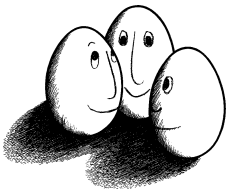
Diplomarbeit

Entwicklung von Optimierungsverfahren für das Lösen verschiedener Lernaufgaben mit der Stützvektormethode

Hendrik Blom



Diplomarbeit
Fakultät Informatik
Technische Universität Dortmund

Dortmund, 13. April 2011

Betreuer:

Prof. Dr. Katharina Morik
Dipl.-Inform. Marco Stolpe

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In der Informationstechnologie können zwei Trends beobachtet werden: Die verstärkte Nutzung von mobilen Computern und die Personalisierung von Inhalten. Laut [19] hat das mobile Computing das Potential, die Interaktion zwischen Individuen, Gruppen, Organisationen und Gesellschaften stark zu verändern. Das am stärksten wachsende Segment der mobilen Computer sind die sogenannten Smartphones. Die Betriebsysteme, mit dem stärksten Wachstum sind iOS von Apple, Android von Google und BlackBerry OS von Research in Motion [1]. Auch wenn die Betriebsysteme selbst in C oder C++ geschrieben sind, können oder werden alle Applikationen in Java implementiert.

Um Applikationen zu personalisieren, müssen Art und Nutzung der Applikationen und durch den Nutzer erstellte Daten, wie beispielsweise Kommunikationsprotokolle mit Webseiten oder Servern, ortsbezogene Daten (GPS) oder Fotos ausgewertet werden.

Ein mögliches Szenario für die Personalisierung einer Anwendung ist die Implementierung einer Gesichtserkennung für eine Applikation zur Verwaltung persönlicher Fotos. Damit ist es möglich, Gesichter auf Fotos zu erkennen und gegebenenfalls Personen zuzuordnen. Aktuelle Smartphones speichern den Aufnahmeort eines Fotos per GPS. Werden mehrere Personen auf einen Foto erkannt, können Relationen zwischen diesen Personen und zwischen ihnen und ihrem Aufenthaltsort festgestellt werden. In Verbindung mit weiteren zugänglichen Daten können nicht nur Rückschlüsse auf den Benutzer, sondern auch auf die anderen Personen auf dem Foto gezogen werden. Die gewonnenen Informationen könnten dann durch Dritte missbraucht werden.

Dieses Beispiel stellt eine mögliche negative Folge der Übertrag von persönlichen Daten an Dritte dar. In [30] wird sogar von einer möglichen Gefährdung der freien und demokratischen Informationsgesellschaft gesprochen. Personalisierungssysteme sollten daher so gestalten werden, dass die Übertragung von persönlichen Daten an Dritte minimiert wird. Hierfür muss die Datenverarbeitung auf den lokalen Systemen erweitert werden, so dass die Menge der extern auszuwertenden persönlichen Daten verringert wird. Da Rechenleistung, Speicherkapazität und Energieversorgung von mobilen Computern beschränkt sind, müssen geeignete Verfahren für diese Systeme implementiert werden.

1.1 Problemstellung

Das Problem der Gesichterkennung gehört zur Gruppe der Klassifikationsprobleme. Ein Klassifikationsproblem stellt sich immer dann, wenn zuvor unbekannte Objekte in vor-

gegebene Klassen eingeordnet werden sollen. Beim Beispiel der Gesichtserkennung sollen Bilder von Gesichtern Personen zugeordnet werden. Die Menge der zuvor bekannten Bilder einer Person definieren dabei eine Klasse. Es können viele weitere Beispiele für Klassifikationsprobleme gefunden werden, die auf Smartphones zu lösen sind (vgl. Abschnitt 1.3).

Es existiert eine Vielzahl von verschiedenen Verfahren, mit denen Klassifikationsprobleme gelöst werden könnten. Durch die Ressourcenbeschränkung von Smartphones oder anderen mobilen Systemen können Verfahren, die für die Lösung des Problems auf weniger beschränkten Systemen problemlos einsetzbar wären, nicht uneingeschränkt verwendet werden. Ein häufig für Klassifikationsprobleme eingesetztes Verfahren ist die Stützvektormethode. Diese liefert zwar gute Ergebnisse, kann aber nicht in ohne Weiteres unter beschränkten Ressourcen eingesetzt werden, da die Anforderungen an die Rechenkapazität und den Speicherbedarf zu hoch sind.

Um jedoch nicht auf dieses geeignete Lernverfahren verzichten zu müssen, können spezielle Varianten der Stützvektormethode eingesetzt werden. Diese Varianten, wie die Core Vector Machine und die Ball Vector Machine können aufgrund ihrer speziellen mathematischen Formulierung schneller gelöst werden, ohne schlechtere Ergebnisse zu liefern. Implentierungen dieser Verfahren liegen aber nicht in Java, sondern nur in C++ vor und können somit auf den oben genannten Betriebsystemen nicht eingesetzt werden. Aufgrund der vielen möglichen Betriebsysteme und Hardwareplattformen muss in der Implementierung der Verfahren auch auf Teilaspekte der jeweiligen Systems und Hardware eingegangen werden können, ohne die gesamte Implementierung ändern zu müssen. Bisher existiert solch ein Framework nicht.

1.2 Ziel

Ein sich häufig stellendes Klassifikationsproblem ist die binäre Klassifikation (vgl. [5]). Da die Stützvektormethode aufgrund ihrer mathmatischen Formulierung besonders dafür geeignet ist, wird diese im Rahmen dieser Arbeit genauer untersucht. Des Weiteren sollen einige Varianten der Stützvektormethode, die Core Vector Machine und die Ball Vector Machine, genauer betrachtet und bzgl. ihre Eignung für ressourcenbeschränkte Systeme untersucht werden. Die strukturelle Stützvektormethode erweitert die Einsetzbarkeit der Stützvektormethode auf eine Vielzahl von weiteren Klassifikationsaufgaben und soll deswegen ebenfalls untersucht werden. Die Lösungsverfahren werden theoretisch beschrieben und miteinander verglichen.

In ihrer mathematischen Beschreibung stellen die Stützvektormethode und ihre Varianten Optimierungsprobleme dar. Dieser Aspekt soll genauer betrachtet und mögliche Lösungsverfahren vorgestellt werden. Um auf einer breiten Palette von Systemen und insbesondere auch unter den vorgestellten Betriebssystemen für mobile Computer lauffähig zu sein, werden die Lernverfahren in Java implementiert und soweit möglich mit den Originalimplementierungen in C und C++ empirisch verglichen.

Für die Implementierung wird ein Framework entwickelt, das eine flexiblere Gestaltung der Verfahren ermöglicht. So können dann Teilaspekte der Implementierungen ausgetauscht werden, ohne dass das gesamte Verfahren angepasst werden muss. Das Framework soll ferner eine transparente Datenhaltung ermöglichen, damit verschiedene Datenquellen ohne große Anpassungen integriert werden können.

Damit die empirische Auswertung möglichst leicht durchgeführt werden kann, wird ein sogenanntes Rapidminer Plugin entwickelt. Rapidminer ist eine der am häufigsten eingesetzten Datamining Applikationen. Dabei handelt es sich um eine Open-Source Software, die ausschließlich in Java implementiert ist und deswegen für diese Arbeit besonders geeignet. Neben einer Vielzahl von schon vorhandenen Lernverfahren, bietet Rapidminer auch eine gute Infrastrukur um Experimente auszuführen. Durch das Plugin können die Implementierungen der Verfahren in Rapidminer integriert und ausgeführt werden. In Rapidminer gibt es eine Implementierung der Stützvektormethode, die MySVM, die durch die Integration in das Framework flexibilisiert werden soll. Die MySVM enthält beispielsweise eine Implementierung der Sequential Minimal Optimization, die aufgrund der starken Verzahnung in der Implementierung weder austauschbar noch für andere Implementierungen einsetzbar ist. Deswegen wird die Implementierung der SVM soweit angepasst, dass andere Optmierungsverfahren mit der restlichen Implementieurung einsetzbar sind, aber auch die Sequential Minimal Optimization für andere Implementierungen verwendet werden kann.

1.3 Klassifikationsprobleme

Klassifikationprobleme lassen sich in Abhängigkeit von der Anzahl der existierenden Klassen in die Ein-Klassen-Klassifikation, die Zwei-Klassen-Klassifikation und die Mehr-Klassen-Klassifikationassifikation unterteilen.

Bei der Ein-Klassen-Klassifikation wird eine Beschreibung der Daten gelernt (vgl. [22]). Dadurch können laut [36] verschiedene Probleme gelöst werden, wobei insbesondere die Detektion von Ausreißern von Interesse ist. Ausreißer beschreiben uncharakteristische Datenpunkte eines Datensatzes. Durch die starke Unähnlichkeit der Ausreißer zum Rest der Daten wird im Allgemeinen die Güte von Klassifikations- oder auch Regressionsmodellen verringert. Ferner ist eine korrekte Vorhersage für unbekannte oder seltene Bereiche im Merkmalsraum sehr unwahrscheinlich (vgl. [28]). Durch das Entfernen von Ausreißern aus der Menge aller Beispiele kann demnach die Güte der Vorhersage für die anderen Beispiele verbessert werden.

Die Verallgemeinerung der Zwei-Klassen-Klassifikation ist die Mehr-Klassen-Klassifikation, bei der die Beispiele mehr als zwei Klassen zugeordnet werden können. Ein Verfahren zur binären Klassifkation kann immer zur Mehr-Klassen-Klassifikation erweitert werden. Dazu lernt das binäre Verfahren für jede Klasse ein eigenes Modell, das zwischen der jeweiligen Klasse und den übrigen Klassen unterscheidet. Durch die Kombination aller Modelle kann für jedes Beispiel entschieden werden zu welcher Klasse es gehört. Es existieren auch Verfahren, die nicht auf die Anzahl der Klassen beschränkt sind und für

die keine Kombination von mehreren Modellen nötig ist. Dazu gehört unter anderem der Baumlerner C4.5 von [26].

1.4 Aufbau

In Kapitel 2 werden die grundlegende Idee und der mathematische Hintergrund der Stützvektormethode beschrieben. Aufgrund der Formulierung der Stützvektormethode kann die Zwei-Klassen-Klassifikation als ihre natürliche Lernaufgabe angesehen werden. Im Rahmen dieser Arbeit werden verschiedene Varianten der Stützvektormethode für die binäre Klassifikation implementiert: die einfache Stützvektormethode, die Core Vector Machine und die Ball Vector Machine (vgl. Kapitel 2, 4 und 8). Weitere Varianten sind etwa die LIBSVM, Pegasos oder die SimpleSVM (vgl. [7], [33], [43]).

Um eine Lösung mit der Stützvektormethode zu finden, muss ein konvexes Optimierungsproblem gelöst werden. Die Definition eines konvexen Optimierungsproblems, dessen Eigenschaften, sowie die Sequential Minimal Optimization (SMO) werden in Kapitel 3 vorgestellt.

Wie bereits erwähnt, werden in Kapitel 4 die Core Vector Machine und die Ball Vector Machine vorgestellt. Durch eine approximative Lösung des Optimierungsproblems erreichen sie eine starke Verbesserung der Laufzeit und ermöglichen damit eine Ausführung auf einem Smartphone. Diese beiden Verfahren, die Support Vector Data Description (vgl. Kapitel 5), aber auch die Global Gaussian Approximation (vgl. [8], [17]) können ferner das Problem der Ein-Klassen-Klassifikation lösen.

Eine Erweiterung der Stützvektormethode, die SVM_{struct} ist in der Vorhersage nicht auf Klassen beschränkt. Sie kann die Zuordnung zwischen Beispielen und beliebigen Mengen von strukturierten Ausgabevariablen lernen. Die Struktur der Ausgabevariablen kann dabei fast beliebig definiert werden. Beispiele dafür sind Sequenzen, Zeichenketten, Bäume, Netze oder Graphen (vgl. [40]). Mit diesem Verfahren kann auch eine Mehr-Klassen-Klassifikation durchgeführt werden. In Kapitel 5 wird die SVM_{struct} und nachfolgend in Kapitel 6 das Schnittebenenverfahren beschrieben. Dieses Verfahren wird in der SVM_{struct} verwendet, um ähnlich zur Core Vector Machine eine Reihe von Subproblemen zu definieren, welche stellvertretend für das Gesamtproblem gelöst werden.

Das Framework zur flexibleren gestaltung der Verfahren, sowie dessen wichtigsten Klassen werden in Kapitel 7 entwickelt. Darauf folgt die Beschreibung der der wichtigsten Aspekte der Implementierung des Frameworks und der Verfahren in Kapitel 8.

Zum Abschluss werden die jeweiligen Implementierungen auf verschiedenen Datensätzen empirisch evaluiert. Hierzu dienen die Güte der Lösung, die Laufzeit und die Anzahl der Supportvektoren jeweils in Abhängigkeit der Größe des Datensatzes als Bewertungskriterien (vgl. Kapitel 9). Im abschließenden Kapitel 10 werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick gegeben.

2 Die Stützvektormethode

In diesem Kapitel werden die grundlegenden Ideen und mathematischen Konzepte der Stützvektormethode erläutert. Zunächst wird eine Einführung in die der Stützvektormethode zugrunde liegenden Theorie gegeben. Danach wird die Stützvektormethode für linear trennbare und nicht linear trennbare Daten erläutert. Zum Abschluss des Kapitels wird die Erweiterung der Stützvektormethode mit Hilfe von Kernen vorgestellt.

2.1 Grundlagen

Seien $\mathcal{X}$ und $\mathcal{Y}$ Teilmengen normierter Vektorräume, beispielsweise $\mathcal{X} \subset \mathbb{R}^d$ und $\mathcal{Y} \subset \mathbb{R}$ mit unbekannter gemeinsamer Wahrscheinlichkeitsverteilung $P(x, y)$. Im Fall der binären Klassifikation sei o.B.d.A. $\mathcal{Y} = \{-1, 1\}$ definiert. Eine Stichprobe $Z \subset \mathcal{X} \times \mathcal{Y} \subset \mathbb{R}^d \times \mathbb{R}$ sei eine Realisation von N unabhängig identisch verteilten Zufallsvariablen mit $z_i = (x_i, y_i),\ i = 1, \ldots, N$. Die Stützvektormethode soll aus der Stichprobe oder auch Trainingsmenge eine mit dem Vektor α parametrisierte Funktion

$$f:\ \mathcal{X} \to \mathcal{Y},\ x \mapsto f(x, \alpha) := y \tag{2.1}$$

lernen, die jedem Beispiel $x \in \mathcal{X}$ genau eine Klasse $\hat{y}$ zuordnet und verglichen mit der tatsächlichen Klassen y über alle Beobachtungen einen möglichst kleinen Fehler verursacht. Um den Fehler für eine Beobachtung zu quantifizieren, wird eine Verlustfunktion

$$\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R} \tag{2.2}$$

definiert, wobei $\Delta(y, y) = 0$, $\Delta(y, y') > 0$ für $y \neq y'$, und der maximale Fehler für alle y mit $\max_y\{\Delta(y^*, y)\}$ nach oben beschränkt ist. Die Verlustfunktion für die Klassifikation ist definiert als $\Delta_{0,1}(y, y') = |y - y'|$ und kann für die binäre Klassifikation nur den Wert 0 oder 1 annehmen. Weitere Verlustfunktion werden in Kapitel 5 vorgestellt.

Der erwartete Fehler oder auch das erwartetes Risiko über alle Beobachtungen $x \in \mathcal{X}$ ist:

$$\mathcal{R}_P^{\Delta}(f(\alpha)) = \int_{\mathcal{X}\times\mathcal{Y}} \Delta(y, f(x, \alpha))dP(x, y). \tag{2.3}$$

Im Allgemeinen ist die Verteilung $P(x, y)$ unbekannt und der Fehler kann nur auf einer Stichprobe mit dem durchschnittlichen empirischen Risko gemessen werden:

$$\mathcal{R}_{emp}^{\Delta}(f(\alpha)) = \frac{1}{n}\sum_{i=1}^{N} \Delta(y_i, f(x_i)). \tag{2.4}$$

Je geringer der Fehler auf einer unabhängigen Testmenge ist, desto höher ist die Generalisierungsgüte.

Wenn die Wahl zwischen verschiedenen gelernten Funktion besteht, ist das empirische Risko kein ausreichend gutes Kriterium, um eine Funktion für alle Beobachtungen $x \in \mathcal{X}$ zu bewerten. Ein wichtiges Ergebnis für die binäre Klassifikation mit einer Verlustfunktion, die nur 0 oder 1 annehmen kann, liefert [42]: Für ein beliebiges $\eta \in [0, 1]$ und eine Stichprobe der Größe N ist der erwartete Fehler mit einer Wahrscheinlichkeit von $1 - \eta$ durch

$$\mathcal{R}_P^{\Delta_{0,1}}(f(\alpha)) \leq \mathcal{R}_{emp}^{\Delta_{0,1}}(f(\alpha)) + \sqrt{\frac{h(\log(2N/h) + 1) - log(\eta/4)}{N}} \tag{2.5}$$

beschränkt, wobei h die sogenannte Vapnik Chervonenkis (VC) Dimension ist, eine nicht negative ganze Zahl, die die Kapaziät einer Funktion (5.1) angibt. Die Kapazität einer gelernten Funktion ist ein Maß für die Fähigkeit der Funktion, beliebige Daten ohne Fehler zu lernen. Die VC-Dimension misst, wie viele Punkte maximal zerschmettert werden können. Es seien l Punkte gegeben, die beliebig im Raum verteilt sein können. Für den Fall der binären Klassifikation existieren 2^l Möglichkeiten, diese Punkte in die beiden Klassen einzuteilen. Kann eine Funktion aus der Familie $f(\alpha)$ gefunden werden, die für alle Belegungen die Punkte beider Klassen trennt, wird die Menge der Punkte durch $f(\alpha)$ zerschmettert.

Je geringer die VC-Dimension der gewählten Funktion, desto kleiner wird also die obere Schranke des Risikos $\mathcal{R}_P^{\Delta}(f(\alpha))$. Diesen Zusammenhang nutzt die strukturelle Risikominimierung (vgl. [41]) aus, um aus einer Menge gewählter Funktionen eine Funktion aus der Teilmenge auszuwählen, die das Risiko $\mathcal{R}_P^{\Delta}(f(\alpha))$ am stärksten beschränkt. Dazu werden die Funktionen nach ihrer VC-Dimension in sich selbst enthaltende Mengen aufgeteilt, mit $h_1 < h_2 < h_3 \ldots$. Kann die VC-Dimension für eine Funktion nicht berechnet werden, muss diese approximiert werden. Dazu wird für jedes h eine Funktion gelernt und die Schranke für das Risiko berechnet. Die Funktion, die das Risiko am stärksten beschränkt, wird gewählt.

2.2 Lineare Stützvektormethode

Die lineare Stützvektormethode ist ein binäres Klassifikationsverfahren, die einer Beobachtung $x \in \mathcal{X}$ einer von zwei Klassen $y \in \{-1, 1\}$ zuordnet. Sei eine Trainingsmenge S_{train} von Paaren $(x_1, y_1,), \ldots, (x_N, y_N)$ gegeben, für die eine lineare Entscheidungsschranke so positioniert werden kann, dass allen Beobachtungen ihrer gegebenen Klasse zugeordnet werden können. Liegen zwei Beobachtungen auf unterschiedlichen Seiten der Entscheidungsschranke, werden ihnen unterschiedliche Klassen zugeordnet. Diese Situation ist beispielhaft in Abb. 2.1 dargestellt. Gehört ein Punkt zur Klasse 1 liegt dieser oberhalb der Entscheidungsschranke, sonst darunter.

Mathematisch lässt sich diese Entscheidungsschranke als Hyperebene L modellieren:

$$L = \{x | f(x) = x^T \beta + \beta_0 = 0\}. \tag{2.6}$$

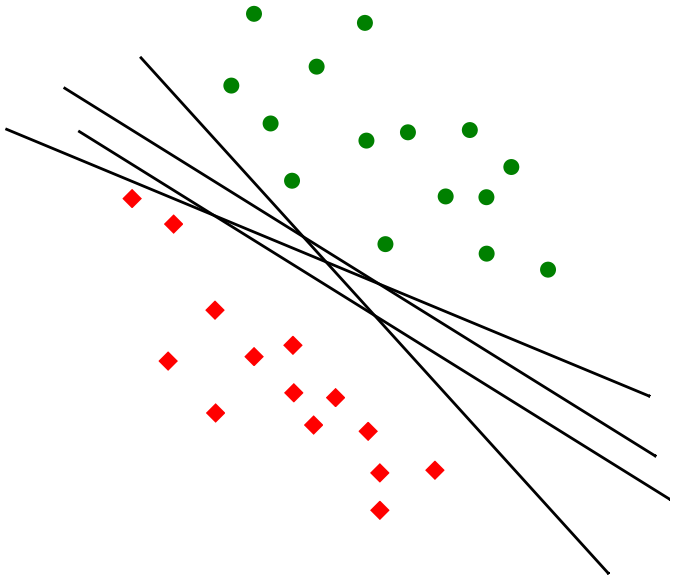


Abbildung 2.1: Eine Menge von möglichen linearen Entscheidungsschranken für die binäre Klassifikation.

Für die Definition der Hyperebene wird ein Skalarprodukt in der Funktion $f(x)$ benötigt. Auf dem normierten Vektorraum $\mathcal{X}$, dem die Beobachtungen entstammen, muss also ein Skalarprodukt definiert sein.

Der Winkel zwischen dem Beispiel x und dem Normalenvektor der Hyperebene β entscheidet über die Klassenzugehörigkeit von x. Ist der Winkel kleiner als 90 Grad, gehört das Beispiel zur positiven Klasse, ist der Winkel gleich 90 Grad, liegt das Beispiel in der Hyperebene und ist der Winkel größer als 90 Grad, gehört das Beispiel zur negativen Klasse (vgl. Abb. 2.2). Da der Winkel zwischen zwei Vektoren über das Skalarprodukt $\langle x_1, x_2 \rangle = \|x_1\| \, \|x_2\| \cos \sphericalangle(x_1, x_2)$ berechenbar ist, kann über dessen Wert die Klassenzugehörigkeit ermittelt werden (vgl. Abb. 2.2). Zusammenfassend kann eine Entscheidungsfunktion definiert werden mit:

$$G(x) = \text{sign} f(x). \tag{2.7}$$

Wenn alle Beispiele durch die gefundene Hyperebene L ihrer Klasse korrekt zugeordnet werden können, wird L als trennende Hyperebene bezeichnet, es gilt:

$$\begin{aligned} f(x_i) &= x_i^T \beta + \beta_0 > 1, \ \forall y = 1 \\ f(x_i) &= x_i^T \beta + \beta_0 < -1, \ \forall y = -1. \end{aligned} \tag{2.8}$$

Diese Bedingungen lassen sich auch zusammenfassen zu:

$$f(x_i) = y_i(x_i^T \beta + \beta_0) > 1, \ \forall y_i = \{1, -1\}. \tag{2.9}$$

Sind die Beispiele linear trennbar, dann existieren unendlich viele solcher trennenden Hyperebenen (vgl. Abb. 2.1). Aus diesen Hyperebenen soll diejenige gewählt werden, die

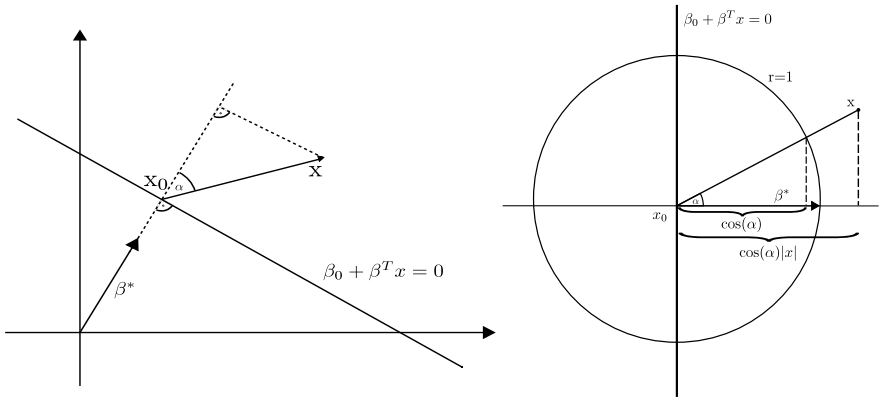
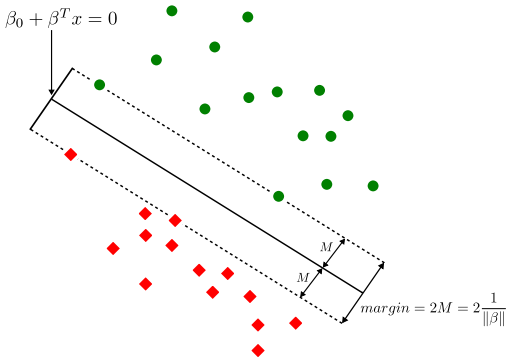


Abbildung 2.2: Abstand zwischen einem Punkt und einer Ebene.

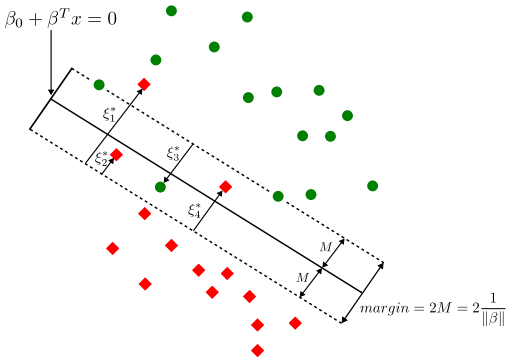
für eine von S_{train} unabhängige Stichprobe S_{test} einen möglist kleinen Fehler verursacht (vgl. Abschnitt 2.1).

Die Stützvektormethode wählt die Hyperebene, die den Abstand zwischen der Hyperebene und den jeweils nächsten Punkten beider Klassen maximiert. Auf diese Weise soll die Generalisierungsgüte des gelernten Modelles maximiert werden.

Der Abstand M zwischen einer Klasse und der Hyperebene wird als Margin bezeichnet. Die Margin ist symmetrisch für beide Klassen. Damit beträgt der Gesamtabstand zwischen beiden Klassen $2M$ (vgl. Abb. 2.3).



(a) Stützvektormethode auf linear trennbaren Daten.



(b) Stützvektormethode auf nichtlinear trennbaren Daten. Mit den Slack-Variablen ξ_i können die Beobachtungen auch in der Margin liegen.

Abbildung 2.3: Darstellung der Datentrennung bei der Stützvektormethode

Werden die Bedingungen (2.9) so angepasst, dass alle Beobachtungen außerhalb der Margin liegen müssen, kann für die Auswahl der Hyperebene folgendes Optimierungsproblem definiert werden:

$$\begin{aligned} &\max_{\beta,\beta_0,\|\beta\|=1} M \\ &\text{u.d.N. } y_i(x_i^T\beta + \beta_0) \geq M,\ \forall i = 1,\ldots,N. \end{aligned} \tag{2.10}$$

Es muss also der Normaleneinheitsvektor einer Hyperebene β^*, die die Margin maximiert, gefunden werden. Werden die Nebenbedingungen des Optinmierungsproblems durch $\|\beta\|$ normiert, kann diese vereinfacht werden:

$$\frac{1}{\|\beta\|} y_i(x_i^T\beta + \beta_0) \geq M \text{ bzw. } y_i(x_i^T\beta + \beta_0) \geq \|\beta\| M,\ \forall i = 1,\ldots,N. \tag{2.11}$$

Erfüllen β und β_0 die Ungleichungen aus (2.11), so erfüllen alle positiven Vielfache von β und β_0 ebenfalls (2.11). Damit kann die Margin willkürlich als $M = \frac{1}{\|\beta\|}$ definiert werden. Um M zu maximieren, genügt es nun, $\|\beta\|$ zu minimieren, so dass das das Optimierungsproblem der SVM (2.10) zu

$$\begin{aligned} \mathbf{SVM}_0 : &\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 \\ &\text{u.d.N. } y_i(x_i^T\beta + \beta_0) \geq 1,\ \forall i = 1,\ldots,N \end{aligned} \tag{2.12}$$

umformuliert werden kann. Da $\|\beta\|$ immer positiv ist, kann das Problem auch durch ein quadratisches Optimierungsproblem beschrieben werden. Durch die Halbierung von $\|\beta\|$ fällt nach Ableiten der Zielfunktion der multiplikative Faktor weg. Damit ist (2.12) ein konvexes Optimierungsproblem mit eindeutiger Lösung (vgl. Abschnitt 3.1).

Mittels des gefunden Normaleneinheitsvektor der Hyperebene $\hat{\beta}$ und β_0 kann nun für Beobachtungen durch

$$\hat{G}(x) = \text{sign}[\hat{f}(x)] = \text{sign}[x^T\hat{\beta} + \hat{\beta}_0] \tag{2.13}$$

entschieden werden, zu welcher Klasse sie gehören.

2.3 Die Stützvektormethode bei nicht linear trennbaren Daten

Reale Daten sind selten linear trennbar, da sich in den meisten Fällen die zu trennenden Klassen überlagern. Um die Daten dennoch, wie zuvor beschrieben, trennen zu können, müssen die Nebenbedingungen (2.9) relaxiert werden. Dazu werden für jedes Beispiel x_i sogenannte Slack-Variablen ξ_i eingeführt. Diese Slack-Variablen erlauben es einigen Punkten, innerhalb der Margin oder sogar auf der falschen Seite der Hyperebene zu liegen (vgl. Abb. 2.3b). Um eine Überanpassung zu verhindern, wird die Summe des Fehlers durch eine Konstante nach oben beschränkt. Die Nebenbedingungen werden wie folgt erweitert:

$$\xi_i \geq 0,\ y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i,\ \forall i = 1,\ldots,N. \tag{2.14}$$

Mit der Erweiterung der Zielfunktion um die Summe aller ξ_i in Verbindung mit den Nebenbedingungen aus (2.14) ergibt sich als neues Optimierungsproblem:

$$\begin{aligned} \mathbf{SVM}_1 : \min_{\beta,\beta_0} \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^{N} \xi_i \\ \text{u.d.N. } \xi_i \geq 0,\ y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i,\ \forall i = 1, \ldots, N. \end{aligned} \tag{2.15}$$

Die Variable $C > 0$ beschränkt den Einfluss der Slack-Variablen. Wird ein kleiner Wert für C vorgegeben, beeinflusst die Erhöhung eines ξ_i den Wert der Zielfunktion nur geringfügig, bei einem großen C hingegen stark. Mit einem großen Wert für C liegt der Fokus mehr auf korrekt klassifizierten Punkten nahe der Entscheidungsschranke und es kann zu Überanpassungen kommen. Für ein kleines C wird weiter entfernten Punkten von der Hyperebene mehr Gewicht gegeben. Für $C = \infty$ sind die Optimierungsprobleme (2.12) und (2.15) identisch.

Eine Lösung des Optimierungsproblems (2.15) kann mit Hilfe der Lagrange-Multiplikator-Methode (vgl. Abschnitt 3.1) gefunden werden. Dazu wird (2.15) in ein äquivalentes Optimierungsproblem umgeformt:

$$\min_{\beta,\beta_0,\xi_i} L_P = \min_{\beta,\beta_0,\xi_i} \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N} \mu_i \xi_i. \tag{2.16}$$

Notwendige Bedingungen für die Optimalität einer Lösung eines beschränkten Optimierungsproblems sind die Karush-Kuhn-Tucker Bedingungen (vgl. Theorem 3.1). Diese beinhalten neben der Bedingung, dass an der Stelle einer optimalen Lösung x, die Ableitung der primalen Funktion L_P (2.16) stationär, also gleich Null ist, weitere Eigenschaften:

$$\frac{\partial L_P}{\partial \beta} \stackrel{!}{=} 0 \Leftrightarrow \beta = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{2.17}$$

$$\frac{\partial L_P}{\partial \beta_0} \stackrel{!}{=} 0 \Leftrightarrow \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.18}$$

$$\frac{\partial L_P}{\partial \xi_i} \stackrel{!}{=} 0 \Leftrightarrow \alpha_i = C - \mu_i,\ \forall i = 1, \ldots, N \tag{2.19}$$

$$\alpha_i,\ \mu_i,\ \xi_i,\ \geq 0,\ \forall i = 1, \ldots, N \tag{2.20}$$

$$\alpha_i [y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0,\ \forall i = 1, \ldots, N \tag{2.21}$$

$$\mu_i \xi_i = 0,\ \forall i = 1, \ldots, N \tag{2.22}$$

$$y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) \geq 0,\ \forall i = 1, \ldots, N. \tag{2.23}$$

Werden (2.17) - (2.20) in L_P eingesetzt, ergibt sich das duale Problem als:

$$\begin{aligned} \max_{\alpha} L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{u.d.N. } 0 \leq \alpha_i \leq C,\ \sum_{i=1}^{N} \alpha_i y_i = 0,\ \forall i = 1, \ldots, N. \end{aligned} \tag{2.24}$$

Die Lösung der dualen Funktion gibt eine untere Schranke für die primale Funktion L_P an. Mit Theorem 3.2 kann gezeigt werden, das im Fall der Stützvektormethode das duale Problem und das primale Problem sogar identische Lösungen liefern. Die Lösung des durch die Stützvektormethode mit Slack-Variablen gegebenen Optimierungsproblems (Formel (2.15)) ergibt sich mit Formel (2.17) als:

$$\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i, \tag{2.25}$$

mit $\hat{\alpha}_i \neq 0$ für alle Beispiele x_i, die auf dem Rand der Margin liegen, für die also (2.23) genau erfüllt ist. Diese Beobachtungen werden als Stützvektoren (Support Vector) bezeichnet, da die Lösung $\hat{\beta}$ nur durch diese Beobachtungen definiert wird. Für die Stützvektoren x_i, die auf dem Rand der Margin liegen ($\xi_i = 0$), gilt mit (2.19) und (2.22) für den Lagrange-Multiplikator: $0 < \hat{\alpha}_i < C$. Für Supportvektoren innerhalb der Margin gilt $\hat{\alpha}_i = C$. Die Lösung für β_0 kann für jeden Stützvektor mit (2.21) errechnet werden.

Aus den gelernten Lösungen für $\hat{\beta}$ und $\hat{\beta}_0$ ergibt sich nun die Entscheidungsfunktion als:

$$\hat{G} = \text{sign}[x^T \hat{\beta} + \hat{\beta}_0]. \tag{2.26}$$

2.4 Nicht lineare Trennung von Daten

Die bisher vorgestellten Methoden trennen die Daten mit einer linearen Entscheidungsschranke. Eine Möglichkeit, eine lineare Methode auf nicht-lineare Entscheidungsschranken zu erweitern, ist die Basis Expansion (vgl. [11]). Mit nicht linearen Entscheidungsschranken können die Klassen besser voneinander getrennt werden. Die eigentliche Methode ändert sich nicht, aber die Klassen werden in einem höher dimensionalen Raum getrennt. Dazu werden die Beobachtungen mit einer Funktion (Feature-Map)

$$\varphi : \mathbb{R}^d \to \mathcal{H} \tag{2.27}$$

in einen höher Dimensionalen Raum $\mathcal{H}$ transformiert. Die Hoffnung ist, dass die Daten in $\mathcal{H}$ linear getrennt werden können. Wird die in $\mathcal{H}$ gelernte, lineare Entscheidungsschranke zurück in den Ursprungsraum transformiert, ergibt sich eine nicht lineare Entscheidungsschranke.

Mit der Feature-Map ergibt sich für das duale Problem (2.24):

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \varphi(x_i), \varphi(x_j) \rangle. \tag{2.28}$$

Die Transformation der Beobachtungen wird also nur in einem Skalarprodukt benötigt. Die Feature-Map $\varphi(x)$ muss nicht explizit bekannt sein, sondern es kann auch eine sogenannte Kern-Funktion

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle \tag{2.29}$$

benutzt werden. Erfüllt die Kern-Funktion Mercer's Bedingung (vgl. [42]), ist sie ein Skalarprodukt im höher dimensionalen Raum $\mathcal{H}$ und die Stützvektormethode kann, eingesetzt in (2.24), mit dieser Kern-Funktion die Klassen in $\mathcal{H}$ trennen.

Beispiele für häufig verwendete Kern-Funktionen sind:

$$\begin{aligned} &\text{polynomischer Kern mit Grad } d : k(x, x') = (1 + \langle x, x' \rangle)^d, \\ &\text{Radiale Basis Funktionen (RBF): } k(x, x') = \exp(-\gamma \left\| x - x' \right\|^2). \end{aligned} \tag{2.30}$$

Welcher Kern zu benutzen ist, kann a priori nicht beantwortet werden. Beispielsweise für den RBF-Kern gilt, dass dieser eine unendliche VC-Dimension besitzt, aber unter gewissen Umständen gute Lösungen liefert (vgl. [5]).

3 Lösungsverfahren für die Stützvektormethode

In diesem Kapitel wird das durch die Stützvektormethode gegebene Optimierungsproblem genauer betrachtet. Bei diesem handelt es sich um ein konvexes Optimierungsproblem, woraus sich Eigenschaften zur Lösbarkeit ableiten lassen. Ein übliches Vorgehen zur Lösung eines beschränkten Optimierungsproblems ist die Verwendung der Lagrange-Multiplikator-Methode. Diese wird zunächst erläutert und die Verbindung der Lösung des primalen und des dualen Problems aufgezeigt. Danach wird eine Übersicht über verschiedene Lösungsverfahren für das Optimimierungsproblem der Stützvektormethode gegeben. Zum Abschluss wird die sogenannte Sequential Minimal Optimization (SMO) eingeführt, die in den Implementierungen zur Anwendung kommt.

3.1 Die Stützvektormethode als Optimierungsproblem

Das durch die Stützvektormethode gegebene Optimierungsproblem (vgl. (2.15)) ist ein konvexes Optimierungsproblem im $\mathbb{R}^d$. Für die Definition eines konvexes Optimierungsproblems werden nachfolgend die Begriffe der konvexen Menge, der Konvexkombination und der konvexen Funktion eingeführt (vgl. [9]).

Für eine konvexe Menge $K \subset \mathbb{R}^d$ können alle Punkte $x_\theta \in K$ mit einer Konvexkombination aller anderen $x_i \in K,\ i = 1, \ldots, N$, dargestellt werden:

$$x_\theta = \sum_{i=1}^{N} \theta_i x_i,\ \sum_{i=1}^{N} \theta_i = 1,\ \theta_i \geq 0. \tag{3.1}$$

Eine konvexe Funktion $f(x)$ ist gegeben als:

$$f(x_\theta) \leq (1-\theta) f(x_0) + \theta f(x_1), \tag{3.2}$$

wobei x_θ eine Konvexkombination aus x_0 und x_1 ist (vgl. (3.1)). Wenn K eine offene Menge ist und $f(x)$ einmal stetig differenzierbar, dann kann die Konvexität von $f(x)$ auch alternativ formuliert werden als:

$$f(x_1) \geq f(x_0) + (x_1 - x_0)^T \nabla f(x_0), \tag{3.3}$$

wobei $x_0, x_1 \in K$. Der Graph der Funktion $f(x)$ liegt also über der Linearisierung von $f(x)$ an der Stelle x_0 (vgl. Abschnitt 6.2).
Ein konvexes Optimierungsproblem minimiert eine konvexe Funktion auf einer konvexen Menge:

$$\begin{aligned} &\min_x f(x) \\ &\quad \text{u.d.N. } x \in K \equiv \{x \mid c_i(x) \geq 0,\ i = 1, \ldots, m\}. \end{aligned} \tag{3.4}$$

Die konvexe Menge K ist durch eine Menge von konkaven Funktionen $c_i(x)$ definiert. Laut [9] ist die Menge $S(k) = \{x|c(x) \geq k\}$ konvex und der Schnitt konvexer Mengen ist wieder eine konvexe Menge. Somit ist K konvex.

Für konvexe Funktionen gelten folgende Eigenschaften:

1. Jedes lokale Minimum eines konvexen Optimierungsproblems (3.4) ist auch eine globales Minimum.

2. Ist die Zielfunktion des Optimierungsproblems $f(x)$ sogar strikt konvex, dann existiert nur ein globales Minimum.

Da jede lineare Funktion gleichzeitig konvex und konvav ist, sind die linearen Nebenbedingungen der Stützvektormethode konvex und konkav. Damit ist der zulässige Bereich, der durch die Nebenbedingungen definiert wird, konvex. Die Zielfunktion ist die Summe konvexer Funktion. Also ist auch die gesamte Funktion konvex und das durch die Stützvektormethode gegebene Optimierungsproblem (2.15) definiert ein konvexes Optimierungsproblem.

Um aus einem Optimierungsproblem mit Nebenbedingungen, wie (2.15), ein unbeschränktes Optimierungsproblem zu konstruieren, wird die Zielfunktion durch Lagrange-Multiplikatoren mit den Nebenbedingungen kombiniert. Die daraus resultierende Funktion wird primale Funktion genannt. Da nur Nebenbedingungen, für die Gleichheit gilt, einen Einfluss auf das Optimierungsergebnis haben, werden die Nebenbedingungen entsprechend umgeformt. Sind $f(x)$ die zu minimierende Funktion, $c_i(x) = 0$ die umgeformten Nebenbedingungen und $\lambda_i,\ i = 1, \ldots, m$, die Lagrange-Multiplikatoren, dann beschreibt

$$L_p = L_p(x, \lambda) = f(x) - \sum_{i=1}^{m} \lambda_i c_i(x) \tag{3.5}$$

die primale Funktion. Im folgenden Theorem 3.1 sind die notwendigen Bedingungen für die Optimalität des Punktes x beschrieben (vgl. Theorem 9.1.1 in [9]):

Theorem 3.1
Besitzt die Funktion $f(x)$ an der Stelle x ein lokales Minimum, dann existieren Lagrange-Multiplikatoren λ, so dass der Punkt (x, λ) die folgenden Bedingungen erfüllt:

$$\begin{aligned} \frac{\partial L_P(x,\lambda)}{\partial x} &= 0 \\ c_i(x) &= 0,\ \forall i = 1, \ldots, m \\ \lambda_i(x) &\geq 0,\ \forall i = 1, \ldots, m \\ \lambda_i c_i(x) &= 0,\ \forall i = 1, \ldots, m. \end{aligned} \tag{3.6}$$

Sind die Nebenbedingungen c_i des konvexen Optimierungsproblems (3.4) einmal stetig differenzierbar, sind sie nicht nur notwendig, sondern sogar hinreichend für ein globales Minimum. Dann ist auch jeder stationäre Punkt der primalen Funktion ein globales

Minimum. Die in Theorem 3.1 genannten Bedingungen werden auch als Karush-Kuhn-Tucker (KKT)-Bedingungen bezeichnet.

Die duale Funktion ist als das Infimum der primalen Funktion über die Lagrange-Multiplikatoren definiert:

$$L_D(\lambda) = \inf_{x \in K} L_P(x, \lambda) = \inf_{x \in K} \left(f(x) - \sum_{i=1}^{m} \lambda_i c_i(x) \right). \tag{3.7}$$

Die duale Funktion gibt eine untere Schranke für den optimalen Wert der primalen Funktion an. Mit Hilfe von Theorem 3.2 kann eine Verbindung zwischen den Lösungen des primalen und des dualen Problems eines konvexen Optimierungsproblems hergestellt werden (vgl. Theorem 9.5.1 in [9]):

Theorem 3.2
Sei x die Lösung für ein primales konvexes Optimierungsproblem (3.4). Wenn $f(x)$ und $c_i(x),\ i = 1, \ldots, m$, einmal stetig differenzierbar sind, dann löst (x, λ) das duale Problem

$$\begin{aligned} &\max_{x,\lambda} L_D(x, \lambda) \\ &u.d.N.\ \frac{\partial L_P(x, \lambda)}{\partial x} = 0\ , \lambda \geq 0. \end{aligned} \tag{3.8}$$

Außerdem sind der minimale Funktionswert von L_P und der maximale von L_D identisch.

3.2 Lösungsverfahren für die Stützvektormethode

Sind N Datenpunkte gegeben und soll (2.28) gelöst werden, dann könnten alle möglichen Kombinationen der Lagrange-Multiplikatoren in einer Laufzeit $\mathcal{O}\left(N^3\right)$ iteriert werden. Der Platzbedarf beträge $\mathcal{O}\left(N^2\right)$. Für die meisten praktischen Anwendungen ergeben sich große Datenmengen und somit ist dieser naive Ansatz für eine Lösung des Optimierungsproblems nicht praktikabel.

Im Folgenden werden zwei Arten von Lösungsverfahren für die Stützvektormethode mit Kernen vorgestellt: Verfahren, die eine Approximation der Kernmatrix zur Lösung des Optimierungsproblems benutzen, und Dekompositionsverfahren, die das Gesamtoptimierungsproblem durch eine Reihe von kleineren Optimierungsproblemen lösen. Insbesondere wird die Sequential Minimal Optimization vorgestellt, die auch in der Implementierung Anwendung findet.

3.2.1 Approximation der Kernmatrix

Es gibt verschiedene Arten die Kernmatrix zu approximieren. Häufig wird die Nyström Methode verwendet. Diese Methode wird dazu benutzt, die Eigenwertzerlegung der Kernmatrix zu approximieren. Mit Hilfe der Eigenwertzerlegung kann eine schnellere Lösung gefunden werden (vgl. [45]). Eine weitere Methode ist das Sampling. Dabei wird durch eine gute Auswahl von Punkten eine Kernmatrix mit kleinerem Rang gebildet (vgl. [2]).

3.2.2 Dekompositionsverfahren

Eines der ersten Verfahren zum Umgang mit großen Datenmengen ist das sogenannte Chunking (vgl. [41]). Dieses Verfahren nutzt aus, dass die Lösung von (2.25) nur auf Lagrange-Multiplikatoren $\alpha > 0$ basiert. Chunking verfolgt ein iteratives Schema. In jeder Iteration wird ein Optimierungsproblem auf den Beobachtungen x_i mit $\alpha_i > 0$ und den M Beobachtungen, die die KKT-Bedingungen (vgl. Theorem 3.1) am stärksten verletzen, gelöst. Werden keine Beobachtungen mehr gefunden, die die KKT-Bedingungen verletzen, ist das Gesamtproblem gelöst. Dieses Verfahren hat zwar einen geringeren Platzbedarf als der naive Ansatz, für große Datenmengen ist die Verringerung aber noch nicht groß genug.

Osunas Algorithmus (vgl. [23]) verfolgt eine zum Chunking ähnliche Vorgehensweise. Es wird eine Reihe von kleineren Optimierungsproblemen mit fester Größe optimiert. In jeder Iteration wird mindestens ein Punkt aus dem Optmierungsproblem entfernt und ein neuer hinzugefügt. In [23] wurde gezeigt, dass der Algorithmus konvergiert, wenn in jeder Iteration mindestens ein Punkt, der die KKT-Bedingungen verletzt, in das Optimierungsproblem aufgenommen wird. Durch die Betrachtung eines Optimierungsproblems mit geringerer und fester Größe reduziert sich der Platzbedarf gegenüber dem Chunking weiter. Allerdings benötigt dieser Algorithmus einen numerischen Optimierer, der Probleme mit der Genauigkeit mit sich bringen kann.

3.2.3 Sequential Minimal Optimization

Eine Erweiterung von Osunas Idee ist die Sequential Minimal Optimization (SMO) (vgl. [25]), die in jeder Iteration ein quadratisches Problem für genau zwei Beobachtungen löst. Der Vorteil ist, dass kein zusätzlicher Speicher für die Kernmatrix benötigt wird und für jeweils zwei Beobachtungen ein Subproblem analytisch gelöst werden kann. Für die Konvergenz wird Osunas Theorem (vgl. [23]) ausgenutzt, so dass in jeder Iteration mindestens ein Punkt ausgewählt wird, der die KKT-Bedingungen verletzt. Der Algorithmus löst also endlich viele Subprobleme für jeweils zwei Beobachtungen analytisch, um das Gesamtproblem zu lösen. Demnach wird für die Lösung kein numerischer Optimierer benötigt.

Nach Auswahl zweier Lagrange-Multiplikatoren o.B.d.A. α_1 und α_2 muss eine Lösung für das Subproblem berechnet werden. Für die Optimalität der Lösung sind für alle Lagrange-Multiplikatoren $0 \leq \alpha_i \leq C$ (vgl. (2.19)) und $\sum_{i=1}^{N} \alpha_i y_i = 0$ (vgl. (2.18)) notwendige Bedingungen. Die erste Bedingung beschränkt die beiden Lagrange-Multiplikatoren auf eine Box mit maximaler Kantenlänge C und die zweite Bedingung führt dazu, dass α_1 und α_2 auf einer Geraden liegen. In Kombination beider Bedingungen liegt das Minimum auf der Geraden innerhalb der Box (vgl. Abb. 3.1).

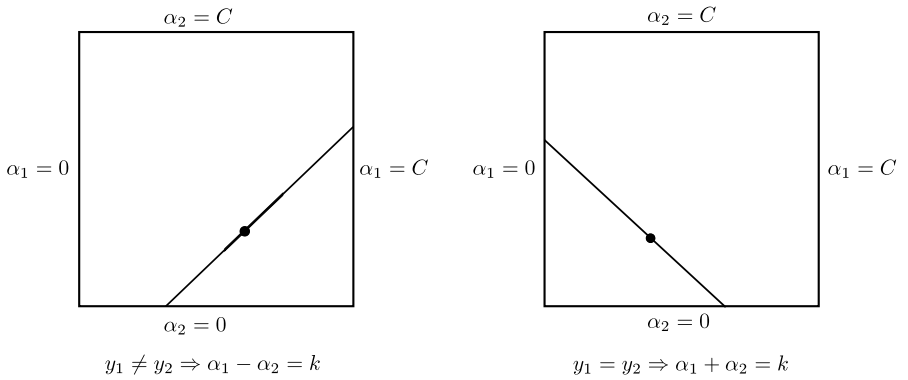


Abbildung 3.1: Lage des Optimums für zwei Lagrange-Multiplikatoren.

O.B.d.A. wird zuerst eine Lösung für α_2 berechnet. Der Geradenabschnitt in der Box wird durch α_2 ausgedrückt. Befinden sich die Beobachtungen der beiden Lagrange-Multiplikatoren in verschiedenen Klassen, gelten folgende Beschränkungen für α_2:

$$\begin{aligned} L(\alpha_2)_{y1 \neq y2} &= \max\{0, \alpha_2 - \alpha_1\} && \Leftrightarrow \alpha_2 \geq 0 \wedge \alpha_1 \geq 0 \\ H(\alpha_2)_{y1 \neq y2} &= \min\{C, C + \alpha_2 - \alpha_1\} && \Leftrightarrow \alpha_2 \leq C \wedge \alpha_1 \leq C. \end{aligned} \tag{3.9}$$

Für Beobachtungen in gleichen Klassen gilt:

$$\begin{aligned} L(\alpha_2)_{y1 = y2} &= \max\{0, \alpha_2 + \alpha_1 - C\} && \Leftrightarrow \alpha_2 \geq 0 \wedge \alpha_1 \leq C \\ H(\alpha_2)_{y1 = y2} &= \min\{C, \alpha_2 + \alpha_1\} && \Leftrightarrow \alpha_2 \leq C \wedge \alpha_1 \geq 0, \end{aligned} \tag{3.10}$$

wobei L die untere Schranke und H die obere Schranke beschreiben. Die zweite Ableitung der Zielfunktion entlang der diagonalen Linie kann mit der Kernfunktion k ausgedrückt werden als:

$$\eta = k(x_1, x_1) + k(x_2, x_2) - 2k(x_1, x_2). \tag{3.11}$$

Ist die Zielfunktion positiv definit, liegt ein Minimum in der Richtung der linearen Gleichungsbedingung und η ist größer als 0. In diesem Fall wird das Minimum berechnet als:

$$\alpha_2^{new} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}, \tag{3.12}$$

wobei $E_i = u_i - y_i$, der Fehler der $i$ten Beobachtung ist mit u_i als Vorhersage für y_i (vgl. (2.26)). Falls das Minimum außerhalb der Schranken liegt, muss der Wert zusätzlich beschnitten werden:

$$\alpha_2^{new,clipped} = \begin{cases} H, & \text{wenn } \alpha_2^{new} \geq H \\ \alpha_2^{new}, & \text{wenn } L < \alpha_2^{new} < H \\ L, & \text{wenn } \alpha_2^{new} \leq L. \end{cases} \tag{3.13}$$

Sei $s = y_1 y_2$, dann berechnet sich der Wert von α_1 aus dem neu berechneten beschnittenem α_2 als:

$$\alpha_1^{new} = \alpha_1 + s(\alpha_2 - \alpha_2^{new,clipped}). \tag{3.14}$$

Um die Konvergenz zu beschleunigen, wird eine spezielle Auswahl der beiden Lagrange-Multiplikatoren durchgeführt. Für die Auswahl der beiden Multiplikatoren existieren zwei verschiedene Heuristiken. Die erste Heuristik iteriert zuerst über die gesamte Trainingsmenge und überprüft für alle Beobachtungen, ob die zugehörigen Lagrange-Multiplikatoren die KKT-Bedingungen verletzen. Nachdem einmal über die gesamte Trainingsmenge iteriert wurde, wird über alle Beobachtungen mit $0 < \alpha_i < C$ iteriert, solange bis für alle die KKT-Bedingungen erfüllt sind. Danach iteriert die Heuristik erneut über die gesamte Trainingsmenge. Sind für Beobachtungen die KKT-Bedingungen bis auf ein $\epsilon > 0$ erfüllt, stoppt der Algorithmus.

Der zweite Lagrange-Multiplikator wird so gewählt, dass die Verbesserung durch die Optimierung der beiden Punkte maximal ist. Da eine Berechnung einer maximalen Lösung zeitaufwändig ist, wird die Verbesserung durch $|E_2 - E_1|$ approximiert (vgl. (3.12)). Dadurch wird entweder die Beobachtung mit minimalem oder maximalem Fehler E_2 gewählt.

Damit nach der Optimierung α_1 und α_2 die KKT-Bedingungen erfüllen und der Algorithmus konvergiert, muss die Verschiebung β_0 nach jeder Iteration neu berechnet werden (vgl. [25]).

4 Laufzeitverbesserungen der Stützvektormethode mittels approximativen Lösungen

Mit einer empirischen Laufzeit zwischen $\mathcal{O}(N)$ und $\mathcal{O}\left(N^{2.2}\right)$ (vgl. [25]) löst der SMO-Optimierer (vgl. Abschnitt 3.2.3) das durch die Stützvektormethode gegebene Optimierungsproblem schneller als der naive Optimierer aus Abschnitt 3.1. Auf großen Datenmengen ist der Einsatz des Optimierers aber nicht praktikabel.

Anstatt den für die Lösung der Stützvektormethode genutzten Optimierer zu verbessern, wird in diesem Abschnitt ein anderer Ansatz vorgestellt. Das Optimierungsproblem wird zu einem äquivalenten Problem, dem Minimum Enclosing Ball Problem (MEB-Problem), umformuliert und dann gelöst.

Im Folgenden wird das MEB-Problem genauer erläutert, sowie mögliche Laufzeitverbesserungen begründet. Um die Verbindung zwischen dem Optimierungsproblem der Stützvektormethode und dem MEB-Problem herzustellen, wird in Abschnitt 4.2 die Support Vector Data Description eingeführt und der Zusammenhang beider Probleme beschrieben. Danach wird die Core Vector Machine (CVM) vorgestellt, die den Zusammenhang der beiden Probleme ausnutzt und damit eine Verbesserung der Laufzeit erreicht. Die Core Vector Machine ist besonders für große Datenmengen geeignet, da sie eine lineare Laufzeit und einen von der Menge der Beispiele unabhängigen Platzbedarf hat. Zum Abschluss des Kapitels wird die Ball Vector Machine (BVM) eingeführt. Dieses heuristische Verfahren kommt im Gegensatz zu den anderen vorgestellten Verfahren ohne jeglichen Optimierer aus. Es wird sich zeigen, dass die BVM eine zu den anderen Verfahren vergleichbare Klassifikationsgüte liefert und eine geringere Laufzeit dafür benötigt.

4.1 Minimum Enclosing Ball Problem

Die erste Formulierung des MEB-Problems kann auf [35] zurück geführt werden. Das MEB-Problem lässt sich wie folgt beschreiben: Sei S eine Menge von Beispielen mit $S = \{x_1, \ldots, x_N\},\ x_i \in \mathbb{R}^d$. Der Minimum Enclosing Ball von S MEB(S) ist der Ball $B(c^*, R^*)$ mit Zentrum c^* und minimalem Radius R^*, der alle Datenpunkte aus S enthält. Es existieren einige Verfahren, die versuchen eine exakte Lösungen für das MEB-Problem zu finden (siehe beispielsweise [44]). Diese Verfahren können aber für Räume mit mehr als 30 Dimensionen ($R^d, d > 30$) eine exakte Lösung nicht effizient berechnen. Da in der Praxis jedoch häufig Datensätze mit mehr als 30 Attributen existieren, muss eine effizientere Lösung gefunden werden, wenn durch eine Umformulierung des durch die Stützvektormethode gegebenen Optimierungsproblems in ein MEB-Problem eine echte

Laufzeitverbesserung erreicht werden soll. Wenn die Bedingung der Exaktheit der Lösung fallen gelassen wird, kann ein effizienteres Lösungsverfahren gefunden werden (vgl. [39]). Dieses Verfahren muss nicht mehr den minimalen Ball $B(c^*, R^*)$ finden, sondern es genügt ein um $\epsilon > 0$ vergrößerter Ball als Lösung. Der Ball $B(c, (1+\epsilon)R)$ für ein gegebenes $\epsilon > 0$ ist eine $(1+\epsilon)$-Approximation des MEB(S), wenn $R \leq R_{MEB(S)}$ und $S \subset B(c, (1+\epsilon)R)$.

Eine Möglichkeit für das effiziente Auffinden einer solchen approximativen Lösung liefert der in [3] vorgestellte Algorithmus. Der Algorithmus folgt einem einfachen iterativem Schema: Nachdem ein zufälliger Startpunkt gewählt wurde, wird in jeder Iteration ein Punkt dem sogenannten CoreSet hinzugefügt. Eine Teilmenge aller Beispiele $Q \subseteq S$ heißt CoreSet, wenn S in dem um $\epsilon > 0$ erweiterten Ball MEB(Q) enthalten ist, also $S \subset B(c, (1+\epsilon)R)$ mit $B(c, R) = \text{MEB}(Q)$. Um in möglichst wenigen Iteration eine Approximation zu finden, wird in jeder Iteration der Punkt dem CoreSet hinzugefügt, der am weitesten vom Zentrum des bisher gefundenen Balls entfernt ist. Dieser Vorgang wird solange wiederholt, bis kein Punkt aus S mehr außerhalb des Balls $B(c, (1+\epsilon)R)$ liegt. Obwohl das CoreSet weniger Punkte als die Ausgangsmenge S enthält, hat die $(1+\epsilon)$-Approximation des CoreSets mindestens ein genauso großes Volumen, wie der MEB der Ausgangsmenge. Das Auffinden des kleinsten Balls wird also nicht einmal auf allen Punkten durchgeführt, sondern in jeder Iteration wird für weniger Punkte der MEB gesucht und dann überprüft, ob sich noch Punkte außerhalb des Balls befinden. Anstatt eines großen Optimierungsproblems wird also eine Reihe von kleineren Optimierungsproblemen gelöst. Ein Beispiel für diese Situation ist im rechten Teil von Abb. 4.2 dargestellt. Die markierten Punkte sind Teil des CoreSets. Da kein Punkt mehr außerhalb von $B(c, (1+\epsilon)R)$ gefunden werden kann, stoppt der Algorithmus.

Eine wichtige Eigenschaft des Algorithmus für die Anwendbarkeit für die Lösung des durch die Stützvektormethode gestellten Optimierungsproblems ist, dass die Anzahl der Iterationen unabhängig von der Dimensionalität des Merkmalsraums ist, da durch die Basis Expansion mit Hilfe einer Kern-Funktion der Merkmalsraum unendlich dimensional werden kann. Wird ein Punkt außerhalb gefunden, wird der Ball in alle Dimensionen gleichermaßen vergrößert. Die Laufzeit hängt nur von der gewählten Approximationsgüte ϵ und der Anzahl der Beobachtungen ab. Wird für eine feste Anzahl von zufällig ausgewählten Punkten der Abstand zum Ball überprüft, kann eine von der Anzahl der Beobachtungen unabhängige Laufzeit erreicht werden. Der Algorithmus wird im Detail in Abschnitt 4.3 vorgestellt.

4.2 Support Vector Data Description

Die Support Vector Data Description ist als Zwischenschritt für die Umformulierung des durch die Stützvektormethode gegebenen Optimierungsproblems in ein MEB-Problem nötig. In diesem Abschnitt wird zuerst die Support Vector Data Description und deren Verwendungsmöglichkeiten beschrieben. Analog zu Abschnitt 2.3 wird danach die mathematische Formulierung des zu lösenden Problems vorgestellt und soweit umgeformt, dass

eine Verwendung von Kern-Funktionen möglich ist. Zum Abschluss des Abschnitts wird sich herrausstellen, dass nicht alle Kern-Funktionen für die Support Vector Data Description und damit auch nicht für die noch zu beschreibenden Verfahren geeignet sind. Einige wichtige Eigenschaften der geeigneten Kern werden beschrieben.

Anders als bei der Stützvektormethode wird bei der Support Vector Data Description keine binäre sondern eine Ein-Klassen-Klassifikation durchgeführt (vgl. Abschnitt 2.2). Da die Daten in der Support Vector Data Description ebenfalls durch einen minimalen Ball beschrieben werden, ist die Äquivalenz zum MEB-Problem direkt ersichtlich. Es wird der Ball gesucht, der mit minimalem Volumen alle gegebenen Beispiele enthält. Durch die Minimierung des Volumens werden Punkte, die keine Ähnlichkeit zu den beschriebenen Beispielen haben, leichter gefunden.

4.2.1 Datenbschreibung

Ein einfaches Modell zur Beschreibung einer Menge von Beispielen ist, einen Ball B mit minimalem Radius R um alle Beispiele zu legen. Die Ähnlichkeit der Beispiele wird dann durch die maximale Distanz zum Zentrum c des Balls, also durch den Radius, ausgedrückt. Der Ball B kann definiert werden als:

$$\begin{aligned} \mathbf{SVDD}_0 : & \min_{R,c} R^2 \\ & \text{u.d.N. } \|x_i - c\|^2 \leq R^2, \ \forall i = 1, \ldots, N. \end{aligned} \tag{4.1}$$

Auch Ausreißer können in der Menge der Beipiele enthalten sein. Um deren Einfluss zu verringern, können wie bei der Formulierung der SVM (vgl. (2.15)) Slack-Variablen ξ_i aufgenommen werden. Mit der Einführung der Slack-Variablen werden die Nebenbedingungen gelockert:

$$\begin{aligned} \mathbf{SVDD}_1 : & \min_{R,c} R^2 + C \sum_{i=1}^{N} \xi_i \\ & \text{u.d.N. } \|x_i - c\|^2 \leq R^2 + \xi_i, \ \xi_i \geq 0, \ \forall i = 1, \ldots, N. \end{aligned} \tag{4.2}$$

Analog zu Formel (2.15) kontrolliert der Parameter C die Summe aller benutzen Slack-Variablen und damit auch das Volumen des Balls. Mit einem größeren Volumen steigt die Wahrscheinlichkeit für Ausreißer. Mit C kann also der Trade-off zwischen einem minimalen Ball und der Anzahl falsch klassifizierter Ausreißer kontrolliert werden.

Die Lösung des Problems beruht, wie Formel (2.15), auf der Lagrange-Multiplikator-Methode. Für die primale Funktion gilt:

$$L_P = R^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [R^2 + \xi_i - (\|x_i\|^2 - 2cx_i + \|c\|^2)] - \sum_{i=1}^{N} \mu_i \xi_i, \tag{4.3}$$

wobei $\alpha_i \geq 0$ und $\mu_i \geq 0, \ \forall i = 1, \ldots, N$.

Die Ableitung der primalen Funktion ergibt die folgenenden notwendigen Bedingungen für die Optimalität der Lösung:

$$\frac{\partial L_P}{\partial R} \stackrel{!}{=} 0 \Leftrightarrow \sum_{i=1}^{N} \alpha_i = 1 \tag{4.4}$$

$$\frac{\partial L_P}{\partial c} \stackrel{!}{=} 0 \Leftrightarrow c = \sum_{i=1}^{N} \alpha_i x_i \tag{4.5}$$

$$\frac{\partial L_P}{\partial \xi_i} \stackrel{!}{=} 0 \Leftrightarrow C - \alpha_i - \mu_i = 0. \tag{4.6}$$

Durch die zusätzliche Bedingung

$$0 \leq \alpha_i \leq C \tag{4.7}$$

kann in Verbindung mit den geltenden Bedingungen $\alpha_i \geq 0, \mu_i \geq 0$ analog zu Formel (2.24) μ_i aus Formel (4.6) entfernt werden. Durch Einsetzen der Bedingungen (4.4) bis (4.7) in die primale Funktion ergibt sich die duale Funktion:

$$\begin{aligned} L_D &= \sum_{i=1}^{N} \alpha_i (x_i^T x_i) - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j x_i^T x_j \\ &\text{u.d.N. } 0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, N. \end{aligned} \tag{4.8}$$

Wird die duale Funktion L_D maximiert, lösen die optimalen α_i auch das primale Problem (vgl. Abschnitt 2.3). Es ergeben sich folgende Eigenschaften der α_i und μ_i aus der Distanz der Beipiele x_i zum Zentrum des Balls:

$$\|x_i - c\|^2 < R^2 \Rightarrow \alpha_i = 0, \mu_i = 0 \tag{4.9}$$

$$\|x_i - c\|^2 = R^2 \Rightarrow 0 < \alpha_i < C, \mu_i = 0 \tag{4.10}$$

$$\|x_i - c\|^2 > R^2 \Rightarrow \alpha_i = C, \mu_i > 0. \tag{4.11}$$

Formel (4.5) zeigt, dass c eine Linearkombination der α_i ist. Nur Beispiele, die auf dem Ball (4.10) liegen, haben einen Einfluss auf das Ergebnis von (4.5). Diese Beispiele werden Stützvektoren genannt, da das Zentrum nur durch sie beschrieben wird (vgl. (2.25)).

Um die Ähnlichkeit von Beobachtungen einer unabhängigen Stichprobe zur Menge der durch das gelernte Modell beschriebenen Daten zu überprüfen, wird die Distanz der Beobachtungen zum Zentrum des Balls errechnet. Damit kann eine Beobachtung ggf. als Ausreißer identifiziert werden. Der Abstand einer Beobachtung z zum Zentrum kann errechnet werden als:

$$\|z - c\|^2 = z^T z - 2 \sum_{i=1}^{N} \alpha_i x_i^T x_i + \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j x_i^T x_j. \tag{4.12}$$

Der Radius kann aus dem Abstand zwischen dem Zentrum c und irgendeinem Stützvektor x_k mit $0 < \alpha_k < C$ abgeleitet werden so dass gilt:

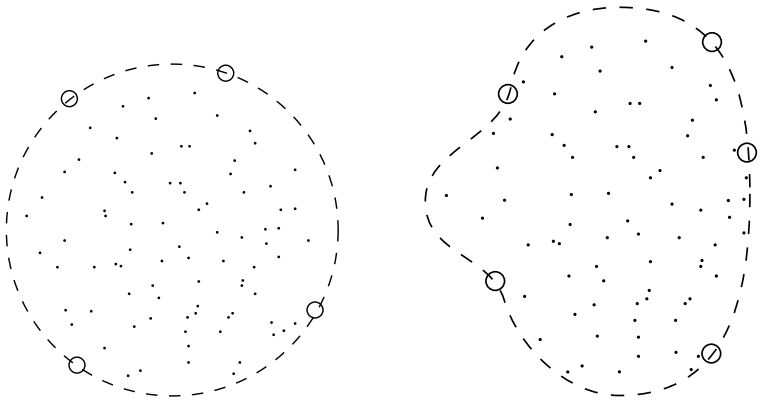
$$R^2 = x_k^T x_k - 2 \sum_{i=1}^{N} \alpha_i x_k^T x_i + \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j x_i^T x_j. \tag{4.13}$$

4.2.2 Flexiblere Modellierung der Daten mit Kernen

In Kapitel 2 wurde eine Erweiterung der Stützvektormethode vorgestellt, die eine Trennung der Daten im Ursprungsraum durch eine nicht lineare Entscheidungsschranke ermöglicht. Dafür wurde eine Kern-Funktion in die duale Formulierung des durch die Stützvektormethode gegebenen Optimierungsproblems integriert. Auch die duale Formulierung des durch die Support Vector Data Description gestellten Optimierungsproblems (4.8) verwendet die Beobachtungen ausschließlich in einem Skalarprodukt. Anstelle des einfachen Skalarprodukts kann ebenfalls eine beliebige Kern-Funktion $k(x_i, x_j)$ eingesetzt werden:

$$\begin{aligned} L_D &= \sum_{i=1}^{N} \alpha_i (x_i^T x_i) - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \\ &\text{u.d.N. } 0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, N. \end{aligned} \tag{4.14}$$

Wie in Abb. 4.1 beispielhaft zu sehen, kann durch die Verwendung eines Kerns die starre Kugelform der Entscheidungschranke verformt werden.



(a) Kreuzprodukt-Kern **(b)** polynomieller Kern

Abbildung 4.1: Verwendung verschiedener Kerne zur Beschreibung des Balls

In Abschnitt 2.3 wurden einige für die Stützvektormethode geeignete Kerne vorgeschlagen. Laut [36] sind aber nicht alle Kerne geeignet, die Beispiele in einen beschränkten Bereich im Merkmalsraum zu transformieren. Der RBF-Kern ist besonders geeignet, da die Beispiele im höher dimensionalen Merkmalsraum alle die gleiche Länge haben. Es kann gezeigt werden, dass $C = 1$ für den RBF-Kern eine gute Wahl ist, unter der Annahme, das der Trainingsdatensatz ohne Fehler beschrieben werden kann (vgl. [36]).

4.3 Core Vector Machine

Im Folgenden wird zuerst die Verbindung zwischen dem durch die Stützvektormethode gegebenen Optimierungsproblem und dem MEB-Problem über die Support Vector Data Description hergestellt. Danach wird der eigentliche Algorithmus der Core Vector Machine und seine einzelnen Schritte zur Lösung des Optimierungsproblems erläutert. Zum

Abschluss werden Aussagen über die Laufzeit und den Platzbedarf getroffen.

Das MEB-Problem sieht keine Ausreißer vor und wird damit analog zu (4.1) formuliert:

$$\begin{aligned} \textbf{MEB} : \min_{R,c} \; & R^2 \\ \text{u.d.N.} \quad & \|\varphi(x_i) - c\|^2 \leq R^2, \forall i = 1, \ldots, N, \end{aligned} \tag{4.15}$$

wobei $\varphi(x_i)$ ein transformierter Punkt im Merkmalsraum und c das zu wählende Zentrum des Balls ist. Die zugehörige duale Funktion aus Formel (4.8) mit einem Kern anstelle des Skalarprodukts ist:

$$\begin{aligned} L_D &= \sum_{i=1}^{N} \alpha_i k(x_i, x_i) - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \\ \text{u.d.N. } \alpha_i &\geq 0, \; \sum_{i=1}^{N} \alpha_i = 1 \; \forall i = 1, \ldots, N \end{aligned} \tag{4.16}$$

In Matrixschreibweise ergibt sich die duale Funktion als:

$$\begin{aligned} \max_{\alpha} \; & \alpha^T \text{diag}(\mathbb{K}) - \alpha^T \mathbb{K} \alpha \\ & \alpha \geq \mathbf{0}, \; \alpha^T \mathbf{1} = 1, \end{aligned} \tag{4.17}$$

wobei $\mathbf{1} = [1, \ldots, 1]$, $\mathbf{0} = [0, \ldots, 0]$, $\alpha = [\alpha_1, \ldots, \alpha_N]$ die Lagrange-Multiplikatoren, $\mathbb{K}_{N \times P} = k(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle$ die Kernmatrix und $\varphi(x)$ die Feature-Map beschreibt. Durch die Formulierung des Optimierungsproblems in Matrixschreibweise werden die weiteren Umformungen schlüssiger.

Das Zentrum c (vgl. Formel (4.5)) und der Radius R des gefundenen Balls können mit den optimalen α_i bestimmt werden:

$$\begin{aligned} c &= \sum_{i=1}^{N} \alpha_i \varphi(x_i) \\ R &= \sqrt{\alpha^T \text{diag}(\mathbb{K}) - \alpha^T \mathbb{K} \alpha}. \end{aligned} \tag{4.18}$$

4.3.1 Zusammenhang zwischen Kern-Methoden und dem MEB-Problem

Die Verbindung zwischen Kern-Methoden im Allgemeinen und dem MEB-Problem kann mit einer speziellen Anforderung an die Kern-Funktion hergestellt werden:

$$k(x, x) = \kappa, \; \forall \, x. \tag{4.19}$$

Ein Kern definiert ein Skalarprodukt auf dem Merkmalsraum und induziert damit eine Norm. Besitzen alle Beispiele eine konstante Norm, dann liegen sie auf einem Ball. Alle Kern-Funktionen, die (4.19) erfüllen, transformieren die Beipiele also auf einen Ball. Dies gilt für:

1. isotrope Kern-Funktionen $k(x, y) = K(\|x - y\|)$, beispielsweise den RBF-Kern
2. Skalarprodukt Kern-Funktionen $k(x, y) = K(x^T y)$, beispielsweise den polynomiellen Kern mit normalsierten Eingaben
3. jede normalisierte Kern-Funktion $k(x, y) = \frac{K(x,y)}{\sqrt{K(x,x)}\sqrt{K(y,y)}}$.

Mit den Eigenschaften der Support Vector Data Description (vgl. [36]) wird in üblichen Anwendungen aber eine RBF-Kern-Funktion gewählt.

Die Verbindung zwischen Kern-Methoden und dem MEB-Problem wird über die in [31] gezeigte Verbindung zwischen Ein-Klassen- und Zwei-Klassen-Problemen hergestellt. Kann für eine Menge von Beispielen aus einer Klasse eine Hyperebene $w^T x > 0$ gefunden werden, die die Beispiele vom Ursprung trennt, dann existiert laut Proposition 1 in [31] eine eindeutige Supporting Hyperplane $w^T \varphi(x) = \rho$, die die Beispiele mit maximaler Distanz vom Ursprung trennt. Angenommen für ein binäres Klassifikationsproblem existiert eine trennende Hyperebene, die durch den Ursprung verläuft. Dann kann laut Proposition 2 in [31] eine Supporting Hyperplane mit gleichem Abstand wie der Margin der trennenden Hyperebene gefunden werden, so dass alle Beispiele multipliziert mit ihrer Klasse maximal vom Ursprung getrennt werden können. Liegen die Beispiele wie zuvor beschrieben auf einem Ball, dann ist das Auffinden der Supporting Hyperplane äquivalent zur Suche nach einem Ball mit minimalem Radius, der alle Beispiele enthält. Die Punkte, die auf der Supporting Hyperplane liegen, befinden sich auch auf dem Ball mit minimalem Radius (vgl. Abb. 4.2). Der Normalenvektor w der Supporting Hyperplane ist dabei identisch zum Zentrum c des Balls, wie in (4.15) verwendet.

Unter der Annahme, dass die Beispiele auf einem Ball liegen und die trennende Hyperebene durch den Ursprung verläuft, kann die Suche nach einer trennenden Hyperebene äquivalent zu einer Suche nach einem Ball mit minimalem Radius umgeformt werden. Da das durch die Support Vector Data Description gegebene Optimierungsproblem äquivalent zum MEB-Problem ist, kann das durch die Stützvektormethode gegebene Optimierungsproblem als MEB-Problem angesehen werden.

Damit das durch die Stützvektormethode gegebene Optimierungsproblem in ein MEB-Problem umgeformt werden kann, muss jedoch das Problem der Support Vector Data Description noch angepasst werden. Durch die Kombination von Formel (4.19) mit der Bedingung $\alpha^T \mathbf{1} = 1$ ergibt sich $\alpha^T \mathbb{K} = \kappa$. Diese Konstante κ kann im dualen Problem (4.17) vernachlässigt werden, da die optimale Lösung α beider Optimierungsprobleme identisch ist. Damit ergibt sich das vereinfachte Optimierungsproblem:

$$\begin{aligned} &\max_{\alpha} \; -\alpha^T \mathbb{K} \alpha \\ &\text{u.d.N. } \alpha \geq \mathbf{0},\ \alpha^T \mathbf{1} = 1. \end{aligned} \tag{4.20}$$

Eine weitere nötige Anpassung ist die quadratische Bestrafung eines Fehlers, da so die Bedingung der Slack-Variablen $\xi_i > 0$ vernachlässigt werden kann (vgl. [15]). Auch wenn durch diese Anpassung die statistische Robustheit in der Theorie (vgl. [34]) gegenüber

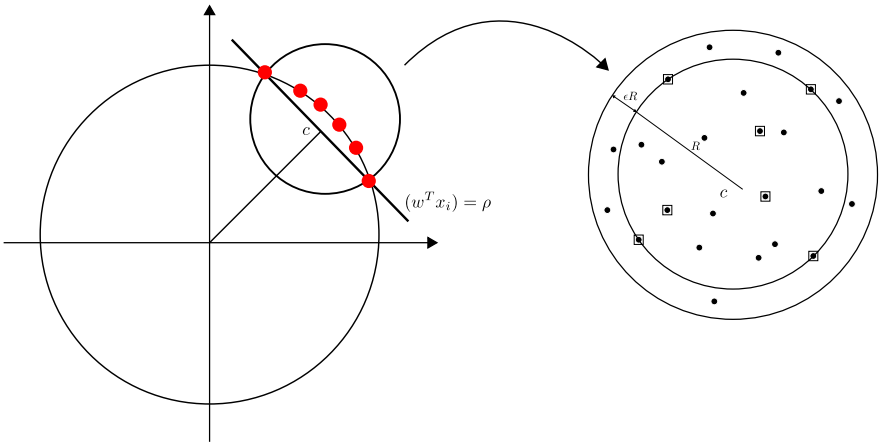


Abbildung 4.2: Zusammenhang zwischen Ein-Klassen-Klassifikation, Support Vector Data Description und Minimum Enclosing Ball Problem.

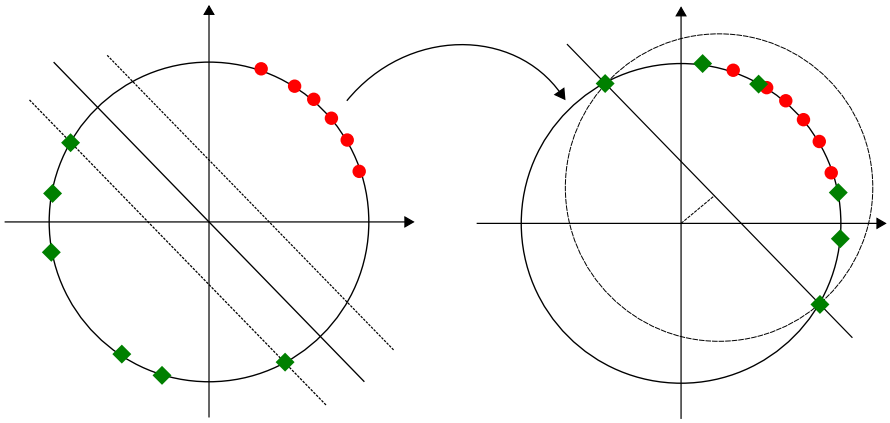


Abbildung 4.3: Übergang von einem Zwei- in ein Ein-Klassen-Problem

Ausreißern leidet, kann experimentell eine vergleichbare Güte zwischen der einfachen und der quadratischen Bestrafung festgestellt werden (vgl. [20]).

4.3.2 Ein-Klassen SVM

Die Formulierungen der Optimierungsprobleme der folgenden Ein-Klassen SVM und Zwei-Klassen SVM unterscheiden sich von der Darstellung der SVM in Kapitel 2, damit sie in ein äquivalentes MEB-Problem umgeformt werden können. Im Fall der Ein-Klassen SVM gibt der zusätzlich eingeführte Parameter ρ den Abstand der gelernten Hyperebene vom Ursprung an. Das Optimierungsproblem der veränderten Ein-Klassen SVM ist dann definiert als:

$$\begin{aligned} \mathbf{SVM}_1^c: \quad & \min_{w,\rho,\xi_i} \|w\|^2 - 2\rho + C\sum_{i=1}^{N} \xi_i^2 \\ & \text{u.d.N. } w^T\varphi(x_i) \geq \rho - \xi_i, \ \forall i = 1,\ldots,N, \end{aligned} \tag{4.21}$$

wobei $w^T\varphi(x_i) = \rho$ die gelernte Hyperebene darstellt und der Parameter C die Verwendung der Slack-Variablen kontrolliert. Durch das Einfügen von ρ in die Zielfunktion wird der Abstand der gelernten Hyperebene vom Ursprung maximiert.

Das zugehörige duale Problem in Matrixschreibweise lautet:

$$\begin{aligned} &\max_{\alpha} -\alpha^T \left(\mathbb{K} + \frac{1}{C}\mathbb{I}\right)\alpha \\ &\text{u.d.N. } \alpha \geq \mathbf{0}, \alpha^T\mathbf{1} = 1, \end{aligned} \tag{4.22}$$

wobei $\mathbb{I}$ die $N \times N$ Identitätsmatrix und $\mathbb{K}$ die Kernmatrix ist. Die Herleitung des dualen Problems ist sehr ähnlich zur Herleitung des dualen Problems der Zwei-Klassen SVM und wird nur für letzteren durchgeführt (vgl. (4.26)).

Wird das duale Problem umformuliert, wird die Äquivalenz zur Support Vector Data Description (4.20) direkt ersichtlich:

$$\begin{aligned} &\underset{\alpha}{max} -\alpha^T\tilde{\mathbb{K}}\alpha \\ &\text{u.d.N. } \alpha \geq \mathbf{0}, \alpha^T\mathbb{1} = 1 \end{aligned} \tag{4.23}$$

mit der erweiterten Kernmatrix

$$\tilde{\mathbb{K}} = \left[\tilde{k}(z_i, z_j)\right] = \left[k(x_i, x_j) + \frac{\delta_{ij}}{C}\right], \tag{4.24}$$

wobei $\delta_{ij} = \begin{cases} 1 & y_i = y_j,\ \forall i,j = 1,\ldots,N \\ 0 & sonst \end{cases}$. Auch diese erweiterte Kern-Funktion ist für alle Beispiele mit $\tilde{k}(z,z) = \kappa + \frac{\delta_1}{C} \equiv \tilde{\kappa}$ konstant (vgl. (4.19)). Die lineare Feature-Map φ mit $k(z_i, z_j) = \langle\varphi(z_i), \varphi(z_j)\rangle$ wird durch die nicht-lineare Featuremap $\tilde{\varphi}(z_i) = \begin{bmatrix}\varphi(x_i) \\ \frac{1}{\sqrt{C}}e_i\end{bmatrix}$ ersetzt, wobei e_i den $i$ten Einheitsvektor beschreibt. Die angepasste Ein-Klassen SVM kann also in eine äquivalentes MEB-Problem umgeformt werden.

4.3.3 Zwei-Klassen SVM

Auch das Optimierungsproblem der Zwei-Klassen SVM wird angepasst, damit es in ein zur Support Vector Data Description äquivalentes Optimierungsproblem (4.20) umgeformt werden kann. Es gilt:

$$\begin{aligned} \mathbf{SVM}_1^c: \quad &\min_{w,b,\rho,\xi_i} \|w\|^2 + b^2 - 2\rho + C\sum_{i=1}^{N}\xi_i^2 \\ &\text{u.d.N. } y_i(w^T\varphi(x_i) + b) \geq \rho - \xi_i,\ \forall i = 1,\ldots,N, \end{aligned} \tag{4.25}$$

wobei $\frac{\rho}{\|w\|}$ die Breite der Margin und $\frac{b}{\|w\|}$ den Abstand der Hyperebene vom Ursprung definieren. Das Optimierungsproblem der SVM kann nur zu einem MEB-Problem äquivalent sein, wenn die Hyperebene durch den Ursprung verläuft. Ist die Hyperebene zu

weit vom Ursprung entfernt, erhöht sich der Wert der Slack-Variablen. Neben dem eigentlichen Optimierungsziel der Marginmaximierung muss also zusätzlich der Abstand der Hyperebene zum Ursprung minimiert werden.

Die duale Funktion kann hergeleitet werden in dem zunächst die Nebenbedingungen und die Zielfunktion zur primalen Funktion kombiniert werden:

$$L_P = \|w\|^2 + b^2 - 2\rho + C\sum_{i=1}^{N}\xi_i^2 - \sum_{i=1}^{N}\alpha_i[y_i(w^T\varphi(x_i)+b) - \rho - \xi_i]. \tag{4.26}$$

Danach werden die partiellen Ableitungen bestimmt und gleich Null gesetzt. Daraus ergeben sich folgende Bedingungen für die Optimalität der Lösung:

$$\begin{aligned}
\frac{\partial L_P}{\partial b} &= 2b - \sum_{i=1}^{N}\alpha_i y_i \stackrel{!}{=} 0 \Leftrightarrow 2b = \sum_{i=1}^{N}\alpha_i y_i \\
\frac{\partial L_P}{\partial \rho} &= -2 + \sum_{i=1}^{N}\alpha_i \stackrel{!}{=} 0 \Leftrightarrow \sum_{=1}^{N} i\alpha_i = 2 \\
\frac{\partial L_P}{\partial w} &= 2w - \sum_{i=1}^{N}\alpha_i y_i \varphi(x_i) \stackrel{!}{=} 0 \Leftrightarrow 2w = \sum_{i=1}^{N}\alpha_i y_i \varphi(x_i) \\
\frac{\partial L_P}{\partial \xi_i} &= 2C\xi_i - \alpha_i \stackrel{!}{=} 0 \Leftrightarrow \xi_i = \frac{\alpha_i}{2C}.
\end{aligned}$$

Nach dem Einsetzen der obigen partiellen Ableitungen in die primale Funktion (4.26) ergibt sich die duale Funktion als:

$$\begin{aligned}
L_D &= \frac{1}{4}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j \varphi(x_i)\varphi(x_j) + \frac{1}{4}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j - 2\rho + C\sum_{i=1}^{N}\left(\frac{\alpha_i}{2C}\right)^2 \\
&\quad - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j \varphi(x_i)\varphi(x_j) - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j + \sum_{i=1}^{N}\alpha_i\rho - \sum_{i=1}^{N}\alpha_i\left(\frac{\alpha_i}{2C}\right) \\
&= -\frac{1}{4}\sum_{i,j=1}^{N}\alpha_i\alpha_j[y_i y_j \varphi(x_i)\varphi(x_j) + y_i y_j] + C\sum_{i=1}^{N}\left(\frac{\alpha_i^2}{4C^2}\right) - \sum_{i=1}^{N}\alpha_i\left(\frac{\alpha_i}{2C}\right) \\
&= -\frac{1}{4}\sum_{i,j=1}^{N}\alpha_i\alpha_j[y_i y_j \varphi(x_i)\varphi(x_j) + y_i y_j] - \frac{1}{4}\sum_{i=1}^{N}\left(\frac{\alpha_i^2}{C}\right) \\
&= -\frac{1}{4}\sum_{i,j=1}^{N}\alpha_i\alpha_j\left[y_i y_j \varphi(x_i)\varphi(x_j) + y_i y_j + \frac{\delta_{ij}}{C}\right]
\end{aligned}$$

mit δ_{ij} wie in (4.24).

Die duale Funktion kann wieder in Matrixschreibweise umformuliert werden:

$$\max_{\alpha} -\alpha^T \left(\mathbb{K} \odot yy' + yy' + \frac{1}{C}\mathbb{I} \right) \alpha \tag{4.27}$$

$$\text{u.d.N. } \alpha \geq \mathbf{0}, \alpha^T \mathbf{1} = 1, \tag{4.28}$$

wobei $y = [y_1, \ldots, y_N]$ und $\odot$ das Hadamard-Produkt bezeichnet. Das Hadamard-Produkt berechnet sich als das Produkt der einzelnen Elemente zweier Matrizen, so dass gilt $\mathbb{K} \odot yy' = (\mathbb{K})_{ij} \cdot (yy')_{ij}$. Nun können die primalen Variablen w, b und die Slack-Variablen ξ_i wie folgt berechnet werden:

$$w = \sum_{i=1}^{N} \alpha_i y_i \varphi(x_i) \tag{4.29}$$

$$b = \sum_{i=1}^{N} \alpha_i y_i \tag{4.30}$$

$$\xi_i = \frac{\alpha_i}{C}. \tag{4.31}$$

Die gelernte Hyperebene $\rho = y_i(w^T \varphi(x_i) + b) + \frac{\alpha_i}{C}$ kann für jeden beliebigen Stützvektor bestimmt werden. Im Fall der Ein-Klassen-SVM können (bis auf b) die primalen Variablen analog berechnet werden.

Die Äquivalenz des durch die Zwei-Klassen-SVM gegebenen Optimierungsproblems zum Optimierungsproblem der Support Vector Data Description (4.20), ist direkt ersichtlich aus:

$$\begin{aligned} &\max_{\alpha} \ -\alpha^T \tilde{\mathbb{K}} \alpha \\ &\text{u.d.N. } \alpha \geq \mathbf{0}, \alpha^T \mathbf{1} = 1 \end{aligned} \tag{4.32}$$

mit

$$\tilde{\mathbb{K}} = [\tilde{k}(z_i, z_j)] \text{ mit } \tilde{k}(z_i, z_j) = y_i y_j k(x_i, x_j) + y_i y_j + \frac{\delta_{ij}}{C}. \tag{4.33}$$

Auch dieser erweiterte Kern ist konstant für alle Beispiele mit $\tilde{k}(z_i, z_j) = y_i y_j k(x_i, x_j) + y_i y_j + \frac{1}{C} = \tilde{\kappa}$. Die Gültigkeit aller Aussagen aus Abschnitt 4.3.1 bleibt erhalten. Damit ist (4.25) ein zu Formel (4.20) äquivalentes Optimierungsproblem, wobei die lineare Feature-Map φ mit $k(z_i, z_j) = \langle \varphi(z_i), \varphi(z_j) \rangle$ durch die nichtlineare Featuremap $\tilde{\varphi}(z_i) = \begin{bmatrix} y_i \varphi(x_i) \\ y_i \\ \frac{1}{\sqrt{C}} e_i \end{bmatrix}$ ausgetauscht wird. Dabei beschreibt e_i den $i$ten Einheitsvektor. Die Klassenzugehörigkeit wird mit in die Feature-Map integriert.

4.3.4 Algorithmus der Core Vector Machine

Der nun vorgestellte Algorithmus zur Lösung des MEB-Problems basiert auf einem Algorithmus aus [3]. Im Folgenden wird das CoreSet S, das Zentrum des Balls c und der Radius R der $t$ten Iteration als S_t, c_t, R_t bezeichnet. Alle Punkte, die dem CoreSet hinzugefügt werden, heißen Core-Vektoren.

Algorithm 1: Die Core Vector Methode

Eingabe : Eine Menge von Trainingsbeispielen S_{train} und ein $\epsilon > 0$
Ausgabe: Ein Ball B mit Zentrum c_{t+1} und Radius R_{t+1}

Initialisiere $S_0 = \{\tilde{\varphi}(z_0)\}, c_0 = \tilde{\varphi}(z_0), R_0 = 0$
while *Ein Punkt z existiert, so dass $\tilde{\varphi}(z)$ außerhalb des Balls $B(c_t, (1+\epsilon)R_t)$ liegt* **do**
Finde das z, für das gilt, dass $\tilde{\varphi}(z)$ die größte Distanz d zum Ball B hat:

$$\max_z d(c_t, \tilde{\varphi}(z)) \tag{4.34}$$

Setze $S_{t+1} = S_t \cup \{\tilde{\varphi}(z)\}$
Berechne den neuen $MEB(S_{t+1})$ aus Formel (4.20)
Setze $c_{t+1} = c_{MEB(S_{t+1})}$ und $R_{t+1} = r_{MEB(S_{t+1})}$ aus Formel (4.18)
t = t+1
end

Initialisierung

Die Initialisierung startet mit der Festlegung eines beliebigen Startpunkts $z_0 \in S_{train}$. Zwei weiterere Punkte $z_1 = \arg\max_z d(\tilde{\varphi}(z), \tilde{\varphi}(z_0))$ und $z_2 = \operatorname{argmax}_z d(\tilde{\varphi}(z), \tilde{\varphi}(z_1))$ werden gesucht. Das initiale CoreSet $S_0 = \{z_1, z_2\}$ besitzt folgende Eingeschaften:

$$\begin{aligned}
c_0 &= \frac{1}{2}(\tilde{\varphi}(z_1) + \tilde{\varphi}(z_2)) \\
\alpha_1 &= \alpha_2 = \frac{1}{2} \\
\alpha_i &= 0 \; \forall i \neq 1, 2 \\
R_0 &= \frac{1}{2} \left\| \tilde{\varphi}(z_1) - \tilde{\varphi}(z_2) \right\| \\
&= \frac{1}{2} \sqrt{\left\| \tilde{\varphi}(z_1) \right\|^2 + \left\| \tilde{\varphi}(z_2) \right\|^2 - 2\tilde{\varphi}(z_1)^T \tilde{\varphi}(z_2)} \\
&= \frac{1}{2} \sqrt{2\tilde{\kappa} - k(z_1, z_2)}.
\end{aligned}$$

In einem binären Klassifikation müssen die beiden initialen Punkte aus den verschiedenen Klassen stammen. Der initiale Radius R_0 ergibt sich mit der Definition des Kerns $\tilde{\kappa} = -\kappa - 1$ und von $\tilde{\kappa} = \kappa + 1 + \frac{1}{C}$ als:

$$
\begin{aligned}
R_0 &= \frac{1}{2}\sqrt{2\tilde{\kappa} - k(z_1, z_2)} \\
&= \frac{1}{2}\sqrt{2(\kappa + 1 + \frac{1}{C}) - 2(-k(z_1, z_2) - 1)} \\
&= \frac{1}{2}\sqrt{2\left(\kappa + 2 + \frac{1}{C}\right) - 2k(z_1, z_2)}.
\end{aligned} \tag{4.35}
$$

Da κ und C konstant sind, ergibt sich der maximale Radius R_0 zweier Punkte mit dem minimalen Abstand zueinander. Diese Heuristik wird auch für die Initialisierung der DirectSVM (vgl. [29]) und SimpleSVM (vgl. [43]) eingesetzt.

Distanzberechnungen

Der Abstand $\|c_t - \tilde{\varphi}(z)\|$ zwischen dem Zentrum und einem beliebigen Punkt z muss an zwei Stellen des Algorithmus berechnet werden. Zuerst wird festgestellt, ob noch ein Punkt außerhalb des Balls $B(c_t, R_t)$ existiert und im zweiten Schritt wird die Abstandberechnung zum Auffinden des entferntesten Punktes verwendet. Der entfernteste Punkt wird gewählt, da dieser die Optimalitätbedingungen (4.9) - (4.11) am stärksten verletzt. Dieses Vorgehen ist analog zur Sequential Minimal Optimization (vgl. Abschnitt 3.2.3). Es kann auch analytisch gezeigt werden (vgl. [39]), dass diese Wahl des Beipiels den größten Fortschritt bei der Maximierung des dualen Problems mit sich bringt.

Mit Hilfe der Definition des Zentrums (vgl. (4.18)) ergibt sich für den quadrierten Abstand:

$$
\|c_t - \tilde{\varphi}(z)\|^2 = \sum_{z_i, z_j \in S_t} \alpha_i \alpha_j \tilde{k}(z_i, z_j) - 2 \sum_{z_i \in S_t} \alpha_i \tilde{k}(z_i, z) + \tilde{k}(z, z). \tag{4.36}
$$

Damit basiert die Berechnung der Distanz alleine auf Kernberechnungen und nicht auf expliziten Beispielen im Merkmalsraum. Daher kann die Distanz berechnet werden, ohne das die Lage der transformierten Beispiele im Merkmalsraum und das Zentrum des Balls bekannt sind.

Die Berechnung der Distanzen für alle N Trainingspunkte dauert in der $t$ten Iteration $\mathcal{O}\left(|S_t|^2 + N\,|S_t|\right) = \mathcal{O}\left(N\,|S_t|\right)$. Eine effiziente Berechnung aller Abstände für eine große Menge von Beispielen ist somit nicht möglich. Anstatt für alle Datenpunkte den Abstand zu berechnen, wird eine feste Anzahl von zufälligen Datenpunkten zur Abstandberechnung ausgewählt. Dabei soll gelten, dass mindestens ein Punkt aus den 5% der am weitesten entfernten Punkte mit einer Wahrscheinlichkeit von 95% ausgewählt wird. So müssen laut [39] mindestens 59 zufällige Beispiele ausgewählt werden. Folgende Rechnung für binomialverteilte Zufallsvariablen zeigt, dass diese Behauptung gilt:

$$
\begin{aligned}
P(X \geq 1) &= 1 - P(X \leq 0) \\
&= 1 - \sum_{k=0}^{0} \binom{N}{k} 0,05^k \, 0,95^{(N-k)} \stackrel{!}{=} 0,95 \\
&\Leftrightarrow 1 - \binom{N}{0} 0,05^0 \, 0,95^N = 0,95 \\
&\Leftrightarrow 1 - 1 \cdot 1 \cdot 0,95^N = 0,95 \\
&\Leftrightarrow 0,05 = 0.95^N \\
&\Leftrightarrow \log(0,05) = N \log(0,95) \\
&\Leftrightarrow N = \frac{\log(0,05)}{\log(0,95)} \approx 59
\end{aligned}
\tag{4.37}
$$

Wegen $|S_t| \ll N$ wird eine starke Beschleunigung erreicht und die Laufzeit verringert sich auf $\mathcal{O}\left(|S_t|^2 + |S_t|\right) = \mathcal{O}\left(|S_t|^2\right)$.

Auffinden des Minimum Enclosing Balls

Der Minimum Enclosing Ball kann mit einem beliebigen Optimierungsverfahren gefunden werden. In [39] und in der Implementierung im Rahmen dieser Arbeit wird eine auf das Problem (4.20) angepasste Sequential Minimal Optimization verwendet, da dieser für kleine bis mittelgroße Datenmengen effizient eine Lösung berechnen kann (vgl. [25]).

Zeit- und Platzbedarf des Algorithmus

Zuerst soll die Laufzeit und der Platzbedarf des Algorithmus ohne probabilistische Beschleunigung (4.37) analysiert werden. In [3] wurde bewiesen, dass der Algorithmus in maximal $2/\epsilon$ Iterationen konvergiert. Die maximale Anzahl und damit auch die maximale Größe des CoreSets ist also durch $\tau = \mathcal{O}(1/\epsilon)$ nach oben beschränkt. In einigen praktischen Anwendungen hat sich gezeigt, das die tatsächliche Größe des CoreSets viel kleiner ist als diese theoretische Schranke (vgl. [18]).

Mit der Initialisierung werden zwei Punkte, in jeder Iteration wird ein Punkt hinzugefügt. Die Größe des CoreSets Q_t in Iteration t ist also $|Q_t| = t + 2$. Die Initialisierung benötigt eine Laufzeit von $\mathcal{O}(N)$, die Distanzberechnung zwischen Zentrum und einem beliebigen Punkt benötigt eine Laufzeit von $\mathcal{O}\left((t+2)^2 + tN\right) = \mathcal{O}\left(t^2 + tN\right)$ und das Auffinden eines MEB am Ende der Iteration benötigt $\mathcal{O}\left((t+2)^3\right) = \mathcal{O}\left(t^3\right)$. Die Lösung des Optimierungsproblem kann natürlich durch eine andere Wahl als den naiven Lösungalgorithmus beschleunigt werden (vgl. Kapitel 3). Unter der Annahme, dass alle anderen Opterationen konstante Zeit benötigen (insbesondere die Kernberechnung) kann aus den Laufzeiten eine Gesamtlaufzeit für die Iteration t zusammengefasst werden als $\mathcal{O}\left(tN + t^3\right)$. Die Laufzeit der Kernberechnung ist jedoch abhängig von der Dimension-

alität des Ursprungsraums, aber nicht abhängig von der Dimensionalität des Merkmalsraums. Die Gesamtlaufzeit des Algorithmus kann zusammengefasst werden als:

$$T = \sum_{t=1}^{\tau} \mathcal{O}\left(tN + t^3\right) = \mathcal{O}\left(\tau^2 N + \tau^4\right) = \mathcal{O}\left(\frac{N}{\epsilon^2} + \frac{1}{\epsilon^4}\right). \tag{4.38}$$

Dies ist mit festem ϵ linear in N.

Der Platzbedarf des Algorithmus ist nur abhängig von der Größe des CoreSets. Der Platzbedarf für die Speicherung der Beispiele wird ignoriert, da das zu optimierende Problem nur auf Elementen des CoreSets definiert wird. Die Kernmatrix aus Formel (4.36) für Iteration t hat einen Platzbedarf von $\mathcal{O}\left(|S_t|^2\right)$. Die maximale Anzahl von Iterationen ist $\tau = 1/\epsilon$. Daraus folgt, dass der Platzbedarf nach oben durch $\mathcal{O}\left(1/\epsilon^2\right)$ beschränkt ist. Der Platzbedarf ist also unabhängig von der Gesamtgröße der Trainingsmenge für ein festes ϵ.

Wird im Algorithmus die probabilistische Beschleunigung eingesetzt, ist die Laufzeit unabhängig von der Größe der Trainingsmenge. Die Initialisierung benötigt konstante Zeit und die Distanzberechnung benötigt $\mathcal{O}\left((t+2)^2\right) = \mathcal{O}\left(t^2\right)$. Die Laufzeit für die anderen Operationen bleiben unverändert. Iteration t benötigt also eine Gesamtlaufzeit von $\mathcal{O}\left(t^3\right)$. Die Anzahl der Iterationen erhöht sich, da keine Garantie dafür abgegeben werden kann, ob in jeder Iteration der Punkt mit der größten Distanz gefunden wird. In [4] wird aber mit $\mathcal{O}\left(1/\epsilon^2\right)$ eine obere Schranke angegeben. In Relation zu (4.38) verringert sich die Gesamtlaufzeit also auf:

$$T = \sum_{t=1}^{\tau} \mathcal{O}\left(t^3\right) = \mathcal{O}\left(\tau^4\right) = \mathcal{O}\left(\frac{1}{\epsilon^8}\right). \tag{4.39}$$

Für ein festes ϵ ist die Laufzeit damit unabhängig von N.

4.4 Ball Vector Methode (BVM)

Die Core Vector Machine stellt in der praktischen Anwendung im Vergleich zu vielen anderen SVM-Implementierungen eine Verbesserung bzgl. Zeit- und Platzkomplexität dar (vgl. [39]). Allerdings wird noch in jeder Iteration die Lösung eines quadratischen Optimierungsproblems errechnet. Die Größe des CoreSets kann in Abhängigkeit der Größe des Datensatzes oder auch der Komplexität des Datensatzes sehr groß werden (vgl. [37]). Je größer das CoreSet, desto größer wird auch der Aufwand, um das durch die Core Vector Machine gestellte Optimierungsproblem zu lösen.

Anstatt das MEB-Problem mit dem zuvor beschriebenen Algorithmus zu lösen, wird nun das MEB-Problem vereinfacht. Dieses vereinfachte Problem wird dann mit heuristischen Algorithmen gelöst. Es wird nicht mehr der Ball mit minimalem Radius gesucht, sondern nur noch ein Ball mit einem vorgegebenem Radius, der alle Beispiele umschließt. Die Aktualisierung des Zentrums dieses sogenannten Enclosing Balls (EB) kann aufgrund der

vereinfachten Problemstellung in jeder Iteration analytisch gelöst werden. Dadurch kann, anders als bei der Core Vector Machine, auf den Einsatz eines Optimierers verzichtet werden. Somit verringert sich die Laufzeit. Ferner zeigen [37], dass bei richtiger Wahl des Radius die Lösungsgüte vergleichbar mit der der Core Vector Machine ist.

Ein Ball $B(c, R)$ mit Zentrum c und Radius R heißt Enclosing Ball von S $EB(S)$, wenn alle Beispiele aus S innerhalb des Balls liegen. Es gilt:

$$\|c - \varphi(z_i)\|^2 \leq r^2, \ \forall \varphi(z_i) \in S. \tag{4.40}$$

Genau wie die Core Vector Machine (vgl. Abschnitt 4.3) wird eine iterative Strategie verfolgt: In jeder Iteration t wird ein Beispiel z außerhalb des Balls $B(c_t, (1+\epsilon)r_t)$ gesucht. Wird ein Beispiel z außerhalb der $(1+\epsilon)$-Approximation gefunden, wird das Zentrum des Balls c so verschoben, dass das gefundene Beispiel auf der Hülle des Balls liegt.

In den folgenden Abschnitten wird zunächst eine $(1+\epsilon)$-Approximation für das EB-Problem vorgestellt. Ein Algorithmus für das EB-Problem mit verbesserter Laufzeit wird in Abschnitt 4.4.2 beschrieben. Eine Verbesserung der Lösungsgüte wird durch eine Verkleinerung des Radius des Enclosing Balls erreicht (vgl. Abschnitt 4.4.3). Zum Abschluss wird die Ball Vector Machine definiert.

4.4.1 $(1+\epsilon)$-Approximationsalgorithmus für $EB(S, r)$

Der folgende Algorithmus ist aus dem MEB-Algorithmus in [24] abgeleitet. Anstelle in jeder Iteration das Beispiel z mit der größten Distanz zu wählen, genügt es hier ein beliebiges Beispiel außerhalb des Balls zu finden.

Algorithm 2: $(1+\epsilon)$-Approximationsalgorithmus für $EB(S, r)$

Eingabe : Eine Menge von Beispielen S, ein Radius r und ein $\epsilon > 0$
Ausgabe: Der Ball $B(c_t, r)$
Initialisierung: $c_0 = \varphi(z_0), \ S_0 = \{\varphi(z_0)\}$,
$z_0 \in S$ *beliebig*
while *Es existiert ein Beispiel z, so dass* $\varphi(z)$ *außerhalb des Balls* $B(c_t, (1+\epsilon)R_t)$ *liegt* **do**
    Setze $S_{t+1} = S_{t+1} \cup \{\varphi(z)\}$
    Finde ein neues Zentrum c_{t+1} mit minimalem Abstand zu c_t. Das Beispiel $\varphi(z_t)$ muss auf dem Ball B liegen.
    $t = t + 1$
end

Effiziente Aktualisierung der Ball-Approximation

Wird ein Beispiel $\varphi(z)$ außerhalb des Balls B gefunden, muss das Zentrum c_t aktualisiert werden. Die Aktualisierung von c_t erfolgt effizient ohne die Hilfe eines Optimierers. Dazu wird c_t so verschoben, dass $\varphi(z)$ auf dem neuen Ball liegt und das alte und neue Zentrum

c_{t+1} minimalen Abstand besitzen.

Mathematisch lässt sich dieser Zusammenhang wie folgt darstellen:

$$\begin{aligned} &\min_{c} \ \|c - c_t\|^2 \\ &\text{u.d.N. } R^2 \geq \|c - \varphi(z_t)\|^2 \end{aligned} \tag{4.41}$$

Die primale Funktion lautet:

$$L_P = \|c - c_t\|^2 - \lambda(R^2 - \|c - \varphi(z_t)\|^2), \tag{4.42}$$

wobei λ der Lagrange-Multiplikator ist. Mit dem Gradienten $\nabla \|x\|^2 = 2\frac{x}{\|x\|}\|x\| = 2x$ ergibt sich die Ableitung von L_P zu:

$$\begin{aligned} \frac{\partial L_P}{\partial c} \stackrel{!}{=} 0 &\Leftrightarrow 2(c - c_t) + 2\lambda(c - \varphi(z_t)) \stackrel{!}{=} 0 \\ &\Leftrightarrow c = \frac{c_t + \lambda\varphi(z_t)}{(1+\lambda)}. \end{aligned} \tag{4.43}$$

Dies eingesetzt in die notwendige Bedingung für die Optimalität der Lösung (vgl. Theorem 3.1) ergibt:

$$\begin{aligned} R^2 &= \|c - \varphi(x_t)\|^2 \\ \Leftrightarrow R^2 &= \left\| \frac{c_t + \lambda\varphi(z_t)}{(1+\lambda)} - \varphi(z_t) \right\|^2 \\ \Leftrightarrow R &= \left\| \frac{c_t + \lambda\varphi(z_t) - (1+\lambda)\varphi(z_t)}{(1+\lambda)} \right\| \\ \Leftrightarrow R &= \frac{1}{(1+\lambda)} \|c_t - \varphi(z_t)\| \\ \Leftrightarrow \lambda &= \frac{\|c_t - \varphi(z_t)\|}{r} - 1. \end{aligned} \tag{4.44}$$

Für die weiteren Schritte wird

$$\beta_t = \frac{r}{\|c - \varphi(z_t)\|} \ (\geq 0) \tag{4.45}$$

definiert. Das neue Zentrum ist für alle $t > 0$ eine Konvexkombination aus c_t und $\varphi(z)$:

$$\begin{aligned} c_{t+1} &\stackrel{(4.43)}{=} \frac{c_t + \lambda\varphi(z_t)}{(1+\lambda)} \\ &\stackrel{(4.44)}{=} \frac{c_t + \frac{1}{\beta_t}\varphi(z_t) - \varphi(z_t)}{\frac{1}{\beta_t}} = \varphi(z_t) + \beta_t(c_t - \varphi(z_t)). \end{aligned} \tag{4.46}$$

In jeder Iteration $t > 0$ ist c_{t+1} eine Kombination aus dem ersten Zentrum c_0 und allen bisher außerhalb des Balls gefunden Beispielen.

Effiziente Distanzberechnung

Nach der Aktualisierung des Ballzentrums wird überprüft, ob sich noch weitere Beispiele außerhalb des Balls befinden, indem der Abstand zwischen dem Zentrum c_t und allen übrigen Beispielen $\varphi(z)$ berechnet wird:

$$\begin{aligned}\|c_{t+1} - \varphi(z)\|^2 = \beta_t \|c_t\|^2 + (2 - \beta_t) \|\varphi(z_t)\|^2 + R(R - \|c_t - \varphi(z_t)\|) \\ - 2((1 - \beta_t)\varphi(z_t)\varphi(z) + \beta_t c_t \varphi(z))).\end{aligned} \tag{4.47}$$

Wenn $\|c_t\|$ in jeder Iteration gespeichert wird, kann der Abstand in einer Laufzeit von $\mathcal{O}(|S_{t+1}|)$ berechnet werden (vgl. Abschnitt 8.2.3).

Konvergenz

Der Algorithmus 2 konvergiert in $\mathcal{O}\left(1/\epsilon^2\right)$ Iterationen, er hat eine Laufzeit von $\mathcal{O}\left(1/\epsilon^4\right)$ und einen Platzbedarf von $\mathcal{O}\left(1/\epsilon^2\right)$ (vgl. [37]). Die Laufzeit und der Platzbedarf sind mit einen festen ϵ unabhängig von der Dimensionalität des Merkmalsraums und der Größe der Trainingsmenge. Beides ist somit geringer als bei der Core Vector Machine (vgl. Abschnitt 4.3.4).

4.4.2 Schnellerer Multi-Scale $(1+\epsilon)$-Approximationalgorithmus für $EB(S, r)$

In [24] zeigt sich, dass der Algorithmus 2 in $\mathcal{O}(1/\epsilon)$ Iterationen konvergiert, wenn anstatt eines beliebigen Punktes außerhalb der Kugel der Punkt mit der größten Distanz gewählt wird. Die Suche nach diesem Punkt hätte aber eine Laufzeit von $\mathcal{O}(N|S_t|)$ und ist somit rechenintensiv für große Beispielmengen. Um dennoch in weniger Iterationen eine Lösung zu finden, wird die Suchstrategie angepasst. Anstatt ein festes ϵ zu wählen, wird mit einem sehr großen ϵ gestartet. Wird kein Beispiel außerhalb des Balls gefunden, wird das initiale ϵ verringert, bis ein minimales ϵ erreicht ist. Es kann gezeigt werden, dass durch diese Anpassung der Algorithmus in $\mathcal{O}(1/\epsilon)$ iterationen konvergiert (vgl. [37]).

Algorithm 3: Multi-Scale $1 + (\epsilon)$-Approximationsalgorithmus für $EB(S, r)$

Eingabe : Eine Menge von Beispielen S, ein Radius r und M
Ausgabe: Der Ball $B(c_{EB_m}, r)$
Initialisierung: $c_{EB_0} = \varphi(x_0) = random(S)$
for $m = 1$ *to* M **do**
  Setze $\epsilon_m = 2^{-m}$
  Finde eine $(1+\epsilon)$-Approximation von $EB(S, r)$ mit Algorithmus
  $BVM_2(S, r, \epsilon_m, c_{EB_{m-1}})$ mit $c_{EB_{m-1}}$ als Warmstart
end

4.4.3 Verkleinerung des Enclosing Balls

Da der optimale Radius des Balls nicht bekannt ist, wird für die zuvor vorgestellen Algorithmen ein initialer Radius gewählt. Im folgenden Algorithmus wird eine Strategie

implementiert, um den Radius zu verringern. Zunächst wird $h = \sqrt{r^2 - (R^*)^2}$ definiert. Damit kann dass MEB-Problem umgeschrieben werden zu

$$\|c - \varphi(x)\|^2 + (r^2 - (R^*)^2) \leq r^2. \tag{4.48}$$

Dies kann auch als ein $EB(\tilde{S}, r)$-Problem $\|\tilde{c} - \tilde{\varphi}(x_i)\|^2 \leq r^2$ aufgefasst werden, mit $\tilde{S} = \{\tilde{\varphi(x_i)}\} = \{[\varphi(x_i)^T \ 0]^T\}$ und Zentrum $\tilde{c} = [c^T h]^T$. Nach der Initialisierung mit $\tilde{c}_0 = [c_0^T h_0]^T$, wobei $c_0 = \varphi(x)$ ein zufällig gewählter Punkt und $h_0 = r$ ist, können beide zuvor vorgestellten Algorithmen zur Lösung des angepassten Problems verwendet werden. Durch eine iterative Anpassung des Radius kann der Radius noch weiter verkleinert werden, um die Güte der Lösung weiter zu verbessern (vgl. [37]).

Algorithm 4: Verringerung des initialen Radius

Eingabe : Eine Menge von Beispielen S und ein Radius r
Ausgabe: Der Ball $B(c_{EB_m}, r)$
Initialisierung: $c_{EB_0} = \varphi(x_0) = random(S)$

4.4.4 Verbindung zwischen dem Enclosing Ball und der Stützvektormethode

Die Verbindung zwischen der Stützvektormethode und den hier vorgestellten Algorithmen geschieht über die Wahl der verwendeten Kern-Funktion und dem initialen Radius. Wird die gleiche Kern-Funktion, wie bei der Core Vector Machine gewählt, kann der initiale Radius als $\sqrt{\tilde{\kappa}}$ gewählt werden, da $\tilde{\kappa}$ den quadratischen optimalen Radius des minimalen Balls beschränkt. Aus den Optimaliätsbedingungen der Core Vector Machine folgt:

$$(R^*)^2 = \tilde{\kappa} - \sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \leq \tilde{\kappa}. \tag{4.49}$$

Wird das MEB-Problem mittels eines EB-Problems mit initialem Radius $\sqrt{\tilde{\kappa}}$ gelöst, ergibt sich die Ball Vector Machine.

5 Strukturelle Stützvektormethode

Bisher wurde die Stützvektormethode hier nur für Klassifikationsaufgaben verwendet. Eine mögliche Erweiterung auf eine allgemeinere Lernaufgabe bietet die strukturelle Stützvektormethode (vgl. [40]). Die Lernaufgabe der strukturellen Stützvektormethode besteht darin, basierend auf einer gegebenen Trainingsmenge $S_{train} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subset \mathcal{X} \times \mathcal{Y}$ eine Funktion $f : \mathcal{X} \to \mathcal{Y}$ zu lernen. Anders als bei der Klassifikation oder Regression können die Ausgabevariablen von beliebiger Struktur sein, beispielsweise Sequenzen, Zeichenketten, Bäume, Verbände oder Graphen. Durch die je nach Definition sehr große Menge an möglichen Ausgabevariablen ist das sich ergebende Optimierungsproblem (2.28) mit einem üblichen Optimierer nicht effizient lösbar (vgl. Kapitel 3).

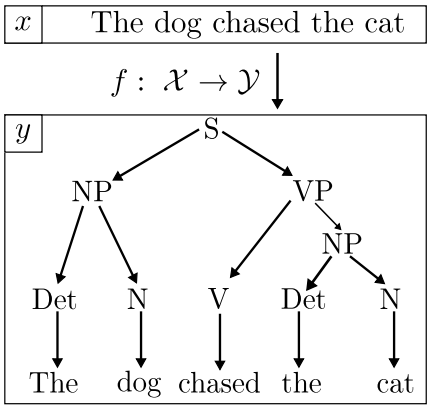
Ähnlich zur Strategie der Core Vector Machine (vgl. Abschnitt 4.1), wird anstatt in jeder Iteration das Optimierungsproblem um eine Beoachtung zu erweitern, wird die Menge der einzuhaltenden Nebenbedingungen in jeder Iteration ausgehend von keiner Nebenbedingung sukzessive vergrößert. Im Folgenden wird gezeigt, dass eine Menge von polynomiell vielen Nebenbedingungen ausreicht, um eine Lösung zu finden, die auf dem Gesamtproblem höchstens um ϵ vom Optimium abweicht.

Die Struktur der Ausgabevariablen wird durch eine für die Beobachtung x und Ausgabevariable y gemeinsame Feature-Map $\Psi(x, y)$ in die Formulierung des Optimierungsproblems aufgenommen. Ein Beispiel dafür ist in Abb. 5.1 gegeben. Der Algorithmus soll für den Satz x den Syntaxbaum y anhand der Funktion $f(x)$ korrekt vorraussagen. Um den Zusammenhang zwischen x und y als Vektor in die Formulierung der Stützvektormethode zu integrieren, wird in dieser Anwendung der die Feature-Map $\Psi(x, y)$ als Histogrammvektor gebildet. Dieser zählt die Anwendung der Regeln. Anstatt auf den Beispielen und den Ausgabevariablen separat zu lernen, wird auf einer gemeinsamen Darstellung beider gelernt.

Im Folgenden werden die Modellierung des Optimierungsproblems erläutert, verallgemeinerte Verlustfunktionen vorgestellt und wie in Kapitel 2 verschiedene Formulierungen der Stützvektormethode hergeleitet.

5.1 Modellierung

Anstatt eine direkte Funktion zwischen den Beobachtungen und den Ausgabevariablen zu lernen, wird bei der strukturellen SVM eine Diskriminanzfunktion $F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R},\ x \mapsto F(x, y; w)$ gelernt. Die Funktion F ist ein Maß für die Kompatibilität zwischen



$$\Psi(x,y) = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \\ \vdots \\ 0 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{matrix} \text{S} \to \text{NP VP} \\ \text{S} \to \text{NP} \\ \text{NP} \to \text{Det N} \\ \text{VP} \to \text{V NP} \\ \\ \text{Det} \to \textit{dog} \\ \text{Det} \to \textit{the} \\ \text{N} \to \textit{dog} \\ \text{V} \to \textit{chased} \\ \text{N} \to \textit{cat} \end{matrix}$$

Abbildung 5.1: Darstellung eines Syntaxbaums für natürliche Sprache.

x und y. Durch die Maximierung von F wird das y vorrausgesagt, das am besten zu x passt:

$$f(x;w) = \arg\max_{y \in \mathcal{Y}} F(x,y;w) \tag{5.1}$$

Das optimale w^* muss so gefunden werden, dass $f(x;w)$ das am besten zu x passende $y \in \mathcal{Y}$ liefert. Es wird angenommen, dass die Funktion F linear in der gemeinsamen Feature-Map $\Psi(x,y)$ ist:

$$F = \langle w, \Psi(x,y) \rangle. \tag{5.2}$$

5.1.1 Verallgemeinerte Verlustfunktionen

Wie in Abschnitt 2.1 wird eine Verlusfunktion zwischen zwei Ausgabevariablen y und y' als $\Delta(y,y')$ definiert. Mit dieser Verlustfunktion kann das empirische Risiko für eine unabhängige Testmenge bestimmt werden. Die einfache 0-1 Verlustfunktion, wie im Fall der Klassifikation, ist im Allgemeinen nicht für strukturierte Ausgaben geeignet.
Beispielsweise können Syntaxbäume, obwohl sie sich in einigen Knoten unterscheiden, eine ähnliche Ausage haben. Diese Ähnlichkeit zwischen Bäumen kann nicht durch eine binäre Verlustfunktion ausgedrückt werden. Für Syntaxbäume wird üblicherweise der F_1-Score verwendet, der das harmonische Mittel von Precision und Recall darstellt (vgl. [14]).

Um den verschiedenen Verlustfunktionen einen Einfluss auf das Optimierungsproblem gegeben zu können, müssen diese darin integriert werden.

5.1.2 Strukturelle SVM mit harten Nebenbedingungen

Analog zu in Kapitel 2 wird zunächst der Fall betrachtet, dass eine Funktion f mit empirischem Risiko $\mathcal{R}^{\Delta}_{emp}(f(\cdot;w)) = 0$ gelernt werden kann. Die Nebenbedingungen der linearen SVM (2.9) sind so modelliert, dass alle Beobachtung der korrekten Klasse zugeordnet werden. Die Nebenbedingungen der strukturellen SVM werden ähnlich erstellt.

Wenn die Funktion f allen Beobachtungen $x_i \in S_{train}$ ihre korrekte Ausgabevariable y_i zuordnet, nimmt die Funktion $F(x, y; w)$ an der Stelle (x_i, y_i) den maximalen Wert an:

$$\max_{y \in \mathcal{Y} \setminus y_i} \{\langle w, \Psi(x_i, y)\rangle\} \leq \langle w, \Psi(x_i, y_i)\rangle, \ \forall i = 1, \ldots, N. \tag{5.3}$$

Um ein konvexes Optimierungsproblem im Sinne von (3.4) modellieren zu können, müssen die nicht linearen Nebenbedingungen (5.3) in lineare umgeformt werden. Da die Ungleichung (5.3) für alle $y \in \mathcal{Y}$ gilt, kann auch für jedes y eine einzelne Nebenbedingung formuliert werden. Insgesamt ergeben sich $N\,|\mathcal{Y}| - N$ lineare Nebenbedingungen der Form:

$$\langle w, \delta\Psi_i(y)\rangle \geq 0, \ \forall i = 1, \ldots, N, \ \forall y \in \mathcal{Y} \setminus y_i, \tag{5.4}$$

mit $\delta\Psi_i(y) = \Psi(x_i, y_i) - \Psi(x_i, y)$. Die Schnittmenge der linearen Bedingungen stellt eine konvexe Menge dar (vgl. Kapitel 3), somit ist die erste Vorraussetzung für ein konvexes Optimierungsproblem erfüllt.

Wenn die Bedingungen aus (5.4) erfüllbar sind, existiert üblicherweise mehr als eine Lösung für y (vgl. [40]). Um eine eindeutige Lösung definieren zu können, wird die Zielfunktion des Optimierungsproblems verändert. Es wird nicht mehr der Abstand zwischen der Hyperebene und den nächsten Punkten beider Klassen maximiert (vgl. Abschnitt 2.2), sondern der minimale Abstand zwischen dem Funktionswert von $F(x_i, y_i; w)$ für das korrekte y_i und das zweitbeste $\hat{y} \neq y_i$ maximiert. Genau wie in Abschnitt 2.2 kann für das Maximierungsproblem ein äquivalentes Minimierungsproblem formuliert werden:

$$\begin{aligned} \mathbf{SVM}_{S_0} : \min_{w} \ & \frac{1}{2} \|w\|^2 \\ \text{u.d.N. } & \forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y)\rangle \geq 1. \end{aligned} \tag{5.5}$$

Die Lösung des Maximierungsproblems in (5.1), um für eine Beobachtung x die Ausgabe y mit dem gelernten w vorherzusagen, hängt vom jeweiligen Problem ab. Für eine Mehr-Klassen-Klassifikation kann beispielsweise angenommen werden, dass die Anzahl der möglichen Klassen gering genug ist, um den Funktionswert von $F(x, y; w)$ für alle Klassen zu berechnen.

5.1.3 Strukturelle SVM mit weichen Nebenbedingungen

Sollen auch Punkte innerhalb der Margin zugelassen werden, müssen die Nebenbedingungen relaxiert werden. Analog zu Kapitel 2 wird für jede lineare Nebenbedingung eine Slack-Variable ξ_i eingeführt (vgl. Abschnitt 2.3):

$$\begin{aligned} \mathbf{SVM}_{S_1} : \min_{w, \xi_i} \ & \frac{1}{2} \|w\|^2 \frac{C}{n} \sum_{i=1}^{N} \xi_i \\ \text{u.d.N. } & \forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y)\rangle \geq 1 - \xi_i, \ \xi_i \geq 0. \end{aligned} \tag{5.6}$$

Wie bei der Core Vector Machine (vgl. Abschnitt 4.3.3) kann die Benutzung einer Slack-Variablen quadratisch bestraft werden, so dass die Bedingung $\xi_i \geq 0$ vernachlässigt werden kann:

$$\begin{aligned} \mathbf{SVM}_{S_2} : \min_{w,\xi_i} \frac{1}{2} \|w\|^2 \frac{C}{2n} \sum_{i=1}^{N} \xi_i^2 \\ \text{u.d.N. } \forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y) \rangle \geq 1 - \xi_i. \end{aligned} \tag{5.7}$$

In beiden Fällen kontrolliert die Konstante $C > 0$ den Trade-off zwischen Trainingsfehler und Margin-Maximierung (vgl. Abschnitt 2.3).

5.1.4 Strukturelle SVM mit verallgemeinerten Verlustfunktionen

Wie in Abschnitt 5.1.1 erwähnt, ist je nach Anwendung die Benutzung verschiedener Verlustfunktionen sinnvoll. Wird die Verlustfunktion in die Nebenbedingungen integriert, kann der Ausgleich bei einer Verletzung einer Nebenbedingung durch eine Slack-Variable bei einem höheren Verlust stärker bestraft werden. Es gibt zwei Möglichkeiten die Verlustfunktion zu integrieren: entweder werden die Slack-Variablen oder die Margin skaliert.

Im Fall der Skalierung der Slackvariablen wird durch ein hohes Risiko $\Delta(y_i, y) > 1$ der Fehler, der durch die Slack-Variable ausgeglichen werden kann, verringert. Um den gleichen Fehler, bei einem größeren Risiko auszugleichen, muss der Wert der Slack-Variablen größer sein. Damit wird die Korrektur durch die Slack-Variable stärker bestraft. Ist das Risiko $\Delta(y_i, y)$ hingegen kleiner als 1, kann durch die Slack-Variable ein größerer Fehler ausgeglichen werden. Wird die $\mathbf{SVM}_{S_1}$ um eine Skalierung der Slack-Variablen erweitert, ergibt sich:

$$\begin{aligned} \mathbf{SVM}_{S_1}^{\Delta s} : \min_{w,\xi_i} \frac{1}{2} \|w\|^2 \frac{C}{n} \sum_{i=1}^{N} \xi_i \\ \text{u.d.N. } \forall i, \forall y \in \mathcal{Y} \setminus y_i : \langle w, \delta\Psi_i(y) \rangle \geq 1 - \frac{\xi_i}{\Delta(y_i, y)},\ \xi_i \geq 0. \end{aligned} \tag{5.8}$$

Die Erweiterung der $\mathbf{SVM}_{S_2}^{\Delta s}$ erfolgt analog.

Anstelle der Skalierung der Slack-Variablen, kann die Verlustfunktion auch die Größe der Margin in der jeweiligen Nebenbedingung skalieren. Muss die Verletzung einer Nebenbedingung durch eine Slack-Variable ausgeglichen werden, steigt der Wert der Slack-Variablen mit der Höhe des Werts der Verlustfunktion. Damit ergibt sich die folgende Formulierung:

$$\begin{aligned} \mathbf{SVM}_{S_1}^{\Delta m} \min_{w,\xi} \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{N} \xi_i \\ \text{u.d.N. } \forall i, \forall y \in \mathcal{Y} : \langle w, \delta\Psi_i(y) \rangle \geq \Delta(y_i, y) - \xi_i. \end{aligned} \tag{5.9}$$

Für die SVM mit quadratischer Bestrafung kann die Formulierung analog angepasst werden und es ergibt sich die $\mathbf{SVM}_{S_1}^{\Delta m}$. Auch für die Skalierung der Margin kann gezeigt werden, dass im Fall der 0-1-Verlustfunktion das empirische Risiko nach oben beschränkt ist (vgl. (2.4)). Werden beliebige Verlustfunktion benutzt, kann im Allgemeinen aber keine obere Schranke des empirischen Risikos angegeben werden. Durch die Integration der Verlustfunktion in die Nebenbedingungen kann das empirische Risiko jedoch nach oben beschränkt werden, wie das folgende Theorem bestätigt.

Theorem 5.1
Sei durch $\xi^*(w)$ die optimale Lösung der Slack-Variablen in $\boldsymbol{SVM}_{S_1}^{\Delta s}$ für gegebenen Gewichtsvektor w gegeben. Dann ist $\frac{1}{n}\sum_{i=1}^{n}\xi_i^*$ eine obere Schranke des empirischen Risikos $\mathcal{R}_{emp}^{\Delta}(w)$.

Beweis. Sei $\xi_i^* = \max\left\{0, \max_{y\neq y_i}\{\Delta(y_i,y)(1-\langle w, \delta\Psi_i(y)\rangle)\}\right\}$.
1. Fall: Die gelernte Funktion ergibt die korrekte Vorraussage mit $f(x_i;w) = y_i$. Damit ist der Wert der Verlustfunktion $\Delta(y_i, f(x_i;w)) = 0$ und durch die Nebenbedingung $\xi_i^* \geq 0$ trivialerweise nach oben beschränkt.
2. Fall: Ist die Vorraussage nicht korrekt $\hat{y} \equiv f(x_i;w) \neq y_i$, dann gilt $\langle w, \delta\Psi_i(\hat{y})\rangle \leq 0$, da der Funktionswert der falschen Ausgabe $\hat{y}$ größer ist als die tatsächliche Ausgabe y_i (vgl. (5.1)). Da der Wert kleiner als 0 ist, muss mindestens eine Abweichung größer als 1 ausgeglichen werden. Damit gilt $\frac{\xi_i^*}{\Delta(y_i,y)} \geq 1 \Leftrightarrow \xi_i^* \geq \Delta(y_i,y)$. □

Für die Skalierung der Margin gilt diese Aussage analog (vgl. [40]).

Ein Vorteil der Skalierung der Slack-Variablen ist, das diese Formulierung invariant gegenüber Skalierungen der Verlustfunktion ist. Wird die Verlustfunktion mit einem multiplikativem Faktor η skaliert, also $\Delta' \equiv \Delta\eta$, dann ergibt eine SVM mit einem skalierten $C' = C/\eta$ dieselbe Lösung (vgl. [40]). Im Gegensatz dazu muss für eine Skalierung der Verlustfunktion im Fall der Margin-Skalierung auch die Feature-Map angepasst werden.

5.1.5 Duales Problem

Der im nächsten Kapitel erläutere Algorithmus zur Lösung des durch die strukturelle SVM gegebenen Optimierungsproblems basiert auf dem dualen Problem der jeweiligen **SVM**. Diese werden nun hergeleitet.
Das duale Problem der $\mathbf{SVM}_{S_0}$ (5.5) ist definiert als:

$$\begin{aligned}
&\alpha^* = \arg\max_{\alpha} \Theta(\alpha) \\
&\Theta(\alpha) \equiv -\frac{1}{2}\sum_{i,y\neq y_i}\sum_{j,\bar{y}\neq y_j}\alpha_{(iy)}\alpha_{(j\bar{y})}J_{(iy)(j\bar{y})} + \sum_{i,y\neq y_i}\alpha_{(iy)} \qquad (5.10)\\
&\text{u.d.N. } \alpha \geq 0
\end{aligned}$$

mit $J_{(iy)(j\bar{y})} = \langle \delta\Psi_i(y), \delta\Psi_j(\bar{y})\rangle$. Die Größe des Optimierungsproblems wird bei genauerer Betrachtung der Kernmatrix J deutlich. Um die gesamte Kernmatrix zu berechnen,

muss für jede Kombination von i und j das Skalarprodukt $\langle \delta\Psi_i(y), \delta\Psi_j(\bar{y})\rangle$ aller Kombinationen von $y \neq y_i$ und $\bar{y} \neq y_j$ bestimmt werden. Jeder Eintrag der Kernmatrix J (Abb. 5.2) ist eine $|\mathcal{Y}| - 1 \times |\mathcal{Y}| - 1$ Matrix. Die Kernmatrix J ist also ein ein Tensor.

J	x_1	...	x_j	...	x_N
x_1	–	...	...	...	...
...	...	...	...	...	...
x_i	...	...	$\langle \delta\Psi_i(y), \delta\Psi_j(\bar{y})\rangle$	...	...
...	...	...	...	...	...
x_N	...	...	...	...	–

Abbildung 5.2: Kernmatrix des dualen Problems der strukturellen Stützvektormethode.

Jedes $\alpha_{(iy)}$ hat für jedes $y \neq y_i$ einen Eintrag: $\alpha_{(iy)} = \begin{pmatrix} \alpha_{(y_i \neq y_1)} \\ \vdots \\ \alpha_{(y_i \neq y_{|\mathcal{Y}|})}. \end{pmatrix}$

Zur Herleitung der dualen Funktion wird zunächst die primale Funktion gebildet:

$$L_P = \Theta_p = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{|\mathcal{Y}|} \sum_{y \neq y_i} \alpha_{(iy)} (\langle w, \delta\Psi_i(y)\rangle - 1). \tag{5.11}$$

Dann wird w in Θ_p durch die Ableitung der primalen Funktion ersetzt:

$$w^*(\alpha) = \sum_{i=1}^{N} \sum_{y \neq y_i} \alpha_{(iy)} \delta\Psi_i(y). \tag{5.12}$$

Die duale Formulierung der SVM_1 ist mit (5.11) mit einer zusätzlichen Nebenbedingung gegeben:

$$\sum_{y \neq y_i} \alpha_{(iy)} \leq \frac{C}{N}, \ \forall i = 1, \ldots, N. \tag{5.13}$$

Das duale Problem der SVM_2 ist mit (5.11) und einer Anpassung der Kern-Funktion gegeben:

$$J_{(iy)(j\bar{y})} = \langle \delta\Psi_i(y), \delta\Psi_j(\bar{y})\rangle + \delta(i,j)\frac{N}{C}. \tag{5.14}$$

wobei $\delta(i,j)$ das Kronecker-Delta ist. Das duale Problem der $\text{SVM}_1^{\Delta s}$ ist mit (5.11) und einer Nebenbedingungen gegeben:

$$\sum_{y \neq y_i} \frac{\alpha_{(iy)}}{\Delta(y_i, y)} \leq \frac{C}{n}, \forall i = 1, \ldots, N. \tag{5.15}$$

Das duale Problem der $\text{SVM}_2^{\Delta s}$ ist mit (5.11) und einer Nebenbedingungen gegeben:

$$J_{(iy)(j\bar{y})} = \langle \delta\Psi_i(y), \delta\Psi_j(\bar{y})\rangle + \delta(i,j)\frac{N}{C\sqrt{\Delta(y_i, y)}\sqrt{\Delta(y_j, \bar{y})}}. \tag{5.16}$$

Das duale Problem der $\text{SVM}_1^{\Delta m}$ ist mit (5.11) und einer veränderten Zielfunktion gegeben:

$$\Theta(\alpha) \equiv -\frac{1}{2} \sum_{i,y \neq y_i} \sum_{j,\bar{y} \neq y_j} \alpha_{(iy)} \alpha_{(j\bar{y})} J_{(iy)(j\bar{y})} + \sum_{i,y \neq y_i} \alpha_{(iy)} \Delta(y_i, y). \tag{5.17}$$

Das duale Problem der $\text{SVM}_2^{\Delta m}$ ist mit (5.11) und einer veränderten Zielfunktion gegeben:

$$\Theta(\alpha) \equiv -\frac{1}{2} \sum_{i,y \neq y_i} \sum_{j,\bar{y} \neq y_j} \alpha_{(iy)} \alpha_{(j\bar{y})} J_{(iy)(j\bar{y})} + \sum_{i,y \neq y_i} \alpha_{(iy)} \sqrt{\Delta(y_i, y)}. \tag{5.18}$$

5.2 Algorithmus

Der im Folgenden vorgestellte Algorithmus 5 der strukturellen SVM wählt in jeder Iteration die Nebenbediungung aus, die am stärksten verletzt ist. Ähnlich zur Core Vector Machine wird das sich in jeder Iteration vergrößernde Subproblem gelöst. Aufgrund der vielen Nebenbedingungen sucht dieses Vorgehen, eine kleine Teilmenge von Nebenbedingungen des Gesamtproblems, auf dem das gegebene Optimierungsproblem mit einer Abweichung ϵ vom Optimum gelöst werden kann. Die Auswahl der Nebenbedingungen basiert auf einem sogenannten Schnittebenenverfahren (vgl. Abschnitt 6.2).

Die $|\mathcal{Y}| - 1$ linearen Nebenbedingungen für jedes Beispiel $\Psi(x, y)$ definieren eine konvexe Menge von Punkten (vgl. Kapitel 3). Der Schnitt aller Nebenbedingungen definiert den zulässigen Bereich, in dem eine optimale Lösung gefunden werden kann. Aufgrund der hohen Dimensionalität des zulässigen Bereichs wird versucht, eine optimale Lösung auf kleineren Subräumen zu berechnen. Der Algorithmus der strukturellen SVM vergrößert ausgehend von der leeren Menge, in jeder Iteration die Dimensionalität des Subraums. Die duale Funktion wird also nicht im Raum mit voller Dimension optimiert, sondern nur eine Projektion der dualen Funktion auf dem Subraum. Der betrachtete Subraum ist von den in der Projektion verwendeten α_i abhängig. Die Dimension des Subraums wird erhöht, indem eine Nebenbedingung mit zugehörigem α_j hinzugefügt wird. Der Algorithmus wählt die Nebenbedingung, die durch die aktuelle Lösung am stärksten verletzt wird, indem eine Schnittebene erstellt wird (vgl. Kapitel 6). Der Normalenvektor dieser Schnittebene wird mit der Ableitung der aktuellen Projektion bestimmt:

$$w \equiv \sum_{j=1}^{N} \sum_{y' \in S_j} \alpha_{(iy')} \delta\Psi_i(y'). \tag{5.19}$$

Die Verletzung wird mit Hilfe einer Kostenfunktion bestimmt. Die Kostenfunktion für die $\mathbf{SVM}_{S_1}^{\Delta s}$ (5.8) ergibt sich beispielsweise als:

$$\langle w, \delta\Psi_i(y) \rangle \geq 1 - \frac{\xi_i}{\Delta(y_i, y)} \Leftrightarrow (1 - \langle w, \delta\Psi_i(y) \rangle) \Delta(y_i, y) \leq \xi_i. \tag{5.20}$$

Für die Berechnung der Kosten wird also der Winkel zwischen dem Normalenvektor w und dem Vektor $\delta\Psi_i(y)$ bestimmt. Der Algorithmus wählt in jeder Iteration das y, das

den geringsten Winkel zur Hypereebene und damit die höchsten Kosten besitzt, da diese Verletzung durch die ξ_i ausgeglichen werden muss. Durch die Hinzunahme von y und der entsprechenden Nebenbedingung wird die Dimension des Subproblems erhöht. Die Nebenbedingung beschneidet die gewählte Dimension. Damit wird ein neuer vergrößerter Bereich gefunden, der die optimale Lösung enthalten kann.

Die Anzahl der Klassen kann sehr groß sein, so dass nicht alle Klassen auf die Verletzung der Nebenbedingungen überprüft werden können. Welche Klassen überprüft werden, muss mit einem auf das jeweilige Problem angepassten Optimierungsverfahren bestimmt werden. Für die Mehr-Klassen-Klassifikation kann dennoch angenommen werden, dass die Menge der Klassen klein genug ist, so dass zur Bestimmung der maximal verletzen Nebenbedingung über sie iteriert werden kann. Für den Fall der 0-1-Verlustfunktion muss das inkorrekte y mit maximalem Wert gefunden werden:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \{1 - \langle w, \delta\Psi_i(y)\rangle\}. \tag{5.21}$$

Des Weitern wird die zweitbeste Lösung benötigt, um Margin-Verletzungen in den Fällen der korrekten Vorhersage mit $\hat{y} = y_i$ und $\langle w, \delta\Psi_i(\tilde{y})\rangle < 1$ aufzufinden. Das bedeutet, dass für Probleme, für die die zwei besten Lösungen für das Ausgangsproblem $f(x; w) = F(x, y; w)$ bestimmt werden können, die 0-1-Verlustfunktion angewendet werden kann. In jeder Iteration optimiert der Algorithmus die Projektion der dualen Funktion über den bisher gefundenen möglichen Bereich. Der Algorithmus stoppt, wenn die gefundene Lösung keine weitere Nebenbedingung mehr als ϵ verletzt.

5.2.1 Korrektheit und Komplexität des Algorithmus

Der Beweis für die Korrektheit und die Komplexität des Algorithmus basiert auf einer unteren Schranke der Verbesserung des Funktionswert des dualen Problems für jede Iteration durch Hinzunahme einer Nebenbedingung. Im Fall der quadratischen Bestrafung ($\mathbf{SVM}^*_{S_2}$) kann die Verbesserung anhand der Hinzunahme eines $\alpha_{i\hat{y}}$ ausgedrückt werden, da es keine zusätzliche Beschränkung der α_i gibt (vgl. (5.13)). Im Fall der linearen Bestrafung ($\mathbf{SVM}^*_{S_1}$) muss die Menge aller α_i gleichzeitig betrachtet werden. Die Verbesserung kann daher nicht für ein α_i alleine ausgedrückt werden, jedoch kann die Verbesserung in eine Richtung η ausgedrückt und nach unten beschränkt werden. Damit kann die Korrektheit des Algorithmus gezeigt werden (vgl. [40]).

Durch das folgende Theorem wird eine obere Schranke für die Anzahl der ausgewählten Nebenbedingungen und damit für die Laufzeit gegeben.

Theorem 5.2
Mit $\bar{R} = \max_i R_i \; (= \max_y \|\delta\Psi_i(y)\|)$, $\bar{\Delta} = \max_i \Delta_i \; (= \max_y \{\Delta(y_i, y)\})$ *und für ein gegebenes* $\epsilon > 0$ *terminiert der Algorithmus nach dem inkrementellen Hinzufügen von*

$$max\left\{\frac{2n\bar{\Delta}}{\epsilon}, \frac{8C\bar{\Delta}^3\bar{R}^2}{\epsilon^2}\right\}, max\left\{\frac{2n\bar{\Delta}}{\epsilon}, \frac{8C\bar{\Delta}\bar{R}^2}{\epsilon^2}\right\},$$

$$\frac{C\bar{\Delta}^2\bar{R}^2 + n\bar{\Delta}}{\epsilon^2} \text{ bzw. } \frac{C\bar{\Delta}\bar{R}^2 + n\bar{\Delta}}{\epsilon^2}$$

Nebenbedingungen zu S für die jeweilienge Stützvektormethoden $SVM_1^{\Delta s}$, $SVM_1^{\Delta m}$, $SVM_2^{\Delta s}$ bzw. $SVM_2^{\Delta m}$.

Algorithm 5: Die Core Vector Methode

Eine Menge von Trainingsbeispielen $(x_1, y_1), \ldots, (x_N, y_N)$, C und ein $\epsilon > 0$

Initialisiere $S_i = \emptyset$, $\forall i = 1, \ldots, N$.

repeat

for $i = 1, \ldots, N$ **do**

`/* Vorbereitung der Kostenfunktion (vgl. (5.20)) */`

$$H(y) \equiv \begin{cases} 1 - \langle w, \delta\Psi_i(y)\rangle & (\mathbf{SVM}_{S_0}) \\ (1 - \langle w, \delta\Psi_i(y)\rangle)\Delta(y_i, y) & (\mathbf{SVM}_{S_1}^{\Delta s}) \\ \Delta(y_i, y) - \langle w, \delta\Psi_i(y)\rangle & (\mathbf{SVM}_{S_1}^{\Delta m}) \\ (1 - \langle w, \delta\Psi_i(y)\rangle)\sqrt{\Delta(y_i, y)} & (\mathbf{SVM}_{S_2}^{\Delta s}) \\ \sqrt{\Delta(y_i, y)} - \langle w, \delta\Psi_i(y)\rangle & (\mathbf{SVM}_{S_2}^{\Delta m}) \end{cases}$$

mit $w \equiv \sum_{j=1}^{N} \sum_{y' \in S_j} \alpha_{(iy')} \delta\Psi_i(y')$

`/* Finde eine Schnittebene */`

Berechne $\hat{y} = \arg\max_{y \in \mathcal{Y}} H(y)$

`/* Berechne den aktuellen Wert der Slack-Variable */`

Berechne $\xi_i = \max\{0, \max_{y \in S_i} H(y)\}$

if $H(\hat{y}) > \xi_i + \epsilon$ **then**

$S_i = S_i \cup \{\hat{y}\}$

`/* Vollständige Optimierung */`

$\alpha_S \leftarrow$ Optimiere das duale Problem der gewählten **SVM** über $S = \bigcup_{i=1}^{N} S_i$

`/* oder Optimierung über Subproblem nur mit` S_i `*/`

$\alpha_{S_i} \leftarrow$ Optimiere das duale Problem der gewählten **SVM** über S_i

end

end

until *Kein S_i hat sich in der Iteration geändert.*

6 Auswahl der Nebenbedingungen der strukturellen SVM als Optimierungsproblem

Die Optimierung der Lagrange-Multiplikatoren im Algorithmus der strukturellen SVM ist abhängig von der Problemstellung. Ein weiteres Optimierungsproblem im Algorithmus ist die Auswahl der Nebenbedingungen. Die Auswahl geschieht mittels eines sogenannten Schnittebenenverfahrens (Cutting Plane Algorithm) (vgl. [16]). Als Erweiterung der strukturellen Stützvektormethode existieren in jüngster Zeit Verfahren, die mittels eines Schnittebenenverfahrens die primale Stützvektormethode lösen (vgl. [13, 10]). Joachims [13] zeigt, dass mit Verwendung eines Schnittebenenverfahrens eine lineare Stützvektormethode für Klassifikationsprobleme in linearer Laufzeit trainierbar ist. Werden jedoch Kern-Funktionen verwendet, dann verlangsamt sich der Algorithmus wieder um den Faktor N, also um die Anzahl der Beobachtungen. Somit ist die Verwendung des Schnittebenenverfahrens in der Core Vector Machine nicht erfolgsversprechend bezüglich einer Laufzeitreduktion. Im Folgenden wird die Grundidee der Schnittebenenverfahren erläutert.

6.1 Auffinden von Schnittebenen

Das durch die strukturelle SVM gegebene Optimierungsproblem und das Auswahlproblem ist konvex, daher werden im Folgenden nur Schnittebenenverfahren für konvexe Optimierungsprobleme beschrieben. Für die Erweiterung auf allgemeine Probleme siehe etwa [9].

Im Allgemeinen versucht ein Schnittebenenverfahren, einen Punkt in einer konvexen Menge $K \subseteq \mathbb{R}^d$ zu finden. Bei einem Optimierungsproblem wird ein Punkt aus der Menge der optimalen Punkte gesucht. Die Annahme ist, dass keine Beschreibung der Menge K bekannt ist, sondern nur Informationen an einzelnen Punkten, wobei der Punkt x entweder in der Menge liegt, oder eine trennende Hyperebene zwischen K und x konstruiert werden kann:

$$\begin{aligned} a^T z \leq b,\ \forall z \in K \\ a^T x \geq b. \end{aligned} \tag{6.1}$$

Dabei ist $a \neq 0$ der Normalenvektor der Hyperebene und b ein beliebiger Wert. Werden Punkte auf einer Menge $\mathcal{P}$ gesucht, dann entfernt diese Hyperebene oder auch Schnittebene den Halbraum $\{z|a^T z > b\}$ aus dem Raum $\mathcal{P}$, $K \subseteq \mathcal{P}$. Nur Punkte die der Ebenengleichung genügen, werden weiter in der Suche betrachtet. Ein Schnittebenenverfahren untersucht solange Punkte $x \in \mathcal{P}$, bis entweder ein Punkt aus K gefunden wird,

oder die Menge der möglichen Punkte $\mathcal{P}$ leer ist. Die Güte des Schnittebenenverfahrens hängt also davon ab, in welcher Reihenfolge die Punkte untersucht werden.

Zunächst werden unbeschränkte konvexe Optimierungsprobleme der Form

$$\min f(x) \tag{6.2}$$

betrachtet, wobei $f(x)$ konvex und differenzierbar ist. Es wird die konvexe Menge der optimalen Punkte gesucht. Die Konstruktion einer Schnittebene am Punkt x mit $\nabla f(x) \neq 0$ folgt direkt aus der Konvexität von $f(x)$ (vgl. (6.3)):

$$f(x_1) \geq f(x_0) + (x_1 - x_0)^T \nabla f(x_0). \tag{6.3}$$

Für einen Punkt z, der $(z-x)^T \nabla f(x)$ erfüllt, gilt $f(z) > f(x)$, so dass der Punkt z nicht optimal sein kann.

Die Auswahl einer Menge von zulässigen Punkten ist ein weiteres Problem:

$$\begin{aligned} &\text{Finde } x \\ \text{u.d.N. } &c_i(x) \leq 0, \ \forall i = 1, \ldots, m, \end{aligned} \tag{6.4}$$

wobei die $c_i(x)$ konvex und differenzierbar sind. Angenommen der Punkt x ist nicht zulässig, so existiert also mindestens eine Nebenbedingung $c_j(x), \ j \in \{1, \ldots, m\}$, die verletzt ist. Aus der Ungleichung

$$c_j(z) \geq c_j(x) + \nabla c_j(x)^T (z - x) \tag{6.5}$$

kann geschlossen werden, dass $c_j(z) > 0$, wenn gilt

$$c_j(x) + \nabla c_j(x)^T (z - x) \geq 0. \tag{6.6}$$

Der Punkt z verletzt also auch die j-te Nebenbedingung. Daraus folgt direkt die Definiton der Schnittebene. Jeder zulässige Punkt muss folgende Ungleichung erfüllen:

$$c_j(x) + \nabla c_j(x)^T (z - x) \leq 0. \tag{6.7}$$

Werden die beiden vorgestellten Ansätze kombiniert, kann für ein Optimierungsproblem in Standardform eine Schnittebene konstruiert werden. Das Optimierungsproblem ist gegeben als:

$$\begin{aligned} &\min f(x) \\ \text{u.d.N. } &c_i(x) \leq 0, \ \forall i = 1, \ldots m, \end{aligned} \tag{6.8}$$

wobei $f(x)$ und alle Nebenbedingungen konvex und differenzierbar sind. Wird ein Punkt x ausgewählt, der die Nebenbedingung j verletzt und damit nicht zulässig ist, wird eine Schnittebene durch

$$c_j(x) + \nabla c_j(x)^T (z - x) \leq 0 \tag{6.9}$$

definiert, da jeder optimale Punkt diese Nebenbedingung erfüllen muss. Ist ein Punkt x hingegen zulässig und für die Ableitung gilt $\nabla f(x) = 0$, so ist x optimal. Ist x zulässig, aber nicht optimal, kann mit Hilfe der Zielfunktion eine weitere Schnittebene mit

$$\nabla f(x)^T(z - x) \leq 0. \tag{6.10}$$

konstuiert werden, da alle Punkte mit größerem Funktionswert nicht optimal sein können.

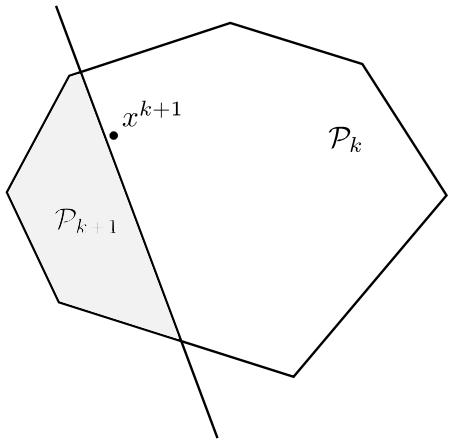


Abbildung 6.1: Ein Polytop $\mathcal{P}_k$ mit Punkt x^{k+1} und Schnittebene.

6.2 Schnittebenenverfahren

Das Schnittebenenverfahren startet mit einer Menge von beschränkten Nebenbedingungen, bei denen bekannt ist, dass alle Punkte aus der konvexen Menge K diese erfüllen. Damit wird ein initialer Bereich $\mathcal{P}_0$ um den gesuchten Bereich K gelegt. Ausgehend von $\mathcal{P}_0$ wird in jeder Iteration für einen Punkt überprüft, ob eine der zuvor definierten Schnittebenen durch ihn verletzt ist. Wird keine weitere Schnittebene gefunden, ist ein optimaler Punkt gefunden. Verletzt der Punkt jedoch eine Nebenbedingung, wird diese Bedingung zu den bisher gefunden Bedingungen hinzugefügt.

7 Softwarearchitektur

Die zuvor beschriebenen Lernverfahren, wie die Stützvektormethode, können als Optimierungsprobleme aufgefasst werden. Diese gehen alle bei der Lösung dieser Optimierungsprobleme ähnlich vor. Bei der Sequential Minimal Optimization, der Core Vector Machine, der Ball Vector Machine und der strukturellen Stützvektormethode wird das Gesamtproblem in eine Reihe von Subproblemen unterteilt. Jedes Subproblem wird durch eine Teilmenge der Daten beschrieben (*WorkingSet*) und für jedes wird ein Lösungsverfahren benötigt. Durch die Lösung aller Subprobleme wird das gesamte Optimierungsproblem gelöst. Das Gesamtproblem wird also durch einen Optimierer (*Optimizer*) gelöst, der zwei Verfahren beinhaltet: ein Auswahlverfahren der Subprobleme (*WorkingSetController*) und ein Lösungsverfahren (*ProblemSolver*). Dieser Zusammenhang wird in Abb. 7.1 veranschaulicht. Im Allgemeinen kann jedes Lernverfahren, auch solche die nicht zu den Dekompositionsverfahren zählen, auf diese Art und Weise beschrieben werden, indem das Gesamtproblem als ein Subproblem verstanden wird.

Im Rahmen dieser Arbeit wird ein Framework entwickelt, welches die Kombinierbarkeit von verschiedenen Arten der Subproblemzusammenstellung und Lösungsverfahren auf einfache Weise ermöglicht. Für die Core Vector Machine kann die Erstellung des CoreSets als Auswahlverfahren für zu lösende Subprobleme verstanden werden. Diese Subprobleme können etwa durch die Sequential Minimal Optimization oder andere Verfahren, wie zum Beispiel Rank-one Aktualisierungsverfahren (vgl. [6]), gelöst werden. Für die Ball Vector Machine als Heuristik wird hingegen kein Lösungsverfahren benötigt, sondern nur eine Auswahl von Subproblemen.

Neben der einfachen Kombinierbarkeit der zwei benötigten Verfahren wird durch das entwickelte Framework eine transparente Datenhaltung ermöglicht. Die Daten können in jeglicher Form vorliegen und werden durch eine Schnittstelle (*DataSource*) den Verfahren transparent zur Verfügung gestellt. Die Verfahen sind damit unabhängig von der Art der Datenquelle. So können die Daten in einer Datenbank, im Hauptspeicher oder über ein Netzwerk zur Verfügung gestellt werden und beispielsweise die Subprobleme durch die Auswahlverfahren außerhalb des Arbeitsspeichers zusammengestellt werden. Im Gegensatz zu Verfahren, die auf den Arbeitspeicher begrenzt sind, können mit diesem Framework auch die immer größer werdenden Datenmengen analysiert werden (vgl. [32]). In den konkreten Implementierungen der Datenquellen können auch Caching-Verfahren verwendet werden. Im Rahmen dieser Arbeit werden etwa häufig verwendete Teile der Kernmatrix zwischengespeichert. Ferner könnten auch fortgeschrittene Datenbank-Caching-Verfahren implementiert werden.

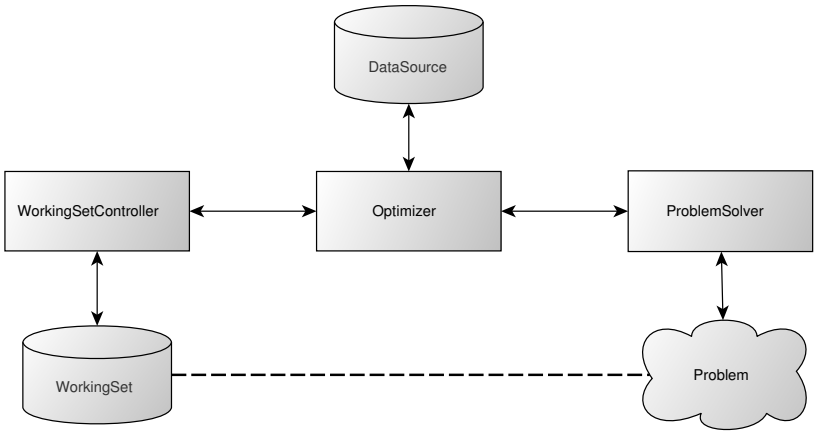


Abbildung 7.1: Zusammenhang der Komponenten des implementierten Frameworks

Die Auswahlverfahren stellen eine Teilmenge der Daten zusammen, die durch ein WorkingSet beschrieben wird. Damit die Daten nicht mehrfach gehalten werden, verweisen die Einträge des WorkingSets nur auf die durch die Schnittstelle zur Verfügung gestellten Daten. Soll dieses WorkingSet einem Lösungsverfahren als Datenquelle dienen, muss die Klasse auch die Schnittstelle DataSource implementieren. Für den WorkingSetController dient diese Klasse somit als WorkingSet und dem ProblemSolver als Datenquelle. Werden Optimierungsalgorithmen externer Anbieter benutzt, die eine eigene Problemdarstellung benötigen, kann die Verbindung zwischen dem hier vorgestellen Framework und dem externen Algorithmus durch die Schnittstelle Problem implementiert werden.

Durch die flexible Gestaltung des Frameworks können einzelne Komponenten sogar auf verschiedenen Computern oder die Lösungsverfahren mit Grafikprozessoren (GPU) gelöst werden. Die Komponenten der Verfahren müssen dafür anders implementiert werden, die Schnittstellen der Verfahren ändern sich dadurch aber nicht.

Nachfolgend werden nun die einzelnen Schnittstellen und die zugehörigen Methoden erläutert.

DataSource

Beschreibung	Das Interface *DataSource* stellt eine zentrale Zugriffsschnittstelle zu allen Daten her, die von allen Komponenten benötigt werden. Die Benutzung dieser Schnittstelle ermöglicht den Zugriff auf verschiedene unterliegende Datenquellen, sowie einen transparenten Zugriff auf Datenbanken oder andere Datenquellen. Eine zentrale Datenschnittstelle garantiert eine konsistente Datenhaltung. Häufig auftretende Datenaggregationen können vereinheitlicht werden. Für die implementierten Stützvektormethoden stellt die *DataSource* alle Informationen über die Beispiele, die Lagrange-Multiplikatoren und Verfahrensparameter, wie die Definition des Epsilons, dar. Werden Daten häufig berechnet, wie die Kernauswertungen, kann die *DataSource* Caching-Verfahren benutzen. Insbesondere bei den im Rahmen dieser Arbeit implementierten Verfahren würde durch eine lokale Datenhaltung der berechneten Werte in verschiedenen Klassen die Übersichtlichkeit leiden. Eine zentrale Schnittstelle für die Daten und berechneten Werte können übersichtlich dokumentiert werden. Ein hoher Grad an Nachvollziehbarkeit der Datenzugriffe und -veränderungen ist somit gewährleistet.

Optimizer

Beschreibung	Die Schnittstelle *Optimizer* stellt einen Optimierer für das Gesamtproblem dar. Für den zuvor beschriebenen Optimierer wurde eine Erweiterung dieser Schnittstelle, der *WorkingSetOptimizer* erstellt.
optimize()	Durch die Methode *optimize()* wird eine eine Lösung für das gesamte Optimierungsproblem erstellt. Handelt es sich um einen iterativen Optimierer, wird nach dem Aufruf eine Iteration durchgeführt. Ein *WorkingSetOptimizer* wählt in jeder Iteration mit Hilfe des *WorkingsetControllers* ein Subproblem und löst dieses mit einem *ProblemSolver*.
boolean convergence()	Diese Methode gibt an, ob die gefundene Lösung des Optimierers eine zuvor definierte Güte besitzt.
initWorkingSet()	Diese Methode berechnet ein initiales *WorkingSet*.
calculate WorkingSet()	Diese Methode stellt ein neues Subproblem zusammen. Bei Verwendung eines *WorkingSetController* wird dieser dazu benutzt, ein neues *WorkingSet* zusammenzustellen.
updateProblem()	Wird ein Optimierer benutzt, der eine eigene Problemdarstellung benötigt, werden durch diese Methode die Daten des *WorkingSets* und andere benötige Parameter in die jeweilige Problemdarstellung überführt.
updateDataSource()	Wird eine Problemdarstellung mit *Problem* implementiert, werden nach der Lösung des Problems die Ergebnisse mit dieser Methode in die *DataSource* übertragen.

WorkingSet

Beschreibung Der *WorkingSetController* stellt eine Teilmenge der Daten zusammen und speichert diese im *WorkingSet*. Im Idealfall ist ein *WorkingSet* nur eine Menge von Indizes. Es werden die Indizes und alle das *WorkingSet* betreffende Informationen dort gespeichert. Enthällt ein *WorkingSet* nur Indizes, dann müssen die Daten nur einmal im Speicher gehalten werden. Im Fall der Core Vector Machine und der Ball Vector Machine wird das CoreSet durch ein *WorkingSet* dargestellt. Die Core Vector Machine benötigt nur Ausschnitte der Kernmatrix. Damit der Speicherbedarf nicht zu groß wird, werdedn nicht die Teile der gesamten Kernmatrix zwischengespeichert, sondern nur die *WorkingSet* betreffenden Einträge. Wird das CoreSet vergrößert, werden die neuen Daten dem Cache hinzugefügt.

WorkingSetController

Beschreibung Der *WorkingSetController* wählt aus allen Daten der *DataSource* eine Untermenge aus und speichert diese in einem *WorkingSet*. Ein Beispiel für die Auswahl eines *WorkingSet* ist das Auffinden eines neuen CoreVectors.

boolean convergence() Diese Methode gibt an, ob ein weiteres Subproblem gefunden werden kann. Im Fall der Core Vector Machine wird durch diese Methode angegeben, ob ein weiterer CoreVector gefunden werden kann. Der *WorkingSetController* kann also auch über die Konvergenz des Gesamtproblems entscheiden.

initWorkingSet() Diese Methode berechnet ein initiales *WorkingSet*.

calculate WorkingSet() Diese Methode stellt ein neues Subproblem zusammen. Für die Core Vector Machine oder Ball Vector Machine wird ein Punkt außerhalb des bisher gefundenen MEBs gesucht, und dem *WorkingSet* hinzugefügt.

Problem

Beschreibung Das Interface *Problem* abstrahiert die Formulierung eines zu lösenden Optimierungsproblems. Arbeitet der *ProblemSolver* direkt auf den Daten einer *DataSource* und implementiert das zu lösende Problem intern, wie die Sequential Minimal Optimization für die Core Vector Machine, wird keine eigene Implementierung von *Problem* benötigt. Wird aber eine eigene Problemdarstellung benötigt, wie bei der Sequential Minimal Optimizationfür die Stützvektormethode, werden die Daten des *WorkingSets* in diese Problemstellung überführt.

int getDimension() Mit dieser Methode wird die Dimension des Suchraums angegeben.

double getX(index) Mit dieser Methode wird die i-te Lösung zurückgegeben.

ProblemSolver

Beschreibung	Die Schnittstelle *ProblemSolver* stellt ein Lösungsverfahren für ein Optimierungsproblem dar. Werden durch einen *WorkingSet-Controller* Subprobleme zusammengestellt, werden diese gelöst. Durch diese Schnittstelle können beliebige Optimierungsalgorithmen eingebunden werden.
solve()	Diese Methode löst das gegebene Optimierungsproblem. Bei einem iterativen Lösungsverfahren wird nach dem Aufruf dieser Methode eine Iteration durchgeführt.
double getMaxAllowedError()	Diese Methode gibt den maximalen erlaubten Fehler des Lösungsverfahren an.
double getMaxIterations()	Diese Methode gibt die Anzahl der Iterationen an, für die das Auffinden der Lösung erlaubt ist.
double getIsZero()	Diese Methode gibt an, ab welchem Wert die Differenz zweier Zahlen als 0 interpretiert wird.

ISvm

Beschreibung	Die Schnittstelle *ISvm* abstrahiert die Stützvektormethode. Innerhalb des Frameworks besteht die *ISvm* nur aus einem Optimierer, der solange aufgerufen wird, bis er konvergiert.
train()	Diese Methode startet das Training der SVM. Die trainierte *ISvm* stellt das gelernte Model zur Vorhersage dar.
predict(index)	Anhand des gelernten Modells wird mittels *predict(index)* für ein Beispiel eines unabhängigen Datensatzes eine Vorhersage getroffen.
predict(dataSource)	für jedes Beispiel der unabhängigen Datenquelle wird die Methode *predict(index)* aufgerufen.

8 Implementierung

Das im letzten Kapitel vorgestellte Framework wird im Rahmen dieser Arbeit in Java implementiert und in Rapiminer durch ein Plugin integriert, um damit Experimente ausführen und die Infrastruktur zur Analyse der Ergebnisse verwenden zu können. Bei der Implementierung wurde darauf geachtet, dass die Umsetzung der Algorithmen unabhängig von Rapidminer ist, so dass alle Verfahren auch in anderen Implementierungen verwendet werden können. Als beispielhafte Umsetzung des Frameworks werden die vorgestellten Verfahren, die bisher nur in C++ existierten, implementiert. Für jedes Verfahren wird ein Optimierer mit einem *WorkingSetController* und einem *ProblemSolver* erstellt. Die Daten werden nur im Arbeitsspeicher verwaltet. Durch den modularen Aufbau des Frameworks können aber in Zukunft weitere Datenquellen und Optimierungsalgorithmen programmiert werden. Im Folgenden wird zunächst die Verbindung zwischen Rapidminer und dem Framework erläutert und dann die Implementierung des Frameworks und der Verfahren vorgestellt.

8.1 Verbindung zwischen Rapidminer und dem Framework

Um ein Lernverfahren in Rapidminer auszuführen, müssen sogenannte Operatoren implementiert werden. Diese stellen in Rapidminer abstrakt ein Verfahren dar, welches Daten verarbeitet, erzeugt oder speichert. Die Daten liegen in einem für Rapiminer spezifischem Datenformat vor. Damit die Daten innerhalb des Frameworks verarbeitet werden können, wird eine spezielle *DataSource* implementiert. Am Ende des Lernverfahrens muss die Verbindung zwischen dem gelernten Modell und einem für Rapidminer spezifischem Modell hergestellt werden, um die Güte des gelernten Modells auf einem unabhängigen Datensatz testen zu können.

Rapidminer Operatoren

Alle implementierten Verfahren werden in einem eigenen Rapidminer Operator integriert, der von einem im Rahmen dieser Arbeit entwickelten abstrakten Operator erbt. Um die Parameter und Einstellungen der Verfahren einheitlich zu modellieren, wird eine Abstraktion des Instanziierungsprozesses eingeführt. Für jede implementierte Komponente, wie eine Stützvektormethode oder ein Kern, wird ein *IType* definiert (vgl. Abb. 8.1). Jeder *IType* besitzt eine Anzahl von *Parametern*. Ein Parameter kann eine Vielzahl von Typen, wie Fließkommazahlen oder boolsche Werte, darstellen. Der abstrakte Operator bietet die Möglichkeit, die Typen und die Parameter automatisch in die graphische Oberfläche des Rapidminers zu integrieren. Wird eine Liste von gleichen Typen durch den abstrakten Operator hinzugefügt, wird in der graphischen Oberfläche eine Auswahlliste erzeugt. Werden die Parameter in der graphischen Oberfläche eingestellt, kann über die Schnittstelle *IType* auf die Werte der Parameter zugegriffen werden. Jeder *IType*

hat eine *IFactory*, die die zugehörige Komponente des Frameworks, wie einen *IKernel* oder *ISVM*, instanziiert. In den Operatoren werden dann die instanziierten Verfahren aufgerufen.

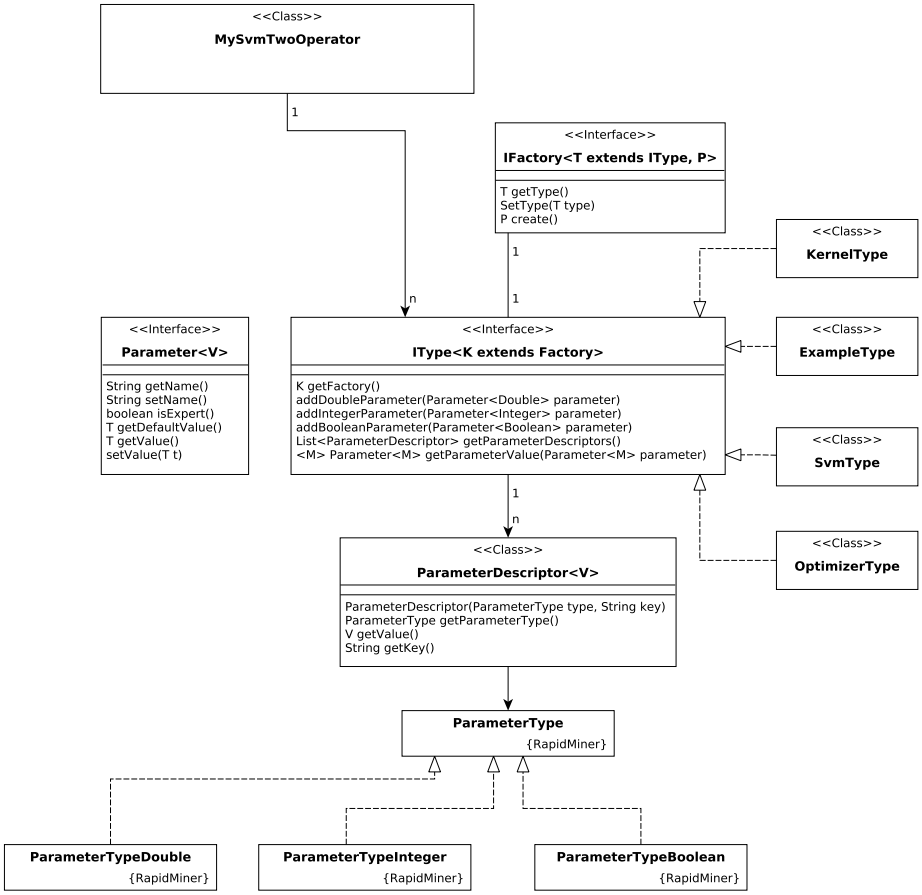


Abbildung 8.1: Komponenten des Instanziierungsprozesses für Rapidminer

Daten

Rapidminer verwendet zur Datenhaltung eine eigene abstrakte Schnittstelle. Die für die Verfahren implementierte *DataSource* könnte diese Schnittstelle einfach an die Verfahren weiterreichen. Jedoch sollen die implementierten Verfahren auch unabhängig von Rapidminer, beispielsweise auf Smartphones, ausführbar sein. Desweiteren ist die Datenschnittstelle von Rapidminer aufgrund der Vielzahl von möglichen Verwendungen nicht effizient implementiert. Daher werden die Beispiele in diesem Framework in ein eigenes Format übertragen, welches anstelle auf Objekten, wie *List*, *Integer* oder *Double*, nur auf primitiven Datentypen und Arrays von primitiven Datentypen operiert. Diese Datenbasis wird durch eine *DataSource* zugreifbar gemacht (vgl. Abschnitt 8.2.1).

Modell

Das gelernte Modell wird durch eine trainierte *ISVM* dargestellt. Um dieses Modell in Rapidminer verwenden zu können, wird eine Klasse implementiert, die von einem *KernelModel* aus Rapidminer erbt und die Methoden der *ISVM* zur Vorhersage nutzt.

8.2 Implementierung des Frameworks

Im Folgenden sollen die entscheidenden in Rahmen dieser Arbeit implementierten Klassen, deren wichtigsten Methoden und die Implementierung der zuvor theoretisch vorgestellten Verfahren beschrieben werden. Ferner werden in den jeweiligen Publikationen nicht genannte Implementierungsdetails vorgestellt.

8.2.1 Stützvektormethode

Die in Rapidminer 5.1 enthaltene MySVM besitzt einen fest integrierten Optimierungsalgorithmus, der nicht austauschbar ist. Da sich der Aufbau der MySVM stark von dem hier entwickelten, flexiblen Framework unterscheidet, wird die Implementierung komplett angepasst. Der Algorithmus der MySVM wählt fest 10 Punkte aus, die die Optimalitätsbedingungen (vgl. Kapitel 2) am stärksten verletzen. Diese Punkte werden in ein quadratisches Optimierungsproblem überführt und dann durch das integrierte Lösungsverfahren, die Sequential Minimal Optimization, gelöst. Die Auswahl der Punkte wird so lange wiederholt, bis eine Lösung für das Gesamtproblem gefunden wurde. Das Vorgehen der MySVM kann also konzeptuell leicht in das hier vorgestellte Framework überführt werden.

Die Implementierung aus dieser Arbeit transformiert die Auswahl der 10 Punkte in einen *WorkingSetController* und das integrierte Lösungverfahren in einen *ProblemSolver*. Das Lösungverfahren erwartet die Daten in Form eines quadratischen Optimierungsproblems. Diese eigene Darstellung wird als *Problem* implementiert.

Durch die starke Verzahnung von Rapidminer, der Auswahl der Punkte und des Lösungsverfahrens in der Implementierung der MySVM können daraus keine Komponenten direkt übernommen werden. So muss eine komplette Anpassung aller Klassen durchgeführt werden. Außerdem werden die Kernfunktionen auf das neu erstellte Datenformat angepasst. Mit der neuen Implementierung können die einzelnen Komponenten problemlos ausgetauscht und sogar in anderen Verfahren benutzt werden. Insbesondere die Implementierung der Sequential Minimal Optimizationist für andere Verfahren von großem Interesse, da es sich dabei um ein Standardlösungsverfahren handelt.

Um gleichzeitig eine flexible Gestaltung des Datenformats und eine hohe Zugriffsgeschwindigkeit zu gewährleisten, werden die eigentlichen Daten zwar in Arrays gehalten, der Zugriff wird aber durch eine Schnittstelle (*AbstractSvmExamples*) gegeben. Ein Beispiel wird durch die Klasse *SvmExample* dargestellt. Die *AbstractSvmExamples* speichern aber keine Menge von *SvmExamples*, sondern einen zwei dimensionalen Array der Attribute. In der Klasse werden nur zwei Referenzen auf Objekte gehalten. Wird

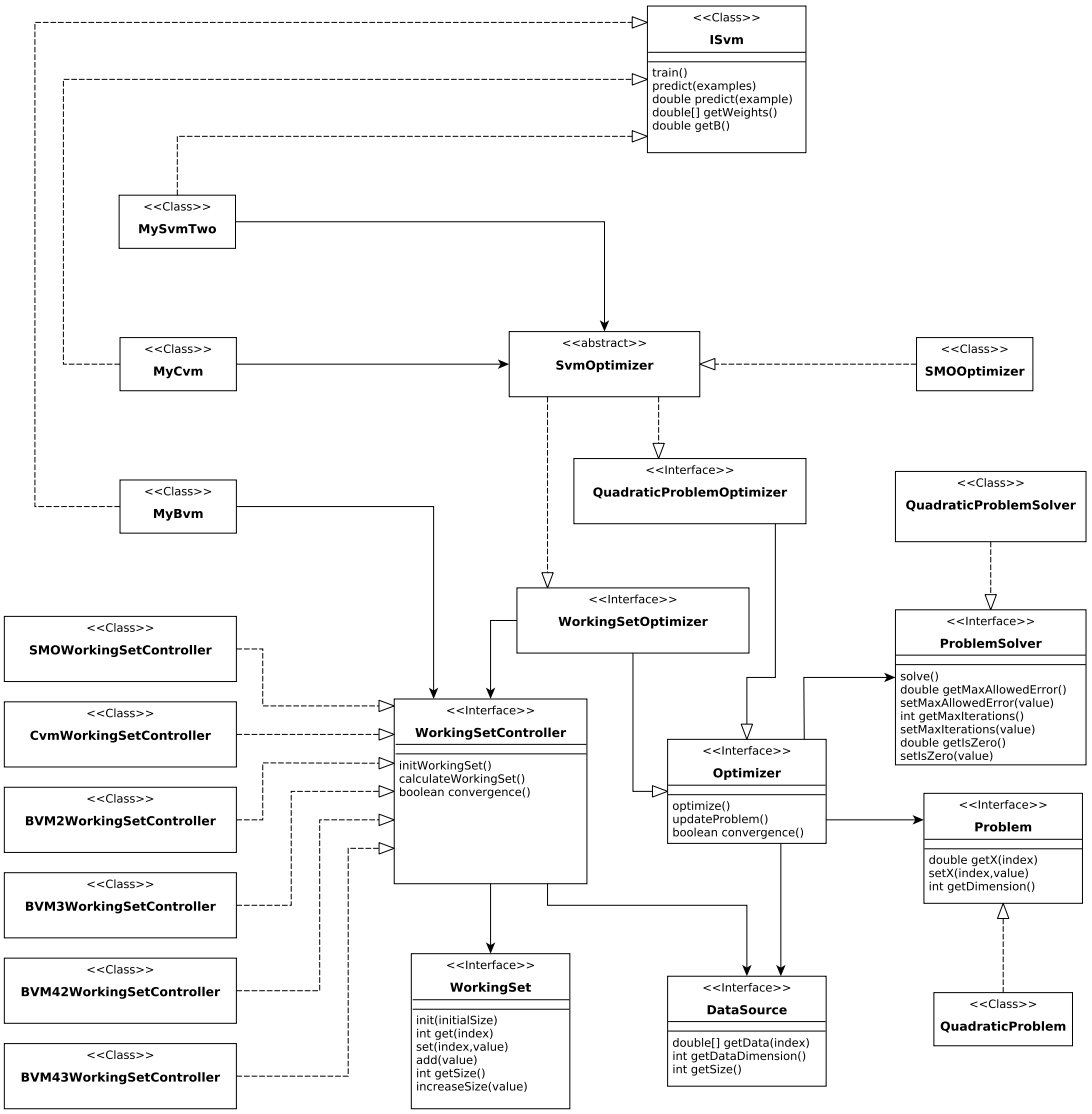


Abbildung 8.2: Architektur der Implementierung

bei der Klasse *AbstractSvmExamples* auf ein *SvmExample* zugegriffen, wird wechselnd die interne Datenstruktur des *SvmExamples* ausgetauscht und das *SvmExample* zurück gegeben. Es werden genau zwei Objekte vorgehalten, da durch die Kernfunktion höchstens zwei Beispiele gleichzeitig benötigt werden. Für den Zugriff auf ein Objekt wird also nur ein Verweis im Speicher geändert. Für spärliche und nicht spärliche Daten kann jeweils eine andere Datenmatrix implementiert werden. Im spärlichen Fall werden die einzelnen Attribute in einer eigenen Datenstruktur, ähnlich zur Implementierung der LibSVM, gehalten. Diese Datenstruktur enthält zwei Informationen: den Index und den Wert des Attributes. Die implementierte *DataSource* bietet einen transparenten Zugriff auf die Berechnungen der Kernfunktion. In der MySVM ist der Aufruf der Kernfunktion direkt abhängig von der gewählten internen Datenstruktur der Beispiele. Anders als bei

der MySVM werden die Ergebnisse nicht im jeweiligen Kern zwischengespeichert, sondern in der Implementierung der *DataSource*. Dadurch können die Implementierungen der Kernfunktionen verändert werden, ohne die Schnittstelle auf die Daten ändern zu müssen.

8.2.2 Core Vector Machine

Die Core Vector Machine ist in zwei neu entwickelten Klassen implementiert: einem *WorkingSetController* und einem *ProblemSolver*. Das CoreSet wird durch ein *WorkingSet* dargestellt. Der *WorkingSetController* sucht in jeder Iteration einen Punkt außerhalb des in der letzten Iteration gefundenen MEBs. Die Suche findet nicht nur auf allen Beispielen, sondern auf einer zufällig gewählten Menge statt. Wie in [38] aufgeführt, ist der von Linux-Systemen angebotene Zufallsegenerator nicht gut für randomisierte Verfahren geeignet. Damit gute Ergebnisse auf allen Betriebssystemen erreicht werden können, wird ein alternativer Zufallsgenerator direkt in das Framework integriert. Dieser wird in Abschnitt 8.2.4 vorgestellt.

Wird ein Beispiel außerhalb des MEBs gefunden, wird dieses dem *WorkingSet* hinzugefügt. Wird kein Beispiel gefunden, wird das Verfahren beendet. Der *WorkingSetController* entscheidet also über die Konvergenz des gesamten Verfahrens. Das neue Subproblem wird dann mit dem *ProblemSolver* gelöst. Das zu lösende Problem besteht aber nicht aus einem Zwei-Klassen-Problem, sondern aus einem Ein-Klassen-Problem (vgl. (4.20)) und kann damit nicht mit der zuvor vorgestellten Sequential Minimal Optimization gelöst werden. Daher muss ein speziell auf das Problem angepasste Lösungsverfahren implementiert werden. Das Zwei-Klassen Problem wird durch eine spezielle Kernfunktion in ein Ein-Klassen Problem transformiert. Um nicht jede Kernfunktion anpassen zu müssen, wird ein Wrapper implementiert, der intern den eigentlichen Kern berechnet und dann das Ergebnis in der gewünschten Form verändert.

Der implementierte *ProblemSolver* arbeitet auf einer *DataSource*. Um die Daten konsistent zu halten, implementiert die Klasse des *WorkingSets* auch die Schnittstelle *DataSource*. Diese Klasse stellt also gleichzeitig ein *WorkingSet* und eine *DataSource* dar. Innerhalb einer Iteration benötigt das Lösungsverfahren nur einen Teil der Kernmatrix, da nur auf dem CoreSet optimiert wird. Im Verhältnis zum Gesamtproblem enthält das CoreSet nur wenige Beispiele enthält, so dass durch die Zwischenspeicherung der Subkernmatrix eine Beschleunigung des Auswahl- und des Lösungsverfahrens erreicht werden kann. In der MySVM wird eine gesamte Zeile der Kernmatrix mit allen überflüssigen Kernauswertungen vorgehalten. Wird beispielsweise eine Zeile nicht im Cache gefunden, wird diese Zeile komplett berechnet, obwohl die Kernauswertungen für Punkte außerhalb des CoreSets nicht benötigt wird. Ändert sich die Größe des CoreSets, müsste in der naiven Implementierung der gesamte Cache gelöscht werden, da die gespeicherten Zeilen der Subkernmatrix nicht alle Einträge enthalten. Es werden alle schon berechneten Kernauswertungen verworfen, obwohl diese wieder im nächsten Optimierungsschritt benötigt werden. In der Imlementierung des *WorkingSets* wird daher eine erweiterte Caching Stratgie verfolgt. Wird auf eine Zeile zugegriffen und ist diese im Cache enthalten, wird überprüft, ob sich die Länge der Zeile von der Größe des *WorkingSets*

unterscheidet. Bei einer Differenz wird ein Array der neuen Größe erzeugt, die alten Werte kopiert und die neuen Werte berechnet. Diese Zeile wird dann anstelle der alten im Cache gespeichert und zurückgegeben. Wird auf diese Zeile nochmals zugegriffen und befindet sie sich immer noch im Cache, muss keine neue Berechnung durchgeführt werden.

8.2.3 Ball Vector Machine

Die Ball Vector Machine ist analog zur Core Vector Machine aufgebaut, benötigt aber als heuristisches Verfahren keinen *ProblemSolver*. Für die Implementierung der Ball Vector Machine müssen zwei Dinge berechnet werden: der Abstand zwischen dem aktuellen Zentrum c_t und einem Punkt $\varphi(x)$ und die Parameter α des zugrundeliegenden Optimierungsproblems. Im Rahmen dieser Arbeit werden alle vorgestellten Algorithmen der Ball Vector Machine implementiert (vgl. Abschnitt 4.4).

In jeder Iteration t in einem der vier Algorithmen muss der Abstand zwischen dem aktuellen Zentrum c_t und einem zufällig gewählten Punkt $\varphi(x)$ berechnet werden. Diese Punkte werden mit dem in Abschnitt 8.2.4 vorgestellem Zufallsgenerator aus der Menge aller Beispiele ausgewählt. Der quadrierte Abstand ist definiert als:

$$d(c_t, \varphi(x))^2 = \|c_t - \varphi(x)\|^2 = \|c_t\|^2 - 2c_t\varphi(x) + \|\varphi(x)\|^2 . \tag{8.1}$$

Mit Formel (4.46) ist bekannt, das sich das neue Zentrum als Konvexkombination aus dem alten Zentrum und dem zuletzt hinzugefügten Punkt errechnen lässt:

$$c_{t+1} = \varphi(x_t) + \beta_t(c_t - \varphi(x_t)). \tag{8.2}$$

Damit lässt sich auch die quadrierte Norm des Zentrums berechnen:

$$\begin{aligned} \|c_{t+1}\|^2 &= \|\beta_t c_t + (1-\beta_t)\varphi(x_t)\|^2 \\ &= \beta_t \|c_t\|^2 + (1-\beta_t)\|\varphi(x_t)\|^2 + (\beta_t^2 - \beta_t)\|c_t - \varphi(x_t)\|^2 . \end{aligned} \tag{8.3}$$

Nach Einsetzen von Formel (8.3) in Formel (8.1) ergibt sich für den quadrierten Abstand:

$$\begin{aligned} d(c_{t+1}, \varphi(x))^2 = \beta_t \|c_t\|^2 + (1-\beta_t)\|\varphi(x_t)\|^2 + (\beta_t^2 - \beta_t)\|c_t - \varphi(x_t)\|^2 \\ - 2c_{t+1}\varphi(x) + \|\varphi(x_t)\|^2 . \end{aligned} \tag{8.4}$$

Zur Vereinfachung von Formel (8.4) wird die Definition von β und c_{t+1} aus Kapitel 4 eingesetzt:

$$\begin{aligned} (\beta_t^2 - \beta_t)\|c_t - \varphi(x_t)\|^2 &= \left(\frac{r^2}{\|c_t - \varphi(x_t)\|^2} - \frac{r}{\|c_t - \varphi(x_t)\|}\right)\|c_t - \varphi(x_t)\|^2 \\ &= r(r - d_t) \end{aligned} \tag{8.5}$$

$$\begin{aligned} c_{t+1}\varphi(x) &= (\varphi(x_t) + \beta_t(c_t - \varphi(x_t)))\varphi(x) \\ &= \varphi(x_t)\varphi(x) + \beta_t c_t\varphi(x) - \beta_t\varphi(x_t)\varphi(x) \\ &= (1-\beta_t)\varphi(x_t)\varphi(x) + \beta_t c_t\varphi(x). \end{aligned} \tag{8.6}$$

Durch Einsetzen von Formel (8.5) und Formel (8.6) in Formel (8.4) wird die Grundlage für die rekursive Berechnung des Abstandes gelegt. Mit $\kappa = \|\varphi(x)\|^2$ folgt:

$$\begin{aligned} d(c_{t+1}, \varphi(x))^2 = \beta_t \|c_t\|^2 + (2 - \beta_t)\kappa + r(r - d_t) \\ - 2((1 - \beta_t)\varphi(x_t)\varphi(x) + \beta_t c_t \varphi(x))). \end{aligned} \tag{8.7}$$

Damit kann der Abstand rekursiv berechnet werden. Der initiale Mittelpunkt $c_0 = \varphi(x_0)$ wird zufällig gewählt. Der quadratische Abstand zwischen Mittelpunkt c_0 und einem beliebigen Punkt z wird in der ersten Iteration berechnet durch:

$$d(c_0, \varphi(z))^2 = 2(\kappa - \varphi(x_0)\varphi(z)). \tag{8.8}$$

Wird ein Punkt außerhalb des gewählten Radius gefunden, wird der Abstand d_t zwischen aktuellem und gefundenem Punkt, sowie die Norm c_{t+1} zwischengespeichert. Die rekursive Berechnung des Abstandes kann durch eine iterative Berechnung ersetzt werden. Dabei seien $x_i,\ i = 1, \ldots, t$, die bisher gefundenen Punkte. Für die Hilfsparameter δ_i gilt:

$$\begin{aligned} \delta_0 = & \quad d(c_0, \varphi(z)) \\ \delta_1 = & \quad \delta_0\beta_1 + (1 - \beta_1)\varphi(x_1)\varphi(z) \\ \vdots & \\ \delta_t = & \quad \delta_{t-1}\beta_t + (1 - \beta_t)\varphi(x_t)\varphi(z). \end{aligned} \tag{8.9}$$

Der quadratische Abstand zwischen dem Punkt z und dem aktuellen Mittelpunkt c_t errechnet sich dann als:

$$d(c_t, \varphi(z))^2 = \|c_t\| + \kappa - 2\delta_t. \tag{8.10}$$

Dadurch ergibt sich eine neue Möglichkeit des Cachings. Die Menge der Punkte, die das aktuelle Zentrum beschreiben, verändern sich nicht mehr. Es kann also für alle Beispiele der maximale Index k und der Wert von δ_k zwischengespeichert werden. Soll der Abstand zwischen dem Beispiel z und einem Zentrum c_t mit $t > k$ berechnet werden, müssen nur noch $\delta_{k+1}, \ldots, \delta_t$ berechnet werden. Nach der Berechnung wird für das Beispiel z der Index t und δ_t gespeichert. Diese neue Caching Strategie speichert keine Kernauswertungen, sondern benötigt nur zwei Zahlen pro Beispiel, um eine schnelle Abstandsberechnung umzusetzten.

Wenn kein Punkt mehr außerhalb des Balls gefunden wurde, werden die Lagrange-Multiplikatoren α des Optimierungsproblems aus den β_t berechnet. Das Zentrum des Balls (vgl. Formel (4.18)) ist definiert als:

$$c = \sum_{i=1}^{N} \alpha_i \varphi(x_i). \tag{8.11}$$

Es müssen also jene Multiplikatoren α berechnet werden, die mit den $\varphi(x_i)$ das aktuell errechnete Zentrum c_{t+1} ergeben. Dann folgt aus Formel (4.46):

$$
\begin{aligned}
c_{t+1} &= \varphi(x_t) + \beta_t(c_t - \varphi(x_t)) = (1-\beta_t)\varphi(x_t) + \beta_t c_t \\
&= (1-\beta_t)\varphi(x_t) + \beta_t((1-\beta_{t-1})\varphi(x_{t-1}) + \beta_{t-1}c_{t-1}) \\
&= (1-\beta_t)\varphi(x_t) + \beta_t(1-\beta_{t-1})\varphi(x_{t-1}) + \beta_t\beta_{t-1}c_{t-1} \\
&= (1-\beta_t)\varphi(x_t) + \beta_t(1-\beta_{t-1})\varphi(x_{t-1}) \\
&\quad + \ldots + \beta_t\beta_{t-1}\ldots(1-\beta_0)\varphi(x_1) + \beta_t\beta_{t-1}\ldots\beta_0 c_0),
\end{aligned}
\tag{8.12}
$$

woraus sich die Multiplikatoren der $\varphi(x_t)$, also die Lösungen α, ablesen lassen.

Die Implementierung des ersten Ball Vector Machine Algorithmus (vgl. Algorithmus 2) verwendet die zuvor beschriebene Abstandsberechnung. Es wird ein *WorkingSetController* implementiert, der in jeder Iteration ein Beispiel außerhalb des aktuellen Balls sucht. Neu gefundene Beispiele werden zum *WorkingSet* hinzugefügt. Wird kein Beispiel mehr gefunden, dann konvergiert das Verfahren. Zum Schluss werden die $\alpha_i,\ i = 1, \ldots, t$, berechnet.

Die Implementierung des zweiten Ball Vector Machine Algorithmus (vgl. Algorithmus 3) wird als ein Wrapper des ersten implementiert. Der Algorithmus startet mit einem $\epsilon = 2^{-1}$. Mit diesem Wert wird der erste Algorithmus aufgerufen. Wird kein Beispiel außerhalb des aktuellen Balls $B(c_t, (1+\epsilon)R)$ gefunden, wird ϵ verkleinert und der erste Algorithmus abermals aufgerufen. Dieses Vorgehen wird solange wiederholt, bis ϵ M mal verkleinert wurde.

Die Implementierung der Algorithmen mit Radiusanpassung (vgl. Algorithmus 4) erbt von den zuvor beschriebenen Algorithmen. Das besondere an der Implementierung ist, dass das *WorkingSet* um eine zusätzliche Dimensionen erweitert wird, um in jeder Iteration den zuvor gültigen Radius zu verkleinern (vgl. Abschnitt 4.4).

8.2.4 Zufallsgeneratoren

Wie bereits erwähnt, ist der Standardzufallsgenerator für die Core Vector Machine und Ball Vector Machine nicht geeignet. Deswegen wird in den implementierten Verfahren eine algemeine Schnittstelle für Zufallsgeneratoren benutzt. Die Schnittstelle kann durch den Standardzufallsgenerator, aber auch durch alternative Zufallsgeneratoren implementiert werden. Für die im Rahmen dieser Arbeit implementierten Verfahren wird der Zufallsgenerator MersenneTwister (vgl. [21]) verwendet. Dieser gilt als schneller und zuverlässiger Zufallsgenerator für gleichverteilte ganze Zahlen. Es wird keine eigene Implementierung durchgeführt, sondern eine bereits existierende Implementierung aus dem Java-Paket Java-Colt verwendet.

8.3 SVM-Struct

Um eine Implementierung der strukturellen Stützvektormethode zu erstellen, muss insbesondere eine eigene *DataSource* erstellt werden. Neue *SvmExamples* müssen implementiert werden, die für die Vielzahl von Anwendungen einsetzbar sind. Die *SvmExamples*

müssen mit einer auf die Anwendung angepassten *ExampleFactory* erzeugt werden. In einem erweiterten *SvmExample* werden neben den abstrakt definierten Attributen, die abstrakt definierten strukturierten Ausgabevariablen und der Feature-Vektor definiert. Die Kern-Funktion kann mit den bereits implementieren *IKernel* weiterhin berechnet werden. Einträge der Kernmatrix werden wie bisher in der *DataSource* zwischengespeichert. Sollen verschiedene Verlustfunktionen benutzt werden, muss eine Schnittstelle dafür definiert werden. Diese wird in die Schnittstelle der *DataSource* integriert. Aufgrund der verschiedenen Varianten des Algorithmus sollte der *WorkingSetController* die Möglichkeit besitzen, mehrere *WorkingSets* gleichzeitig zu besitzen, damit die Optimierung der Mengen $S_i,\ i = 1, \ldots, N$, aus dem Algorithmus parallelisiert werden kann. Der *WorkingSetController* sollte einen *ProblemSolver* für die Maximierung der Kostenfunktion beinhalten. Die Optimierung der durch den *WorkingSetController* gewählten Subprobleme muss mit einem *ProblemSolver* gelöst werden.

9 Experimente

Die Experimente werden auf den in Tabelle 9.1 dargestellten und im Internet frei zugänglichen Datensätzen durchgeführt[1]. Falls nicht anders aufgeführt, gilt für den Parameter der Stützvektormethode $C = 1$. Als Seed wurde mit Ausnahmen auf 2001 gesetzt. Es wird der RBF-Kern mit $exp(-\|x - z\|^2 / \beta)$ mit $(\gamma = 1/\beta)$ verwendet, wobei β die mittlere quadratische Differenz zwischen den Trainingspunkten beschreibt. Um vergleichbare Ergebisse für die Verfahren zu erhalten, wird kein Shrinking verwendet (vgl. [12]). Alle Experimente werden auf der Sun Java JDK 1.6.0_24 unter Ubuntu 10.10 Linux (64-Bit) mit 4 GB Speicher für den Java Prozess ausgeführt. Der Computer, auf dem die Experimente ausgeführt wurden, hat einen Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz und 8 GB Speicher.

Datensatz	#Klassen	Dimension	#Trainingsbeispiele	#Testbeispiele	γ
w3a	2	300	4912	44837	0,0018
reuters	2	8315	7770	3299	6,18 10^{-5}
ijcnn1	2	22	49990	91701	0,0360
usps	2	676	266079	75383	0.0057
intrusion	2	127	4898431	311029	0,0049

Tabelle 9.1: Datensätze für die Experimente

Die Methoden werden für unterschiedliche ϵ auf den genannten Datensätzen durchgeführt. Die Ergebnisgenauigkeit, Anzahl gefundener Stützvektoren und die Laufzeit sind für die verschiedenen Verfahren in den Tabellen 9.2 bis 9.4 zusammengefasst. Experimente einer Methoden, die aufgrund mangelnder Speicherkapazität nicht ausgeführt werden konnten oder eine Trainingszeit länger als 10000 Sekunden besitzen, werden mit „-“ in die Tabellen eingefügt.

9.1 Ergebnisse der SVM

Die Ergebnisse für die MySVM sind für $C = 1$ und $\epsilon = 10^{-4}$ in Tabelle 9.2 dargestellt. Es zeigt sich, dass die MySVM für die drei Datensätze w3a, reuters und ijcnn1 sehr gute Ergebnisse liefert mit Genauigkeiten von über 95%. Im Vergleich zur LIBSVM in [37] war die MySVM jedoch immer langsamer und für den reuters Datensatz in allen drei Zielfunktionen (Genauigkeit, Größe und Laufzeit) schlechter. Für w3a war die MySVM zwar ungenauer, fand aber weniger Stützvektoren. Die MySVM hat beim Datensatz

[1] w3a und ijcnn1 sind von http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/, reuters von http://www.daviddlewis.com/resources/testcollections/reuters21578/, usps und intrusion von http://www.cse.ust.hk/ ivor/cvm.html.

ijcnn1 nahezu identische Genauigkeit zur LIBSVM, ist aber langsamer. Die Anzahl der Stützvektoren ist außerdem geringer. Für den Datensatz usps konnte kein Ergebnis mit der MySVM erzielt werden.

Datensatz	ACC	Recall	Precision	SV	CPU
w3a	98,25	75,27	91,38	592	3
reuters	96,03	65,78	91,30	2632	25
ijcnn1	98,62	94,38	97,47	4987	264
usps	-	-	-	-	-

Tabelle 9.2: Genauigkeit (ACC) [%], Recall [%], Precision [%], Anzahl der Stützvektoren (SV) und Laufzeit (CPU) [s] für die Ergebnisse mit der MySVM

9.2 Ergebnisse der CVM

Die Ergebnisse für die Core Vector Machine sind für $C = 1$ und $\epsilon = 10^{-4}$ in Tabelle 9.3. Für den Datensatz w3a konnten zur C-Implementierung vergleichbare Ergebnisse erzielt werden. Für die anderen Datensätze können die vergleichsweise schlechten Ergebnisse auf zu wenige Stützvektoren zurückgeführt werden. Dies kann an speziellen Parametereinstellungen des implementierten Optimierers liegen oder an der Güte des Zufallsgenerators. [38] hat gezeigt, das es zu Problemen mit dem Zufallsgenerator auf Linuxsystemen kommen kann. Die im Paper verwendeten Implementierungen verwenden einfache Genauigkeit (float), hier wird hingegen mit doppelter Genauigkeit (double) gerechnet. Deswegen könnte es bei gleicher Parametereinstellung zu Problemen bei der Berechnung von Schranken kommen.

Datensatz	ACC	Recall	Precision	SV	CPU
w3a	97,83	81,07	81,23	475	1,3
reuters	94,60	51,68	97,37	1070	359
ijcnn1	91,49	56,14	88,50	164	19
usps	-	-	-	-	-

Tabelle 9.3: Genauigkeit (ACC) [%], Recall [%], Precision [%], Anzahl der Stützvektoren (SV) und Laufzeit (CPU) [s] für die Ergebnisse mit der CVM

9.3 Ergebnisse der BVM

Im Folgenden werden die Ergebnisse für die BVM 4.2 und die BVM 3.4.2 vorgestellt, wobei 4.2 bedeutet, das der Algorithmus 4 in Kombination mit Algorithmus 2 aus [37] ausgeführt wird. In [37] wird nicht genau darauf eingegangen, welche BVM Kombination verwendet wurde, aber vermutlich handelt es sich um den BVM 4.2-Algorithmus.

Es zeigt sich, dass für die BVM 4.2 mit $C = 1$ für alle betrachteten Datensätze sehr gute Ergebnisse mit einer Genauigkeit stets über 95% erzielt werden können (vgl. Tabelle 9.4). In [36] wird gezeigt, dass unter der Annahme eines fehlerfreien Trainingsdatensatz $C = 1$ die beste Wahl ist. Die Tabelle zeigt, dass über die verschiedenen $\epsilon \in \{10^{-4}, 10^{-5}, 10^{-6}\}$ zeigten sich nur minimale Unterschiede in der Genauigkeit. Mit abnehmendem ϵ stiegen die Stützvektoreanzahl und Laufzeiten jedoch stark an.

Für den Datensatz w3a ergeben sich sowohl für die Genauigkeit, die CPU-Zeit, sowie die Anzahl der Stützvektoren nur sehr geringe Unterschiede im Vergleich zu den Auswertungen in [37]. Für den Datensatz reuters ergibt sich eine leicht geringere Genauigkeit, die Ausführung ist langsamer und die Anzahl der Stützvektoren ist erheblich größer. Für den Datensatz ijcnn1 ergeben sich sehr ähnliche Genauigkeiten, in wesentlich geringerer Zeit, aber mit bedeutend mehr Stützvektoren. Die Ball Vector Machine hat auf dem Datensatz eine nahezu identische Genauigkeit, ist jedoch wesentlich langsamer, und benötigt entscheidend mehr Stützvektoren. Für $\epsilon = 10^2$ erreicht die hier implementierte Ball Vector Machine für usps eine leicht geringere Genauigkeit, aber mit erheblich weniger Stützvektoren in sehr kurzer Zeit.

Daten-	$\epsilon = 10^{-2}$					$\epsilon = 10^{-4}$					$\epsilon = 10^{-5}$					$\epsilon = 10^{-6}$				
satz	ACC	Rec	Prec	SV	CPU	ACC	Rec	Prec	SV	CPU	ACC	Rec	Prec	SV	CPU	ACC	Rec	Prec	SV	CPU
w3a						98,18	73,82	91,43	718	1,0	98,26	75,67	91,08	851	1,2	98,36	78,91	90,14	870	1,1
reuters						95,67	61,64	91,64	2640	17	95,67	62,16	90,23	3672	16	95,42	58,78	92,19	3914	17
ijcnn1						98,42	95,78	95,13	5677	62	98,11	92,28	96,49	8972	104	98,04	91,79	96,51	9586	111
usps	97,29	97,32	97,16	109	28	99,54	99,46	99,61	5395	1794	99,53	99,45	99,60	9695	3527	99,51	99,42	99,58	11522	4448

Tabelle 9.4: Genauigkeit (ACC) [%], Recall (Rec) [%], Precision (Prec) [%], Anzahl der Stützvektoren (SV) und Laufzeit (CPU) [s] für Analysen mit der BVM 4.2

	w3a					reuters					usps				
C	ACC	Recall	Prec	SV	CPU	ACC	Recall	Prec	SV	CPU	ACC	Recall	Prec	SV	CPU
0,1	97,95	67,78	93,28	1272	1,94	94,85	52,51	97,42	4429	31,44	99,44	99,34	99,51	7860	2707
10	98,33	76,22	92,26	603	0,83	95,64	61,36	91,50	2179	13,67	99,56	99,48	99,62	4487	1470
100	98,21	72,45	94,03	568	0,78	95,79	63,28	90,84	2093	13,27	99,55	99,47	99,61	4390	1502
1000	98,14	71,36	93,56	534	0,75	95,79	63,54	90,24	2105	13,35	99,52	99,43	99,59	4341	1489
10000	98,14	71,36	93,56	534	0,75	95,79	63,54	90,24	2105	13,29	99,59	99,52	99,65	4310	1408

Tabelle 9.5: Ergebnisse der BVM 4.2 bei verschiedener Einstellungen von C und mit $\epsilon = 10^{-4}$

Datensatz	M=5					M=10					M=13				
	ACC	Recall	Prec	SV	CPU	ACC	Recall	Prec	Sv	CPU	ACC	Recall	Prec	Sv	CPU
w3a	98,28	75,97	91,32	459	0,6	98,15	72,75	91,85	470	0,6	97,97	69,06	91,88	475	0,6
reuters	96,42	87,58	81,38	814	4	95,85	63,57	91,75	1133	6	95,42	58,87	92,19	1322	7
ijcnn1	96,07	84,08	91,74	941	10	96,97	89,09	92,74	1386	14	98,23	92,40	97,10	4670	46
usps	94,60	94,93	94,63	984	279	99,86	99,70	99,78	1442	413	99,51	99,43	99,58	3203	805

Tabelle 9.6: Genauigkeit (ACC) [%], Recall [%], Precision (Prec) [%], Anzahl der Stützvektoren (SV) und Laufzeit (CPU) [s] für Analyen mit der BVM 3.4.2

ϵ	Seed	C	ACC	Recall	Prec	SV	CPU
10^{-1}	101	10^6	93,61	95,16	87,81	2	1,6
10^{-2}	2001	10^6	93,61	95,16	87,81	81	116

Tabelle 9.7: Ergebnisse der BVM 4.2 für den Datensatz intrusion

Ferner wurden Experimente auf den mittelgroßen Datensätzen w3a und reuters, sowie auf dem großen Datensatz usps mit verschiedenen Einstellungen der Variable C durchgeführt. Die Ergebnisse für $C \in [10^{-1}; 10^5]$ sind in Tabelle 9.5 aufgeführt. Es zeigt sich, dass die Ball Vector Machine mit steigendem C nahezu identische Genauigkeiten hervorbringt, sich die Working Set size und die Laufzeit jedoch stark verringern. Die Veränderung des Seeds hat keinen Einfluss auf die Ergebnisse. Dies wird jedoch nicht gesondert tabellarisch aufgeführt.

Die einzelnen Algorithmen wurden so implementiert, das sie untereinander beliebig kombiniert werden können. Der BVM 3.4.2.-Algorithmus besteht somit aus der BVM 3, in die die BVM 4 mit integrierter BVM 2 ausgeführt wird (vgl. Abschnitt 8.2.3). Die Anwendung des BVM 3.4.2.-Algorithmus zeigt ähnliche Ergebnisse bzgl. der Genauigkeit bei den verschiedenen Einstellungen von M. Mit steigendem M steigt die ohnehin schon sehr hohe Genauigkeit weiter an, die Anzahl der Stützvektoren und die Laufzeiten steigen jedoch auch an. Die Einstellung $M = 13$ beschreibt einen Ball, der mindest einen Radius von $2^{-13} \approx 10^{-4}$, so dass diese Einstellungen mit denen des BVM 4.2 mit $\epsilon = 10^{-4}$ verglichen werden können. Die Lösungsgenauigkeit ist in beiden Implementierungen ähnlich, der BVM 3.4.2.-Algorithmus erzeugt jedoch wesentlich weniger Stützvektoren und benötigt eine geringere Laufzeit. Somit ist er besser als der Algorithmus BVM 4.2.

Desweiteren zeigt Tabelle 9.7, dass die Ball Vector Machine auch gute Ergebnisse auf sehr großen Datensätzen erreicht, wobei hier $C = 10^6$ gewählt wurde, da für $C = 1$ keine guten Ergebnisse erzielt werden können (vgl. [37]). Auf dem intrusion Datensatz werden mit der Ball Vector Machine Genauigkeiten von über 93% mit nur sehr wenigen Stützvektoren erreicht.

10 Zusammenfassung

Ziel der Arbeit war die Untersuchung von Varianten der Stützvektormethode auf ihre Eignung für ressourcenbeschränkte Systeme. Zunächst wurden die theoretischen Grundlagen für die Stützvektormethode geschaffen, ihre Verwendbarkeit für linear und nicht linear trennbare Daten erläutert und deren Erweiterung mit Kernen vorgestellt.

Die Stützvektormethode formuliert zur Lösung der Lernaufgabe ein konvexes Optimierungsproblem. Dieser Aspekt wurde theoretisch im Detail betrachtet. Da die Integration von Kernfunktionen nur im dualen Problem möglich ist, wurde der mathematische Zusammenhang zwischen primalem und dualem Problem erläutert. Diesen Zusammenhang nutzt die Lagrange-Multiplikator-Methode aus. Dieses gängige Verfahren zu Lösung beschränkter Optimierungsprobleme wurde ausführlich erläutert und die Verbindung der Lösung des primalen und des dualen Problems aufgezeigt. Für die meisten praktischen Anwendungen mit großen Datenmengen ist der naive Ansatz zur Lösung des Optimierungsproblems (Betrachtung aller möglichen Kombinationen der Lagrange-Multiplikatoren) wegen der benötigten Laufzeit von $\mathcal{O}\left(N^3\right)$ und einem Platzbedarf von $\mathcal{O}\left(N^2\right)$ nicht praktikabel.

Es wurden zwei alternative Arten zur Lösung der Stützvektormethode mit Kernen, Kernmatrixapproximationen und Dekompositionsverfahren, vorgestellt. Die Sequential Minimal Optimization als minimales Dekompositionsverfahren löst endlich viele Subprobleme für jeweils zwei Beobachtungen analytisch, um das Gesamtproblem zu lösen, so dass für die Lösung kein numerischer Optimierer notwendig ist. Die Sequential Minimal Optimization fand Anwendung in den Implementierungen dieser Arbeit.

Weitere Laufzeitverbesserungen, die für eine problemlose Anwendung der Verfahren auf den mobilen Computern notwendig sind, werden durch Umformulierung des ursprünglichen Optimierungsproblems in das äquivalente Minimum Enclosing Ball Problem und dessen approximative Lösung erreicht.

Die Verbindung zwischen dem Optimierungsproblem der Stützvektormethode und dem MEB-Problem wird mit Hilfe der Support Vector Data Description hergestellt. Die Core Vector Machine nutzt diesen Zusammenhang der beiden Probleme aus und erreicht damit schließlich eine Verbesserung der Laufzeit. Sie ist besonders für große Datenmengen geeignet, da sie eine lineare Laufzeit und einen von der Menge der Beispiele unabhängigen Platzbedarf besitzt. Ferner wird ein heuristische Verfahren, die Ball Vector Machine, theoretisch beschrieben und implementiert. Sie kommt im Gegensatz zu den übrigen Verfahren ohne jeglichen Optimierer aus, liefert aber eine zu den anderen Verfahren vergleichbare Klassifikationsgüte und benötigt dafür eine geringere Laufzeit.

Aus diesen Gründen sind die Core Vector Machine und insbesondere die Ball Vector Machine für die binäre Klassifikation auf ressourcenbeschränkten Systemen geeignet.

Eine Erweiterung der Stützvektormethode auf allgemeine Klassifikationsverfahren wurde mit der strukturellen Stützvektormethode vorgestellt. Diese kann anstelle von Klassen auf strukturierten Ausgabevariablen lernen und diese vorhersagen. Die strukturelle Stützvektormethode löst genau wie die Core Vector Machine und die Ball Vector Machine anstelle des Gesamtproblems eine Reihe von Subproblemen. Durch das im Algorithmus integrierte Schnittebenenverfahren kann das durch die strukturelle Stützvektormethode gegebene Optimierunsgproblem effizient bearbeitet werden.

Damit die vorgestellten Verfahren einheitlich in Java implementiert werden können, wurde ein Framework entwickelt, das einen Rahmen für Lernverfahren im Allgemeinen und die Stützvektormethoden im Speziellen bildet. Die zentralen Aspekte und Schnittstellen des Frameworks wurden in dieser Arbeit erläutert. Das Framework wurde so abstrakt formuliert, dass die zentralen Klassen der Implementierungen auf einer Vielzahl von Betriebsystemen angewendet werden können, ohne die gesamten Verfahren verändern zu müssen. Der Grad der Abstraktion ist jedoch so gewählt, dass die Verfahren für ein spezielles System effizient implementiert werden können. Des Weiteren wurde eine Schnittstelle für den Datenzugriff erstellt, so dass die implementierten Verfahren transparent auf unterschiedlichste Datenquellen, wie etwa Datenbanken zugreifen können. Dieses Framework bietet eine Grundlage, um weitere Lernverfahren und Optimierer miteinander flexibel kombinieren zu können.

Aufgrund des mangelhaften Zufallsgenerators auf Linux-Systemen musste ein alternativer Zufallsgenerator verwendet werden. Deswegen wurde im Framework außerdem vorgesehen, dass Zufallsgeneratoren flexibel austauschbar sind.

Der Fokus dieser Arbeit liegt auf der binären Klassifikation. Dies spiegelt sich auch in den implementierten Algorithmen wider. Die MySVM wurde basierend auf der Implementierung aus Rapidminer an das Framework angepasst, um ebenfalls flexibel Teilkomponenten austauschen zu können. Die Core Vector Machine wurde implementiert, da sie ein häufig eingesetztes Approximationsverfahren für die Stützvektormethode ist. Nichtsdestotrotz muss in jeder Iteration der Core Vector Machine ein Optimierungsproblem gelöst werden. Für dieses Optimierungsproblem wurde eigens eine Variante der Sequential Minimal Optimization implementiert. Ferner wurde heuristisches Verfahren die Ball Vector Machine in mehreren Versionen umgesetzt. Aus der mathematischen Formulierung der Abstandsberechnung in der Ball Vector Machine wurde eine neue Caching-Strategie entwickelt. Dadurch liefert die Ball Vector Machine bei großen Datensätzen mit wenigen Dimensionen wesentlich schneller gute Ergebnisse. Kleinere Datensätze mit einer großen Anzahl an Attributen (vgl. Datensatz reuters) provitieren hingegen weniger von dieser Strategie und sollten mit klassischen Caching-Strategien bearbeitet werden.

Im Vergleich zur MySVM und der Core Vector Machine liefern alle Versionen der Ball Vector Machine auf den hier exemplarisch ausgewählten Datensätzen Ergebnisse mit ver-

gleichbarer Genauigkeit in kürzerer Laufzeit und mit weniger Stützvektoren. Ein weiterer Vorteil der Ball Vector Machine ist ihre Einsetzbarkeit bei sehr großen Datensätzen (vgl. Datensatz intrusion).

Neben den hier vorgestellten und implementierten Verfahren könnten ferner online Lernverfahren wie zum Beispiel die ONE-PASS SVM [27] untersucht werden, da diese möglicherweise mit den beschränkten Speicherressourcen der mobilen Geräte gut umgehen könnten. Als Erweiterung des Frameworks sind außerdem die Implementierungen der strukturellen Stützvektormethode und eine Ball Vector Machine mit mehreren Bällen gleichzeitig interessant, um die Verfahren auf Parallelsystemen ausführen zu können.

Abschließend lässt sich sagen, dass zur Zeit der BVM 3.4.2.-Algorithmus der aussichtsreichste Kandidat ist, um effizient Lernaufgaben unter Reccourcenbeschränkung zu lösen.

Literaturverzeichnis

[1] Gartner http://www.gartner.com/it/page.jsp?id=1543014.

[2] D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *Journal of the ACM (JACM)*, 54(2):9–es, 2007.

[3] M. Badoiu and K.L. Clarkson. Optimal Core-Sets for Balls. 2002.

[4] M. Bādoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, page 257. ACM, 2002.

[5] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[6] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in neural information processing systems 13: proceedings of the 2000 conference*, page 409. The MIT Press, 2001.

[7] C.C. Chang and C.J. Lin. LIBSVM: a library for support vector machines. 2001.

[8] D. de Ridder, D. Tax, and R.P.W. Duin. An experimental comparison of one-class classification methods. In *Proc. Fourth Annual Conference of the Advanced School for Computing and Imaging, ASCI, Delft*, 1998.

[9] R. Fletcher. Practical Methods of Optimization: Vol. 2: Constrained Optimization. *JOHN WILEY & SONS, INC., ONE WILEY DR., SOMERSET, N. J. 08873, 1981, 224*, 1981.

[10] V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25th international conference on Machine learning*, pages 320–327. ACM, 2008.

[11] T. Hastie, R. Tibshirani, and J.H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Verlag, 2009.

[12] T. Joachims. Making large scale SVM learning practical. 1999.

[13] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, page 226. ACM, 2006.

[14] M. Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

[15] SS Keerthi, SK Shevade, C. Bhattacharyya, and KRK Murthy. A fast iterative nearest point algorithm for support vector machineclassifier design. *IEEE Transactions on Neural Networks*, 11(1):124–136, 2000.

[16] JE Kelley Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[17] S. Khan and M. Madden. A survey of recent trends in one class classification. *Artificial Intelligence and Cognitive Science*, pages 188–197, 2010.

[18] P. Kumar, J.S.B. Mitchell, and E.A. Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *Journal of Experimental Algorithmics (JEA)*, 8:1–1, 2003.

[19] D.A. Ladd, A. Datta, S. Sarker, and Y. Yu. Trends in Mobile Computing within the IS Discipline: A Ten-Year Retrospective. *Communications of the Association for Information Systems*, 27(1):17, 2010.

[20] Y.J. Lee and O.L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the First SIAM International Conference on Data Mining*, pages 00–07. Citeseer, 2001.

[21] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[22] MM Moya, MW Koch, and LD Hostetler. One-class classifier networks for target recognition applications. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1993.

[23] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, 1997.

[24] R. Panigrahy. Minimum enclosing polytope in high dimensions. *Arxiv preprint cs/0407020*, 2004.

[25] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[26] J.R. Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann, 1993.

[27] P. Rai, H. Daumé III, and S. Venkatasubramanian. Streamed learning: one-pass SVMs. *Arxiv preprint arXiv:0908.0572*, 2009.

[28] S.J. Roberts, W. Penny, and D. Pillot. Novelty, confidence and errors in connectionist systems. In *Intelligent Sensors (Digest No: 1996/261), IEE Colloquium on*, pages 10–1. IET, 1996.

[29] D. Roobaert. DirectSVM: A fast and simple support vector machine perceptron. In *Neural Networks for Signal Processing X, 2000. Proceedings of the 2000 IEEE Signal Processing Society Workshop*, volume 1, pages 356–365. IEEE, 2000.

[30] A. Roßnagel. Personalisierung in der E-Welt Aus dem Blickwinkel der informationellen Selbstbestimmung gesehen. *Wirtschaftsinformatik*, 49(1):8–15, 2007.

[31] B. Schölkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[32] T. Schrádi and S. Juhász. Out-of Core Processing in Preparation Phase of Data Mining Tasks. *10th Symposium of Hungarian Researchers on Computational Intelligence and Informatics*, 2009.

[33] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pages 807–814. ACM, 2007.

[34] A.J. Smola, B. Schölkopf, R. Williamson, and P. Bartlett. New support vector algorithms. *Neural computation*, 12(5):1207–1245, 2000.

[35] J.J. Sylvester. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1, 1857.

[36] D.M.J. Tax and R.P.W. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

[37] I.W. Tsang, A. Kocsor, and J.T. Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th international conference on Machine learning*, pages 911–918. ACM, 2007.

[38] I.W. Tsang and J.T. Kwok. Author's reply to the "comments on the Core Vector Machines: Fast SVM Training on Very Large Data Sets". *Journal of Machine Learning Research*, 2007.

[39] I.W. Tsang, J.T. Kwok, and P.M. Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6(1):363, 2006.

[40] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

[41] V. Vapnik. Estimation of dependences based on empirical data. *Nauka*, 1979.

[42] V.N. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 2000.

[43] SVN Vishwanathan, A.J. Smola, and M.N. Murty. SimpleSVM. 2003.

[44] E. Welzl. Smallest enclosing disks (balls and ellipsoids). *New results and new trends in computer science*, pages 359–370, 1991.

[45] K. Zhang, I.W. Tsang, and J.T. Kwok. Improved Nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1232–1239. ACM, 2008.

Erklärung

Hiermit erkläre ich, Hendrik Blom, die vorliegende Diplomarbeit mit dem Titel *Entwicklung von Optimierungsverfahren für das Lösen verschiedener Lernaufgaben mit der Stützvektormethode* selbständig verfasst und keine anderen als die hier angegebenen Hilfsmittel verwendet, sowie Zitate kenntlich gemacht zu haben.

Dortmund, 13. April 2011