden, welches gut generalisiert und zudem kostensensitiv ist. Verwendet wurde ein einzelner Worker; zu den Ergebnissen der parallelen Parameteroptimierung verweisen wir auf Tabelle 7.39.

C^+	C^-	σ	rp_{valid}	fp_{valid}	$\tilde{tp}_{\text{valid}}$	$\tilde{fp}_{\text{valid}}$	em_1	$\tilde{em}_1$
10000	1	20.00	23	23	**13.35**	**11.42**	0.4458	**0.5677**
10000	1	44.98	32	82	**23.30**	**37.27**	0.5762	**0.5224**
10000	1	32.49	29	47	**19.40**	**21.86**	0.4867	**0.5043**
10000	1	35.61	30	53	**20.49**	**24.78**	0.5000	**0.5018**
10000	1	34.83	30	49	**20.25**	**24.01**	0.4828	**0.5016**
10000	1	35.22	30	51	**20.38**	**24.38**	0.4915	**0.5015**
10000	1	35.02	30	50	**20.31**	**24.19**	0.4872	**0.5015**
10000	1	35.12	30	51	**20.35**	**24.29**	0.4950	**0.5015**
10000	1	35.07	30	51	**20.33**	**24.24**	0.4915	**0.5015**
10000	1	35.08	30	51	**20.33**	**24.25**	0.4915	**0.5015**
Testergebnis: $fn = 0$, $fp = 14$								

Tabelle 7.26: *APPSPACK*-Optimierungsergebnisse für das weiche F-Maß (*ensemble* Datensatz).

Die Tests mit dem neuen F-Maß verliefen sehr positiv. Anders hingegen entwickelten sich die Tests mit dem ROC-Maß für die weiche Fehlergewichtung. Dazu zeigen wir für das obige Beispiel das Endergebnis als auch ein Zwischenergebnis der Optimierung in Tabelle 7.27. Folgende Problematik ist entstanden: für ein sehr schlechtes Ergebnis mit 37 falsch

[52] Wiederum sind Ergebnisse für das harte Maß mit angegeben.

	Zwischenergebnis	Endergebnis
fn	13	37
$\tilde{fn}$	7.38	36.97
fp	18	0
$\tilde{fp}$	9.66	0.00
em	0.39	1.00
$\tilde{em}$	0.60	1.00
$1 - roc$	**0.22**	0.29
$1 - \tilde{roc}$	0.45	**0.29**
fn_{test}	2	11
fn_{test}	7	0

Tabelle 7.27: Schwäche des weichen ROC-Maßes (*ensemble* Datensatz).

negativen Punkten (100%) wird ein ROC-Wert von 0.29 in beiden Fällen (harte und weiche Fehlermessung) erreicht. Bei der harten Fehlermessung wäre dieser zugunsten des dargestellten Zwischenergebnisses verworfen worden ($roc = 0.22$). Bei der weichen Messung wird das Zwischenergebnis jedoch sehr schwach bewertet und entfällt. Kritisch ist also die Tatsache, dass für $se = 0$ oder $sp = 0$ zu gute Werte für roc erreicht werden. Das F-Maß ist hingegen vor Extremwerten geschützt, da die Kennzahlen multiplikativ eingehen und ein einzelner schlechter Wert den anderen quasi aufhebt. Im Folgenden arbeiten wir bei kostensensitiven Anwendungen bevorzugt mit dem F-Maß.

7.4.2 *APPSPACK*-interne Parameter

Neben der SVM hat auch *APPSPACK* interne Parameter, die gewählt werden müssen. Diese sind im Abschnitt 5.2 vorgestellt worden. Ein wichtiger Parameter ist die sogenannte `step tolerance` Δ_{tol}. Für die Tests des letzten Abschnitts galt $\Delta_{\text{tol}} = 0.0001$. Bei einer Erhöhung der Toleranz entstehen wahrscheinlich zwei Effekte: zum Einen ließen sich Iterationen sparen, zum Anderen könnte die Lösung darunter leiden. Den in Tabelle 7.26 gezeigten Test haben wir nun für verschiedene Abbruchkriterien erneut durchgeführt. In Tabelle 7.28 sind die Ergebnisse zu sehen. Wir schlussfolgern, dass eine Erhöhung der Toleranz auf 0.001 bei gleichem Ergebnis 21 Validierungen eingespart hätte. Eine weitere Erhöhung führt zu etwas schlechteren Ergebnissen, spart aber weitere Iterationen. Die *APPSPACK*-Software setzt den Parameter standardmäßig auf 0.01. Dieser Wert hat sich in den Tests mit der SVM-Software und dem F-Maß ebenfalls als geeignet erwiesen.

Δ_{tol}	# Validierungen	$e\tilde{m}_1$ im Optimum
0.1	23	0.5043
0.05	23	0.5043
0.01	32	0.5018
0.005	46	0.5016
0.001	66	0.5015
0.0001	87	0.5015
0.00001	97	0.5015

Tabelle 7.28: Einfluss der `step tolerance` auf Anzahl der Validierungen und Ergebnis.

7.4.3 Gütemasse und Gewichtung

Die Auswahl eines Gütemaßes ist für die Anwendung von Support-Vektor-Maschinen und anderen Lernmethoden sehr wichtig – insbesondere bei kostensensitiven und unausgeglichenen Datensätzen. In [43] haben wir gezeigt, welche Verbesserungen man mit unserem neuen flexiblen Gütemaß erreichen kann. Dabei haben wir mit der Optimierung der Genauigkeit verglichen. Zusätzlich sind gewichtete und ungewichtete Modelle optimiert worden. In Tabelle 7.29 kann man erkennen, wie bei beiden Maßen die Sensitivität durch Gewichtung ansteigt und gleichzeitig die Gesamtfehlerzahl nicht ansteigt – sogar sinkt. Die Kombination von F-Maß und Gewichtung liefert wie erhofft das beste Ergebnis bei der Sensitivität und maximiert zusätzlich sogar die Genauigkeit. Im Gegensatz zu kostensensitiven Techniken, bei denen höhere Sensitivität stark auf Kosten der Genauigkeit geht, kann man also mittels geeigneter Gütemaße kostensensitiv und dennoch sehr genau lernen. Die Parameteroptimierung wurde mit *APPSPACK* durchgeführt. Die Anzahl der Validierungen ist für das F-Maß um mehr als die Hälfte höher als für das einfache Maß. Die Gewichtung und damit ein weiterer Parameter kostet bei beiden Maßen auch fast 50% mehr Validierungen. Damit unterscheidet sich der Aufwand zwischen dem einfachsten Modell rechts und unserem Modell links um mehr als 100 Prozent. Eine Bemerkung noch zur Tabelle: es werden stets mehr Testpunkte generiert, als dann Validierungen stattfinden. Das entsteht durch das Caching bei *APPSPACK*. Schon ausgewertete Kombinationen werden gespeichert und müssen bei erneutem Bedarf nicht mehr ausgewertet werden. Außerdem kommt es vor, dass Testpunkte aus der Warteschleife gelöscht werden, wenn diese nicht mehr benötigt werden.

7.4.4 Gütemaß mit Parameter

In [40] haben wir uns damit beschäftigt, welchen Einfluß der interne Parameter unseres Gütemaßes haben kann. Für verschiedene Werte von β wurde mittels *APPSPACK* eine Parameteroptimierung mit anschließendem Test durchgeführt. Tabelle 7.30 zeigt Ergebnisse

Gütemaß	$\tilde{fm}_1$		ac	
SVM-Modell	gewichtet	ungewichtet	gewichtet	ungewichtet
Optimierungsphase				
# *APPSPACK*-Testpunkte	104	54	41	36
# Validierungen	50	34	32	22
Optimale Parameterwerte				
σ	25	23	50	100
C	—	95	—	25
C^+	97	—	100	—
C^-	3	—	50	—
Testergebnisse				
fk_{test}	10	10	11	16
fn_{test}	0	4	5	11
fp_{test}	10	6	6	5
ac	0.97	0.97	0.97	0.95
se	1.00	0.97	0.96	0.91

Tabelle 7.29: Vergleich von Ergebnissen der Parameteroptimierung mit neuem und altem Gütemaß für gewichtete und ungewichtete SVM-Modelle (*cancer* Datensatz).

für Werte von β die kleiner und größer als eins (Default) sind. Wir haben festgestellt, dass Änderungen des Parameters während der Validierung zu den gewünschten Ergebnissen in den Tests führen. Die Sensitivität konnte für $\beta > 1$ deutlich verbessert werden. Da der *thyroid* Datensatz extrem unausgeglichen ist, nehmen die Fehler zweiter Art deutlich zu, was jedoch bei medizinischen Anwendungen im Allgemeinen toleriert wird. Insgesamt liegt die Testgenauigkeit mit 94% ($\beta = 2.5$) für dieses schwere Klassifikationsproblem[53] durchaus in einem guten Bereich. Letztendlich muss der Nutzer entscheiden, wieviele Fehler in jeder Klasse tolerierbar sind. Was wir zeigen wollen ist, welche Methoden sich eignen, um die Klassifikatoren passend zu den eigenen Präferenzen robust zu beeinflussen. Die Anzahlen der Validierungen sind sehr unterschiedlich und wachsen mit fallendem Wert für β an.

Eine Untersuchung weiterer Gütemaße wurde in [133] für den *cancer* Datensatz durchgeführt. Dabei hat sich gezeigt, dass neben dem F-Maß auch das parametergesteuerte ROC-Maß mit harter Fehlermessung gut zur kostensensitiven Parameteroptimierung von SVM's geeignet ist. Mittels Variierung des internen Parameters konnten ähnliche Ergebnisse wie mit dem F-Maß erzielt werden. Insgesamt haben unsere Untersuchungen gezeigt, welchen enormen Einfluss Gütemaße haben. Dabei sehen wir den positiven Effekt, dass der Einsatz eines passenden Gütemaßes den Prozess der Parameteroptimierung effizient und robust macht und in vertretbarer Zeit das gewünschte Modell liefert. Im Übrigen

[53] Merten Joost, persönliche Kommunikation, 2003

β	0.5	0.75	1.0	1.5	2.5
Optimierungsphase					
# *APPSPACK*-Testpunkte	125	79	75	78	62
# Validierungen	101	62	58	60	46
$\tilde{fm}_\beta$	0.908	0.898	0.892	0.901	0.928
Optimale Parameterwerte					
σ	91.15	52.05	28.84	25.75	64.42
C^+	100000	100000	10280	39670	100000
C^-	21790	19560	1000	1000	1000
Rate C^+/C^-	4.6	5.1	10.3	39.7	100
Testergebnisse					
fk_{test}	70	73	103	137	197
fn_{test}	7	5	4	3	1
fp_{test}	63	68	99	134	196
ac	0.98	0.98	0.97	0.96	0.94
se	0.97	0.98	0.98	0.99	1.00
sp	0.98	0.98	0.97	0.96	0.94

Tabelle 7.30: Vergleich von Ergebnissen der Parameteroptimierung für verschiedene Werte von β im flexiblen F-Maß (*thyroid* Datensatz).

ist der Einsatz eines Gütemaßes unabhängig von der Methode der Parameteroptimierung. Auch eine einfache Gitter-Suche profitiert davon [133]. Der zugrunde liegende Lernalgorithmus ist ebenfalls austauschbar. Unsere Untersuchungen sind somit auch auf andere Klassifikationsverfahren übertragbar nach dem Schema in Abbildung 7.7.

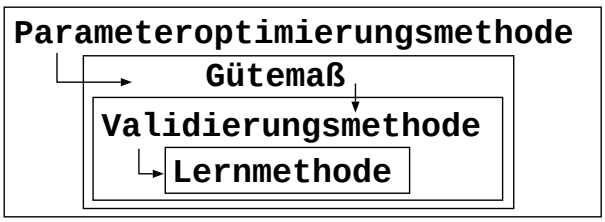

Abbildung 7.7: Allgemeine Struktur zur Parameteroptimierung.

7.4.5 Multiparameter-Kerne

In einem Multiparameter-Kern wird den Variablen nicht immer der gleiche Kernparameter zugewiesen, wir haben diese Kerne auf Seite 104 vorgestellt. In [40] haben wir für den *thyroid* Datensatz getestet, ob die Optimierung von mehr als einem Kernparameter eine

Ergebnisverbesserung für diesen unausgeglichenen Datensatz mit sich bringt. Wir haben dabei je einen Parameter für die binären und einen für die reellwertigen Variablen eingesetzt. Mit der *APPSPACK*-Software wurde ein gewichtetes SVM-Modell optimiert. Wie man anhand der Ergebnisse in Tabelle 7.31 erkennen kann, führte die Optimierung zu anderen Werten bei den Kern- sowie den Strafparametern. Die Testergebnisse zeigten eine Verbesserung in beiden Klassen. Die Tests wurden für $\beta = 1.5$ durchgeführt. Vergleich man jetzt mit Tabelle 7.30, erkennt man den positiven Effekt. Obwohl die Sensitivität gestiegen ist, gab es deutlich weniger Fehler zweiter Art, was durch Variierung von β allein nicht möglich war. Negativ zu benennen ist die von 60 auf 140 gestiegene Anzahl von Validierungsschritten während der Optimierung mit *APPSPACK*. Der Trend, dass feinere Methoden bei der Optimierung deutlich teurer sind, zog sich durch alle unsere Tests.

Kern	Standardkern	Multiparameter-Kern
# Validierungen	60	140
$\tilde{fm}_{1.5}$	0.892	0.902
σ	25.75	—
σ_{bin}	—	72.16
σ_{cont}	—	31.38
C^+	39670	13380
C^-	1000	1000
Verhältnis C^+/C^-	39.7	13.4
fk_{test}	137	112
fn_{test}	3	2
fp_{test}	134	110
ac	0.960	0.967
se	0.988	0.992
sp	0.958	0.965

Tabelle 7.31: Vergleich von Ergebnissen der Parameteroptimierung für den einfachen und den Multiparameter-Gauß-Kern (*thyroid* Datensatz).

7.4.6 Vergleichende Anwendungen zur Parameteroptimierung

Wir haben in den letzten Abschnitten gezeigt, wie wir mit der gekoppelten Anwendung von *APPSPACK* und der SVM-Software im Verbindung mit dem neuen Gütemaß Modelle mit gewünschten Eigenschaften, insbesondere mit hoher Sensitivität, erlernen können. Die Modelle haben auch die unabhängigen Tests gemeistert. Im Abschnitt 7.5.3 werden wir auf die Eigenschaften der parallelen Anwendung von *APPSPACK* eingehen. Dabei wird auch auf die Verbindung zur parallelen Support-Vektor-Maschine eingegangen werden. Die bisher vorgestellten Ergebnisse waren für uns, beispielsweise innerhalb des GALA-Projektes,

von großem praktischen Nutzen. Eine vergleichende Bewertung mit anderen Ergebnissen war dabei aber nicht möglich. Um dennoch zu zeigen, das *APPSPACK* allgemein gut geeignet ist, um SVM-Parameter anzupassen, führen wir in diesem Abschnitt Benchmark-Tests durch.

Der *astro* Datensatz wurde von den LIBSVM-Entwicklern untersucht [67]. Die Eigner dieser Astrophysik-Anwendung konnten nur 75.2% Genauigkeit auf den Testdaten erreichen. Eine gezielte Optimierung wurde durchgeführt. Für den Parameter σ des Gauß-Kerns[54] und den Strafparameter C wurde eine zweistufige Gittersuche vorgeschlagen. Ein grobes Gitter wurde zur Identifizierung einer interessanten Region des Parameterraumes eingesetzt. Im Anschluss wurde die Optimierung mit einem feinen Gitter in diesem Bereich beendet. Dafür ist das folgende Ergebnis dokumentiert:

- 5-fache Kreuzvalidierung mit 96.892% Genauigkeit im Optimum (96 Fehler bei 3089 Punkten),
- 96.875% Genauigkeit im unabhängigen Test (125 Fehler bei 4000 Punkten).

Über den genauen Aufwand der Optimierung, die Gütemessung und die Verteilung der Fehler auf die Klassen ist nichts bekannt. Es wird jedoch gesagt, dass ausschließlich feine Gittersuche nicht möglich ist. Weiterhin ist die Prozedur eingeschränkt auf moderate Datengrößen und maximal 2 Parameter [67]. Wir haben eine Optimierung mit *APPSPACK* durchgeführt. Dabei haben wir diesmal das einfache Maß der Genauigkeit gewählt, da es zu dem Datensatz keine Informationen über Kostensensitivität gibt und die publizierten Ergebnisse die Fehlerarten nicht unterscheiden. Es wurde kein Startpunkt vereinbart. Insgesamt fanden 54 Validierungen statt. Die erzielten Ergebnisse waren:

- 4-fache Kreuzvalidierung[55] mit 96.892% Genauigkeit im Optimum (96 Fehler bei 3089 Punkten),
- 96.750% Genauigkeit im unabhängigen Test (130 Fehler bei 4000 Punkten).

Bei gleicher Validierungsgüte ist unser Test etwas schlechter verlaufen, wobei der Unterschied vernachlässigbar ist.

Der *vehicle* Datensatz wurde ebenfalls von den LIBSVM-Entwicklern optimiert [67]. Die Eigner dieser Industrie-Anwendung konnten nur 4.88% Genauigkeit auf den Testdaten erreichen[56]. Eine gezielte Optimierung mit groben und feinen Grids brachte das folgende Ergebnis:

- 5-fache Kreuzvalidierung mit 84.875% Genauigkeit (188 Fehler bei 1243 Punkten),
- 87.805% Genauigkeit im unabhängigen Test (5 Fehler bei 41 Punkten).

[54] Man beachte auch die Hinweise in [67] zur Motivation der Wahl des Gauß-Kerns.

[55] 5-fache Kreutvalidierung ist bei unserer Software nicht vorgesehen, sodass wir eine 4-fache gewählt haben. Diese liefert im Allgemeinen etwas schlechtere Ergebnisse, sodass der Vergleich fair ist.

[56] typischer Effekt bei realen Daten

Über den genauen Aufwand der Optimierung und die Verteilung der Fehler auf die Klassen ist ebenfalls nichts bekannt. Erneut wurde mittels *APPSPACK* ohne Startpunkt optimiert. Insgesamt fanden 50 Validierungen statt. Die Ergebnisse waren:

- 4-fache Kreuzvalidierung mit 84.875% Genauigkeit (188 Fehler bei 1243 Punkten),
- 87.805% Genauigkeit im unabhängigen Test (5 Fehler bei 41 Punkten).

Sowohl das Validierungs- als auch das Testergebnis sind identisch. Insgesamt verliefen unsere vergleichenden Optimierungen sehr erfolgreich und dokumetieren die Robustheit unseres Ansatzes.

Die Vorteile der Optimierung mit *APPSPACK*, die in diesen Tests deutlich wurden, sind:

- Es können deutlich mehr als 2 verschiedene Parameter betrachtet werden, ohne dass der notwendige Aufwand die Möglichkeiten übersteigt.
- Durch die iterative Verbesserung während der Laufzeit ist auch ein früherer Abbruch nach einer festen Iterationszahl möglich. Bei der Gittersuche gibt es keine Abhängigkeiten zwischen den Iterationen, sodass ein Abbruchzeitpunkt a priori schwierig zu bestimmen ist.
- Es muss zu Beginn keine Einschränkung auf ein grobes Gitter stattfinden, sodass keine interessanten Bereiche verloren gehen können.
- Die Kapazitäten, die bei der aufwändigen Suche auf dem feinen Gitter verbraucht werden und unter Umständen keinen Effekt erzielen, können zur Suche im ganzen Raum eingesetzt werden.
- Es gibt keine Zwischenschritte und keine Interaktion, wie z.B. den Wechsel vom groben zum feinen Gitter.

7.5 Parallelisierung

Ein wichtiger Aspekt unserer Arbeit war auch die Entwicklung einer parallelen SVM-Software. In Kapitel 6 haben wir mehrere parallele Ebenen vorgestellt. Uns stehen drei parallele Modi zur Verfügung:

- paralleles SVM-Training basierend auf OpenMP und ESSLSMP,
- parallele Kreuzvalidierung basierend auf MPI,
- parallele Parameteroptimierung basierend auf MPI.

Die drei Ebenen sind von innen nach außen verschachtelt, sodass die parallelen Versionen in beliebiger Kombination sehr flexibel einsetzbar sind.

Die Zeitmessungen wurden für unsere Implementierung mit den vier Fortran-Routinen

```
1: cpu_time
2: system_clock
3: rtc
4: omp_get_wtime
```

durchgeführt. Die CPU-Zeit (1) liegt im Allgemeinen etwas unter der Wall-Clock-Zeit (2,3). Die Funktion 4 ist eine OpenMP-Funktion zur Messung der Wall-Clock-Zeit.

7.5.1 Innere Ebene

Die innere parallele Ebene ist besonders für große Daten, bei denen ein einzelnes Training sehr lange dauert, von Interesse. Allgemein ist eine Reduzierung der Trainingszeit auch bei der iterativen Parameteroptimierung wertvoll. Dieser Abschnitt basiert auf den Arbeiten [41, 42, 45, 46], in denen wir verschiedene Tests für paralleles SVM-Training durchgeführt haben.

In Tabelle 7.32 untersuchen wir die Skalierbarkeit für den *bonds* Datensatz mit 10 bzw. 50 Variablen bei einer Arbeitsmenge mit 5000 Punkten und 20000 Trainingspunkten, vgl. [45]. Die Rechenzeiten für den Datensatz mit 50 Variablen sind etwa dreimal so hoch, wie für den Fall mit 10 Variablen. Da die Trainingszeit von Support-Vektor-Maschinen allgemein linear mit der Anzahl der Variablen steigt, ist dieses Verhalten normal. Wir haben die

Variablen	10	50
t^1_{train}	319.4	980.9
t^2_{train}	161.0	546.2
t^3_{train}	117.1	371.6
t^4_{train}	99.2	279.9
t^5_{train}	80.3	229.5
t^6_{train}	69.8	195.8
t^7_{train}	62.7	169.0
t^8_{train}	55.5	151.9
s^2_{train}	2.0	1.8
s^3_{train}	2.7	2.6
s^4_{train}	3.2	3.5
s^5_{train}	4.0	4.3
s^6_{train}	4.6	5.0
s^7_{train}	5.1	5.8
s^8_{train}	5.8	6.5

Tabelle 7.32: Skalierbarkeit des parallelen SVM-Trainings.

Ergebnisse in Abbildung 7.8 dargestellt. Die Skalierbarkeit bis 8 Threads ist als gut anzusehen. Linearer Speedup kann nicht erreicht werden, da die Zerlegungsmethode serielle Anteile enthält. Für den großen Datensatz entwickelt sich die Skalierbarkeit für 4 und mehr Threads günstiger. Das liegt an dem höheren Anteil der Kernberechnungen an der Gesamtarbeit. Die Laufzeit wird stärker durch die Berechnung der Kerne beeinflusst, sodass der serielle Anteil kleiner ist und die Parallelität besser greift.

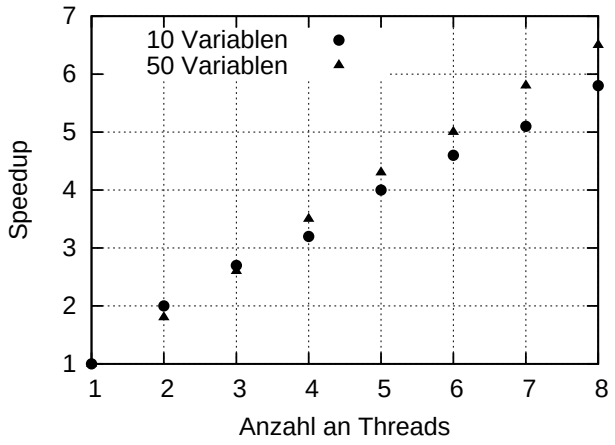

Abbildung 7.8: Vergleich der Speedup-Werte für die Daten aus Tabelle 7.32.

Wie haben gesehen, dass die Parallelität der inneren Ebene gut skaliert. Wie schon mehrfach erwähnt, hat die Größe der Arbeitsmenge enormen Einfluss auf die Rechenzeit. Für den seriellen Fall haben wir im Abschnitt 7.2.4 gezeigt, dass sehr große Arbeitsmengen nicht zu den besten Rechenzeiten führen. Es stellt sich nun die Frage, wie sich die Skalierbarkeit in Abhängigkeit der Arbeitsmenge entwickelt. Dafür testen wir die optimierte Version des Zerlegungsalgorithmus. In Tabelle 7.33 sind die Trainingszeiten für den *a9a* Datensatz für verschieden große Arbeitsmengen mit 1, 2 und 4 Threads angegeben. Besonders auffällig ist der superlineare Speedup für die kleinste Arbeitsmenge bei 2 Threads. Dieser wird auf dem JUMP-Cluster oft durch den schnellen Cache-Speicher hervorgerufen, dessen Vergrößerung bei parallelem Rechnen dazu führt, dass einzelne Teilprobleme in den Cache passen. Auch für 4 Threads kann man erkennen, dass eine bessere Effizienz vorliegt, als für die Tests mit $ws > 50$. In Abbildung 7.9 sind die Ergebnisse graphisch dargestellt, wobei wir weitere Tests durchgeführt haben, die in Tabelle 7.33 aus Platzgründen nicht angegeben sind. Bei den drei Kurven ist die Form jeweils ähnlich, sodass das parallele Verhalten dem seriellen Fall entspricht. Es zeigt sich, dass analog zu den bisherigen Ergebnissen die Arbeitsmenge mit 1200 Punkten zur besten Performance in allen Tests führt (fett dargestellt). Für eine Arbeitsmenge mit 50 Punkten konnte eine sehr gute Skalierbarkeit erreicht werden. Vergleicht man jedoch die Rechenzeiten, wird deutlich, dass diese Größe im Vergleich zu $ws = 1200$ nicht verwendet werden sollte. Trotz der seriellen Anteile des Verfahrens sind die Speedup-Werte gut.

ws	50	100	300	500	1000	1200	1600	2000	2400	2800	3000	3500
t^1_{train}	2477	1986	1063	874	651	**592**	793	962	1064	1346	1477	2176
t^2_{train}	1141	1006	571	460	379	**347**	458	558	617	795	879	1224
s^2_{train}	2.2	2.0	1.9	1.9	1.7	1.7	1.7	1.7	1.7	1.7	1.7	1.9
t^4_{train}	659	610	318	276	200	**191**	250	318	356	457	486	713
s^4_{train}	3.8	3.3	3.3	3.2	3.3	3.1	3.2	3.0	3.0	2.9	3.0	3.1

Tabelle 7.33: Ausgewählte Ergebnisse des parallelen Trainings mit 1, 2 und 4 Threads bei variierender Größe der Arbeitsmenge (O5-Optimierung, *a9a* Datensatz, $C = 1$, $\sigma = 3$).

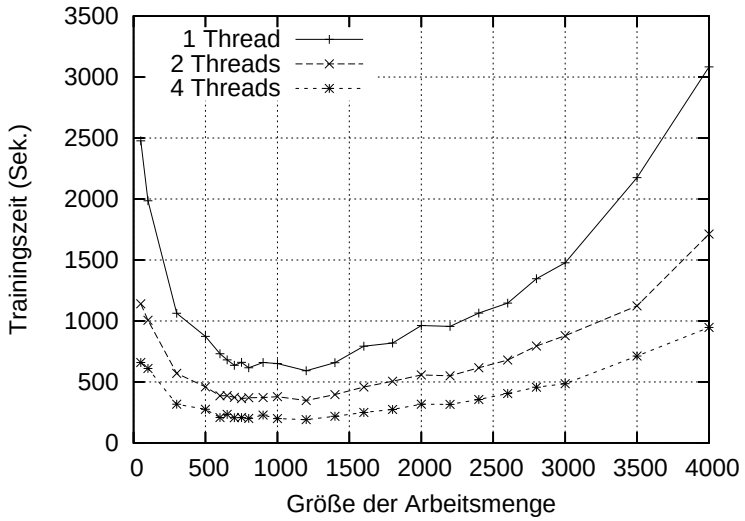

Abbildung 7.9: Darstellung der Skalierbarkeit in Abhängigkeit der Arbeitsmenge (*a9a* Datensatz, $C = 1$, $\sigma = 3$).

Die durchgeführten Tests haben gezeigt, dass für alle Programmversionen parallele Skalierbarkeit vorliegt. Zunächst für sehr große Arbeitsmengen nachgewiesen [41], haben wir gezeigt, dass auch kleinere Arbeitsmengen gute parallele Skalierbarkeit zeigen [46]. Mit diesen konnte dann eine bessere Gesamtperformance erreicht werden. Die Nutzung extrem großer Arbeitsmengen kann auf dem JUMP-System ohnehin nur durch Anforderung von sehr viel Speicher realisiert werden. Auf anderen Cluster-Systemen mit weniger Speicherverfügbarkeit können keine so großen Working-Sets eingesetzt werden.

7.5.2 Mittlere Ebene

Die mittlere parallele Ebene setzt eine verteilte Validierung, ähnlich der WEKA-Software [18] um und wurde im Abschnitt 6.2.3 vorgestellt. Das Zusammenführen mit der inneren parallelen Ebene ermöglicht eine hybrid-parallele Nutzung der Kreuzvalidierung. Für kleine Daten kann auch die serielle Version des Trainings verwendet bzw. nur ein

Thread für die innere Ebene zur Verfügung gestellt werden. Die parallele Skalierbarkeit der Validierung hängt von zwei Faktoren ab:

- Anzahl der Validierungsschritte und
- unterschiedliche Dauer der einzelnen Schritte.

Wir arbeiten typischerweise mit einer 8-maligen Validierung, sodass der Speedup auf 8 beschränkt ist. Für sehr feine Modelle, wie die *leave-one-out* Validierung kann man auch mehr CPU's einsetzen. Die Dauer der Validierungsschritte sollte sich nicht stark unterscheiden. Wie auch schon in [37] begründet, ist die Dauer eines Trainings abhängig von der Anzahl der Trainingsdaten und Variablen sowie von den externen Lernparametern. Beispielsweise kann die Rechenzeit für unterschiedliche Kernparameter drastisch variieren. Der Vorteil bei der Validierung ist, dass jeder Schritt etwa gleich viele Trainingspunkte bekommt, die Anzahl an Variablen konstant und alle Parameterwerte gleich sind. Das führt zu einer ausgeglichenen Last, sodass nicht weiter eingegriffen werden muss. Einzig der Fortschritt der Optimierung kann in Einzelfällen ungünstig verlaufen und so zu starkem Anwachsen der Rechenzeiten führen. Man sollte insbesondere für große Daten beachten, dass die Anzahl der CPU's möglichst ein Teiler der Anzahl der Validierungsschritte ist, um keine Ressourcen zu verschwenden.

Wir haben die verteilte Validierung zunächst für das serielle Training getestet. In Tabelle 7.34 sind die Validierungszeiten für den *astro* Datensatz mit 1, 2, 4 und 8 CPU's für eine 8-fache Validierung angegeben. Die Ergebnisse sind, wie zu erwarten, identisch (letzte Spalte).

CPU's	t_{valid}	s_{valid}	fk_{valid}
1	26.4	–	4.75%
2	13.6	1.9	4.75%
4	6.8	3.9	4.75%
8	3.8	6.9	4.75%

Tabelle 7.34: Gute Skalierbarkeit der parallelen Validierung durch ausgeglichene Last (*astro* Datensatz, $ws = 100$, $C = 50$, $\sigma = 2$).

Eine komplexere Anwendung für beide parallele Ebenen wurde für den *bonds* Datensatz durchgeführt [37]. Wiederum wurde eine 8-fache Validierung durchgeführt. Nach und nach wurde die Anzahl der Threads für das parallele Training erhöht. Dabei vermischen sich die Effekte in beiden Ebenen, siehe Tabelle 7.35. Für eine bessere Übersicht haben wir auch die Werte der parallelen Effizienz abgetragen. Die Effizienz eff_{par} für eine Anwendung mit insgesamt c CPU's/Threads definieren wir wie üblich als

$$\text{eff}_{\text{par}} := \frac{\text{Speedup}}{c} .$$

Die Effizienz nimmt mit wachsender Anzahl an CPU's/Threads ab, ist aber bei allen Tests als gut anzusehen. Im Maximum haben wir zwei ganze Knoten des JUMP-Systems eingesetzt (64 Prozessoren). Auffällig ist der Sprung von einem auf zwei Knoten in der letzten Spalte. Dabei steigt die Effizienz von 0.67 auf 0.82 und sinkt dann wieder auf 0.67. Das liegt an der limitierten Bandbreite des Speichers auf JUMP. Sobald man auf zwei Knoten rechnet, jedoch nicht alle Prozessoren arbeiten, ist das System entlastet und liefert bessere Ergebnisse.

		# CPU's			
		1	2	4	8
# Threads	1	6105 : 1.0 : 1.00	3074 : 2.0 : 1.00	1566 : 3.9 : 0.98	834 : 7.3 : 0.91
	2	3157 : 1.9 : 0.95	1599 : 3.8 : 0.95	815 : 7.5 : 0.94	453 : 13.5 : 0.84
	3	2168 : 2.8 : 0.93	1109 : 5.5 : 0.92	577 : 10.6 : 0.88	348 : 17.5 : 0.73
	4	1641 : 3.7 : 0.93	847 : 7.2 : 0.90	444 : 13.7 : 0.86	284 : 21.5 : **0.67**
	5	1362 : 4.5 : 0.90	703 : 8.7 : 0.87	366 : 16.7 : 0.84	187 : 32.7 : **0.82**
	6	1172 : 5.2 : 0.87	609 : 10.0 : 0.83	326 : 18.7 : 0.78	165 : 37.0 : 0.77
	7	1054 : 5.8 : 0.83	549 : 11.1 : 0.79	299 : 20.4 : 0.73	155 : 39.4 : 0.70
	8	978 : 6.2 : 0.78	518 : 11.9 : 0.74	290 : 21.9 : 0.68	158 : 42.9 : **0.67**

Tabelle 7.35: Vergleich von (Validierungszeit in Sekunden : Speedup : Effizienz) für eine 8-fache Kreuzvalidierung (*bonds* Datensatz mit 50 Variablen, L_1-Norm, Gauß-Kern, $C^+ = 100$, $C^- = 20$, $\sigma = 1$).

7.5.3 Äußere Ebene

APPSPACK, die von uns verwendete serielle und parallele Software zur Parameteroptimierung, wurde mittels eines neuen Kommunikationsschemas an die beiden parallelen Ebenen der SVM-Software angebunden. Details dazu sind in Kapitel 6 zu finden. Zunächst wurde die Software zusammen mit der seriellen SVM getestet. Es sollte gezeigt werden, welche Effekte die parallele Parameteroptimierung auf die konkreten Ergebnisse hat. Dazu wurden Tests in [40] durchgeführt. Abermals wurde der *thyroid* Datensatz verwendet. In Tabelle 7.36 haben wir die Ergebnisse zusammengefasst. Die Anzahl der Funktionsauswertungen stieg im parallelen Fall drastisch an, war aber für 8 CPU's am größten. Die optimalen Parameterwerte unterschieden sich zwischen seriellem und parallelem Lauf, und für den Kernparameter auch innerhalb der parallelen Läufe. Die Ergebnisse für 16 und 32 CPU's sind gleich. Interessant sind die folgenden Merkmale:

- Die erreichten Werte für das Gütemaß sind fast identisch.
- Die Verhältnisse C^+/C^- sind fast identisch.

APPSPACK-Modus	keine Worker	7 Worker	15 Worker	31 Worker
# Funktionsauswertungen	49	80	62	62
$e\tilde{m}_{0.75}$	0.102	0.101	0.102	0.102
fk_{train}	50	50	51	51
σ	62.88	72.16	52.05	52.05
C^+	69060	100000	100000	100000
C^-	13380	19560	19560	19560
Verhältnis C^+/C^-	5.2	5.1	5.1	5.1
fk_{test}	78	78	73	73
fn_{test}	4	4	5	5
fp_{test}	74	74	68	68

Tabelle 7.36: Einfluß der Parallelität von *APPSPACK* auf die Ergebnisse (*thyroid* Datensatz, variables F-Maß mit $\beta = 0.75$).

- Die Anzahlen falsch negativer Testpunkte in den unabhängigen Tests nach der Optimierung sind fast identisch. Für die Tests mit einem Fehler mehr, wurden im Gegenzug 6 Punkte mehr richtig in die negative Klasse eingeordnet. Die Kosten entsprechen dabei jeweils $6 : 1$ und ähneln dem Verhältnis C^+/C^- sowie dem Verhältnis negativer zu positiver Trainingspunkte $(7 : 1)$.

Die Optimierung der SVM-Lernparameter mittels der *APPSPACK*-Software hat sich als sehr praktisch herausgestellt, insbesondere mit dem neuen flexiblen Gütemaß, siehe Abschnitt 5.1.6 und [43]. Die verschiedenen Lösungen entstehen durch die vorhandenen lokalen Minima der Gütefunktion im Parameterraum. Solange die Testergebnisse gleich sind, hat das keine Nachteile. Dennoch gibt es eine negative Seite der parallelen Optimierung. Die Tests zeigten, dass bei Nutzung vieler CPU's mit fortschreitender Zeit immer weniger CPU's aktiv an der Optimierung beteiligt sind. Das liegt am iterativen Charakter des Verfahrens. Gibt es keine weiteren Testpunkte in einer Iteration, stehen die CPU's still und warten auf weitere Aufgaben. Bei unseren Tests nahmen diese Aufgaben aber mit der Zeit immer weiter ab, da zum Ende hin nur wenige Richtungen untersucht werden. Bei der Gittersuche kann das beispielsweise nicht passieren, da die Aufgaben dabei unabhängig voneinander sind. Einen Ausweg aus diesem Problem bietet der Einsatz der mittleren und/oder inneren parallelen Ebene der SVM-Software. Die Idee ist einfach: kann eine bestimmte Anzahl von CPU's dazu verwendet werden, einen einzelnen *APPSPACK*-Schritt, also eine vollständige Validierung, schneller zu machen, kommt es theoretisch zu folgenden positiven Effekten:

- Die CPU's von *APPSPACK* sind besser ausgelastet.
- Es gibt eine deutliche Zeitersparnis durch schnellere Validierungsschritte.
- Die Gesamtzahl an Iterationen nimmt leicht ab.

Diese Annahmen sind in [48] an einem Beispiel überprüft worden. Angenommen seien die in Tabelle 7.37 angegebenen Testszenarien für den *cancer* Datensatz bei 8-facher Validierung. Für eine konstante Anzahl von 13 CPU's haben wir drei mögliche Aufteilun-

Szenario	A	B	C
Master	1	1	1
Worker	12	6	3
Slaves je Worker	0	1	3
CPU's gesamt	13	13	13

Tabelle 7.37: Szenarien für angepasste *APPSPACK*-Tests.

gen auf mittlere und äußere Ebene vorgenommen. In Tabelle 7.38 geben wir die Anzahl der Auswertungen pro Worker, die Gesamtzahl an Auswertungen für jedes Szenario sowie die verbrauchte Gesamtzeit (in Sekunden) der parallelen Optimierung an. Die Ergebnis-

Szenario	A	B	C
Worker 1	18	18	23
Worker 2	17	19	23
Worker 3	15	19	22
Worker 4	17	17	–
Worker 5	6	6	–
Worker 6	5	5	–
Worker 7	3	–	–
Worker 8	3	–	–
Worker 9	2	–	–
Worker 10	1	–	–
Worker 11	1	–	–
Worker 12	1	–	–
Gesamtzahl	89	84	68
Ersparnis	–	6%	24%
Gesamtzeit	28.2	16.3	12.4
Ersparnis	–	42%	56%

Tabelle 7.38: *APPSPACK*-Tests mit paralleler Kreuzvalidierung. Gegeben ist jeweils die Anzahl der Auswertungen von Testpunkten.

se bestätigen die Vermutungen. Es kommt zu einem Ausgleich der Last für die Worker, insbesondere für das Szenario C. Die Zeitersparnis ist sehr positiv, wenn man bedenkt, dass in jedem Test die gleiche Anzahl an CPU's eingesetzt wurde. Bei den Tests mit dem

einfachen F-Maß wurden stets die gleichen Ergebnisse erzielt, d.h. die gleichen Parameterkombinationen sind generiert worden. Im Folgenden soll ein ähnlicher Test mit mehr Kombinationen für das weiche Maß durchgeführt werden.

Die Kombination von paralleler Optimierung und paralleler SVM-Validierung führen dazu, dass die Optimierung je nach Anzahl und Verteilung der CPU's unterschiedlich verlaufen kann. Bei zunehmenden Workern werden parallel mehr Punkte getestet. Durch parallele Validierung verändern sich die einzelnen Ausgabezeiten des Gütemaßes. In Anlehnung an den Test in Tabelle 7.26 haben wir für den *ensemble* Datensatz und das weiche F-Maß nun einige Szenarien der gekoppelt-parallelen Optimierung mit $1, 2, 4, 8$ und 16 Workern durchgespielt. Die Ergebnisse sind in Tabelle 7.39 zu finden. Wir mussten feststellen,

CPU's	Worker	+Slaves	Validierungen	C^+ : C^- : σ	$fn_{\text{test}} : fp_{\text{test}}$
2	1	–	87	10000 : 1 : 35.08	0 : 14
3	2	–	93	10000 : 1 : 35.08	0 : 14
3	1	1	87	10000 : 1 : 35.08	0 : 14
5	4	–	98	10000 : 1 : 35.08	0 : 14
5	2	1	93	10000 : 1 : 35.08	0 : 14
5	1	3	87	10000 : 1 : 35.08	0 : 14
9	8	–	87	10000 : 0.2677 : 20.24	0 : 20
9	4	1	98	10000 : 1 : 35.08	0 : 14
9	2	3	91	10000 : 1 : 35.08	0 : 14
9	1	7	87	10000 : 1 : 35.08	0 : 14
17	16	–	95	10000 : 0.2677 : 20.24	0 : 20
17	8	1	91	10000 : 0.2677 : 20.24	0 : 20
17	4	3	100	10000 : 1 : 35.08	0 : 14
17	2	7	93	10000 : 1 : 35.08	0 : 14

Tabelle 7.39: Unterschiedliche Lösungen bei Variierung der Anzahl von Workern.

dass für Konstellationen mit 8 und 16 Workern eine andere Lösung gefunden wird. Die Lösung schneidet im Test schlechter ab. Eine Auswertung der Ergebnisse zeigt, dass auch das erreichte Gütemaß mit 0.5095 schlechter ist, als für die anderen Ergebnisse (0.5015, vgl. Tabelle 7.26). Die Optimierung führt in ein unerwünschtes lokales Minimum, welches nicht wieder verworfen wird. *APPSPACK* stellt Parameter zur Verfügung, die helfen, verfrühte Abbrüche zu vermeiden. Die a priori Optimierung dieser Parameter ist jedoch schwer.

7.6 Zusammenfassung

Dieses Kapitel diente dazu, unsere Software anhand geeigneter Beispielanwendungen – teils aus dem GALA-Projekt, teils aus frei verfügbaren Quellen – exemplarisch zu testen und die Ergebnisse kritisch auszuwerten. Zunächst wurden Eigenschaften der Zerlegungsmethode untersucht. Dabei haben wir Tests zu den Fragestellungen

- Wie hoch sind die Kosten von Multiparameter-Kernen?
- Wie hoch ist die Zeiteinsparung durch Datentransformation?
- Gibt es Zeiteinsparung durch den neuen Gradienten?
- Welchen Einfluß hat die Größe der Arbeitsmenge bei unterschiedlich großen Datensätzen auf Zeit und Ergebnisse, insbesondere die Anzahl der Support-Vektoren?
- Wie ist die Aufteilung der Rechenzeit auf einzelne Routinen bei unterschiedlich großen Datensätzen?

durchgeführt.

Im Anschluss sind wir auf Fragestellungen im Zusammenhang mit Kostensensitivität und Genauigkeit eingegangen. Untersucht worden sind:

- gewichtete SVM-Modelle,
- Unterschiede zwischen L_1- und L_2-Norm-Modellen,
- Schwellwertverschiebung,
- Sampling, und
- Kombinationen dieser Methoden.

Dabei haben wir kostensensitive Datensätze verwendet.

Zu Gütemaßen und Parameteroptimierung sind aufbauend auf den bis dahin gewonnenen Erkenntnissen folgende Untersuchungen durchgeführt worden:

- Unterschiede von ROC-Maß und F-Maß,
- Einfluß *APPSPACK*-interner Parameter auf die Optimierung,
- Optimierung gewichteter und ungewichteter SVM's,
- Optimierung einfacher und komplizierter Gütemaße,
- Flexibilität durch Gütemaß-interne Parameter,
- Optimierung von Multiparameter-Kernen, und
- vergleichende Benchmark-Tests.

Die erzielten Ergebnisse waren sehr positiv. Das Zusammenspiel von gewichteten und flexiblen SVM's mit gewichteten und flexiblen Gütemaßen bei der Optimierung mit *APPSPACK* funktionierte gut und es konnten sensitive Testergebnisse erzielt werden. Wir haben gezeigt, welche konkreten Verbesserungen der Einsatz des neuen Gütemaßes in Verbindung mit gewichteter Klassifikation für kostensensitive Probleme bringt.

Zu unserer Parallelisierung haben wir im Anschluss Tests mit allen drei Ebenen durchgeführt:

- Für die innere parallele Ebene haben wir die Skalierungseigenschaften eines einzelnen Trainings untersucht. Ein wichtiger Aspekt dabei war die Untersuchung unterschiedlich großer Arbeitsmengen, da schon aus den vorangegangengen Tests der seriellen Software klar war, wie unterschiedlich die Rechenzeiten in Abhängigkeit dieser Größe sein können. Wir haben gezeigt, dass der Algorithmus für optimale Größen der Arbeitsmengen gut skaliert.
- Für die Validierungsebene haben wir gezeigt, dass die parallele Version gut skaliert und haben erklärt, warum alle Validierungschritte etwa gleich lange rechnen.
- Für eine größere Anwendung haben wir die beiden inneren parallelen Ebenen auf bis zu 64 CPU's auf dem JUMP-System getestet und gute Speedup-Werte erreicht.
- Die parallele *APPSPACK*-Software haben wir zur Parameteroptimierung von SVM's eingesetzt. Dabei haben wir Experimente mit der seriellen und der parallelen SVM-Software durchgeführt. Wir haben gesehen, wie gut *APPSPACK* und SVM zusammenarbeiten und welche Vorteile die Kopplung der parallelen Ebenen mit sich bringt. Die Tests bestätigten, dass die drei neuen parallelen Ebenen zu einer flexiblen und effizienten Software geführt haben.

Trotz aller Vorteile gibt es noch offene Fragen bezüglich der parallelen Optimierung. Beispielsweise wäre es von Vorteil, wenn stets die gleiche Lösung gefunden werden würde. Dazu sind weitere Optimierungen am Gütemaß und an den internen Parametern von *APPSPACK* vorzunehmen. Die Schwierigkeiten bei der Nutzung vorhandener CPU's durch die Restriktionen der Teilbarkeit und die Verschwendung von Threads durch den Master (vgl. Anhang B) sind zu bemängeln und könnten verbessert werden.

Kapitel 8

Zusammenfassung und Ausblick

In dieser Arbeit wurde unsere dreistufig parallele Software zur kostensensitiven Parameteroptimierung von Support-Vektor-Maschinen vorgestellt. Sie adressiert Anwendungen der binären Klassifikation unterschiedlicher Fachgebiete, die durch große und unausgeglichene oder kostensensitive Daten gekennzeichnet sind. Solche Daten treten zunehmend in der Bioinformatik und der Pharmaforschung, aber auch bei Text-, Schrift- und Bildklassifikationsproblemen sowie bei der Entwicklung technischer Diagnosesysteme auf.

Der Kern unserer Software ist ein effizienter serieller Zerlegungsalgorithmus für das Training einer Support-Vektor-Maschine. Für kostenintensive Matrix-Vektor-Operationen werden Funktionen aus der ESSL-Bibliothek verwendet. Im Inneren haben wir schnelle Sortieralgorithmen implementiert. Um kostensensitive und unausgeglichene Daten klassifizieren zu können, sind flexible Komponenten entwickelt worden, die optional einsetzbar sind. Dazu gehören die Methoden der Klassengewichtung, der Schwellwertverschiebung und des Samplings sowie Multiparameter- und Ensemble-Kerne. Der Trainingsalgorithmus wurde um eine Routine zur Kreuzvalidierung ergänzt, wobei beginnend bei der groben zweifachen Kreuzvalidierung auch feinere Validierungsmethoden gewählt werden können. Die Validierung endet mit der Berechnung eines Gütemaßes, das dem Nutzer ein Gefühl für die Performance des Modells gibt. Neben der Genauigkeit als wichtiges Standardmaß haben wir kostensensitive parametrisierte Maße vorgestellt und implementiert. Zusätzlich kann auch noch ein unabhängiger Recall durchgeführt werden, der ebenfalls bewertet wird.

Support-Vektor-Maschinen werden mittels verschiedener Parameter konfiguriert. Für erfolgreiche Anwendungen ist es wichtig, diese Parameter möglichst gut auf neue Daten einzustellen. Die Grid-Suche, das Testen aller durch gegebene Werte einzelner Parameter möglichen Kombinationen, ist zum Einen sehr aufwändig und zum Anderen für den Nutzer, der wenig Erfahrung hat, zu schwierig, da die Grid-Suche nur erfolgreich ist, wenn zwischen groben und feinen Grids gewechselt wird. Wir haben deshalb vorgeschlagen, die Parameteroptimierung mittels der frei verfügbaren Optimierungssoftware *APPSPACK* durchzuführen. Wir haben gezeigt, dass sich diese Software dazu eignet, das

SVM-Parameteroptimierungsproblem zu lösen, da wir eine Zielfunktion mit Schrankennebenbedingungen vorliegen haben. Wir haben dargestellt, wie wir die Verbindung zwischen unserer SVM-Software und *APPSPACK* hergestellt haben. Gesteuert wird die Optimierung über die Validierungstests und die daraus resultierenden Werte eines Gütemaßes. Um die Optimierung für kostensensitive und unausgeglichene Daten zu verbessern, haben wir ein vielversprechendes Gütemaß ausgewählt, parametrisiert und eine flexible Fehlersummierung basierend auf der Marge vorgeschlagen und implementiert. Wir konnten zeigen, dass die flexible, gewichtete Optimierung gut mit flexiblen, gewichteten SVM-Modellen harmoniert und generalisiert.

Für große Datensätze und bei der Optimierung vieler Parameter ist man bei der Anwendung von SVM's mit dem Problem der Rechenzeit konfrontiert. Wir haben dargestellt, welche Entwicklungen es auf dem Gebiet der Parallelverarbeitung bei Data-Mining-Methoden gab und aktuell gibt. Für Support-Vektor-Maschinen befinden sich die Arbeiten noch im Anfangsstadium. Wir haben unsere SVM-Implementierung auf dem JUMP-System parallelisiert. In Anlehnung an die Struktur des SMP-Systems wurde das SVM-Training im Shared-Memory-Modus parallelisiert. Dabei wurden sowohl OpenMP-Direktiven als auch ESSLSMP-Routinen verwendet, um die aufwändigsten Teile des Trainings zu parallelisieren. Die äußere Validierungsroutine arbeitet im MPI-Modus mit verteiltem Speicher. Während ein einzelnes Training nur auf einem SMP-Knoten läuft, kann die Validierung auch auf mehrere Knoten verteilt werden. Zusätzlich zu unserer hybrid-parallelen SVM-Software, stellt auch die *APPSPACK*-Software einen MPI-parallelen Modus zur Verfügung, der asynchron optimiert, indem mehrere Parametertupel gleichzeitig getestet werden. Wir haben diesen Modus an die hybrid-parallele SVM gekoppelt. Dazu musste das MPI-Kommunikationsschema von *APPSPACK* geändert werden, um zu verhindern, dass alle verfügbaren CPU's direkt von *APPSPACK* verbraucht werden, um Parametertupel zu testen.

Zu den von uns vorgestellten Methoden und Entwicklungen haben wir ausführliche Tests durchgeführt. Dabei spielten effizientes Training, kostensensitive Klassifikation, Parameteroptimierung und der Einsatz der parallelen Ebenen wichtige Rollen. Eine Auswahl interessanter Tests haben wir in Kapitel 7 zusammengestellt und interpretiert. Unsere Ideen, deren Umsetzung und Anwendung haben wir auf internationalen Workshops, Konferenzen und in Zeitschriften vorgestellt und diskutiert.

Neben den binären Klassifikationsaufgaben gibt es auch Anwendungen für multiple Klassifikation, bei der mehr als zwei Klassen zu erkennen sind. Auf dem Gebiet der SVM's werden multiple Probleme typischerweise über die Kombination mehrerer binärer Klassifikatoren gelöst. Daher ist die Implementierung einer Parameteroptimierung für den unausgeglichenen Mehrklassenfall eine interessante Erweiterung.

Anhang A

Ergänzungen zu Abschnitt 5.2

A.1 Bug Fixing

In einem unserer vielen Tests verursachte *APPSPACK* einen Fehler. Bei der Generierung eines Testpunktes kam es zur Division durch Null. Dieser Fehler setzte sich fort und einer der neuen Parameterwerte war dann `NotaNumber`. Bei der Suche nach der Ursache fanden wir den folgenden Bug in `solver.cpp`.

```
// correct violations of the constraints
for (int j=0;j<n;j++)
{
   if ((isLower[j])&&((lower[j]-x[j])>0))            <-- korr
   {
      if (lower[j]-x[j]>epsMach)                     <-- ok
      {
         tmpStep=(lower[j]-parentX[j])/direction[j];
         for (int k=0;k<n;k ++)
         x[k]=parentX[k]+tmpStep*direction[k];
      }
   else
      // nudge onto exact boundary
      x[j]=lower[j];
   }
   else if ((isUpper[j])&&((x[j]-upper[j])>0))       <-- korr
   {
      if ((x[j]-upper[j])>0)                         <-- bug
      {
         tmpStep=(upper[j]-parentX[j])/direction[j]; <-- error
         for (int k=0;k<n;k++)
         x[k]=parentX[k]+tmpStep*direction[k];
```

```
        }
        else
           // nudge onto exact boundary
           x[j]=upper[j];
     }
  }
```

Für jeden neu berechneten Parameter wird geprüft, ob er die vorgegebenen Schranken verletzt. Falls das der Fall ist, wird die Korrekturumgebung `korr` betreten. Die weitere Behandlung ist davon abhängig, wo genau der Wert liegt. Falls er in unmittelbarer Nähe der Schranke liegt (Maschinenepsilon), wird er mit dem Wert der Schranke überschrieben. Falls er weiter davon entfernt ist, wird eine Korrektur durchgeführt, bei der der Vaterpunkt `parentX` und die aktuelle Richtung `direction` wieder ins Spiel kommen. Im Fall `ok` stimmt der Code; im Fall `bug` fehlte jedoch das Maschinenepsilon `epsMach`. Das führte in unserem Test dazu, dass Parameter, die größer als die zulässige obere Schranke waren und für die `direction[j]=0` zutraf, Division durch Null erzeugten (`error`). Je nach Compiler und System führt das entweder zum Abbruch des Programms oder zu Fortpflanzungen von Fehlern. Wir haben das Maschinenepsilon eingefügt und alle Tests erneut durchgeführt, ohne dass dieser Fehler wieder auftrat.[57]

A.2 Kommunikation mit externem Programm

Damit *APPSPACK* das SVM-Gütemaß und somit die SVM selbst optimieren kann, müssen die folgenden Punkte erfüllt sein, die wir auch umgesetzt haben:

- Die SVM-Software liest einen von *APPSPACK* übermittelten Namen, öffnet die zugehörige Inputdatei und liest die darin enthaltenen Parameterwerte. Am Ende des Programms öffnet es eine Outputdatei und schreibt den Wert des Gütemaßes. Der Name dieser Datei wird ebenfalls von *APPSPACK* übermittelt.
- Damit *APPSPACK* die entsprechenden Inputdateien mit Inhalt füllen kann, muss der Nutzer eine `.apps` Datei anlegen, in der er Informationen [optional] über
  - Namen des aufzurufenden Programms,
  - Anzahl zu optimierender Parameter,
  - obere und untere Schranken für die Parameter,
  - [Skalierung],
  - [Startpunkt],

[57] Der Fehler wird in der nächsten Version der *APPSPACK*-Software behoben sein.

- [maximale Anzahl an Iterationen],
- [Abbruchgenauigkeit],
- ...

ablegt.

Anhang B

Ergänzungen zu Abschnitt 6.2.6

B.1 *APPSPACK* main.cpp

Im main.cpp haben wir neue MPI-Kommunikatoren aufgebaut. Bisher wurde der Standard-Kommunikator `MPI_COMM_WORLD` eingesetzt. Da jetzt die ursprünglichen Worker in zwei Klassen eingeteilt werden, benötigen wir zwei neue Kommunikatoren. Ein Zeilenkommunikator (`row_comm`) wird für die Kommunikation zwischen Master und den echten Workern benötigt. Die Slaves sind nicht in diesem Kommunikator enthalten. Ein Spaltenkommunikator
(`col_comm`) wird für die Kommunikation zwischen den Workern und den Slaves benötigt, wobei der Master diesen Kommunikator nicht nutzen darf, da er streng genommen nichts von der Existenz der Slaves weiß. Wir haben die folgenden Zeilen in den Code eingefügt.

```
int cols;        //number of workers
int rows;        //number of slaves per worker
int members;     //master and workers together
int col;
int col_comm;    //new column communicator
int row;
int row_comm;    //new row communicator
int number;      //master:0,worker:1,...,w
int slave;       //worker:0,slaves:1,...,c-1

cols=w;
rows=c;
members=cols+1;

//create row communicator
if (rank<members){
   row=0;}
```

```
  else{
     row=MPI_UNDEFINED;}
  MPI_Comm_split(MPI_COMM_WORLD,row,rank,&row_comm);
  //rank is the number in MPI_COMM_WORLD
  number=-1;
  if (rank<members) MPI_Comm_rank(row_comm,&number);

  //create column communicator
  if (rank>0){
     col=(rank-1)%cols;}
  else{
     col=MPI_UNDEFINED;}
  MPI_Comm_split(MPI_COMM_WORLD,col,rank,&col_comm);
  slave=-1;
  if (rank>0) MPI_Comm_rank(col_comm,&slave);
```

B.2 *APPSPACK* master.cpp

In der master.cpp Routine wird der neue Zeilenkommunikator benutzt, um Nachrichten ausschließlich an die Worker zu senden.

```
  for (int i=1;i<members;i++) //problem size
    MPI_Send(&n,1,MPI_INT,i,SIZE,row_comm);
[...]
  for (int i=1;i<members;i++) //quit message
    MPI_Send(&n,1,MPI_INT,i,QUIT,row_comm);
```

B.3 *APPSPACK* executor.cpp

Der Executor in der schon angepassten Form musste für unsere Zwecke nochmals geändert werden. Neue Argumente sind eingefügt worden. Der Executor sendet die Testpunkte nun ausschließlich zu den CPU's, die als Worker agieren.

```
  MPI_Send(&tag_in,1,MPI_INT,idx,XTAG,row_comm);
  MPI_Send(x,n,MPI_DOUBLE,idx,XVEC,row_comm);
[...]
  MPI_Iprobe(MPI_ANY_SOURCE,XTAG,row_comm,&flag,&mpiStatus);
[...]
  MPI_Recv(&tag,1,MPI_INT,source,XTAG,row_comm,&mpiStatus);
  MPI_Recv(&f,1,MPI_DOUBLE,source,FVAL,row_comm,&mpiStatus);
```

B.4 *APPSPACK* worker.cpp

In der modifizierten Routine, die *APPSPACK* zur Verfügung stellt, haben wir die C++ interne Funktion entfernt und durch einen Aufruf der F90 Funktion `feval` ersetzt.

```
#define feval __mainx_NMOD_feval //link to F90 source

extern "C" double feval(const int *n,const double *x,
const int *col_comm,const int *slave,const int *rows);
```

Die Worker in `row_comm` empfangen Nachrichten vom Master und dem Executor. Jeder Worker leitet die Nachrichten an seine Slaves unter der Nutzung von `col_comm` weiter. Die zentrale While-Schleife in der Routine muss modifiziert werden, um zu sichern, dass jeder Worker seine Slaves über das Ende der Optimierung informiert, bevor er selbst die Schleife verläßt. Dazu haben wir eine Variable `stop` definiert.

```
  int stop;
  if (number>0)
    MPI_Recv(&n,1,MPI_INT,0,SIZE,row_comm,&status);
  MPI_Bcast(&n,1,MPI_INT,0,col_comm);
[...]
  while (1)
  {
    if (number>0){
      MPI_Probe(0,MPI_ANY_TAG,row_comm,&status);
      msgtag=status.MPI_TAG;
      if (msgtag==QUIT) stop=1; //don't leave here
      for (int i=1;i<rows;i++)  //stop message
        MPI_Send(&stop,1,MPI_INT,i,0,col_comm);
      if (msgtag==QUIT) break;  //worker can leave
    }
    if (slave>0){
      MPI_Recv(&stop,1,MPI_INT,0,0,col_comm,&status);
      if (stop==1) break;
    }
    if (number>0){
      MPI_Recv(&tag,1,MPI_INT,0,XTAG,row_comm,&status);
      MPI_Recv(x,n,MPI_DOUBLE,0,XVEC,row_comm,&status);
    }
    //send point to other slaves
    MPI_Bcast(x,n,MPI_DOUBLE,0,col_comm);
[...]
    f=feval(&n,x,&col_comm,&slave,&rows);
    if (number>0){
```

```
      //send result to executor
      MPI_Send(&tag,1,MPI_INT,0,XTAG,row_comm);
      MPI_Send(&f,1,MPI_DOUBLE,0,FVAL,row_comm);
    }
  }
```

B.5 *APPSPACK* solver.cpp

Der *APPSPACK*-Löser selbst arbeitet nicht im parallelen Modus. Ausschließlich der Master generiert neue Testpunkte und hat Zugriff auf die internen Routinen und den Löser, sodass Parallelität nicht benötigt wird. Weitere Änderungen sind nicht notwendig.

Ein Bug-Fixing für den *APPSPACK*-Löser haben wir bereits im Abschnitt A.1 erwähnt.

B.6 SVM main.f90

Die Auswertung der SVM-Validierung erfolgt nicht mehr über einen externen Call. Wir haben unsere Fortran90 SVM-Software in eine Library umgewandelt. Das Hauptprogramm, in dem die Validierung aufgerufen wird, haben wir in eine Funktion.

```
feval(n,x,col_comm,slave,rows)
```

umgewandelt. Dabei sind `n` die Anzahl der zu optimierenden Parameter, `x` der aktuelle Parametervektor (*APPSPACK*-Testpunkt), `col_comm` der Spaltenkommunikator, `slave` der Rang im Spaltenkommunikator und `rows` die Anzahl der Slaves pro Validierung. Alle Daten werden von der C++ Routine `worker.cpp` übernommen. Im Fortran-Code wird in allen inneren Routinen nur noch mit dem neuen Kommunikator gearbeitet. Am Ende der Validierung sammelt der Worker die lokalen Ergebnisse der Slaves zusammen und berechnet zusammen mit dem eigenen Ergebnis das Gütemaß, welches dann dem C++ Teil über den Funktionswert `feval` übergeben wird. Der Worker sendet das Ergebnis dann an den Master weiter (siehe worker.cpp).

B.7 *APPSPACK* Kommandozeile und Initialisierung

In der ursprünglichen *APPSPACK*-Software wird, wie im Abschnitt A.2 beschrieben, eine `.apps` Datei angelegt, um die Kommunikation zwischen *APPSPACK* und dem externen Executable ohne Änderungen des Codes zu ermöglichen. In unserer modifizierten Version wird dieses Schema nicht mehr benötigt. Der Master liest die notwendigen Informationen direkt in einer Parameterdatei. Festgelegt werden beispielsweise die Anzahl der zu

optimierenden Parameter, der Startpunkt, die Schranken, die Anzahl der Iterationen, das Debug-Level und weitere Parameter. Für die gekoppelt parallele Version haben wir die Kommandozeile um neue Argumente erweitert. Über `-cols <int>` lesen wir die Anzahl der Worker und über `-rows <int>` die Gesamtzahl an Slaves pro Worker ein. Das ermöglicht eine flexible Nutzung des Programms.

```
//read number of workers
sscanf(argv[2],"%d",&cols);
//read number of slaves for validation
sscanf(argv[4],"%d",&rows);
```

B.8 Loadleveler-Script

Um die dreistufig parallele Software auf dem JUMP-System zu starten, muss ein Loadleveler-Script geschrieben werden. Ein Beispiel aus unseren Tests ist:

```
# created : Thur Jul 06 13:07:59 MEZ 2006
# job-ensemble-2.5.3.sh - LoadLeveler CMD File
# @ job_type = parallel
# @ node = 1
# @ tasks_per_node = 5                                          (**)
# @ resources = ConsumableCpus(4)ConsumableMemory(14GB)         (**)
# @ wall_clock_limit = 01:00:00
# @ notification = always
# @ notify_user = t.eitrich@fz-juelich.de
# @ initialdir = ~/appspack/ensemble/tests-2.5
# @ output = out.$(schedd_host).$(jobid).$(stepid)
# @ error = err.$(schedd_host).$(jobid).$(stepid)
# @ restart = yes
# @ queue export OMP_NUM_THREADS=4
poe ./appspack -cols 2 -rows 2                                   (*)
```

Dabei wird eine Optimierung für den *ensemble* Datensatz gestartet (siehe Kapitel 7). Das Script zeigt einen Nachteil unserer Software, der aber hauptsächlich durch das Master-Worker-Prinzip von *APPSPACK* entsteht. Wir fordern zwei Worker an, die insgesamt noch jeweils einen Slave bekommen, sodass jede Validierung von 2 CPU's durchgeführt wird. Das sind zusammen 4 CPU's ($\star$). Jede CPU soll mit 4 Threads trainieren. Zusammen mit dem Master brauchen wir also $16 + 1 = 17$ CPU's, fordern aber 20 an ($\star\star$). Der Master bekommt die Threads ebenfalls zugewiesen, diese liegen aber brach, denn der Master hat innerhalb von *APPSPACK* neben der Definition von Punkten, Kommunikation und Auswertung keine Aufgaben. Auf der Ebene der Validierung haben wir dafür gesorgt, dass sich dieses Problem bei den Workern nicht fortpflanzt, d.h. jeder Worker der Validierung

nimmt auch aktiv am Training teil, anstatt nur auf Ergebnisse zu warten. Eine Änderung der *APPSPACK*-Software in dieser Hinsicht liegt nicht in unserer Hand, sodass wir die Problematik offen lassen. Allgemein ist eine optimierte Nutzung schwierig. Beispielsweise könnten wir einen JUMP-Knoten (32 CPU's) gut aufteilen in 4 Worker mit je 3 zusätzlichen Slaves, die alle jeweils mit 1 weiteren Thread trainieren. Der Master verhindert das leider. Strukturell ist das eine Schwachstelle, über die man sich in Zukunft Gedanken machen sollte.

Anhang C

Ergänzungen zu den Abschnitten 7.2.4 und 7.2.5

Im Abschnitt 7.2 hatten wir uns mit dem Problem der Wahl einer Arbeitsmengengröße beschäftigt. Für kleine Datensätze ist diese Problematik vernachlässigbar, gewinnt aber mit wachsender Anzahl an Trainingsdaten stark an Bedeutung. Die drei ineinander verschachtelten Löser mit ihren Optimalitätskriterien beeinflussen sich gegenseitig sehr stark und führen zu einer komplizierten Optimierungsstruktur. In Ergänzung zu den bisherigen Tests haben wir weitere Versuche durchgeführt. Für die zwei Datensätze *astro* und *w8a* haben wir in Tabelle 7.13 schon erste Ergebnisse gezeigt. Wir haben weitere Tests nach folgendem Schema durchgeführt. Neben der Größe der Arbeitsmenge ws haben wir auch die Anzahl der Trainingsdaten stark variiert, sodass nun direkte Vergleiche möglich sind. Bei den in Kapitel 7 durchgeführten Tests mit verschiedenen Datensätzen ist ein Vergleich schwer, da auch die Anzahl der Variablen, die Werte der Parameter und weitere Merkmale des Klassifikationsproblems die Trainingszeit stark beeinflussen.

Für den mittelgroßen *astro* Datensatz mit wenigen Variablen verhielten sich die Trainingszeiten für verschiedene Trainingsgrößen ähnlich (Tabelle C.1). Bei allen Tests lag das gesuchte Minimum innerhalb des Intervalls [0,100]. Für sehr kleine Arbeitsmengen wuchs die Zeit wie erwartet drastisch an.

Für den großen *w8a* Datensatz mit vielen Variablen verhielten sich die Trainingszeiten nicht so gleichmäßig (Tabelle C.2). Es sind Schwankungen in Abhängigkeit von der Größe der Arbeitsmenge zu beobachten. Bei den Tests mit 5000 und 10000 Trainingsdaten lag das gesuchte Minimum innerhalb des Intervalls $ws \in [500, 1000]$, bei den Tests mit 15000 bis 30000 Trainingsdaten im Intervall $ws \in [1000, 1500]$ und für 40000 Punkte im Intervall $ws \in [1500, 2000]$. Tendenziell lohnen sich große Arbeitsmengen bei diesem Datensatz.

Wir schlussfolgern, dass die Datensatzgröße und die Anzahl der Variablen Einfluß auf eine sinnvolle Wahl von ws haben. Eine Empfehlung kann allgemein nicht gegeben werden.

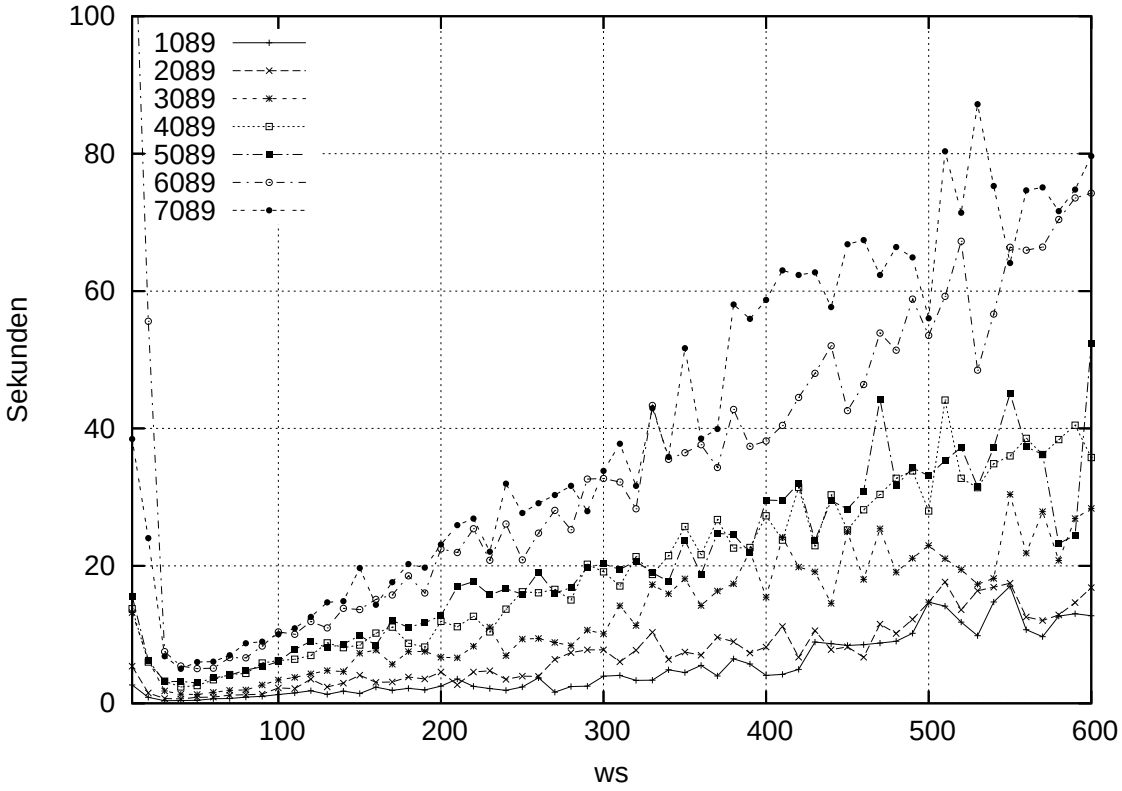

Abbildung C.1: Abhängigkeit der Trainingszeit von der Größe der Arbeitsmenge (*astro* Datensatz mit variierender Größe).

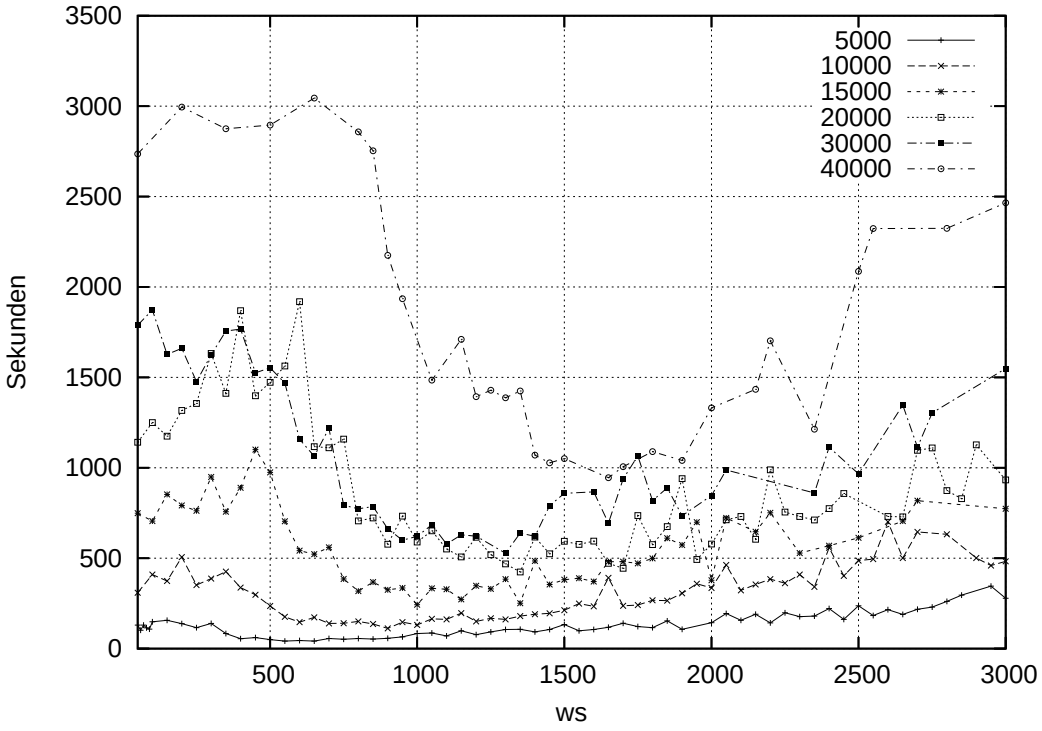

Abbildung C.2: Abhängigkeit der Trainingszeit von der Größe der Arbeitsmenge (*w8a* Datensatz mit variierender Größe).

Anhang D

Notationen

D.1 Allgemein

$[a,b] \leq [c,d]$	$\forall x \in [a,b] : x \leq c$
$\mathbf{1}$	Indikatorfunktion
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}_+$	Menge der positiven reellen Zahlen (ohne Null)
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{N}$	Menge der natürlichen Zahlen (mit Null)
$\mathbf{1}$	Vektor, der nur aus Einsen besteht
$\mathbf{0}$	Vektor, der nur aus Nullen besteht
∇	Gradient
n	Anzahl an Variablen
f	Klassifikationsfunktion
h	Hypothesenfunktion
$\delta_{i,j}$	Kroneckersymbol mit $\delta_{i,j} = 1 \Leftrightarrow i = j \quad (i,j \in \mathbb{Z})$
$\mathcal{SV}$	Menge der Support-Vektoren $\Rightarrow \alpha_i > 0$
$\mathcal{FSV}$	Menge der freien Support-Vektoren $\Rightarrow \alpha_i \in (0, C_i)$
$\mathcal{BSV}$	Menge der beschränkten Support-Vektoren $\Rightarrow \alpha_i = C_i$
$\mathcal{M}$	hochdimensionaler Merkmalsraum
sv	Anzahl der Support-Vektoren $\Rightarrow \alpha_i > 0$
b	Schwellwert der linearen Zielfunktion f
b_r	robuster Schwellwert der linearen Zielfunktion f
k^G	Gauß-Kern
k^P	Polynomialkern
k^S	Slater-Kern
k^T	Taniomoto-Kern
$k(\cdot,\cdot)$	Kernfunktion

$\phi(\cdot)$	implizite Datentransformationsfunktion
$\sigma,\ c,\ d$	Kernparameter
k_a	verallgemeinerter Kern
k_a^G	verallgemeinerter Gauß-Kern
k_a^P	verallgemeinerter Polynomialkern
k_a^S	verallgemeinerter Slater-Kern
$I_{ij}(\cdot,\cdot)$	Zählfunktion des Tanimoto-Kerns
$\boldsymbol{\xi}$	Schlupfvariablen im Softmargin-Modell
$\boldsymbol{x}$	Datenpunkt allgemein
y	Klassenlabel allgemein
$\boldsymbol{x}^i$	i-ter Trainingspunkt
$(\boldsymbol{x}^i, y_i)$	i-tes Trainingspaar
$\boldsymbol{w}$	Vektor der primalen linearen Zielfunktion
C	Strafparameter
C^+	Strafparameter für positive Punkte
C^-	Strafparameter für negative Punkte
$\boldsymbol{C}$	Strafvektor für die Nebenbedingung von Softmargin-Aufgaben

D.2 Zerlegungsalgorithmus

ws	Anzahl der Punkte in einem Working-Set
$\mathcal{I}$	Menge zur Bestimmung von Working-Set-Paaren
$\mathcal{J}$	Menge zur Bestimmung von Working-Set-Paaren
$\mathcal{D}$	Menge aktiver Punkte
$\mathcal{F}$	Menge inaktiver Punkte
$\boldsymbol{\alpha}$	Vektor von Lagrange-Multiplikatoren allgemein
$\boldsymbol{\lambda},\ \boldsymbol{\mu},\ \nu$	Multiplikatoren für KKT-Bedingungen
$\boldsymbol{s}$	Abstiegsrichtung allgemein
k	Iterationsindex für Decompositionschritte
W	Zielfunktion der dualen Aufgabe, Maximierungsproblem
W^-	Zielfunktion der dualen Aufgabe, Minimierungsproblem
Ω	Bereich zulässiger Punkte einer Optimierungsaufgabe
$\boldsymbol{K}$	Kernmatrix allgemein
$\boldsymbol{Q}$	Grammatrix der globalen SVM-Aufgabe
$\boldsymbol{v}$	Vektor beim Verfahren von Zoutendijk
$n_{\text{akt}},\ n_{\min},\ n_{\max}$	VVP-Parameter
$\boldsymbol{\alpha}_{\text{d}}^k$	Vektor der Lagrange-Multiplikatoren aller Punkte im k-ten Working-Set
$\boldsymbol{Q}_{\text{dd}}$	Grammatrix des aktuellen Subproblems
$\boldsymbol{y}_{\text{d}}^k$	Klassenvektor der im k-ten Decompositionschritt betrachteten Punkte
k'	Iterationsindex für VVP-Schritte bei festem k

$\boldsymbol{q}$	Vektor im Teilproblem
e	Konstante im Teilproblem
ρ	Projektionsparameter bei VVPM
$\rho_{\min}$	untere Grenze für ρ
$\rho_{\max}$	obere Grenze für ρ
W_{d}^{-}	Zielfunktion der dualen Aufgabe in einem QP-Teilproblem
θ_u	unterer Wert zur Definition des schlechten Abstiegs
θ_o	oberer Wert zur Definition des schlechten Abstiegs
$\boldsymbol{I}_{ws}$	Einheitsmatrix der Größe ws
$\boldsymbol{I}_{\rho}^{k}$	spezielle Diagonalmatrix bei VVPM
$\boldsymbol{t}$	Vektor für VVPM
$\boldsymbol{C}_{\mathrm{d}}$	Strafvektor für das aktuelle Working-Set
g	Funktion zur Berechnung von Schrittweite θ
$\tilde{\boldsymbol{s}}$	Vektor bei VVPM
$\boldsymbol{P}$	Projektionsmatrix allgemein
ι	Wechselindikator bei VVPM
θ	Schrittweitenparameter allgemein
IP	Bruchstellen bei Pardalos
L	Linke Intervallgrenze bei Pardalos
R	Rechte Intervallgrenze bei Pardalos
M	Median bei Pardalos
$\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$	interne Vektoren bei Pardalos
d	interne Variable bei Pardalos
$\hat{\boldsymbol{u}}$	Vektor im inneren Teilproblem
$\boldsymbol{z}$	Vektor beim Pardalos-Löser
κ	Lösungspunkt bei Pardalos

D.3 Modifikationen und Parameteroptimierung

rd	Anzahl der Referenzdaten
rd^{+}	Anzahl positiver Referenz- oder Trainingspunkte
rd^{-}	Anzahl negativer Referenz- oder Trainingspunkte
$\mathcal{RD}$	Menge der Referenzdaten bzw. der Trainingsdaten zur Validierung
td	Anzahl der unabhängigen Testdaten pro Test
$\mathcal{TD}$	Menge von Testdaten
pp	Anzahl positiver Punkte in den Testdaten
np	Anzahl negativer Punkte in den Testdaten
fk	Anzahl falsch klassifizierter Punkte in den Testdaten
rp	Anzahl richtig positiver Punkte in den Testdaten
fp	Anzahl falsch positiver Punkte in den Testdaten

rn	Anzahl richtig negativer Punkte in den Testdaten
fn	Anzahl falsch negativer Punkte in den Testdaten
m	Anzahl der betrachteten SVM-Parameter
Λ	Gewichtungsfaktor in einem (C^+, C^-)-Modell bei Beachtung der Klassen
ac	Genauigkeit, *accuracy*
se	Sensitivität, *sensitivity*
sp	Spezifität, *specificity*
pr	Präzision, *precision*
fm	F-Maß
β	Faktor zur Variabilität des F-Maßes
fm_β	variables F-Maß
$\tilde{fm}_\beta$	variables weiches F-Maß
ef	Anreicherungsfaktor
ω_{rp}	Kosten für richtig positiven Punkt
ω_{fp}	Kosten für falsch positiven Punkt
ω_{rn}	Kosten für richtig negativen Punkt
ω_{fn}	Kosten für falsch negativen Punkt
$\boldsymbol{KFM}$	Konfusionsmatrix
$\boldsymbol{KSM}$	Kostenmatrix
$\boldsymbol{T}$	Menge von *APPS*-Testpunkten
$\mathcal{R}$	Indexmenge der Suchrichtungen zur Parameteroptimierung
$\boldsymbol{r}$	Suchrichtung bei *APPS*
$\boldsymbol{su}$	Indexvektor der Suchrichtungen bei *APPS*
$\boldsymbol{\tau}$	Statusvektor für *APPS*-Testpunkte
$\boldsymbol{u}$	untere Schranke zur Parameteroptimierung
$\boldsymbol{o}$	obere Schranke zur Parameteroptimierung
$\boldsymbol{N}$	Marke für *APPS*-Testpunkte
$\boldsymbol{\pi}$	SVM-Parametervektor
mg	Marge
mg^+	Marge zur Klasse 1 hin
mg^-	Marge zur Klasse -1 hin
$h_{\boldsymbol{\pi}}$	Hypothesenfunktion (für ein Parameterbündel $\boldsymbol{\pi}$)
$f_{\boldsymbol{\pi}}$	Zielfunktion (für ein Parameterbündel $\boldsymbol{\pi}$)
$\Phi(\cdot)$	Gütefunktion zur Parameteroptimierung
$\boldsymbol{\Delta}$	Schrittweitenvektor bei *APPSPACK*
ς	*APPSPACK*-Parameter

Literaturverzeichnis

[1] S. Abe. *Support vector machines for pattern classification*. Springer, 2005.

[2] R. Agarwal and M. V. Joshi. PNrule: a new framework for learning classifier models in data mining (a case-study in network intrusion detection). Technical Report TR 00-015, University of Minnesota, Dept. of Computer Science, 2000.

[3] R. Akbani, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Proceedings of the 15th European Conference on Machine Learning (ECML 2004), Pisa, Italy, 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2004.

[4] P. J. Angeline. *Evolutionary algorithms and emergent intelligence*. PhD thesis, Ohio State University, 1993.

[5] A. V. Anghelescu and I. B. Muchnik. Optimization of SVM in a space of two parameters: weak margin and intercept - application in text classifier design, 2003. online.

[6] C. Audet and J. E. Dennis. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[7] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8:141–148, 1988.

[8] A. Basermann. Iterative Verfahren für dünnbesetzte Matrizen zur Lösung technischer Probleme auf massiv parallelen Systemen. Berichte des Forschungszentrums Jülich JUEL-3015, Forschungszentrum Jülich, 1995.

[9] M. S. Bazaraa and C. M. Shetty. *Nonlinear programming – theory and algorithms*. John Wiley & Sons, New York, 1979.

[10] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, Belmont, MA, 1995.

[11] E. G. Birgin, J. M. Martinez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[12] C. M. Bishop. *Neural networks for pattern recognition*. Clarendon Press, Oxford, 1995.

[13] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, New York, 1999.

[14] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[15] P. Brucker. An O(n) algorithm for quadratic knapsack problems. *Operations Research Letters*, 3(3):163–166, 1984.

[16] P. B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proceedings of the 34th Symp. Foundations of Computer Science*, pages 332–340. IEEE Computer Society Press, 1993.

[17] A. Cannon, L. Cowen, and C. E. Priebe. Approximate distance classification. In *Proceedings of the Symposium on the Interface between Computer Science and Statistics*, 1998. online http://citeseer.ist.psu.edu/cannon98approximate.html.

[18] S. Celis and D. R. Musicant. Weka-parallel: machine learning in parallel. Computer Science Technical Report 2002b, Carleton College, 2002.

[19] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

[20] O. Chapelle, V. N. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

[21] N. V. Chawla, N. Japkowicz, and A. Kolcz. Editorial: special issue on learning from imbalanced data sets. *SIGKDD Explorations*, 6(1):1–6, 2004.

[22] N. Chen, W. Lu, J. Yang, and G. Li. *Support vector machine in chemistry*. World Scientific Pub Co Inc, 2004.

[23] R. Collobert and S. Bengio. SVMTorch: a support vector machine for large-scale regression and classification problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

[24] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

[25] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334, 1965.

[26] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, UK, 2000.

[27] Y. H. Dai and R. Fletcher. New algorithms for singly linearly constrained quadratic programming problems subject to lower and upper bounds. *Math. Prog.*, 106(3):403–421, 2006.

[28] J. Dennis and V. Torczon. Derivative-free pattern search methods for multidisciplinary design problems. In *The Fifth AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 922–932, American Institute of Aeronautics and Astronautics, Reston, VA, 1994.

[29] U. Detert. Introduction to the JUMP architecture, 2004. online http://jumpdoc.fz-juelich.de.

[30] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In M. Zaki and C. Ho, editors, *Large-Scale Parallel Data Mining*, volume 1759 of *Lecture Notes in Computer Science*, pages 245–260, 2000.

[31] T. Dickhaus. Statistische Verfahren für das Data Mining in einem Industrieprojekt. Technical Report FZJ-ZAM-IB-2003-08, Research Centre Jülich, 2003.

[32] J.-X. Dong, A. Krzyzak, and C. Y. Suen. A fast parallel optimization for training support vector machines. In P. Perner and A. Rosenfeld, editors, *Proceedings of 3rd International Conference on Machine Learning and Data Mining*, volume 2734 of *Lecture Notes in Computer Science*, pages 96–105. Springer, 2003.

[33] Y.-S. Dong and K.-S. Han. Boosting SVM classifiers by ensemble. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1072–1073. ACM Press, 2005.

[34] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14(1):1–17, 1988.

[35] V. Dvorak and P. Matousek. Highly efficient parallel ANN implementation for real-time processing. In *Proceedings of Conference of Computer Engineering and Informatics CE&I 1999*, pages 186–191. Faculty of Electrical Engineering and Informatics, University of Technology Kosice, 1999.

[36] T. Eitrich. Support-Vektor-Maschinen und ihre Anwendung auf Datensätze aus der Forschung. Berichte des Forschungszentrums Jülich JUEL-4096, Forschungszentrum Jülich, 2003.

[37] T. Eitrich, W. Frings, and B. Lang. HyParSVM – a new hybrid parallel software for support vector machine learning on SMP clusters. In W. E. Nagel, W. V. Walter, and W. Lehner, editors, *Euro-Par 2006, Parallel Processing, 12th International Euro-Par Conference*, volume 4128 of *LNCS*, pages 350–359. Springer, 2006.

[38] T. Eitrich, A. Kless, C. Druska, J. Grotendorst, and W. Meyer. Classification of highly unbalanced CYP450 data of drugs using cost sensitive machine learning techniques. *Journal of Chemical Information and Modeling*, 47(1):92–103, 2007.

[39] T. Eitrich and B. Lang. Analysis of support vector machine training costs for large and unbalanced data from pharmaceutical industry. In A. Aboshosha, editor, *Proceedings of the International Conference on Artificial Intelligence and Machine Learning (AIML 2005), Cairo, Egypt*, pages 58–64. ICGST, 2005.

[40] T. Eitrich and B. Lang. Parallel tuning of support vector machine learning parameters for large and unbalanced data sets. In M. R. Berthold, R. Glen, K. Diederichs, O. Kohlbacher, and I. Fischer, editors, *Computational Life Sciences, First International Symposium (CompLife 2005), Konstanz, Germany*, volume 3695 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 2005.

[41] T. Eitrich and B. Lang. Data mining with parallel support vector machines for classification. In T. Yakhno and E. Neuhold, editors, *ADVIS 2006*, volume 4243 of *LNCS*, pages 197–206. Springer, 2006.

[42] T. Eitrich and B. Lang. Efficient implementation of serial and parallel support vector machine training with a multi-parameter kernel for large-scale data mining. In *Proceedings of the 11. International Conference on Computer Science (ICCS), February 24-26, 2006, Prague, Czech Republic*, pages 6–11, 2006.

[43] T. Eitrich and B. Lang. Efficient optimization of support vector machine learning parameters for unbalanced datasets. *Journal of Computational and Applied Mathematics*, 196:425–436, 2006.

[44] T. Eitrich and B. Lang. On the advantages of weighted L_1-norm support vector learning for unbalanced binary classification problems. In *Proceedings of the IEEE Conference on Intelligent Systems (IS), London*, pages 575–580. IEEE Computer Society Press, 2006.

[45] T. Eitrich and B. Lang. On the efficient implementation of a serial and parallel decomposition algorithm for fast support vector machine training including a multi-parameter kernel. *International Journal of Computational Intelligence*, 3(2):91–98, 2006.

[46] T. Eitrich and B. Lang. On the optimal working set size in serial and parallel support vector machine learning with the decomposition algorithm. In P. Christen, P. Kennedy, J. Li, S. Simoff, and G. Williams, editors, *Australasian Data Mining Conference, Sydney (AusDM2006)*, volume 61 of *CRPIT*, pages 121–128. ACS, 2006.

[47] T. Eitrich and B. Lang. Parallel cost-sensitive support vector machine software for classification. In J. Meinke, O. Zimmermann, S. Mohanty, and U. Hansmann, editors, *Proceedings of the Workshop From Computational Biophysics to Systems Biology, John von Neumann Institute for Computing, Jülich, NIC Series Vol. 34*, pages 141–144, 2006.

[48] T. Eitrich, B. Lang, and A. Streit. Customizing the APPSPACK software for parallel parameter tuning of a hybrid parallel support vector machine. In G. D. Fatta, M. R. Berthold, and S. Parthasarathy, editors, *Proceedings of the Workshop on Parallel Data Mining, ECML, Berlin*, pages 38–50. ECML/PKDD, 2006. online http://www.ecmlpkdd2006.org/ws-pdm.pdf.

[49] T. Fawcett. ROC graphs: notes and practical considerations for researchers. Technical Report HPL-2003-4, HP Labs, 2003.

[50] R. Fletcher. On the Barzilai-Borwein method. Research report, University of Dundee, Dept. of Mathematics, 2001.

[51] D. J. Garcia, O. Hall, D. B. Goldgof, and K. Kramer. A parallel feature selection algorithm from random subsets. In G. D. Fatta, M. R. Berthold, and S. Parthasarathy, editors, *Proceedings of the Workshop on Parallel Data Mining, ECML, Berlin*, pages 64–75. ECML/PKDD, 2006. online http://www.ecmlpkdd2006.org/ws-pdm.pdf.

[52] C. Geiger and C. Kanzow. *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben*. Springer, Berlin, 1999.

[53] C. Geiger and C. Kanzow. *Theorie und Numerik restringierter Optimierungsaufgaben*. Springer, Berlin, 2002.

[54] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: the cascade SVM. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 521–528. MIT Press, Cambridge, MA, 2005.

[55] G. A. Gray and T. G. Kolda. APPSPACK 4.0: asynchronous parallel pattern search for derivative-free optimization. Sandia Report SAND2004-6391, Sandia National Laboratories, Livermore, CA, 2004.

[56] G. A. Gray and T. G. Kolda. Algorithm 8xx: APPSPACK 4.0: asynchronous parallel pattern search for derivative-free optimization. *ACM Transactions on Mathematical Software*, 32(3), 2006.

[57] C. Großmann and J. Terno. *Numerik der Optimierung*. B. G. Teubner, Stuttgart, 1997.

[58] M. K. Gupta and D. A. Padua. Effects of program parallelization and stripmining transformation on cache performance in a multiprocessor. In *Proceedings of the International Conference on Parallel Processing (ICPP '91), Texas, USA*, pages 301–304. CRC Press, 1991.

[59] I. Gutheil. Performance of single-processor BLAS on IBM p690. Technical Report FZJ-ZAM-IB-2004-08, Research Centre Jülich, 2004.

[60] L. O. Hall, K. W. Bowyer, R. E. Banfield, D. Bhadoria, W. P. Kegelmeyer, and S. Eschrich. Comparing pure parallel ensemble creation techniques against bagging. In *Proceedings of the 3rd IEEE Conference on Data Mining (ICDM 2003), Melbourne, Florida*, pages 533–536. IEEE Computer Society Press, 2003.

[61] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2001.

[62] D. M. Hawkins. The problem of overfitting. *J. Chem. Inf. Comput. Sci.*, 44:1–12, 2004.

[63] S. Hettich, C. L. Blake, and C. J. Merz. *UCI repository of machine learning databases*, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[64] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.

[65] T. K. Ho and E. M. Kleinberg. Building projectable classifiers of arbitrary complexity. In M. E. Kavanaugh and B. Werner, editors, *Proc. of the 13th Int. Conf. on Pattern Recognition, Vienna, Austria*, pages 880–885. IEEE Computer Society Press, 1996.

[66] P. D. Hough, T. G. Kolda, and V. J. Torczon. Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing*, 23(1):134–156, 2001.

[67] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2003. http://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf.

[68] T. Huckle and S. Schneider. *Numerische Methoden*. Springer, 2006.

[69] IBM. ESSL - Engineering and Scientific Subroutine Library for AIX version 4.1.

[70] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[71] A. Jakulin, M. Mozina, J. Demsar, I. Bratko, and B. Zupan. Nomograms for visualizing support vector machines. In *Proceeding of the 11. ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD'05)*, pages 108–117. ACM Press, 2005.

[72] R. Jin, G. Yang, and G. Agrawal. Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):71–89, 2005.

[73] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 169–185. MIT Press, 1998.

[74] T. Joachims. SVM-light support vector machine, 2004. online http://svmlight.joachims.org/.

[75] M. V. Joshi, R. C. Agarwal, and V. Kumar. Predicting rare classes: can boosting make any weak learner strong? In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Canada*, pages 297–306. ACM Press, 2002.

[76] M. V. Joshi, G. Karypis, and V. Kumar. ScalParC: a new scalable and efficient parallel classification algorithm for mining large datasets. In *IPPS: 11th International Parallel Processing Symposium*, pages 573–579. IEEE Computer Society Press, 1998.

[77] S. Kantabutra and A. L. Couch. Parallel k-means clustering algorithm on NOWs. *NECTEC Technical Journal*, 1:243–248, 2000.

[78] G. Karakoulas and J. Shawe-Taylor. Optimizing classifiers for imbalanced training sets. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 253–259. MIT Press, 1999.

[79] S. S. Keerthi. Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms. *IEEE Transactions on Neural Networks*, 13:1225–1229, 2002.

[80] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13:637–649, 2001.

[81] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks, Perth, Australia*, pages 1942–1948. IEEE Computer Society Press, 1995.

[82] A. Kless and T. Eitrich. Cytochrome P450 classification of drugs with support vector machines implementing the nearest point algorithm. In J. A. López, E. Benfenati, and W. Dubitzky, editors, *Knowledge Exploration in Life Science Informatics, International Symposium (KELSI 2004), Milan, Italy*, volume 3303 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2004.

[83] A. Kless and T. Eitrich. Cytochrome P450 classification of drugs with maximum entropy methods, 2006. submitted, Poster presented at the 16. European Symposium on QSAR and Molecular Modelling, Italy.

[84] A. Kless, T. Eitrich, W. Meyer, and J. Grotendorst. Data Mining in F&E. *BioWorld: Magazin für Molekularbiologische und Biotechnologische Applikationen*, 9(2):22–23, 2004.

[85] A. Kless and J. Grotendorst, editors. *GALA Grünenthal Applied Life Science Analysis*. NIC series 30. John von Neumann Institute for Computing, 2006.

[86] T. G. Kolda. Revisiting asynchronous parallel pattern search for nonlinear optimization. Technical Report SAND2004-8055, Sandia National Laboratories, Livermore, CA 94551, 2004.

[87] J. M. Kriegl, T. Arnhold, B. Beck, and T. Fox. Prediction of human cytochrome P450 inhibition using support vector machines. *QSAR Comb. Sci.*, 24:491–502, 2005.

[88] J. M. Kriegl, T. Arnhold, B. Beck, and T. Fox. A support vector machine approach to classify human cytochrome P450 3A4 inhibitors. *Journal of Computer-Aided Molecular Design*, 19:189–201, 2005.

[89] D. Kun, L. Yih, and A. Perera. Parallel SMO for training support vector machines, 2003. SMA 5505 Project Final Report.

[90] G. R. G. Lanckriet, M. Deng, N. Cristianini, M. I. Jordan, and W. S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 300–311. World Scientific Pub Co Inc, 2004.

[91] P. Laskov. Feasible direction decomposition algorithms for training support vector machines. *Machine Learning*, 46(1-3):315–349, 2002.

[92] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Softw.*, 5(3):308–323, 1979.

[93] A. Lazarevic and Z. Obradovic. The distributed boosting algorithm. In *KDD 2001: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*, pages 311–316, New York, NY, USA, 2001. ACM Press.

[94] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 246–254, Seattle, Washington, 1995. ACM Press.

[95] S. Leyffer. The return of the active set method. *Oberwolfach Report*, 2(1), 2005.

[96] C.-J. Lin. Linear convergence of a decomposition method for support vector machines. Technical report, National Taiwan University, Dept. of Computer Science and Information Engineering, 2001. http://www.csie.ntu.edu.tw/∼cjlin/papers/linearconv.pdf.

[97] C.-J. Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

[98] A. Y. Liu. The effect of oversampling and undersampling on classifying imbalanced text datasets. Master's thesis, University of Texas at Austin, 2004.

[99] M. A. Maloof. Learning when data sets are imbalanced and when costs are unequal and unknown. In *Workshop on Learning from Imbalanced Data Sets II, ICML, Washington DC*, 2003. online http://www.site.uottawa.ca/ nat/Workshop2003/maloof-icml03-wids.pdf.

[100] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23:1–18, 1990.

[101] F. Markowetz. Support vector machines in bioinformatics. Master's thesis, University of Heidelberg, 2001.

[102] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London A*, 209:415–446, 1909.

[103] C. Merkwirth, H. Mauser, T. Schulz-Gasch, O. Roche, M. Stahl, and T. Lengauer. Ensemble methods for classification in cheminformatics. *J. Chem. Inf. Comput. Sci.*, 44:1971–1978, 2004.

[104] R. Mettu. *Approximation algorithms for NP-hard clustering problems*. PhD thesis, Department of Computer Science, University of Texas at Austin, 2002.

[105] M. Momma and K. P. Bennett. A pattern search method for model selection of support vector regression. In R. L. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors, *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*. SIAM, 2002.

[106] K. Morik, P. Brockhausen, and T. Joachims. Combining statistical learning with a knowledge-based approach - a case study in intensive care monitoring. In I. Bratko and S. Dzeroski, editors, *Proc. 16th International Conf. on Machine Learning*, pages 268–277. Morgan Kaufmann, San Francisco, CA, 1999.

[107] B. A. Murtagh and M. A. Saunders. MINOS 5.4 user's guide. *Journal of Global Optimization*, 1995.

[108] D. R. Musicant, V. Kumar, and A. Ozgur. Optimizing F-measure with support vector machines. In I. Russell and S. M. Haller, editors, *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference, May 12-14, 2003, St. Augustine, Florida, USA*, pages 356–360. AAAI Press, 2003.

[109] S. S. Nielsen and S. A. Zenios. Massively parallel algorithms for singly constrained convex programs. *ORSA Journal on Computing*, 4(2):166–181, 1992.

[110] C. S. Nunes, T. Mendonca, P. Amorim, D. L. Ferreira, and L. Antunes. Stochastic and neuro-fuzzy modelling for predicting return of consciousness after general anaesthesia. In *Proceedings of the 3rd European Symposium on Intelligent Technologies, Hybrid Systems and their Implementation on Smart Adaptive Systems (EUNITE 2004, Aachen, Germany)*, 2004.

[111] M. Oppel. *Quantenchemische und quantendynamische Rechnungen zur Schwingungsanregung und Photodissoziation von $\mathbb{HNO}_3$ durch ultrakurze Laserpulse*. PhD thesis, Freie Universität Berlin, Fachbereich Chemie, 1998.

[112] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *IEEE Workshop on Neural Networks and Signal Processing*, pages 276–285. IEEE Computer Society Press, 1997.

[113] E. Osuna, R. Freund, and F. Girosi. Support vector machines: training and applications. AI Memo 1602, Massachusetts Institute of Technology, 1997.

[114] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 130–136. IEEE Computer Society Press, 1997.

[115] U. Paquet and A. P. Engelbrecht. Training support vector machines with particle swarms. In *International Joint Conference on Neural Networks, Portland*, pages 1593–1598. IEEE Computer Society Press, 2003.

[116] P. M. Pardalos and N. Kovoor. An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Mathematical Programming*, 46(3):321–328, 1990.

[117] S. Parthasarathy, M. Zaki, M. Ogihara, and W. Li. Parallel data mining for association rules on shared-memory systems. *Knowledge and Information Systems*, 3(1):1–29, 2001.

[118] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[119] F. Poulet. Multi-way distributed SVM algorithms. In *Parallel and Distributed Computing for Machine Learning, Workshop in Conjunction with the 14th European Conference on Machine Learning (ECML03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD03)*, 2003. online.

[120] F. Provost. Learning with imbalanced data sets 101. In *Proceedings of the AAAI Workshop on Imbalanced Data Sets (invited paper)*. AAAI Press, 2000. AAAI Technical Report WS-00-05.

[121] S. Qiu and T. Lane. Parallel computation of RBF kernels for support vector classifiers. In *SIAM International Conference on Data Mining, Newport Beach, USA*. SIAM, 2005.

[122] J. R. Quinlan. Simplifying decision trees. In B. Gaines and J. Boose, editors, *Knowledge Acquisition for Knowledge-Based Systems*, pages 239–252. Academic Press, London, 1988.

[123] M. L. Rekart, M. Krajden, D. Cook, G. McNabb, T. Rees, and J. Isaac-Renton. Problems with the fast-check HIV rapid test kits. *Canadian Medical Association Journal*, 167(2):119, 2002.

[124] J. D. M. Rennie. Derivation of the F-measure. http://people.csail.mit.edu/∼jrennie/writing, 2004.

[125] V. Ruggiero and L. Zanni. On the efficiency of splitting and projection methods for large strictly convex quadratic programs. *Nonlinear Optimization and Related Topics, Applied Optimization*, 36:401–413, 1999.

[126] V. Ruggiero and L. Zanni. A modified projection algorithm for large strictly-convex quadratic programs. *J. Optim. Theory Appl.*, 104(2):281–299, 2000.

[127] V. Ruggiero and L. Zanni. Variable projection methods for large convex quadratic programs. *Recent Trends in Numerical Analysis*, pages 299–313, 2000.

[128] V. Ruggiero and L. Zanni. An overview on projection-type methods for convex large-scale quadratic programs. *Nonconvex Optimization and Its Applications*, 58:269–300, 2001.

[129] T. P. Runarsson and S. Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing - Letters and Reviews*, 3(3):59–67, 2004.

[130] S. Rüping. mySVM-manual, 2000. online http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/mysvm-manual.pdf.

[131] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1999.

[132] S. Schnitzler and T. Eitrich. Eine Studie zur kostensensitiven Klassifikation unausgeglichener Datensätze mit Support-Vektor-Maschinen. Technical Report FZJ-ZAM-IB-2006-05, Research Centre Jülich, 2006.

[133] S. Schnitzler and T. Eitrich. Gütemaße zur Optimierung von Support-Vektor-Maschinen. Technical Report FZJ-ZAM-IB-2006-06, Research Centre Jülich, 2006.

[134] B. Schölkopf. The kernel trick for distances. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 301–307. MIT Press, 2001.

[135] B. Schölkopf and A. J. Smola. *Learning with kernels*. MIT Press, Cambridge, MA, 2002.

[136] T. Serafini, G. Zanghirati, and L. Zanni. Large quadratic programs in training Gaussian support vector machines. *Rendiconti di Matematica*, 7(23):257–275, 2003.

[137] T. Serafini, G. Zanghirati, and L. Zanni. Parallel decomposition approaches for training support vector machines. In G. R. Joubert, W. E. Nagel, F. J. Peters, and W. V. Walter, editors, *Proceedings of the International Conference on Parallel Computing (ParCo 2003), Dresden, Germany*, pages 259–266. Elsevier, 2004.

[138] T. Serafini, G. Zanghirati, and L. Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, 20(2-3):353–378, 2005.

[139] T. Serafini and L. Zanni. On the working set selection in gradient projection-based decomposition techniques for support vector machines. *Optimization Methods and Software*, 20(4-5), 2005.

[140] D. B. Skillicorn. Strategies for parallelizing data mining. *IEEE Concurrency*, 7(4):26–35, 1999.

[141] J. C. Slater. Atomic shielding constants. *Phys. Rev.*, 36:57–64, 1930.

[142] J. C. Slater. Analytic atomic wave functions. *Phys. Rev.*, 42:33–43, 1932.

[143] A. Srivastava, E. Han, V. Kumar, and V. Singh. Parallel formulation of decision-tree classification algorithms. *Data Mining and Knowledge Discovery*, 3(3):237–261, 1999.

[144] C. Staelin. Parameter selection for support vector machines. Technical Report HPL-2002-354, HP Laboratories, Israel, 2002.

[145] R. G. Susnow and S. L. Dixon. Use of robust classification techniques for the prediction of human cytochrome P450 2D6 inhibition. *J. Chem. Inf. Comput. Sci.*, 43(4):1308–1315, 2003.

[146] S. J. Swamidass, J. Chen, J. Bruand, P. Phung, L. Ralaivola, and P. Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 21(1):359–368, 2005.

[147] J. A. Swets, R. M. Dawes, and J. Monahan. Better decisions through science. *Scientific American*, 283(4):70–75, 2000.

[148] A. C. Tan, D. Gilbert, and Y. Deville. Integrative machine learning approach for multi-class SCOP protein fold classification. In H.-W. Mewes, D. Frishman, V. Heun, and S. Kramer, editors, *Proceedings of the German Conference on Bioinformatics (GCB 2003)*, pages 153–159, 2003.

[149] K. Torkkola. Linear discriminant analysis in document classification. In *IEEE ICDM 2001 Workshop on Text Mining (TextDM 2001), San Jose, CA*, 2001.

[150] R. J. Vanderbei. LOQO: an interior point code for quadratic programming. *Optimization Methods and Software*, 11:451–484, 1999.

[151] V. N. Vapnik. *Statistical learning theory*. John Wiley & Sons, New York, 1998.

[152] N. P. Vermeulen. Prediction of drug metabolism: the case of cytochrome P450 2D6. *Curr. Top. Med. Chem.*, 3(11):1227–1239, 2003.

[153] G. Weiss and F. Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[154] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Royal Holloway University of London, 1998.

[155] P. Willett. Similarity-based approaches to virtual screening. *Biochemical Society transactions*, 31(3):603–606, 2003.

[156] I. H. Witten and E. Frank. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005.

[157] R. Yan, Y. Liu, R. Jin, and A. Hauptmann. On predicting rare classes with SVM ensembles in scene classification. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 21–24. IEEE Computer Society Press, 2003.

[158] C. W. Yap and Y. Z. Chen. Prediction of cytochrome P450 3A4, 2D6, and 2C9 inhibitors and substrates using support vector machines. *J. Chem. Inf. Model.*, 45:982–992, 2005.

[159] H. Yu, J. Yang, and J. Han. Classifying large data sets using SVMs with hierarchical clusters. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, editors, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 306–315. ACM Press, 2003.

[160] M. J. Zaki. Parallel and distributed association mining: a survey. *IEEE Concurrency*, 7(4):14–25, 1999.

[161] M. J. Zaki, C.-T. Ho, and R. Agrawal. Parallel classification for data mining on shared-memory multiprocessors. In *ICDE*, pages 198–205, 1999.

[162] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, 29(4):535–551, 2003.

[163] G. Zanghirati and L. Zanni. Variable projection methods for large quadratic programs in training support vector machines. Preprint 339, Department of Mathematics, University of Modena and Reggio Emilia, Italy, 2003.

[164] L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7(1):1467–1492, 2006.

[165] Z.-H. Zhou and X.-Y. Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):63–77, 2006.

[166] Z. Zhu, Y. S. Ong, K. W. Wong, and K. T. Seow. Experimental condition selection in whole-genome functional classification. In *Proceedings of IEEE Conference on Cybernetics and Intelligent Systems (CIS 2004)*, pages 295–300. IEEE Computer Society Press, 2004.

[167] G. Zoutendijk. *Methods of feasible directions: a study in linear and non-linear programming*. Elsevier, 1960.

Bisher sind erschienen:

Modern Methods and Algorithms of Quantum Chemistry - Proceedings
Johannes Grotendorst (Hrsg.)
Winterschule, 21. - 25. Februar 2000, Forschungszentrum Jülich
NIC-Serie Band 1
ISBN 3-00-005618-1, Februar 2000, 562 Seiten
nicht mehr lieferbar

Modern Methods and Algorithms of Quantum Chemistry - Poster Presentations
Johannes Grotendorst (Hrsg.)
Winterschule, 21. - 25. Februar 2000, Forschungszentrum Jülich
NIC-Serie Band 2
ISBN 3-00-005746-3, Februar 2000, 77 Seiten
nicht mehr lieferbar

Modern Methods and Algorithms of Quantum Chemistry - Proceedings, Second Edition
Johannes Grotendorst (Hrsg.)
Winterschule, 21. - 25. Februar 2000, Forschungszentrum Jülich
NIC-Serie Band 3
ISBN 3-00-005834-6, Dezember 2000, 638 Seiten
nicht mehr lieferbar

Nichtlineare Analyse raum-zeitlicher Aspekte der hirnelektrischen Aktivität von Epilepsiepatienten
Jochen Arnold
NIC-Serie Band 4
ISBN 3-00-006221-1, September 2000, 120 Seiten

Elektron-Elektron-Wechselwirkung in Halbleitern: Von hochkorrelierten kohärenten Anfangszuständen zu inkohärentem Transport
Reinhold Lövenich
NIC-Serie Band 5
ISBN 3-00-006329-3, August 2000, 146 Seiten

Erkennung von Nichtlinearitäten und wechselseitigen Abhängigkeiten in Zeitreihen
Andreas Schmitz
NIC-Serie Band 6
ISBN 3-00-007871-1, Mai 2001, 142 Seiten

Multiparadigm Programming with Object-Oriented Languages - Proceedings
Kei Davis, Yannis Smaragdakis, Jörg Striegnitz (Hrsg.)
Workshop MPOOL, 18. Mai 2001, Budapest
NIC-Serie Band 7
ISBN 3-00-007968-8, Juni 2001, 160 Seiten

Europhysics Conference on Computational Physics - Book of Abstracts
Friedel Hossfeld, Kurt Binder (Hrsg.)
Konferenz, 5. - 8. September 2001, Aachen
NIC-Serie Band 8
ISBN 3-00-008236-0, September 2001, 500 Seiten

NIC Symposium 2001 - Proceedings
Horst Rollnik, Dietrich Wolf (Hrsg.)
Symposium, 5. - 6. Dezember 2001, Forschungszentrum Jülich
NIC-Serie Band 9
ISBN 3-00-009055-X, Mai 2002, 514 Seiten

Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms - Lecture Notes
Johannes Grotendorst, Dominik Marx, Alejandro Muramatsu (Hrsg.)
Winterschule, 25. Februar - 1. März 2002, Rolduc Conference Centre, Kerkrade, Niederlande
NIC-Serie Band 10
ISBN 3-00-009057-6, Februar 2002, 548 Seiten

Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms - Poster Presentations
Johannes Grotendorst, Dominik Marx, Alejandro Muramatsu (Hrsg.)
Winterschule, 25. Februar - 1. März 2002, Rolduc Conference Centre, Kerkrade, Niederlande
NIC-Serie Band 11
ISBN 3-00-009058-4, Februar 2002, 194 Seiten

Strongly Disordered Quantum Spin Systems in Low Dimensions: Numerical Study of Spin Chains, Spin Ladders and Two-Dimensional Systems
Yu-cheng Lin
NIC-Serie Band 12
ISBN 3-00-009056-8, Mai 2002, 146 Seiten

Multiparadigm Programming with Object-Oriented Languages - Proceedings
Jörg Striegnitz, Kei Davis, Yannis Smaragdakis (Hrsg.)
Workshop MPOOL 2002, 11. Juni 2002, Malaga
NIC-Serie Band 13
ISBN 3-00-009099-1, Juni 2002, 132 Seiten

Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms - Audio-Visual Lecture Notes
Johannes Grotendorst, Dominik Marx, Alejandro Muramatsu (Hrsg.)
Winterschule, 25. Februar - 1. März 2002, Rolduc Conference Centre, Kerkrade, Niederlande
NIC-Serie Band 14
ISBN 3-00-010000-8, November 2002, DVD

Numerical Methods for Limit and Shakedown Analysis
Manfred Staat, Michael Heitzer (Hrsg.)
NIC-Serie Band 15
ISBN 3-00-010001-6, Februar 2003, 306 Seiten

Design and Evaluation of a Bandwidth Broker that Provides Network Quality of Service for Grid Applications
Volker Sander
NIC-Serie Band 16
ISBN 3-00-010002-4, Februar 2003, 208 Seiten

Automatic Performance Analysis on Parallel Computers with SMP Nodes
Felix Wolf
NIC-Serie Band 17
ISBN 3-00-010003-2, Februar 2003, 168 Seiten

Haptisches Rendern zum Einpassen von hochaufgelösten Molekülstrukturdaten in niedrigaufgelöste Elektronenmikroskopie-Dichteverteilungen
Stefan Birmanns
NIC-Serie Band 18
ISBN 3-00-010004-0, September 2003, 178 Seiten

Auswirkungen der Virtualisierung auf den IT-Betrieb
Wolfgang Gürich (Hrsg.)
GI Conference, 4. - 5. November 2003, Forschungszentrum Jülich
NIC-Serie Band 19
ISBN 3-00-009100-9, Oktober 2003, 126 Seiten

NIC Symposium 2004
Dietrich Wolf, Gernot Münster, Manfred Kremer (Hrsg.)
Symposium, 17. - 18. Februar 2004, Forschungszentrum Jülich
NIC-Serie Band 20
ISBN 3-00-012372-5, Februar 2004, 482 Seiten

Measuring Synchronization in Model Systems and Electroencephalographic Time Series from Epilepsy Patients
Thomas Kreutz
NIC-Serie Band 21
ISBN 3-00-012373-3, Februar 2004, 138 Seiten

Computational Soft Matter: From Synthetic Polymers to Proteins - Poster Abstracts
Norbert Attig, Kurt Binder, Helmut Grubmüller, Kurt Kremer (Hrsg.)
Winterschule, 29. Februar - 6. März 2004, Gustav-Stresemann-Institut Bonn
NIC-Serie Band 22
ISBN 3-00-012374-1, Februar 2004, 120 Seiten

Computational Soft Matter: From Synthetic Polymers to Proteins - Lecture Notes
Norbert Attig, Kurt Binder, Helmut Grubmüller, Kurt Kremer (Hrsg.)
Winterschule, 29. Februar - 6. März 2004, Gustav-Stresemann-Institut Bonn
NIC-Serie Band 23
ISBN 3-00-012641-4, Februar 2004, 440 Seiten

Synchronization and Interdependence Measures and their Applications to the Electroencephalogram of Epilepsy Patients and Clustering of Data
Alexander Kraskov
NIC-Serie Band 24
ISBN 3-00-013619-3, Mai 2004, 106 Seiten

High Performance Computing in Chemistry
Johannes Grotendorst (Hrsg.)
Bericht des Verbundprojekts:
High Performance Computing in Chemistry - HPC-Chem
NIC-Serie Band 25
ISBN 3-00-013618-5, Dezember 2004, 160 Seiten

Zerlegung von Signalen in unabhängige Komponenten: Ein informationstheoretischer Zugang
Harald Stögbauer
NIC-Serie Band 26
ISBN 3-00-013620-7, April 2005, 110 Seiten

Multiparadigm Programming 2003
Joint Proceedings of the
3rd International Workshop on Multiparadigm Programming with Object-Oriented Languages (MPOOL'03)
and the
1st International Workshop on Declarative Programming in the Context of Object-Oriented Languages (PD-COOL'03)
Jörg Striegnitz, Kei Davis (Editors)
NIC-Serie Band 27
ISBN 3-00-016005-1, Juli 2005, 300 Seiten

Integration von Programmiersprachen durch strukturelle Typanalyse und partielle Auswertung
Jörg Striegnitz
NIC-Serie Band 28
ISBN 3-00-016006-X, Mai 2005, 306 Seiten

OpenMolGRID - Open Computing Grid for Molecular Science and Engineering
Final Report
Mathilde Romberg (Editor)
NIC-Serie Band 29
ISBN 3-00-016007-8, Juli 2005, 86 Seiten

GALA Grünenthal Applied Life Science Analysis
Achim Kless und Johannes Grotendorst (Hrsg.)
NIC-Serie Band 30
ISBN 3-00-017349-8, November 2006, 204 Seiten

Computational Nanoscience: Do It Yourself!
Lecture Notes
Johannes Grotendorst, Stefan Blügel, Dominik Marx (Hrsg.)
Winterschule, 14. - 22. Februar 2006, Forschungszentrum Jülich
NIC-Serie Band 31
ISBN 3-00-017350-1, Februar 2006, 528 Seiten

NIC Symposium 2006 - Proceedings
G. Münster, D. Wolf, M. Kremer (Hrsg.)
Symposium, 1. - 2. März 2006, Forschungszentrum Jülich
NIC-Serie Band 32
ISBN 3-00-017351-X, Februar 2006, 384 Seiten

Parallel Computing: Current & Future Issues of High-End Computing
Proceedings of the International Conference ParCo 2005
G.R. Joubert, W.E. Nagel, F.J. Peters,
O. Plata, P. Tirado, E. Zapata (Hrsg.)
NIC-Serie Band 33
ISBN 3-00-017352-8, Oktober 2006, 930 Seiten

From Computational Biophysics to Systems Biology 2006 Proceedings
U.E.H. Hansmann, J. Meinke, S. Mohanty, O. Zimmermann (Hrsg.)
NIC-Serie Band 34
ISBN-10 3-9810843-0-6, ISBN-13 978-3-9810843-0-6,
September 2006, 224 Seiten

Alle Bände stehen online zur Verfügung unter

http://www.fz-juelich.de/nic-series/.