

Studiengang Informationstechnik

**Bachelorarbeit**

2014

**Implementierung von Support Vector Machines in der Umgebung  
eines evolutionären Optimierers**

von Timo Schumacher

- Matrikelnr.: 4360263 -

- Kurs: Tinf11 -

Betreuer: Prof. Dr. Harald Kornmayer



Deutsches Zentrum für  
Luft- und Raumfahrt e.V.  
in der Helmholtz-Gemeinschaft

Standort: Köln-Porz

Institut für Antriebstechnik  
Abteilung Fan und Verdichter

Christian Voß

# **Eidesstattliche Erklärung**

Hiermit versichere ich, dass ich  
die vorliegende Arbeit selbstständig und nur unter  
Verwendung der angegebenen Quellen und  
Hilfsmittel angefertigt habe.

---

Köln-Porz, der 16. September 2014

## **Zusammenfassung**

Die Aufgabe eines Klassifikators, Individuen aufgrund ihrer Eigenschaften verschiedenen Klassen zuzuordnen, findet ein breites Anwendungsgebiet in der Informationstechnik. So auch im Institut für Antriebstechnik des Deutschen Zentrums für Luft- und Raumfahrt.

Der institutseigene evolutionäre Optimierer für Verdichterschaufeln verwendet bereits ein Meta-Modell, basierend auf dem Kriging Verfahren, um den Wert der Zielfunktion an unbekannten Orten des Parameterraums abschätzen zu können. Ein weiteres Meta-Modell, das ungültige Bereiche des Parameterraums erkennen kann, würde die Performance des Optimierers weiter verbessern. Dazu ist eine Vorhersage nötig, ob für diesen Bereich des Parameterraums die Prozesskette erfolgreich durchlaufen werden kann oder nicht. Diese Vorhersage soll auf den Informationen der bereits berechneten Individuen erfolgen.

Im Rahmen dieser Arbeit wurde für diese Aufgabe ein Support-Vector-Machine Algorithmus implementiert. Als Besonderheit findet hier ein Kriterium zur Auswahl und Kalibrierung der Kernelfunktion seinen Einsatz. Dadurch ist auch die Verwendung einer komplexen Kernelfunktionen mit hohem Freiheitsgrad möglich.

Dazu wird zunächst der mathematische Hintergrund von Support-Vector-Machines erläutert. Es wird das gewählte Qualitätskriterium vorgestellt sowie der bereits zu Beginn des Projektes bestehende implementierte Kriging-Algorithmus erwähnt.

Die softwaretechnische Umsetzung erfolgt innerhalb des Kriging-Projektes. Die Rahmenbedingung, in der die Implementierung erfolgte, war dadurch bereits gegeben und werden, wie auch die Umsetzung, in dieser Arbeit beschrieben.

Zuletzt wurden Tests durchgeführt, um die Performance des Projektes zu validieren. Dabei wurde die Laufzeit sowie die Qualität der Vorhersage untersucht.

Die Vermutung, das komplexe Kernel eine bessere Vorhersage leisten können, bestätigt sich. Das Problem der Kalibrierung eines solchen Kernels kann durch den Einsatz des Qualitätskriteriums auf ein Bruchteil Aufwands des meist verwendeten Grid-Searches reduziert werden.



## **Abstract**

A classifiers ability of identifying the category of an object based on its properties is of wide use in the field of computer science. The Institute of Propulsion of the German Aerospace Center provides on of these uses.

For the optimization of the shape of compressor blades the institute uses an evolutionary algorithm which already applies a meta-modell. This meta-modell, based on the Kriging algorithm, provides a prediction of the objective function without the need of the costly calculation. Another meta-modell which successfully predicts inoperative areas in the parameterspace could improve the performance of the optimization even further. For this purpose a prediction is needed, whether a new Member will provide a valid result. This prediction is based on the database of existing Members.

In this work a Support Vector Machine Algorithm was implemented to undertake this task. A special feature of this implementation is the use of a criteria for the selection and calibration of the kernel function. Through this it is possible to use a more complex kernel with a high degree of freedom.

This work first describes the mathematical background of Support Vector Machines. The criteria is presented and the already implemented Kriging algorithm is laid out.

The implementation was done as a part of the existing Kriging project. Because of this, the general framework was already given. This framework and the implementation are described.

At last, tests were made to validate the performance of the project. Analyzed were both, the runtime and the quality of the prediction.

The assumption that a more complex kernel will yield better predictions could be confirmed. Until now, the problem of calibrating a kernel was mostly solved with the use of a grid search. The effort needed for this calibration were successfully reduced by the use of the mentioned criteria.

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>X</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Das DLR und das Institut für Antriebstechnik . . . . .	1
1.2 Klassifikatoren . . . . .	2
1.3 AutoOpti . . . . .	3
1.4 Einsatz eines Klassifikators in AutoOpti . . . . .	6
<b>2 Theoretische Grundlagen</b>	<b>8</b>
2.1 Support Vector Machines . . . . .	8
2.2 Quadratic Programming . . . . .	27
2.3 Kriging . . . . .	30
<b>3 Softwaretechnische Umsetzung</b>	<b>33</b>
3.1 Der SVM-Algorithmus . . . . .	33
3.2 Benutzung . . . . .	38
3.3 Programmstruktur und -ablauf . . . . .	39
<b>4 Tests</b>	<b>46</b>
4.1 Test-Setting . . . . .	47
4.2 Testskript . . . . .	49
4.3 Test 1 . . . . .	52
4.4 Test 2 . . . . .	55
4.5 Test 3 . . . . .	57
4.6 Test 4 . . . . .	59
<b>5 Fazit und Ausblick</b>	<b>62</b>

**Literaturverzeichnis**

**64**

# Abbildungsverzeichnis

1	Aufgabenverteilung zwischen Master- und Slaveprozessen <i>Bildquelle:[Sch13]</i> . . . . .	4
2	Nutzung eines Ersatzmodells im Optimierungsprozess <i>Bildquelle:[Sch13]</i> . . . . .	6
3	Die Aufgabe eines Klassifikators . . . . .	7
4	Auswahl der besten Trennebene . . . . .	9
5	Nicht lineare Support Vector Machine . . . . .	15
6	Der des Einfluss von $C$ auf das Modell . . . . .	18
7	Beispiele für zunehmend weniger kompakt werdende Bereiche im Merkmalsraum <i>Bildquelle:[Nie83]</i> . . . . .	22
8	Visuelle Darstellung des Qualitätskriterium für einen Kernel . . . . .	23
9	Gradient Projection kann leicht auf Box-Constraints eingehen . . . . .	30
10	Verschiedene Kombinationsmöglichkeiten der beiden Optimierungen. Eine „Update“-Aktivität entspricht einem einzigen Iterationsschritt. . . . .	36
11	Aktivitätsdiagramm des Trainings . . . . .	37
12	Klassendiagramm der im Training eingesetzten Klassen . . . . .	40
13	Die Beziehung zwischen Trainings- und Vorhersagemodus kann über die verwendeten Klassen dargestellt werden. Die Trainer-Klasse wird nur beim Training eingesetzt, die Kriging-Klasse nur bei der Vorhersage. Durch das Verwenden von einer identischen Instanz der DensityFunction wird sicher gestellt, dass sich das Training und die Vorhersage auf das gleiche Modell beziehen. Diese identische Instanz wird bei der Vorhersage durch das Einlesen der Ergebnisdatei des Trainings erzeugt. . . . .	42
14	Die Form des Verdichters vor und nach der Optimierung <i>Bildquelle:[VAR14]</i> . . . . .	48
15	Für weitere Tests am Prüfstand wurde die berechnete Form aus einem Aluminiumblock gefräst <i>Bildquelle:[VAR14]</i> . . . . .	48
16	Der Ablauf des Skriptes, dargestellt als Struktogramm . . . . .	50
17	Testergebnisse mit der Bibliothek libSVM. Die Knoten des Gitters sind hierbei die Messpunkte <i>Bildquelle:[Sch14]</i> . . . . .	53

---

18	Testergebnisse des Projektes mit fest eingestellten Parametern. Die schwarze Linie in (a) verdeutlicht die Umrisse des Plateaus aus (b) . . . . .	54
19	Die Ergebnisse von SupportVektor Machines bei varierender Größe der Trainingsmenge . . . . .	56
20	Test 2: Die Qualität der Vorhersage und die Trainingsdauer in Abhängigkeit des voreingestellten $C$ . . . . .	58
21	Test 2: Ausgewählte $\theta$ in Abhängigkeit von $C$ . . . . .	58
22	Testergebnisse für einen $\theta$ -Vektor in dem die Einträge unabhängig voneinander sind. (a) und (b) verwenden verschiedene Ansätze für die Optimierung . . . . .	60

# Abkürzungsverzeichnis

<b>DLR</b>	Deutsches Zentrum für Luft- und Raumfahrt
<b>SVM</b>	Support Vector Machine
<b>AT</b>	Institut für Antriebstechnik
<b>AT-FUV</b>	Abteilung Fan und Verdichter
<b>CFD</b>	Computational Fluid Dynamics
<b>TRACE</b>	Turbomachinery Research Aerodynamic Computational Environment
<b>RBF</b>	Radial Basis Funktion

# 1 Einleitung

## 1.1 Das DLR und das Institut für Antriebstechnik

Das Deutsches Zentrum für Luft- und Raumfahrt (DLR) ist eines der größten deutschen Forschungszentren und konzentriert sich auf die Bereiche Luftfahrt, Raumfahrt, Energie und Verkehr. Das Institut für Antriebstechnik (AT) ist hierbei im Bereich Luftfahrt und Energie angesiedelt und forscht auf dem Gebiet der Luftfahrtantriebe und Gasturbinen.

Die Abteilung Fan und Verdichter (AT-FUV) des Instituts für Antriebstechnik entwickelt und verbessert die vorderen Komponente einer Turbomaschine. Dabei handelt es sich um das Gebläse (engl. Fan) und den Verdichter (engl. Compressor). Neben dem Durchführen von Experimenten an mehreren großen Prüfständen ist auch hier, wie in vielen anderen Forschungsbereichen auch, die Simulation ein wichtiges Werkzeug.

Die Simulation von Strömungen wird als Computational Fluid Dynamics (CFD) bezeichnet. In AT-FUV kommt dazu der Strömungslöser Turbomachinery Research Aerodynamic Computational Environment (TRACE) zum Einsatz. TRACE basiert auf den Navier-Stokes-Gleichungen und ist auf Turbomaschinen spezialisiert. Er wird im Institut für Antriebstechnik entwickelt und wird im DLR als Standardverfahren für Innenströmungen eingesetzt [TRA12].

Bei der aerodynamischen Auslegung von Verdichterschaufeln kann es durch numerische Instabilitäten zu einem Abbruch der Prozesskette kommen. Je nach gewählten Parametern kann es auch vorkommen, dass es physikalisch keine Lösung gibt. Ziel dieser Arbeit ist die Entwicklung eines Klassifikators für dieses Umfeld. Dieser soll auf Basis bereits berechneter Verdichterschaufeln bewerten, ob für die gewählten Parameter eine erfolgreiche Strömungssimulation durchgeführt werden kann.

### 1.2 Klassifikatoren

„Bei der Klassifikation von (einfachen) Mustern wird jedes Muster als ein Ganzes betrachtet und unabhängig von anderen Mustern genau einer Klasse  $\Omega_k$  von  $k$  möglichen Klassen  $\Omega_\lambda, \forall_{\lambda \in \{1, \dots, l\}}$  zugeordnet. [...]“ [Nie83]

Um den Stellenwert dieser Aufgabe zu erkennen reicht ein Blick in die Natur. Schon einfachste Lebensformen vermögen die Klassifikation von Sinneseindrücken in Klassen wie „gefährlich“ und „ungefährlich“. Auch der Mensch ordnet neu gewonnene Eindrücke, meist unbewusst, in bereits bekannte Kategorien bzw. Klassen ein.

Die Klassifizierung ist am leichtesten wenn die Eigenschaft der Klassendefinition direkt zugänglich ist. So reicht es beispielsweise bei der Unterscheidung zwischen 3 und 5 Türen Autos aus, die Türen abzuzählen. Oft ist dies jedoch nicht der Fall. Für die Unterscheidung zwischen „giftig“ und „ungiftig“ ist ausprobieren jeder einzelnen Beere nicht sinnvoll. Eine Tierart mit dieser Vorgehensweise hätte sicherlich nicht lange Bestand. Denkbar wäre zum Beispiel das zu Hilfe nehmen der Farbe. Ist sie ein intensives Rot ist die Zuordnung der Beere in „giftig“ eine gute Abschätzung.

Die Zusammenhänge zwischen solch verschiedenen Eigenschaften eines Musters kann durch verschiedene Methoden bestimmt werden. Ein gutes Verständnis der zugrundeliegenden Mechanismen kann ausreichen, um Regeln für die Klassifikation aufzustellen. Sind die Mechanismen jedoch zu komplex für eine detaillierte Beschreibung müssen diese Regeln anders gefunden werden. Es liegt nahe, dabei auf bekannte Muster zu setzen und aus Gemeinsamkeiten und Unterschieden zwischen ihnen auf Regeln zu schließen. Dieser Abstraktionschritt wird auch Training (des Klassifikators) genannt und ist eng mit dem Fachgebiet des maschinellen Lernen (engl. Machine Learning) verwandt. Das oben genannte Beispiel der giftigen Beeren legt eine solche Vorgehensweise Nahe; Hatte ein Tier mehrmals Bauchschmerzen nach den Verzehr von roten Beeren, gebietet ihm sein Instinkt diese das nächste Mal hängen zu lassen, ohne das es einen direkten Zusammenhang zwischen Farbe und Unverträglichkeit kennen würde.

Diese Vorgehensweise, aus einem bekannten Satz an Daten auf allgemeine Regeln zu schließen, findet viele Anwendungen in der modernen Informationstechnik. Hier werden die Regeln durch Algorithmen bestimmt. Beispielsweise wurde die automatische Erkennung handgeschriebener Texte durch Einsatz von trainierten Klassifikatoren deutlich verbessert. Moderne Spam-Filter nutzen bayessche Filter und erzielen deutlich höhere Trefferquoten als mit dem Blacklist-Verfahren.

## 1 Einleitung

---

Auch Gebiete die aufgrund ihrer Komplexität ohne den Einsatz computergestützter Algorithmen nicht untersucht werden konnten, haben durch die moderne Informationstechnik das Werkzeug des maschinellem Lernen hinzugewonnen. So lässt sich zum Beispiel die Art eines Tumors aus den Genexpressionsprofilen des Patienten erkennen [Mar03]. Erst durch das Training anhand einer großen Menge an Krankenakten können die Regeln für einen solchen Sachverhalt ermittelt werden.

Es gibt verschiedene Algorithmen um ein solches Training durchzuführen. Weit verbreitet sind zum Beispiel neuronale Netze, die den Zusammenhängen im Gehirn nachempfunden sind.

Die in dieser Arbeit verwendete Support Vector Machine ist ein weiterer solcher Algorithmus und arbeitet mit der Position der Muster im Parameterraum. Er legt eine Trennebene zwischen die Trainingsmuster, daher kann er lediglich zwischen zwei Klassen unterscheiden. Ein großer Vorteil ist jedoch die Geschwindigkeit. Er steigt lediglich linear mit der Anzahl der Parameter der Samples.

### 1.3 AutoOpti

Durch die hohen Ansprüche in Bezug auf Sparsamkeit, Leistung und Umweltbelastung an moderne Triebwerke ist deren Optimierung ein Hauptaufgabengebiet des Instituts für Antriebstechnik. In der Abteilung Fan und Verdichter optimiert man hierfür in erster Linie die Aerodynamik der Schaufeln. Zum Einsatz kommt dabei der institutseigene automatisierte Optimierer AutoOpti. Bei AutoOpti handelt es sich um einen evolutionären Algorithmus, der kommende Generationen anhand einer Zielfunktion aus den vorherigen Generation ableitet. Dabei wird versucht, eine oder mehrere Zielfunktionen zu maximieren/minimieren. "Da die aerodynamische Auslegung von Triebwerkskomponenten immer ein Kompromiss zwischen den unterschiedlichsten Anforderungen darstellt, wurde bei der Entwicklung von AutoOpti besonderer Wert auf die Möglichkeit zur simultanen Optimierung mehrerer Zielfunktionen gelegt." [Sch13].

Der Optimierungsalgorithmus kann, um ein Optimum zu erreichen, oft auf mehrere Hundert freie Parameter zugreifen. Dabei handelt es sich meist um Angaben die Einfluss auf die Schaufelgeometrien haben, z.B. die Dicke oder Länge. Ein Satz an Parametern bezeichnet man als Member.

Die zu optimierende Eigenschaft wird Zielfunktion genannt. Das können verschiedene Größen

## 1 Einleitung

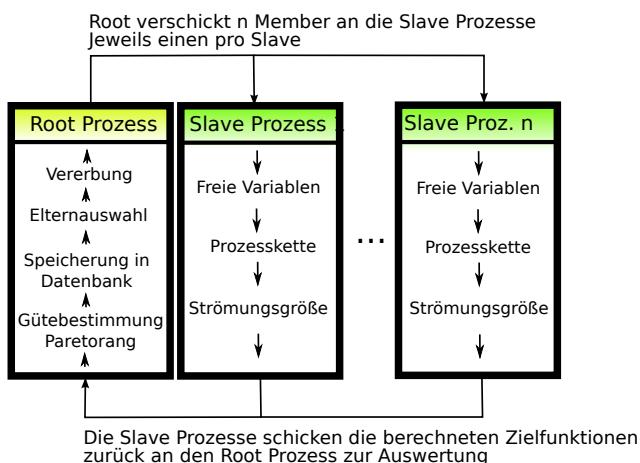
---

wie beispielsweise der Wirkungsgrad oder Spritverbrauch sein. In AutoOpti geschieht die Berechnung der Zielfunktion der Member anhand einer ganzen Reihe von Prozessen. Um eine hohe Flexibilität beizubehalten muss diese von Benutzer definiert werden. Dort kann der Nutzer die Programme und deren Verwendung einstellen.

In den meisten Fällen beinhaltet die Prozesskette:

- Die Geometrieerzeugung
- Die Erzeugung des Rechennetzes
- Durchführung der Strömungssimulation, zum Beispiel mit TRACE
- Durchführung der mechanischen Simulation (Calculix)
- Auswertung der Simulation

AutoOpti wird meist auf einem Rechencluster ausgeführt. Um eine effektive Parallelisierung zu erreichen, wird zwischen einem Root-Prozess und mehreren Slave-Prozessen unterschieden. Auf dem Root-Prozess wird der eigentliche Optimierungsalgorithmus durchgeführt und die Slave-Prozesse koordiniert. Außerdem kümmert sich der Root-Prozess um die Verwaltung der Datenbank. Die Slave-Prozesse durchlaufen die Prozesskette und berechnen dadurch die Zielfunktionen sowie weitere Werte, anhand derer die Einhaltung von Restriktionen überprüft werden kann (vgl. Abb. 1).



**Abbildung 1:** Aufgabenverteilung zwischen Master- und Slaveprozessen *Bildquelle:[Sch13]*

Sind in der Datenbank bereits Member enthalten (z.B. durch zufällig bestimmte Parameter), ist der weitere Ablauf von AutoOpti:

## 1 Einleitung

---

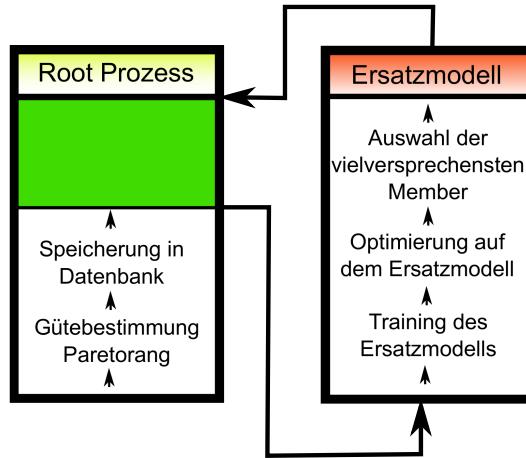
1. Auswählen einiger Member aus der Datenbank als Eltern.
2. Verwendung von Vererbungsoperatoren (Mutation, Kreuzung usw.) auf die Eltern. Dadurch wird ein neuer Satz an freien Variablen erzeugt. Dieser neue Member wird an einen Slave-Prozess geschickt.
3. Der Slave-Prozess durchläuft die Prozesskette zum Ermitteln der Zielfunktionen und Restriktionen.
4. Die berechnete Zielfunktionen und Restriktionen gehen zurück an den Root-Prozess. Dieser trägt ihn gemäß einem Gütwert in die Datenbank ein. Von hier aus kann er bei der Erzeugung neuer Member mitwirken.

Vor- und Nachteile[Sch13]

- + Die Suche nach dem Optimum wird global durchgeführt.
- + Die Zielfunktion muss nicht differenzierbar sein, was die Umsetzung dieser Strategie stark vereinfacht.
- + Die Zielfunktion muss nicht immer bewertbar sein, beispielsweise wenn die Prozesskette abbricht und dadurch kein Wert für die Zielfunktion ermittelt werden kann.
- + Die Strategie ist sehr gut parallelisierbar, z.B. durch die oben genannte Einteilung in Root und Slave Prozesse.
- Für jeden Member muss die gesamte, rechenintensive Prozesskette durchlaufen werden. Evolutionäre Algorithmen brauchen in der Regel jedoch viele Member um ein zufriedenstellendes Ergebnis zu erhalten.
- Bei einer nicht bewertbaren Zielfunktion wird die Information darüber verworfen. Theoretisch ist es möglich, dass der Algorithmus immer wieder in den gleichen Bereich geht, in dem die Member nicht konvergieren.

Um dem ersten Nachteil vorzubeugen wird ein auf den berechneten Membern basierendes Ersatzmodell verwendet. Dadurch lassen sich Vorhersagen über den Wert der Zielfunktion eines Members treffen, ohne die Prozesskette durchlaufen zu müssen (vgl. Abb. 2). Die Berechnung der Zielfunktion kann sonst durchaus mehrere Stunden bis Tage dauern. Das Ersatzmodell basiert dabei auf dem Kriging-Verfahren oder bayesisch trainierten neuronalen Netzen.

Der zweite Nachteil kann dadurch jedoch nicht behoben werden. Bricht die Prozesskette ab, z.B. da die Vernetzung aufgrund der gegebenen Parameter nicht durchgeführt werden kann,



**Abbildung 2:** Nutzung eines Ersatzmodells im Optimierungsprozess *Bildquelle:[Sch13]*

fließt diese Information nicht in das Ersatzmodell mit ein. Das die Prozesskette für einen Member abbricht ist keine Ausnahme. In den Parameterräumen gibt es immer große Bereiche, in denen das Berechnen der Geometrie oder des Rechennetzes nicht möglich ist oder die Strömungslösung fehlschlägt. Um diesen Nachteil zu beheben wird also eine Vorhersage für einen neuen Member benötigt wie wahrscheinlich die Prozesskette erfolgreich abgeschlossen wird.

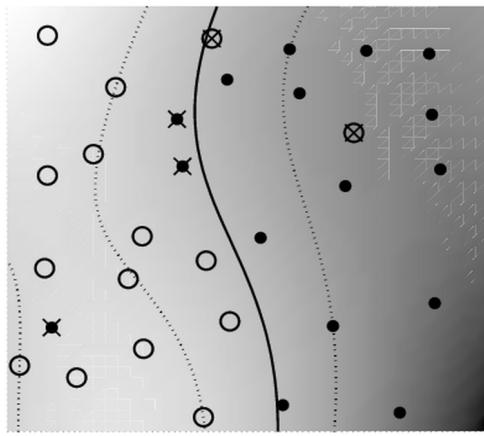
## 1.4 Einsatz eines Klassifikators in AutoOpti

Unabhängig von der Zielfunktion lassen sich die Parametersätze, die der Root-Prozess an die Slave-Prozesse übergibt, in zwei Klassen unterteilen: Die Prozesskette konnte erfolgreich beendet werden oder nicht. Die Aufgabe eines Klassifikators ist immer das Finden von Trends in Bereichen des Parameterraums (vgl. Abb. 3). Dadurch ließe sich für neue Member vorhersagen, ob es sich lohnen könnte, die Prozesskette durchzuführen, ehe diese überhaupt gestartet wurde. Man würde die Prozesskette nur starten wenn die Wahrscheinlichkeit, dass sie erfolgreich durchlaufen wird, über einem gewissen Wert liegt.

Eine besonderen Herausforderungen beim Einsatz eines Klassifikators in der hier vorliegenden Anwendung ist die große Zahl an freien Variablen. Der Aufwand sollte daher linear mit der Dimension des Parameterraums steigen. Das ist z.B. bei Support Vector Machines (SVM) der Fall; ist die Kernel-Matrix berechnet hat die Dimension des Parameterraums keinen Einfluss

## 1 Einleitung

---



**Abbildung 3:** Die Aufgabe eines Klassifikators ist das Erkennen von Trends anhand von Trainingspunkten mit bekannter Zugehörigkeit. Dadurch ist es möglich für neue Punkte Vorhersagen zu treffen. *Bildquelle:[SS02]*

mehr auf den Aufwand (vgl. Kapitel 2.1).

Eine weitere Herausforderung liegt darin, dass die Parameter sehr verschieden beschaffen sein können. Es handelt sich um verschiedene Größen (z.B Winkel, Entferungen oder dimensionslose Größen wie Verhältnisse). Außerdem erstrecken sich manche Parameter über mehrere Größenordnungen, andere hingegen haben sehr enge Grenzen. Diese Umstände machen ein Abstandsmaß das alle Parameter gleich behandelt wie es die meisten Kernel tun, zunächst unbrauchbar.

## 2 Theoretische Grundlagen

### 2.1 Support Vector Machines

Support Vector Machines bezeichnen ein binäres Klassifikationsverfahren und sind im Umfeld von maschinellem Lernen angesiedelt. Klassifikationsverfahren sind Modelle um Objekte Klassen zuzuordnen. Das Training dieser Modelle erfolgt durch einen Satz an Objekten, den Trainingsdaten, deren Klasse bekannt ist. Im Allgemeinen gilt: je mehr Trainingsdaten vorliegen, desto besser ist das Modell. Des Weiteren sollte darauf geachtet werden, dass die Trainingsdaten möglichst gut der späteren Anwendung entsprechen bzw. sogar einer realen Situation entnommen wurden. In dem späteren Anwendungsgebiet dieses Projektes sind die Trainingsdaten die Population des evolutionären Algorithmus.

Support Vector Machines sind eine vergleichsweise neue Methode und wurde in den 90er Jahren unter anderen von Vladimir N. Vapnik [Vap98] entwickelt. Die Idee hinter ihnen ist es, zwei Klassen durch eine Hyperebene zu trennen. Dabei wird der Ansatz des größten Abstands verfolgt: Es wird eine Ebene gesucht, die die Klassen trennt und den größtmöglichen Abstand (engl. margin) zu diesen hat.

#### 2.1.1 Lineare Support Vector Machines

Es liegen  $l$  Trainingsindividuen  $\{\mathbf{x}_i, y_i\}, i = 1, \dots, l$  vor. Hierbei gilt:  $\mathbf{x}_i \in \mathbb{R}^d$ .  $y_i \in \{-1, 1\}$  zeigt die entsprechende Klassenzugehörigkeit. Im Folgendem werden die Individuen meist als Member oder auch Sample bezeichnet.  $d$  ist die Dimension des Parameterraums, d. h. die Anzahl an frei wählbaren Parametern der Optimierung. Eine Hyperebene im Parameterraum  $\mathbb{R}^d$  kann wie gewohnt durch die Gleichung  $E : \langle \mathbf{w}, \mathbf{x} \rangle + b = 0$  dargestellt werden<sup>1</sup>. Dabei ist  $\mathbf{w}$  ein Normalenvektor der senkrecht zur Ebene steht,  $b$  stellt eine Verschiebung vom Ursprung dar.

---

<sup>1</sup>In der folgenden Arbeit stellen fette Buchstaben Vektoren dar,  $\langle \cdot, \cdot \rangle$  bezeichnet das Skalarprodukt

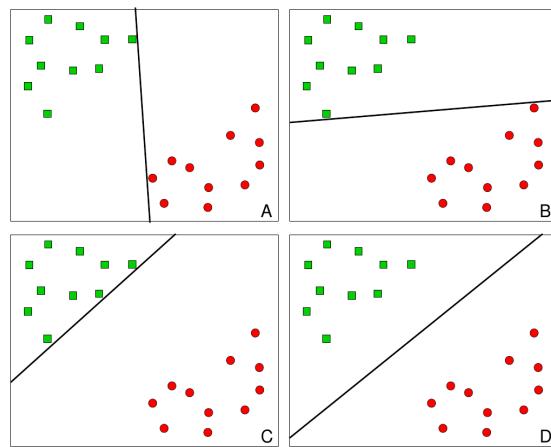
## 2 Theoretische Grundlagen

---

Die Gleichung

$$f(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \quad (2.1)$$

gibt an, auf welcher Seite der Ebene  $E(\mathbf{w}, b)$  der Punkt  $\mathbf{x}$  liegt. Gesucht ist eine Ebene, die die beiden Klassen voneinander trennt, sodass gilt  $f(\mathbf{x}_i) = y_i$ . Dadurch wird für jeden Punkt  $\mathbf{x}$  mit unbekannter Klasse eine Vorhersage  $y = f(\mathbf{x})$  möglich. Dabei trifft diejenige Ebene die besten Vorhersagen, die den größten minimalen Abstand (engl. margin) zu den Klassen hat [SS02] (vgl. Abb. 4). Das Finden dieser Hyperebene ist die Aufgabe des Trainings. Im Folgenden soll eine Übersicht über den Trainingsalgorithmus und dessen Herleitung gegeben werden. In der beschriebenen Art findet dieser sich auch in [CST00].



**Abbildung 4:** Auswahl der besten Trennebene: Die Ebene mit dem größten Abstand zu den Klassen eignet sich am besten zur Klassifizierung unbekannter Punkte  
Bildquelle:[Mar03]

Die Darstellung der Ebene mithilfe eines Normalenvektors ist nicht eindeutig. Die Multiplikation von  $\mathbf{w}$  und  $b$  mit einem Faktor  $c \neq 0$  verändert sie nicht.

$$\{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\} = \{\mathbf{x} | \langle c\mathbf{w}, \mathbf{x} \rangle + cb = 0\} \quad (2.2)$$

Wählt man  $c = \frac{1}{\|\mathbf{w}\|}$ , erhält man  $\|\mathbf{w}_E\| = 1$  und damit die Definition für den gerichteten Abstand eines Punktes/Trainingsmembers zu einer Ebene.

$$d = \langle \mathbf{w}_E, \mathbf{x}_i \rangle + b_E \quad (2.3)$$

## 2 Theoretische Grundlagen

---

Multipliziert man den gerichteten Abstand mit der Klassenzugehörigkeit  $y_i \in \{-1, +1\}$  des Trainingsmember ergibt sich

$$M = \min_{\mathbf{x}_i} y_i (\langle \mathbf{w}_E, \mathbf{x}_i \rangle + b_E) \quad (2.4)$$

und damit die Definition des minimalen gerichteten Abstandes der Trainingsmember zur trennenden Ebene. Negative Werte von  $M$  weisen auf eine Ebene hin, die die Klassen nicht korrekt voneinander trennt.

Den Wandabstand gilt es durch die Wahl von  $\mathbf{w}_E$  und  $b_E$  zu maximieren.

$$\begin{aligned} & \max_{\mathbf{w}_E, b_E, \|\mathbf{w}_E\|=1} M \\ & \text{unter der Bedingung } y_i (\langle \mathbf{w}_E, \mathbf{x}_i \rangle + b_E) \geq M, \quad \forall i \in \{1, \dots, l\} \end{aligned} \quad (2.5)$$

Wählt man  $\mathbf{w}$  so, das  $\|\mathbf{w}\| = \frac{1}{M}$  gilt, lässt sich das Maximierungs- als ein Minimierungsproblem darstellen. Teilt man beide Seiten der Bedingung durch  $M$  mit  $\frac{\mathbf{w}_E}{M} = \mathbf{w}_E \cdot \|\mathbf{w}\| = \mathbf{w}$  und  $\frac{b_E}{M} = b_E \|\mathbf{w}\| = b$  wird die Normbedingung( $\|\mathbf{w}_E\| = 1$ ) eliminiert und das Minimierungsproblem lautet

$$\begin{aligned} & \min_{\mathbf{w}} \|\mathbf{w}\| \quad \text{bzw.} \quad \min_{\mathbf{w}} \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle \\ & \text{unter der Bedingung } y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \quad \forall i \in \{1, \dots, l\} \end{aligned} \quad (2.6)$$

Optimierungsaufgaben lassen sich in ihre so genannte duale Form wandeln. Dabei handelt es sich lediglich um eine andere Darstellungsweise. Manchmal lassen sich aus dieser Form jedoch Vorteile ziehen.

Zur Überführung von Gleichung 2.6 in seine duale Form bietet sich hier der Einsatz von Lagrange-Multiplikatoren  $\alpha_i \quad \forall i \in \{1, \dots, l\}$  an. Daraus folgt die Lagrange-Funktion

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle - \sum_{i=1}^l \alpha_i [y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1] \quad (2.7)$$

Da es sich bei der Nebenbedingung um eine Ungleichung handelt, muss zusätzlich  $\alpha_i >$

## 2 Theoretische Grundlagen

---

$0 \quad i = 1, \dots, l$  gelten [Har08]. Gesucht ist die Minimierung von  $L$  in Bezug auf  $\mathbf{w}, b$  und die Maximierung in Bezug auf  $\boldsymbol{\alpha}$ .

An der gesuchten Stelle ist der Gradient  $\nabla L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0$

$\nabla_{\mathbf{w}} L = 0$  zeigt

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^l \alpha_i y_i \mathbf{x}_i = 0 \rightarrow \mathbf{w} = \sum_{i=1}^l y_i \alpha_i \mathbf{x}_i \quad (2.8)$$

aus  $\nabla_b L = 0$  folgt direkt

$$\nabla_b L = \sum_{i=1}^l \alpha_i y_i = 0 \quad (2.9)$$

Multipliziert man die Lagrange-Funktion zunächst aus und setzt dann Gleichung 2.8 ein, ergibt sich daraus

$$W(\boldsymbol{\alpha}) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + \sum_{i=1}^l \alpha_i - \sum_{i=1}^l \alpha_i y_i \langle \mathbf{w}, \mathbf{x}_i \rangle + \sum_{i=1}^l \alpha_i y_i b$$

$$\begin{aligned} W(\boldsymbol{\alpha}) &= \frac{1}{2} \left\langle \sum_{i=1}^l (y_i \alpha_i \mathbf{x}_i), \sum_{j=1}^l (y_j \alpha_j \mathbf{x}_j) \right\rangle - \sum_{i=1}^l \left( \alpha_i y_i \left\langle \mathbf{x}_i, \sum_{j=1}^l (y_j \alpha_j \mathbf{x}_j) \right\rangle \right) \\ &\quad + \sum_{i=1}^l \alpha_i + b \underbrace{\sum_{i=1}^l \alpha_i y_i}_{=0} \quad (2.10) \end{aligned}$$

Da bei dem Skalarprodukt das Distributivgesetz  $\langle (\mathbf{a} + \mathbf{b}), \mathbf{c} \rangle = \langle \mathbf{a}, \mathbf{c} \rangle + \langle \mathbf{b}, \mathbf{c} \rangle$  gilt, gilt auch

$$\langle \mathbf{a}, \sum_i \mathbf{b}_i \rangle = \sum_i \langle \mathbf{a}, \mathbf{b}_i \rangle \text{ und } \langle \sum_i \mathbf{a}_i, \sum_j \mathbf{b}_j \rangle = \sum_i \sum_j \langle \mathbf{a}_i, \mathbf{b}_j \rangle$$

Dadurch lässt sich Gleichung 2.10 schreiben als

$$W(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i,j=1}^l \langle \alpha_i y_i \mathbf{x}_i, \alpha_j y_j \mathbf{x}_j \rangle - \sum_{i,j=1}^l \langle \alpha_i y_i \mathbf{x}_i, \alpha_j y_j \mathbf{x}_j \rangle + \sum_{i=1}^l \alpha_i \quad (2.11)$$

Fasst man die ersten beiden Terme zusammen und beachtet man die Homogenität des Skalarproduktes ( $\langle \mathbf{a}r, \mathbf{b} \rangle = r\langle \mathbf{a}, \mathbf{b} \rangle$  mit  $r \in \mathbb{R}$ ) folgt daraus

## 2 Theoretische Grundlagen

---

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i,j=1}^l \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (2.12)$$

was unter der Berücksichtigung von  $\alpha_i \geq 0$  und  $\sum_i \alpha_i y_i = 0$  zu maximieren ist.

Die Umwandlung der ursprünglichen Optimierungsaufgabe 2.6 in die duale Form von 2.12 mag zunächst verwundern. Es ist lediglich eine andere Darstellungsweise; ist eine Lösung gefunden ist auch die andere Optimierung gelöst. Zu beachten ist jedoch, dass  $\mathbf{w}$  eliminiert wurde und der Parameterraum  $\mathbb{R}^d$  lediglich innerhalb eines Skalarproduktes auftaucht. Diese Eigenschaften kommen bei den nicht-linearen Support Vector Machine (SVM) (vgl. Kapitel 2.1.2) noch zum Einsatz und sind nötig für den Einsatz des so genannten Kernel-Tricks.

In der Matrixschreibweise verdeutlicht sich die quadratische Natur der Optimierungsaufgabe

$$\min W(\boldsymbol{\alpha}) = \mathbf{1}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \quad (2.13)$$

sodass:

$$\mathbf{y}^T \boldsymbol{\alpha} = 0$$

$$\alpha_i \geq 0 \quad \forall i$$

dabei gilt:

$$\mathbf{K}_{ij} = y_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle y_j$$

$$\mathbf{1}^T = (1 \ 1 \ \dots \ 1)$$

Algorithmen für Optimierungen dieser Form fasst man unter dem Begriff *Quadratic Programming* zusammen. Wie effektiv diese arbeiten beeinflusst maßgeblich den Rechenaufwand. Der in diesem Programm verwendete Ansatz ist in Kapitel 2.2 beschrieben.

Sind die idealen  $\alpha_i$  gefunden, muss  $b$  bestimmt werden.  $\forall i$  gilt die Kuhn-Tucker Bedingung [Har08, S. 53] für Ungleichungen als Nebenbedingungen.

$$\alpha_i [y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1] = 0 \quad (2.14)$$

Ist  $\alpha_i = 0$  spricht man von einer inaktiven Nebenbedingung; In dem Fall der Support Vector Machine entspricht das einem Punkt dessen Abstand zur Trennebene größer ist als die Margin. Ist  $\alpha_i > 0$  folgt aus der Bedingung 2.14 jedoch das dann gelten muss:

$$\langle \mathbf{w}, \mathbf{x}_i \rangle + b = y_i, \quad y_i \in \{-1, +1\} \quad (2.15)$$

Punkte, die diese Gleichung erfüllen, nennt man Stützvektoren (engl. Support Vectors). Nur diese haben Einfluss auf die Hyperebene. Alle anderen mit  $\alpha_i = 0$  lassen sich entfernen, ohne das sich die Ebene verändert.

Um  $b$  zu ermitteln genügt es mathematisch gesehen, die Parameter eines einzigen Stützvektors  $\mathbf{x}_i$  in Gleichung 2.15 einzusetzen. Um die numerische Stabilität zu erhöhen wird in der Regel ein gewichteter Mittelwert von alle Stützvektoren benutzt [PTVF07, S.887]. Dazu werden die oben gefundenen  $\alpha_1, \dots, \alpha_l$  genutzt; ein großes  $\alpha_i$  eines Stützvektors erhöht seine Gewichtung.

$$b = \frac{\sum_{i=1}^l \alpha_i (y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle)}{\sum_{i=1}^l \alpha_i} \quad (2.16)$$

$$= \frac{\sum_{i=1}^l \alpha_i \left( y_i - \sum_{j=1}^l \alpha_j y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle \right)}{\sum_{i=1}^l \alpha_i} \quad (2.17)$$

Ist  $b$  gefunden, lässt sich eine Entscheidungsfunktion bilden:

$$f(\mathbf{x}) = \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b) = \text{sgn} \left( \sum_{i=1}^l \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b \right) \quad (2.18)$$

Mit der Bestimmung von  $\alpha$  und  $\beta$  ist die Erstellung einer Support Vector Machine abgeschlossen. Die Entscheidungsfunktion kann für jeden beliebige Punkt im Parameterraum einer Klasse vorhersagen.

### 2.1.2 Nicht lineare Support Vector Machines und Kernelfunktionen

In vielen Anwendungsfällen lassen sich Klassen nicht linear trennen. Um Klassen nicht nur Anhand einer Ebene trennen zu können, kann man die Trainingsdaten zunächst in einen höher dimensionalen Raum transformieren. Dieser Raum wird *feature-Space*  $H$  (deut. Merkmalsraum) genannt.  $\Phi : \mathbb{R}^d \rightarrow H$  ist die entsprechende Transformation. Im *feature-Space* werden die Klassen zwar weiterhin nur von einer Ebene getrennt, durch die nicht-lineare Transformation  $\Phi$  lassen sich aber nun auch Fälle, die im Parameterraum  $\mathbb{R}^d$  nicht linear separierbar sind, im *feature-Space*  $H$  trennen (vgl. Abb. 5).

Beachtet man die Beobachtung, dass im linearen SVM-Algorithmus die Trainingsdaten lediglich innerhalb von Skalarprodukten auftreten, lässt das eine wichtige Schlussfolgerung zu: Die Transformation  $\Phi$  selber kann vernachlässigt werden, lediglich das Skalarprodukt innerhalb des *feature-Spaces* muss bekannt sein.

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

Eine Funktion  $K(\mathbf{x}_i, \mathbf{x}_j)$  wird als Kernel-Funktion bezeichnet. Oft hängt die Kernelfunktion zusätzlich von weiteren Parametern ab. Die Wahl der Kernelfunktion und deren Parameter hat erheblichen Einfluss auf die Leistung der Support Vector Machine. Die Kernelfunktion ist ein Maß der Ähnlichkeit, jede Funktion  $K(x_i, x_j)$  kann als Kernelfunktion verwendet werden, wenn für sie gilt: Die Matrix  $\mathbf{K}$  mit  $K_{i,j} = K(x_i, x_j)$  ist immer positiv semidefinit.

Neben dem Skalarprodukt werden jedoch vor allem drei Kernelfunktionen genutzt:

**Polynom-Kurven d. Grades:**  $K(\mathbf{x}_i, \mathbf{x}_j) = (r + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^d$

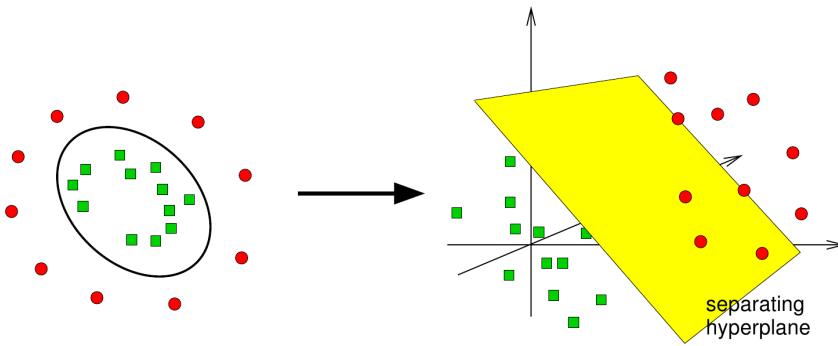
**Radial Basis Funktion (RBF):**  $K(\mathbf{x}_i, \mathbf{x}_j) = e^{\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$

**Neuronales Netz:**  $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa_1 \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \kappa_2)$

Auch für die Berechnung von  $b$  muss lediglich das Skalarprodukt  $\langle \cdot, \cdot \rangle$  durch  $K(\cdot, \cdot)$  ersetzt werden. Es kann weiterhin die Gleichung 2.17 genutzt werden. Das gleiche gilt auch für die Entscheidungsfunktion 2.18.

### 2.1.3 Soft Margin SVM

Durch die Wahl eines geeigneten Kernels ließe sich zwar jede Menge an Trainingsdaten sauber trennen. Für das Erkennen von Mustern ist dies jedoch oft unerwünscht. So führt es



**Abbildung 5:** Oft lassen sich Klassen im eigentlichen Parameterraum nicht linear trennen. Eine geeignete, nicht lineare Transformation in einen höher-dimensionalen Raum kann dies jedoch ermöglichen. Bildquelle:[Mar03]

in Anwendungen, die mit großem Jitter behaftet sind, zum so genannten Overfitting (vgl. Kapitel 2.1.3).

Der Trainingsalgorithmus, wie er in Abschnitt 2.1.1 beschrieben ist, lässt keinen Trainingspunkt mit  $y_i \neq f(\mathbf{x}_i)$  zu. Sind Muster der Trainingsdaten jedoch in einem Gebiet, das ansonsten von Mustern der anderen Klasse dominiert wird, wäre ein Algorithmus besser geeignet der diese Ausreißer erkennt und zulässt. Durch das Einführen eines Strafterms, der solche Fehlerkennungen zulässt, jedoch bestraft, lässt sich ein besseres Abstraktionsvermögen erreichen. Solche Support Vector Machines nennt man auch *Soft Margin-SVM*. Das Minimierungsproblem verändert sich zu:

$$\min_{\mathbf{w}, b} \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{i=1}^l \xi_i$$

$$\text{unter der Bedingung } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad \forall i \quad (2.19)$$

und    $\xi_i \geq 0, \quad \forall i$

Für jeden Trainingspunkt wird ein  $\xi_i$  eingesetzt. Ist der Punkt auf dem Rand der Margin oder auf der richtigen Seite gilt  $\xi_i = 0$ . Ist er jedoch auf der falschen Seite der Trennebene oder nicht weit genug entfernt, muss  $\xi_i > 0$  gelten damit die Bedingung trotzdem erfüllt werden kann. Solche Punkte werden in dem Bestrafungsterm mit dem Faktor  $C$  zur Minimierungsaufgabe

## 2 Theoretische Grundlagen

---

addiert. Die Größe von  $C$  ist hierbei vom Nutzer bestimmt. Dieser kann damit Einfluss auf die Größe der Bestrafung nehmen.

Der verwendete Ansatz ist ähnlich dem des separierbaren Falles.  $\mu_i$  ist dabei der Lagrange-Multiplikator, der sich auf die zusätzliche Nebenbedingung  $\xi_i \geq 0$  bezieht.

$$L(\boldsymbol{\alpha}, \mathbf{w}, b, \boldsymbol{\mu}, \boldsymbol{\xi}) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{i=1}^l \xi_i - \sum_{i=0}^l \alpha_i [y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i] - \sum_{i=1}^l \mu_i \xi_i \quad (2.20)$$

Auch hier gilt

$$\nabla_{\mathbf{w}} L = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^l \alpha_i y_i \mathbf{x}_i$$

und

$$\nabla_b L = 0 \Rightarrow \sum_{i=1}^l \alpha_i y_i = 0$$

Weiterhin gilt es jetzt jedoch auch  $\nabla_{\xi_i} L = 0$  zu berücksichtigen.

$$\nabla_{\xi_i} L = C - \alpha_i - \mu_i = 0 \Rightarrow C = \alpha_i + \mu_i \quad (2.21)$$

Zusätzlich muss wieder  $\alpha_i \geq 0$  gelten. Da auch  $\boldsymbol{\mu}$  sich auf ein Ungleichung bezieht gilt hier das gleiche;  $\mu_i \geq 0$  muss erfüllt sein.

Aus diesen beiden Bedingungen und Gleichung 2.21 lässt sich auf eine Obergrenze für  $\alpha_i$  schließen. Damit 2.21 überhaupt erfüllt werden kann, muss

$$\alpha_i \leq C \quad (2.22)$$

gelten.

Um  $\xi_i$  aus dem Optimierungsproblem zu eliminieren genügt es,  $L(\boldsymbol{\alpha}, \mathbf{w}, b, \boldsymbol{\mu}, \boldsymbol{\xi})$  ein wenig anders darzustellen und  $C - \alpha_i - \mu_i = 0$  einzusetzen.

$$L(\boldsymbol{\alpha}, \mathbf{w}, b, \boldsymbol{\mu}, \boldsymbol{\xi}) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle - \sum_{i=1}^l \alpha_i [y_i \langle \mathbf{x}_i, \mathbf{w} \rangle + b - 1] + \sum_{i=1}^l \xi_i \underbrace{(C - \alpha_i - \mu_i)}_{=0} \quad (2.23)$$

Es zeigt sich, dass sich die Optimierungsaufgabe für eine Soft Margin SVM fast nicht ändert.

Lediglich die Restriktion  $\alpha_i \leq C$  kommt hinzu.

$$\min_{\alpha} W(\alpha) = \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T \mathbf{K} \alpha \quad (2.24)$$

sodass

$$\mathbf{y}^T \alpha = 0 \quad (2.26)$$

$$0 \leq \alpha_i \leq C \quad \forall i \in \{1, \dots, l\} \quad (2.27)$$

(2.28)

### Over/Underfitting

Durch den Faktor  $C$  des Bestrafungsterms aus Gleichung 2.19 lässt sich das Verhalten des Trainingsalgorithmus ändern. Für  $C \rightarrow 0$  haben Trainingsmuster auf der falschen Seite kaum Einfluss. Das trainierte Modell ist nicht komplex genug, um den zugrunde liegenden Sachverhalt ausreichend darzustellen. Man spricht vom so genannten Underfitting.

$C \rightarrow \infty$  bewirkt ein Verhalten wie in den ursprünglich beschriebenen SVMs. Sind die Klassen tatsächlich separierbar, mag ein solches Verhalten gewünscht sein. Bei einer Überschneidung der Klassen, z.B. durch Rauschen, führt ein solcher Trainingsalgorithmus zum so genannten Overfitting. Das heißt es werden zwar alle Trainingsdaten korrekt wiedergegeben, die Generalisierungsfähigkeit leidet jedoch darunter.

Das Beste Ergebnis erzielt man in der Regel mit einem Kompromiss zwischen diesen beiden Extremen. Für das finden des idealen  $C$  kann zum Beispiel die Kreuzvalidierung eingesetzt werden (vgl. Abb. 6).

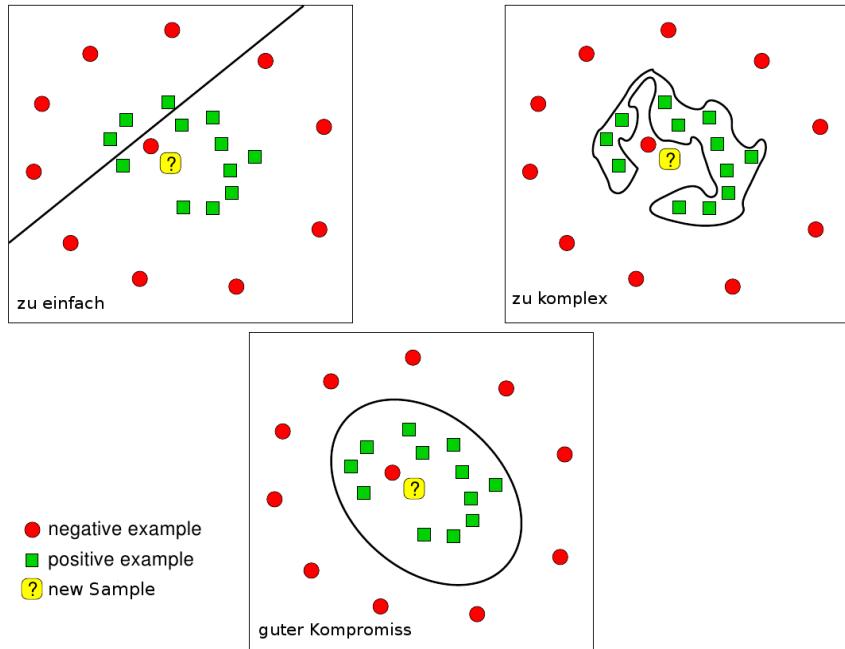
#### 2.1.4 Wahrscheinlichkeitsabschätzung

Der Standard Algorithmus einer SVM lässt zunächst keinen Rückschluss darauf zu, mit welcher Wahrscheinlichkeit die ermittelte Klasse der tatsächlichen entspricht. In der Anwendung in AutoOpti ließe sich eine solche Wahrscheinlichkeit sinnvoll einbinden.

In der Arbeit von Platt [Pla99] wird eine Methode beschrieben, in der der Algorithmus zunächst normal durchgeführt wird, um eine Entscheidungsfunktion der Form  $g(\mathbf{x}) = \text{sgn}(f(\mathbf{x}))$  zu erhalten.  $f(\mathbf{x})$  bezeichnet in diesem Kontext den gerichteten Abstand zur Hyperebene.

## 2 Theoretische Grundlagen

---



**Abbildung 6:** Verschieden große C hat verschiedene Eigenschaften der SVM zufolge. (l.o.: kleines C, r.o.: großes C)  
Die besten Ergebnisse liefert ein Kompromiss zwischen den beiden Extremfällen(unten) *Bildquelle:[Mar03]*

Platt zeigt, dass die gesuchte Wahrscheinlichkeit mithilfe der Sigmoidfunktion aus  $f(\mathbf{x})$  berechnet werden kann:

$$P(y = 1|f) = \frac{1}{1 + e^{Af+B}} \quad (2.29)$$

Die Parameter  $A$  und  $B$  müssen zunächst über einen Maximum-Likelihood Ansatz bestimmt werden. Hierbei wird eine Fehlerfunktion minimiert. Statt  $y_i \in \{-1, +1\}$  wird hierbei  $t_i = \frac{y_i+1}{2} \in \{0, 1\}$  verwendet.

$$\min - \sum_{i=1}^l (t_i \log(p_i) + (1 - t_i) \log(1 - p_i)) \quad (2.30)$$

Dabei ist  $p_i = \frac{1}{1+e^{Af_i+B}}$ . Benutzt man für das Finden von  $A$  und  $B$  Member, die bereits

beim Berechnen der SVM zum Einsatz kam, kann es zu einer Verzerrung kommen (ähnlich dem Testen). Daher sollte hier entweder ein vorher entnommener Teil benutzt werden oder die  $f_i$  durch Kreuzvalidierung erstellt werden (vgl. Kapitel 2.1.5).

Platt zeigt, dass mit dieser Methode gute Abschätzungen für die tatsächliche Wahrscheinlichkeit von Membern geliefert werden können[Pla99].

Der Aspekt der Wahrscheinlichkeit ist in diese Arbeit noch nicht eingeflossen. Es ist aber denkbar, das dies in absehbarer Zeit Teil des Projektes wird.

### 2.1.5 Güte eines Klassifikators

Das Messen der Güte verschiedener Kernelfunktionen mit verschiedenen Einstellungen ist eine zentrale Aufgabenstellung dieser Arbeit. Daher soll in diesem Abschnitt erläutert werden welche Methoden zum ermitteln der Kennzahlen es gibt und welche hier Verwendung finden.

Das Hauptmerkmal zur Beschreibung der Güte einer SVM ist die Genauigkeit der Vorhersagen. Bei der Feststellung von Klassenzugehörigkeiten gibt es zunächst lediglich korrekt/inkorrekt, daher muss man diese Genauigkeit als mittlere Fehlerquote verstehen.

Die Fehlerquote  $T$  lässt sich ausdrücken als

$$T = \frac{\text{falscheTests}}{\text{AnzahlTests}} = \frac{1}{l} \sum_{i=1}^l \Theta(-y_i f(\mathbf{x}_i)) \quad (2.31)$$

Hierbei ist  $\Theta$  die Sprungfunktion.

$$\Theta(x) : \mathbb{R} \rightarrow \{0, 1\}$$

$$\Theta(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Je mehr Samples getestet werden, desto näher liegt dieser Ausdruck an der tatsächlichen Fehlerquote. Bei der Auswahl der Testsamples muss man jedoch beachten, dass es sich hier nicht um Trainingssamples handeln sollte. Das würde zu einer Verzerrung der Ergebnisse führen. Overfitting (vgl. Kapitel 2.1.3) würde bei einem solchen Test nicht erkannt werden. Des Weiteren sollten die Test-, wie auch die Trainingsdaten, die spätere Anwendung möglichst gut widerspiegeln.

Ein gängiges Mittel zur Durchführung solcher Tests ist die Kreuzvalidierung. Dabei wird

## 2 Theoretische Grundlagen

---

ein Datensatz in  $k$  möglichst gleich große Teilmengen ( $T_1 \dots T_k$ ) unterteilt. Anschließend werden  $k$ -Testdurchläufe durchgeführt, in denen jeweils  $\{T_i\}$  die Test- und  $\{T_1 \dots T_k\} \setminus \{T_i\}$  die Trainingsdaten sind. Bei dieser Art spricht man auch von der  $k$ -fachen Kreuzvalidierung ( $k$ -fold Cross Validation) [HTF09, S. 241].

Für die Durchführung der Performance-Tests wurde eine Kreuzvalidierung durchgeführt (vgl. Kapitel 2.1.5). Im Folgenden werden die Teilmengen, die in einer Iteration Teil des Trainings sind, als Trainingsmember bezeichnet. Die Testdaten werden Testmember genannt. Nach den  $k$  Durchläufen war dadurch jeder Member der Datenbank 1 mal Testmember und  $k - 1$  mal Trainingsmember.

### Leave-One-Out Estimate

Ist  $k = l$  (Anzahl der Trainingssamples) spricht man dabei von dem *Leave-One-Out (LOO) Estimator*.

In diesem Fall wird für jedes Sample untersucht, ob es richtig kategorisiert wird, wenn es selber kein Element der Trainingsmenge ist. Die neue Trainingsmenge besteht aus allen anderen Samples der ursprünglichen Trainingsmenge. Der *LOO Estimator*  $T$  entspricht sehr gut dem tatsächlichen Erwartungswert der Fehlerquote. Die einzige Verfälschung röhrt daher, das die Trainingsmengen jeweils eine Größe  $l - 1$  statt  $l$  hat. Bei großen Trainingsmengen kann dies jedoch vernachlässigt werden.

### Performance

Ein weiteres Qualitätsmerkmal ist die Laufzeit bei großen Trainingsdatensätzen. Der Aufwand der Klassifizierung eines unbekannten Samples durch eine Bereits trainierte SVM ist überschaubar, das Hauptaugenmerk liegt daher auf dem Trainingsalgorithmus.

Die zentrale Aufgabe des Trainingsalgorithmus ist die Minimierung der Gleichung 2.13. Wie effektiv die Lösung gefunden wird beeinflusst maßgeblich die benötigte Rechenzeit. Da die möglichst effektive Implementierung dieser Optimierung ein Schwerpunkt dieser Arbeit war, ist der mathematische Hintergrund in Kapitel 2.2 näher erläutert.

### 2.1.6 Kriterium für die Auswahl geeigneter Kernelfunktion

Um die Qualität zwischen verschiedenen Kernelfunktionen oder Kernelfunktionen mit verschiedenen Parametern vergleichen zu können bedarf es einer Form, die Qualität zu beschreiben.

Der *LOO Estimator* ist die genaueste Aussage die auf Basis der Trainingsmenge getroffen werden kann, wie zuverlässig die Vorhersage der SVM in der Anwendung sein wird. Der große Nachteil des *LOO Estimator* ist jedoch die kostenintensive Berechnung. Der Trainingsalgorithmus muss für jeden Punkt einmal durchgeführt werden. Vereinfachend kann man bei SVM beachten das nicht-Support-Vektoren ( $\alpha_i = 0$ ) keinen Einfluss auf die Entscheidungsfunktion haben, also auch nach einer Entfernung noch korrekt klassifiziert werden. Je nach eingesetzter Kernelfunktion ist der Anteil der Support Vektoren jedoch meist so groß, dass dies kaum eine Erleichterung darstellt.

Es gilt also ein leicht zu berechnenden Wert zu finden, der die Zuverlässigkeit widerspiegelt. Dazu lohnt sich eine Betrachtung der Aussage von Heinrich Niemann aus „Klassifikation von Mustern“: Als allgemeine Voraussetzung, um Klassen leicht voneinander trennen zu können, gelte:

„Die Merkmale bilden für Muster einer Klasse einen einigermaßen kompakten Bereich im Merkmalsraum. Die von Merkmalen verschiedener Klassen eingenommenen Bereiche sind einigermaßen getrennt. [...]“ [Nie83, S. 20]

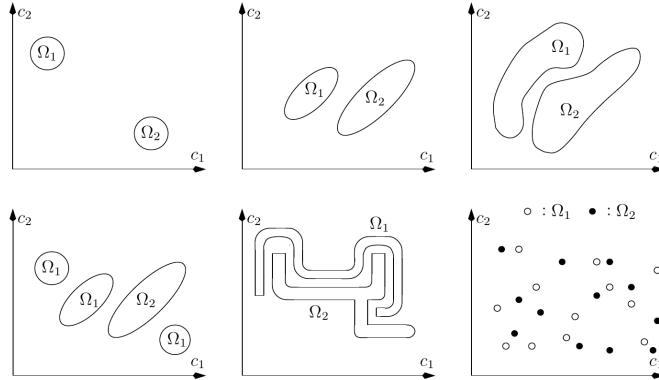
Verdeutlicht wird diese Aussage in Grafik 7. Die Aussage beschreibt das nicht alleine ein großer Abstand zueinander ausreicht, um Klassen gut voneinander trennen zu können. Darüber hinaus müssen sie auch ausreichend kompakt sein, d.h. Member einer Klasse müssen sich möglichst ähnlich sein. In einem Parameterraum spiegelt sich das darin wieder, dass eine Klasse nur über einen kleinen Raum verteilt ist.

Da man in dem Fall der nichtlinearen Support Vector Machine durch die Transformation  $\Phi : \mathbb{R} \rightarrow H$  bzw. der Kernelfunktion  $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$  direkt Einfluss auf die Position der Klassen im Merkmalsraum  $H$  hat, ist es sinnvoll,  $K$  so zu wählen, das die oben genannten Bedingungen möglichst gut erfüllt sind.

Vladimir Vapnik und Olivier Chapelle schlagen dazu vor, das Verhältnis zwischen dem kleinsten Radius der alle Member beinhaltet und dem gefundenen Wandabstand zwischen den Klassen als Maß zu verwenden[VC00]. In ihrer weiteren Arbeit „Choosing Mutiple Parameters for Support Vector Machines“[CVBM02] beweisen sie außerdem, dass es sich dabei um eine

## 2 Theoretische Grundlagen

---



**Abbildung 7:** Beispiele für zunehmend weniger kompakt werdende Bereiche im Merkmalsraum  
Bildquelle:[Nie83]

obere Schranke des LOO-Estimator  $T$  handelt.

$$T \leq \frac{1}{l} \frac{R^2}{M^2} = \frac{1}{l} R^2 \|\mathbf{w}\|^2 = \frac{1}{l} R^2 \left( \sum_{i,j=1}^l \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right) = \frac{1}{l} R^2 (\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}) \quad (2.32)$$

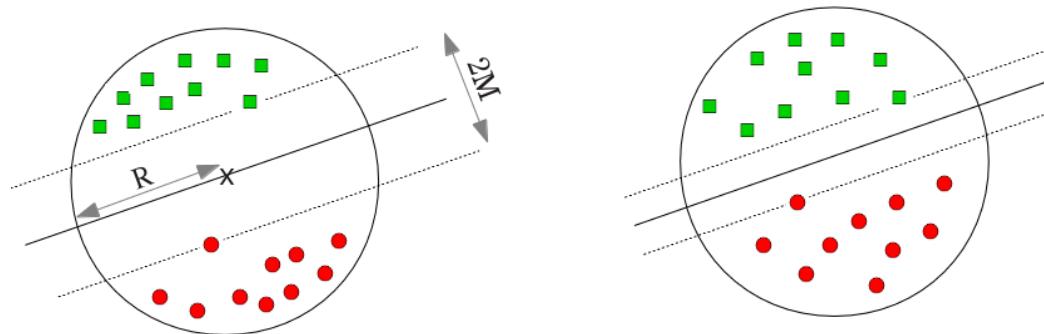
$R$  ist hier der Radius eines minimalen Kreises im *feature-Space*, der alle Trainingssample beinhaltet.

Bei der Ermittlung von  $R^2$  muss man berücksichtigen, das der *feature-Space* und die Transformation  $\Phi : \mathbb{R} \rightarrow H$  nicht bekannt sind. Jedoch lässt sich auch hier eine duale Form einsetzen, in der lediglich Kenntnis über das Skalarprodukt  $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$  vorausgesetzt wird. [Yil07] beschreibt, wie  $R^2$  gefunden werden kann durch:

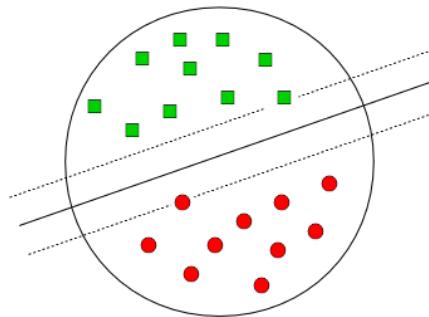
$$R^2 = \max_{\boldsymbol{\beta}} \sum_{i=1}^l \beta_i K(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^l \beta_i \beta_j K(\mathbf{x}_i, \mathbf{x}_j)$$

unter den Bedingungen  $\sum_{i=1}^l \beta_i = 1$  (2.33)

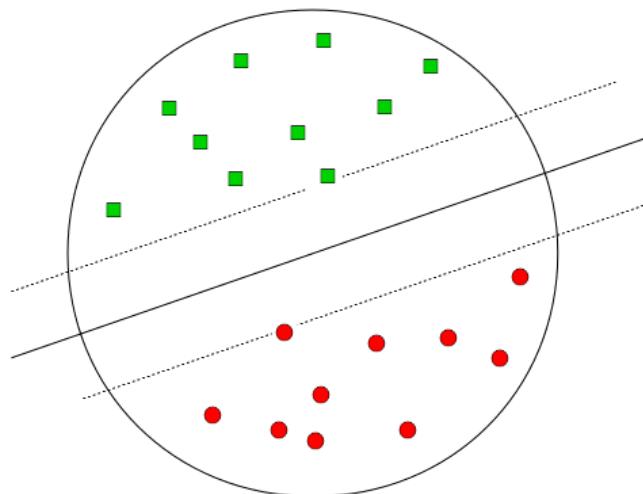
und  $\beta_i \geq 0$



(a) Kernel mit dem besten Margin-Radius-Verhältnis



(b) Gleich großer Radius, jedoch eine kleinere Margin



(c) Gleiche Margin, jedoch größerer Radius

**Abbildung 8:** Visuelle Darstellung des Qualitätskriterium für einen Kernel. Das Verhältnis zwischen Margin und Radius ist für die obere Schranke des LOO-Estimators ausschlaggebend. (a) wäre in diesem Beispiel die beste Wahl.

## 2 Theoretische Grundlagen

---

Mit dieser oberen Schranke liefern Vapnik und Chapelle ein gutes Kriterium für die Auswahl eines Kernels. Je größer die gefundene Margin im Verhältnis zum Radius ist, desto besser ist der genutzte Kernel geeignet. Dabei bezieht sich diese Auswahl nicht nur auf die verwendete Kernelart (vgl. Kapitel 2.1.2). Es ist auch möglich den Kernel anhand von Parametern „einzustellen“ und diese Einstellungen anhand der vorgestellten oberen Schranke aus 2.32 zu bewerten. Die Kernelparameter werden im Folgenden mit  $\theta$  bezeichnet<sup>2</sup>.

So wurde zum Beispiel für Kapitel 4 ein RBF-Kernel gewählt, der nicht über einen einzigen Parameter  $\gamma$  verfügt. Stattdessen verfügt die Funktion über den Vektor  $\theta$ .

$$K(\mathbf{x}, \tilde{\mathbf{x}}) = e^{\gamma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2} = e^{\gamma \sum_i (x_i - \tilde{x}_i)^2} \xrightarrow{\text{wird zu}} K_{\theta}(\mathbf{x}, \tilde{\mathbf{x}}) = e^{\sum_i \theta_i (x_i - \tilde{x}_i)^2} \quad (2.34)$$

$\theta_i$  verändert somit den Einfluss des  $i$ -ten Parameters auf das Gesamtmodell.

Abbildung 8 verdeutlicht, dass eine gleichmäßige Ausdehnung des feature-Space nicht ausreicht. Der Abstand der Klassen zueinander wird zwar vergrößert, der Radius aber in gleichem Maße. Es gilt die Parameter zu finden die einen kleinen Einfluss auf die Klassenzugehörigkeit der Trainingsdaten haben. Deren Einfluss kann auch im Modell gesenkt werden, ohne das die Margin schrumpft.

Üblicherweise werden die Kernelparameter für die Kalibrierung des Kernels vom Nutzer gewählt. Oft wird dabei ein Grid-Search verwendet. Dazu werden viele Einstellungen ausprobiert. Diejenige, die die besten Ergebnisse erzielt (z.B. in einer Kreuzvalidierung) wird von da an verwendet. Aufgrund des dadurch hohen Aufwands, den schon kleine Dimensionen des Suchraums mit sich bringen, wird dem Kernel dabei selten ein hoher Freiheitsgrad gestattet. Im Falle des RBF-Kernels handelt es sich dabei meist lediglich um  $\gamma$  (vgl. Gleichung 2.34, links).

Die Optimierung von  $\frac{R^2}{M^2}$  anhand von  $\theta$  in den SVM Algorithmus mit einzubeziehen verspricht eine Verbesserung der Vorhersage. Ist das Kriterium eine gute Abschätzung der Güte des Kernels, könnte so der für die Anwendung bestmögliche Kernel automatisiert gefunden werden. Da die Berechnung des Kriteriums einen Bruchteil des Aufwands der Kreuzvalidie-

---

<sup>2</sup>Analog dazu verfügen die Korrelationsfunktionen im Kriging über *Hyperparameter*. Aufgrund der großen Ähnlichkeit der Kernel- und Korrelationsfunktionen können sie in der Implementierung gemeinsam umgesetzt werden. Im Kapitel der Umsetzung 3 wird daher nicht zwischen Kernel- und Hyperparameter unterschiedet, meist wird die Bezeichnung Hyperparameter benutzt.

## 2 Theoretische Grundlagen

---

rung darstellt, ist es nun auch denkbar, eine komplexe<sup>3</sup> Kernfunktion mit einem größeren Freiheitsgrad zu nutzen. So erst wird es möglich statt des einfachen  $\gamma$  in dem Kernel einen Vektor  $\boldsymbol{\theta} \in \mathbb{R}^d$  einzusetzen.

Da die Kernfunktion  $K(\mathbf{x}_i, \mathbf{x}_j)$  nun auch von  $\boldsymbol{\theta}$  abhängt wird durch die neue Schreibweise  $K_{\boldsymbol{\theta}}(\mathbf{x}_i, \mathbf{x}_j)$  gekennzeichnet. Analog dazu wird für die Kernelmatrix mit  $\mathbf{K}_{\boldsymbol{\theta}}$  verwendet.

Ob diese Einbeziehung möglich und sinnvoll ist eine der zentralen Fragestellungen dieser Arbeit.

### Differenzierung des Terms $R^2 \|\mathbf{w}\|^2$

Um die Konvergenz bei der Minimierung von  $T^2$  zu beschleunigen soll der Gradient zum Einsatz kommen. Dies ist nur möglich, wenn man  $\boldsymbol{\alpha}$  und  $\boldsymbol{\beta}$  als konstant betrachtet.

$$\frac{\partial T}{\partial \theta_p} \Big|_{\alpha, \beta \text{ konst.}} = \frac{\partial R^2 \|\mathbf{w}\|^2}{\partial \theta_p} = \underbrace{\frac{\partial R^2}{\partial \theta_p}}_{\text{III}} \cdot \underbrace{\|\mathbf{w}\|^2}_{\text{I}} + \underbrace{R^2}_{\text{II}} \cdot \underbrace{\frac{\partial \|\mathbf{w}\|^2}{\partial \theta_p}}_{\text{IV}} \quad (2.35)$$

$$\|\mathbf{w}\|^2 = 2 \sum_{i=1}^l \alpha_i - \sum_{i,j=1}^l \alpha_i \alpha_j y_i y_j \mathbf{K}_{\boldsymbol{\theta}}{}_{i,j} = 2 \cdot \mathbf{1}^T \boldsymbol{\alpha} - \boldsymbol{\alpha}^T \mathbf{K}_{\boldsymbol{\theta}}^y \boldsymbol{\alpha} \quad (\text{I})$$

$$R^2 = \sum_{i=1}^l \beta_i \mathbf{K}_{\boldsymbol{\theta}}{}_{i,i} - \sum_{i,j=1}^l \beta_i \beta_j \mathbf{K}_{\boldsymbol{\theta}}{}_{i,j} = \underbrace{\mathbf{k}^T}_{\text{Diagonale von } \mathbf{K}} \boldsymbol{\beta} - \boldsymbol{\beta}^T \mathbf{K}_{\boldsymbol{\theta}} \boldsymbol{\beta} \quad (\text{II})$$

$$\frac{\partial R^2}{\partial \theta_p} = \sum_{i=1}^l \beta_i \frac{\partial \mathbf{K}_{\boldsymbol{\theta}}{}_{i,i}}{\partial \theta_p} - \sum_{i,j=1}^l \beta_i \beta_j \frac{\partial \mathbf{K}_{\boldsymbol{\theta}}{}_{i,j}}{\partial \theta_p} = \frac{\partial \mathbf{k}^T}{\partial \theta_p} \boldsymbol{\beta} - \boldsymbol{\beta}^T \frac{\partial \mathbf{K}_{\boldsymbol{\theta}}}{\partial \theta_p} \boldsymbol{\beta} \quad (\text{III})$$

$$\frac{\partial \|\mathbf{w}\|^2}{\partial \theta_p} = - \sum_{i,j=1}^l \alpha_i \alpha_j y_i y_j \frac{\partial \mathbf{K}_{\boldsymbol{\theta}}{}_{i,j}}{\partial \theta_p} = - \boldsymbol{\alpha}^T \frac{\partial \mathbf{K}_{\boldsymbol{\theta}}^y}{\partial \theta_p} \boldsymbol{\alpha} \quad (\text{IV})$$

Dabei entspricht die Ableitung der Matrix  $\mathbf{K}$  der Ableitung der einzelnen Elemente nach  $\theta_p$ .

---

<sup>3</sup> „komplex“ bezieht sich hier darauf, dass der Kernel über eine Vielzahl von Einstellparametern verfügt. Es besteht kein Zusammenhang zu den komplexen Zahlen.

## 2 Theoretische Grundlagen

---

$$\left[ \frac{\partial \mathbf{K}_\theta}{\partial \theta_p} \right]_{i,j} = \frac{\partial K_\theta(\mathbf{x}_i, \mathbf{x}_j)}{\partial \theta_p} \quad (2.36)$$

Ein hochgestelltes  $\mathbf{y}$  soll darauf hinweisen, das hier die Information der Klassenzugehörigkeit bereits eingeflossen ist.

$$\left[ \frac{\partial \mathbf{K}_\theta^y}{\partial \theta_p} \right]_{i,j} = y_i y_j \frac{\partial K_\theta(\mathbf{x}_i, \mathbf{x}_j)}{\partial \theta_p} \quad (2.37)$$

Aus den Gleichungen wird offensichtlich, das I und II unabhängig von dem Index  $p$  des Kernelparameters sind und es daher ausreicht, diese ein mal für alle partiellen Ableitungen zu berechnet. III und IV machen es aber für jeden Kernelparameter notwendig, die gesamte Kernelmatrix abzuleiten.

Das Berechnen der Ableitung ist zwar sehr Rechenintensiv. Die schnellere Konvergenz erlaubt es aber, in weniger Schritten zu einem Optimum zu kommen. Dadurch sind deutlich weniger kostspielige Aktualisierungen von  $\alpha$  und  $\beta$  notwendig.

### Soft Margin als Kernelparameter

Der Bestrafungsfaktor  $C$ , mit dem falsch klassifizierte Trainingsmember in die Maximierung der Margin einfließen, hat hohen Einfluss auf das Abstraktionsvermögen des Modells und muss sorgfältig ausgewählt werden.

Auch dieser Bestrafungsterm kann jedoch als Kernelparameter aufgefasst werden, der anhand des oben genannten Kriteriums angepasst werden kann[CV95].

Dazu wird der Bestrafungsterm leicht abgeändert. Fließt er quadratisch ein

$$\min_{\mathbf{w}, b} \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + C \left( \sum_{i=1}^l \xi_i \right)^2 \quad (2.38)$$

lässt sich zeigen das dies einer Addition von  $\frac{1}{C}$  auf der Hauptdiagonalen der Kernelmatrix entspricht.

$$\tilde{\mathbf{K}}_\theta \leftarrow \mathbf{K}_\theta + \frac{1}{C} \mathbf{I} \quad (2.39)$$

$\mathbf{I}$  ist die Einheitsmatrix.

Dies ermöglicht die automatische Optimierung des Bestrafungsterms. In der Implementierung wird  $C$  als zusätzlicher Hyperparameter  $\theta_{d+1}$  umgesetzt.

Da  $K_\theta(\mathbf{x}_i, \mathbf{x}_j)$  unabhängig von  $C$  ist kann leicht  $\frac{\partial \mathbf{K}_\theta^y}{\partial C}$  bestimmt werden.

$$\frac{\partial \mathbf{K}_\theta^y}{\partial C} = -\frac{1}{C^2} \mathbf{I} \quad (2.40)$$

## 2.2 Quadratic Programming

In vielen Bereichen der Informatik ist ein Problem der Form 2.13 zu lösen. Aufgrund der quadratischen Natur wird ein solches Problem auch „Quadratic Programming“ genannt.

Sowohl das finden der besten Hyperebene als auch die Bestimmung des Radius  $R^2$  ist ein „Quadratic Problem“. Da die Form sich nur leicht unterscheidet, wurde ein Algorithmus gesucht und implementiert, der einen allgemeineren Fall löst. Dabei wurden die Bezeichner aus 2.13 beibehalten bzw. um den Vektor  $\mathbf{p}$  ergänzt. Negative Vorzeichen der ursprünglichen Aufgabe werden durch Multiplikation der betroffenen Vektoren/Matrizen mit  $-1$  eliminiert:

$$\min_{\alpha} W(\alpha) = \frac{1}{2} \alpha^T \mathbf{K} \alpha + \alpha^T \mathbf{p} \quad (2.41)$$

$$\text{unter der Bedingung } \mathbf{y}^T \alpha = \text{konst.} \quad (2.42)$$

$$\text{und } 0 \leq \alpha_i \leq C \quad (2.43)$$

Dieses Problem ließe sich durch verschiedene Ansätze lösen. Dazu zählen zum Beispiel der recht naive Line Search-Algorithmus sowie das Newton-Verfahren. Diese beachten jedoch nicht die vielen positiven Eigenschaften, die ein Problem dieser Form hat.

Darunter fällt

- das eine solche quadratische Funktion immer konvex ist, wenn die Matrix  $\mathbf{K}$  positiv semidefinit ist. Da das auf  $\mathbf{K}$  immer zutrifft lässt sich daraus schließen, dass es neben dem globalen Minimum keine weiteren lokalen Minima gibt. Die notwendige wird damit zur Hinreichenden Bedingung, die Gegenrichtung des Gradienten führt immer zu einer Verbesserung der Zielfunktion und dadurch einer Annäherung an das globale Minimum.

## 2 Theoretische Grundlagen

---

- die einfache Form der Nebenbedingungen, speziell die der Ungleichungen. Beschränkungen der Form  $l \leq \alpha_i \leq u$  mit  $l, u \in \mathbb{R}$  werden auch Box-Constraints genannt. Ihre Gültigkeit lässt sich nur durch die Betrachtung einer Komponente überprüfen und einhalten.
- die leichte Bestimmung des Gradienten für den lediglich eine Matrix-Vektor-Multiplikation und Vektor-Vektor-Addition notwendig ist.

$$\nabla_{\alpha} W(\alpha) = \mathbf{K}\alpha + \mathbf{p} \quad (2.44)$$

Eine Eigenschaft die die Berechnung ein wenig erschwert ist die Gleichheitsbedingung  $\sum_{i=1}^l y_i \alpha_i = \mathbf{y}^T \alpha = const.$  Obgleich leicht auf Gültigkeit zu überprüfen, so verhindert es doch, das jeder Punkt innerhalb der Box-Constraints einen gültigen Punkt darstellt.

Auch in der SVM-Bibliothek libSVM, mit deren Hilfe Support Vector Machine auf ihre Einsetzbarkeit überprüft wurde, kommt ein Algorithmus zum Einsatz, der einige der positiven Eigenschaften ausnutzt. Mit seinem Ansatz, pro Iteration lediglich zwei der Variablen zu verändern, kann leicht Gleichung 2.42 sichergestellt werden. Bei einer großen Anzahl an Trainingspunkten nimmt die Effektivität dieses Algorithmus jedoch rapide ab. Das gilt vor allem dann, wenn wie im Fall der SVM mit RBF-Kernel, nur sehr wenige der Vektorelemente  $\alpha_i = 0 \vee \alpha_i = C$  erfüllen[CL02].

Ist eine Ungleichung in einem Punkt als Gleichung erfüllt, d.h. der Punkt liegt auf einer „Wand“ der Box-Constraint, spricht man auch von einer aktiven Nebenbedingung. Viele Optimierungsalgorithmen merken sich die Menge  $\mathcal{A}(\alpha^k)$  der aktuell aktiven Nebenbedingungen. Diese werden als Gleichheitsbedingungen aufgefasst, alle anderen werden nicht berücksichtigt. Die Menge  $\mathcal{A}(\alpha^k)$  muss jedoch dynamisch bleiben, pro Iteration können Nebenbedingungen hinzugefügt oder entfernt werden.

### 2.2.1 Gradient-Projection-Method

Handelt es sich bei den Nebenbedingungen lediglich um Box-Constraints, stellt die Gradient-Projection-Methode wie sie in *Numerical Optimization* von Nocedal J. und S.J. Wright[NW99] vorgestellt wird eine effektive Möglichkeit zur Berechnung des Minimums dar. Anders als z.B. die Active-Set Methode benötigt er keine Invertierungen, außerdem kann sich hier die Menge der aktiven Bedingungen pro Iterationsschritt rasch ändern. Die Active-Set Methode erlaubt nur das Hinzufügen oder Entfernen einer aktiven Methode pro Iteration.

Von einer beliebigen Schätzung  $\alpha^0$ , die alle Nebenbedingungen erfüllt, wird iterativ in Gegenrichtung des Gradienten  $g = \nabla W(\alpha^0)$  gegangen, bis sich der Wert von  $W(\alpha)$  nicht mehr verbessert<sup>4</sup>.

Die Suchrichtung (Gegenrichtung des Gradienten) wird in der Parameterdarstellung dargestellt:

$$\alpha(t) = \alpha^k - tg, \quad t \geq 0 \quad (2.45)$$

Um jedoch die Box-Constraints einzuhalten wird die Suchrichtung  $-g$  auf  $-p$  geknickt, sobald  $\alpha_i = 0$  oder  $\alpha_i = C$  ist.

Fasst man alle aktiven Nebenbedingung an Punkt  $\alpha^k$  als Menge  $\mathcal{A}(\alpha^k)$  auf, stellt sich die neue Projektionsrichtung  $p$  dar als

$$p_i = \begin{cases} 0 & \text{wenn } i \in \mathcal{A}(\alpha^k) \\ g_i & \text{wenn } i \notin \mathcal{A}(\alpha^k) \end{cases} \quad (2.46)$$

Sobald die Projektion auf eine weitere Wand trifft, wird diese in die Menge  $\mathcal{A}(\alpha^k)$  mit aufgenommen (vgl. Abb. 9).

Jedes Teilstück der Projektion wird nun untersucht. Befindet sich ein Minimum (in Abhängigkeit von  $t$ ) auf diesem Teilstück, ist  $\alpha^{k+1}$  gefunden. Der Algorithmus kann mit der neuen Schätzung begonnen werden.

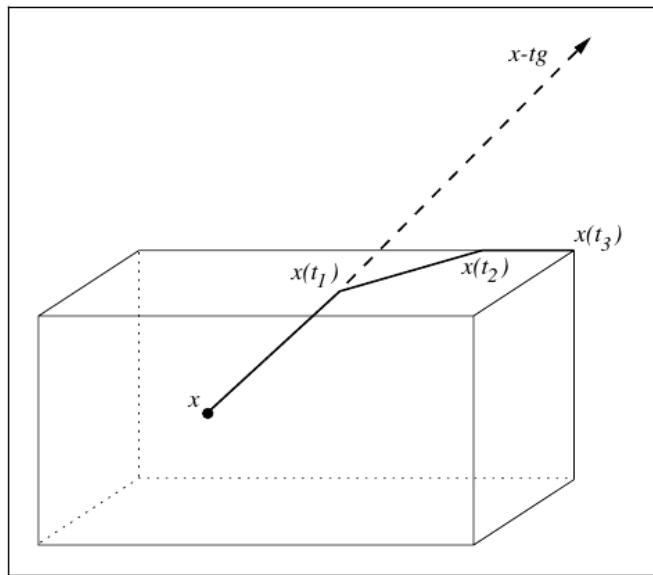
Leider stellt [NW99] keine Lösung für das Beachten der Gleichheitsbedingung 2.42 auf. Die Einbindung kann jedoch durch eine leichte Abwandlung erreicht werden. Dazu kann man zunächst die Beobachtung machen, dass eine Projektion in Richtung aller Vektoren  $\tilde{p}$  die

$$\sum_{i=1}^d y_i \tilde{p}_i = \mathbf{y}^T \tilde{\mathbf{p}} = 0 \quad (2.47)$$

erfüllen nie die Gleichheitsbedingung verletzen können. Das folgt aus

---

<sup>4</sup>Im folgenden stellen hoch gestellte Indizes den Iterationsschritt dar, tief gestellte Indizes bezeichnen die Komponente des Vektors



**Abbildung 9:** Gradient Projection kann leicht auf Box-Constraints eingehen

$$\mathbf{y}^T \boldsymbol{\alpha}^{k+1} = \mathbf{y}^T (\boldsymbol{\alpha}^k + t\tilde{\mathbf{p}}) = \mathbf{y}^T \boldsymbol{\alpha}^k + \mathbf{y}^T (t\tilde{\mathbf{p}}) = \mathbf{y}^T \boldsymbol{\alpha}^k + t\mathbf{y}^T \tilde{\mathbf{p}} = \mathbf{y}^T \boldsymbol{\alpha}^k \quad (2.48)$$

Projiziert man also nicht direkt in Richtung  $\mathbf{g}$  bzw.  $\mathbf{p}$  sondern in eine Richtung  $\tilde{\mathbf{p}}$ , die Gleichung 2.47 erfüllt und möglichst nah an  $\mathbf{g}$  liegt, kann man den Algorithmus der Gradient-Projection anwenden, ohne das sich die Summe aus der Nebenbedingung 2.42 ändert.

## 2.3 Kriging

Das Softwareprojekt wurde als Erweiterung einer Anwendung geschrieben, die das Kriging-Verfahren einsetzt. Der kommende Abschnitt soll lediglich dazu dienen, die Grundidee hinter diesem Verfahren zu erläutern. Besonderes Gewicht liegt schlussendlich auf den Gemeinsamkeiten zwischen Support Vector Machines und Kriging. Für eine detailliertere Beschreibung ist der interessierte Leser an die Arbeit von Andreas Schmitz verwiesen [Sch13], die im Rahmen des genannten Softwareprojektes entstand.

Ähnlich wie ein Klassifikator trifft auch das Kriging-Verfahren Vorhersagen über Eigenschaften eines Members, ehe die Prozesskette gestartet wurde. Es wird allerdings nicht die

Klassenzugehörigkeit sondern der Wert der Zielfunktion vorhergesagt. Somit wird es zur Erstellung eines Metamodelles der Zielfunktion verwendet. Der große Vorteil ist die Laufzeit, die meist ein Bruchteil der Prozesskette beträgt. Erst wenn die Vorhersage des Krigings vielversprechend ist wird die eigentliche Prozesskette gestartet.

Das Kriging-Verfahren, oft auch nur Kriging genannt, ist ein statistisches Verfahren, welches 1951 von Daniel Krige entwickelt und 12 Jahre später von dem Mathematiker Georges Matheron formalisiert wurde. Krige, ein südafrikanischer Bergbauingenieur, suchte eine Möglichkeit, aus Probebohrungen auf geologische Verläufe zu schließen. Es lässt sich jedoch leicht auch auf andere Bereiche anwenden. Als Parameter werden dann statt der Position der Bohrungen im Feld der Parametersatz des Trainingsmembers genutzt.

### 2.3.1 Mathematischer Ansatz

Kriging basiert auf mehreren Annahmen. Das es einen statistischen Ansatz verfolgt erkennt man daran, das die Trainingsmember  $y(\mathbf{x}_i)$  als eine Realisierung von Zufallsvariablen  $Z(\mathbf{x}_i)$  interpretiert werden. Außerdem wird angenommen, das sich der zu schätzende Wert  $y(\mathbf{x})$  aus einer Linearkombination von den bekannten Membern darstellen lässt.  $y^*(\mathbf{x}) = \sum_{i=1}^n w_i y(\mathbf{x}_i)$  bzw.  $Z^*(\mathbf{x}) = \sum_{i=1}^n w_i Z^*(\mathbf{x}_i)$

Um eine gute Vorhersage geben zu können, wird der Fehler zwischen dem tatsächlichen Wert und der ermittelten Zufallsvariable  $F(\mathbf{x}) = Z(\mathbf{x}) - Z^*(\mathbf{x})$  betrachtet. Der Erwartungswert des Fehlers soll Null betragen, die Varianz soll möglichst klein sein.

$$E [F(\mathbf{x})] = 0 \quad (2.49)$$

und

$$\text{var} [F(\mathbf{x})] = \min_{\mathbf{w}} \text{var} \left[ Z(\mathbf{x}) - \sum_{i=1}^n w_i Z^*(\mathbf{x}_i) \right] \quad (2.50)$$

Diese Art von Schätzer wird auch BLUE (Best Linear Unbiased Estimator) Schätzer genannt. Das Best ergibt sich aus der Minimierung der Varianz des Fehlers, Linear aus der gewichteten Summe und Unbiased aus der Erwartungstreue (Bedingungen 2.49)[Sch13].

Aus diesen Bedingungen und der weiteren Annahme, das  $Z(\mathbf{x}) = \text{const}$  ist, lässt sich  $\mathbf{w}$  direkt berechnen. Dazu ist jedoch noch eine Korrelationsmatrix  $\mathbf{R}$  notwendig, die den linearen

## 2 Theoretische Grundlagen

---

Zusammenhang zwischen den als Zufallsvariablen interpretierten Trainingsmembern enthält.

Da von jedem Punkt nur eine Realisierung vorliegt, lässt sich dieser nur aufgrund des Abstandes im Parameterraum abschätzen. Meist wird eine Gauß'sche Verteilung angenommen. Es ist allerdings auch jede andere Funktion denkbar, bei der die Korrelation mit zunehmendem Abstand sinkt. Des Weiteren muss  $\mathbf{R}$  positiv definit sein und  $\mathbf{R}_{i,i} = 1 \forall_i$  gelten. Durch Hyperparameter  $\theta$  kann der Einfluss der Parameter verändert werden, sodass auch die Korrelationsmatrix von diesen Hyperparametern abhängig wird  $\mathbf{R}(\theta)$ .

$\theta$  wird mit Maximum-Likelihood-Ansatz so gewählt, dass die Wahrscheinlichkeit das die durch  $\theta$  beschrieben Verteilung die Stichprobe verursacht hat, am höchsten ist. Das finden der idealen Hyperparameter ist der rechenintensivste Teil des Kriging-Verfahrens und erfolgt über eine numerische Optimierung. Zwar ist der zu maximierende Term differenzierbar, was effektive Methoden wie das QuasiNewton-oder das Resilient-Propagation-Verfahren ermöglicht. Die Dimension des Raumes der Optimierung ist jedoch gleich der Dimension des Raumes der Parameter. In der Anwendung innerhalb von AutoOpti kann dieser jedoch recht groß werden. Außerdem wird pro Iterationsschritt die kostspielige Invertierung der Korrelationsmatrix  $\mathbf{R}(\theta)$  benötigt.

### 2.3.2 Nutzbare Elemente des Kriging-Projektes

Die größte Gemeinsamkeit in der Struktur der Verfahren ist die Matrix, die das Verhältnis aller Punkte zueinander beinhaltet. Die Funktionen, die man zu schätzen der Korrelationen zueinander nutzen kann, ähneln in ihrer Form den gebräuchlichen Kernelfunktionen. So ist der Aufbau des RBF-Kernel der gleiche wie der einer Gauß'schen Normalverteilung.

Des Weiteren gibt es bei beiden Algorithmen einen Term, der während des Trainings minimiert werden muss. Dazu können Minimierer mit verschiedenen Ansätzen eingesetzt werden. Einige existieren bereits innerhalb des Kriging-Projekts und wurden dort bereits erfolgreich eingesetzt.

Auch die Infrastruktur zum Einlesen der Optionen und Datenbank kann von dem bestehenden Projekt übernommen werden. Das gleiche gilt für die Anbindung an AutoOpti.

# 3 Softwaretechnische Umsetzung

Ein Schwerpunkt dieser Arbeit liegt in der Implementierung der vorgestellten Support Vector Machines im Rahmen des bereits bestehenden Kriging-Projektes. Dabei konnten verschiedene softwaretechnische Ansätze erfolgreich eingesetzt werden. Durch den Einsatz von C++ und Interfaces musste der bestehende Code nur minimal verändert werden. Der Großteil des SVM Algorithmus konnte in neu hinzugefügten Klassen behandelt werden.

## 3.1 Der SVM-Algorithmus

Aus Kapitel 2.1 wird deutlich, dass bereits wenige Schritte für den Einsatz von Support Vector Machine ausreichen.

Der Ablauf des Trainings einer klassischen Support Vector Machine ist:

1. Wahl der Kernelfunktion (durch Benutzer). Falls sie Parameter beinhaltet müssen auch diese vor Beginn festgelegt werden.
2. Wahl von  $C$  (durch Benutzer)
3. Einlesen der Trainingspunkte  $\{\mathbf{x}_1, y_1\}, \dots, \{\mathbf{x}_l, y_l\}$
4. Erstellen der Kernelmatrix  $\mathbf{K}^y$ . Dabei gilt

$$\mathbf{K}_{ij}^y = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

5. Finden der Trennebene. Dazu wird der  $\alpha$ -Vektor durch den Einsatz eines Optimierers gesucht.  $\alpha$  ist das Optimum des quadratischen Problems:

$$\begin{aligned} \min_{\alpha} W(\alpha) &= \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T \mathbf{K}^y \alpha \\ \text{sodass } \mathbf{y}^T \alpha &= 0 \\ \text{und } 0 \leq \alpha_i &\leq C \quad \forall i \in \{1, \dots, l\} \end{aligned}$$

6. Berechnen von  $b$  unter Zuhilfenahme mehrerer Support-Vektoren (vgl. Gleichung 2.17)
7. Ausschreiben der  $\alpha_i$  und  $b$

Die Umsetzung der Vorhersage unbekannter Member erfolgt meist über folgende Schritte:

1. Einlesen der Trainingspunkte und der entsprechenden  $\alpha_i$   
Für jeden vorherzusagenden Punkt  $\mathbf{x}$ :
2. Berechnen des Vektors  $\mathbf{k}^y$  mit  $k_i = K(\mathbf{x}_i, \mathbf{x})$
3. Ausrechnen der Entscheidungsfunktion  $f(\mathbf{x})$

$$f(\mathbf{x}) = \operatorname{sgn} (\alpha^T \mathbf{k} + b) \quad (3.1)$$

Da die Vorhersage selber nicht sehr komplex ist und hier kaum Verbesserungsmöglichkeiten existieren wird in diesem Kapitel das Hauptaugenmerk auf das Training gelegt.

Anders als in klassischen Support-Vektor-Maschine Algorithmen soll in dieser Arbeit zusätzlich die Kernelfunktion automatisch eingestellt werden. Die Wahl der Kernelparameter durch den Benutzer fällt weg. Durch den Einsatz des Kriteriums für Kernel aus 2.1.6 entsteht eine zweite Optimierungsaufgabe. Es gilt also durchzuführen:

- Die Minimierung des Terms  $R^2 \|\mathbf{w}\|^2$  durch Veränderung des  $\theta$ -Vektors (vgl. Kapitel 2.1.6)
- Das Finden der besten Trennebene durch Optimierung des  $\alpha$ -Vektors

Zwischen den beiden Problemen besteht ein enger Zusammenhang. In der einen Optimierungsaufgabe tauchen auch die Parameter der jeweils anderen als Konstanten auf. Daher müssen sie gemeinsam betrachtet werden. Da beide durch iterative Algorithmen gelöst werden sind verschiedene Verschachtelungen denkbar. Diese unterscheiden sich darin, ob und wie die Optimierung der einen Aufgabe unterbrochen wird, um die andere zu starten.

In Abbildung 10 sind mehrere Lösungsmöglichkeiten zu sehen, die in die Überlegung mit

### 3 Softwaretechnische Umsetzung

---

eingeflossen sind. Diagramm (a) stellt den sequentiellen Weg dar. Eine Optimierung wird vollständig gelöst, ehe die nächste gestartet wird. Da durch jede Optimierung Konstante der jeweils anderen Zielfunktion geändert werden ist es notwendig, diese danach neu durchzuführen. Es entsteht ein Schleife, die in einem günstigen Fall zum Optimum beider Parametersätze führt. Aufgrund der Struktur des bestehenden Projektes war eine Umsetzung dieses Ansatzes aufwendig und hätte viele Anpassungen benötigt. Des weiteren ist zu erwarten, dass die ständigen Durchführungen kompletter Optimierungsalgorithmen sehr rechenintensiv ist.

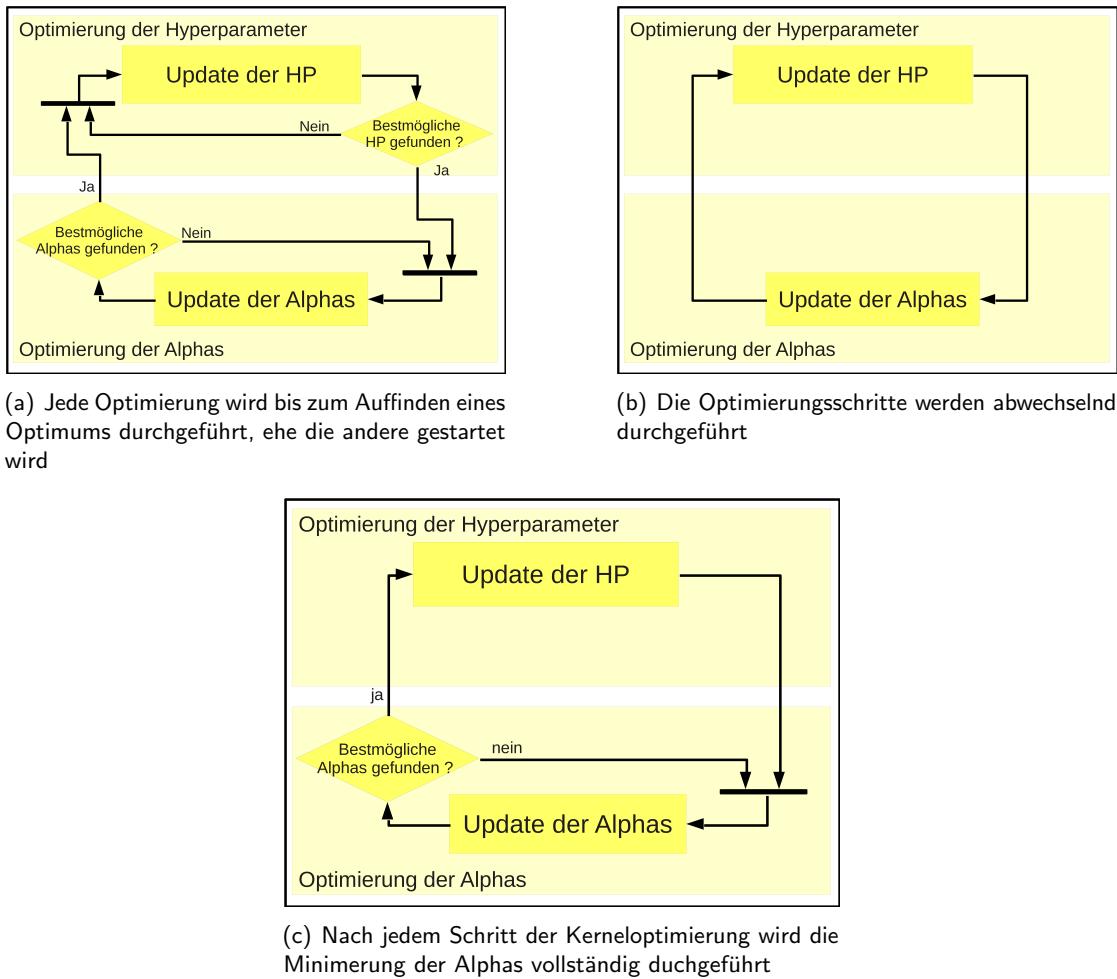
Eine fast parallele Lösung ist in Diagramm 10 (b) dargestellt. Auf jedes Update des einen Parametersatzes folgt ein Update des anderen, bis die Lösung konvergiert. Da für die Minimierung der beiden Probleme verschiedene Ansätze gewählt werden können, birgt (b) die Gefahr, dass einzelne Iterationsschritte sehr unterschiedlich dimensioniert sind und die Gesamtaufgabe schlecht konvergieren würde. Es müssten daher zunächst wechselseitige Effekte bei dem gleichzeitigen Einsatz verschiedener Minimierungsverfahren untersucht werden. Das ist im Rahmen dieser Arbeit nicht geschehen.

Umgesetzt wurde Schema 10 (c), ein Kompromiss aus (a) und (b). Dabei werden nach jedem Update der Hyperparameter  $\theta$  die idealen  $\alpha_i$  gesucht, ehe ein erneutes Update von  $\theta$  erfolgt. Dieser Kompromiss wurde gewählt, da das Finden der  $\alpha_i$  effektiv durchgeführt werden kann.

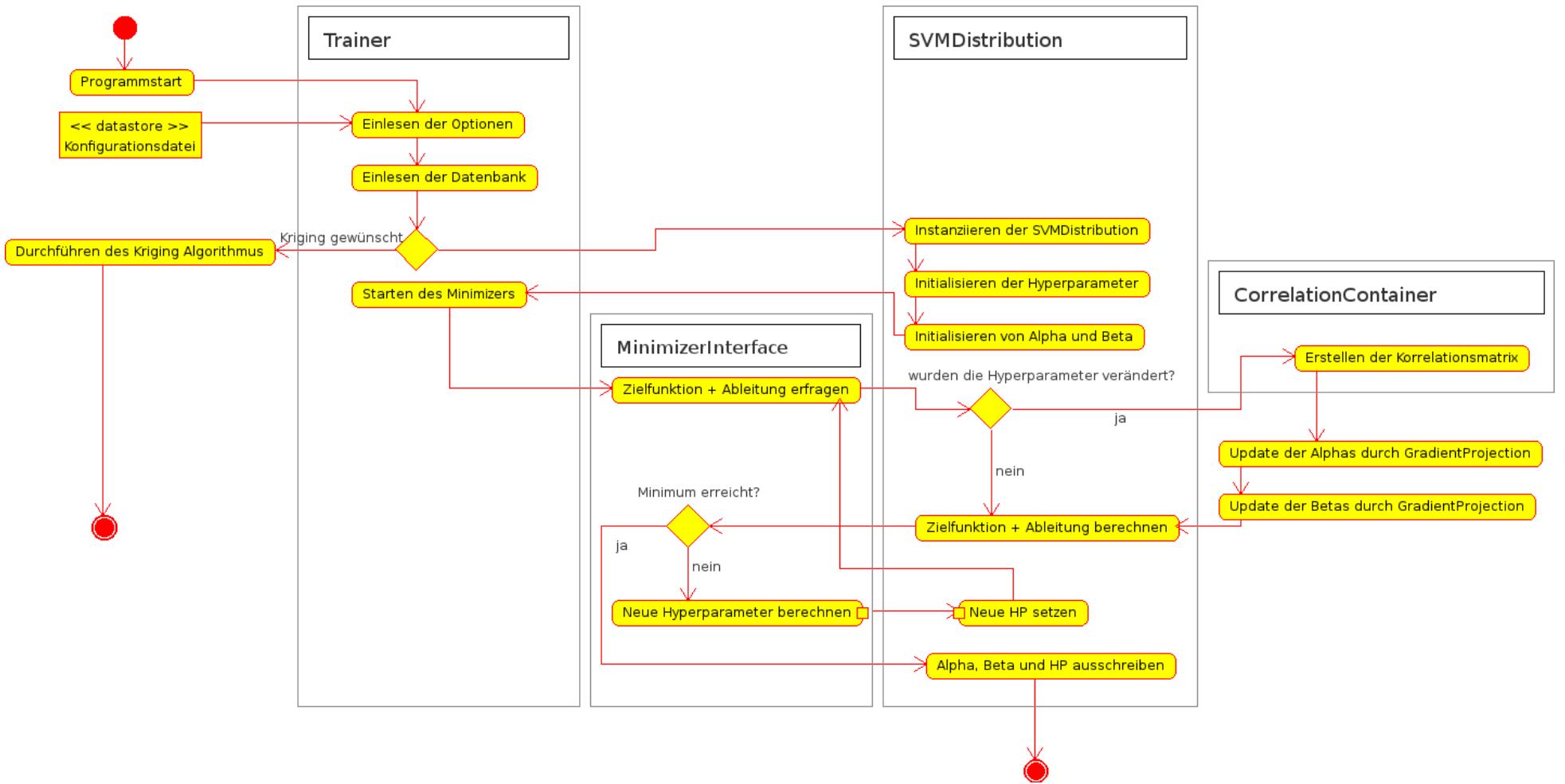
Aus der Wahl für Schema (c) und einigen Aspekten des klassischen SVM-Algorithmus lassen sich bereits die Anforderungen an das Projekt erarbeiten. Zusammen mit dem bekannten Aufbau des Kriging-Projektes ergibt sich die in dem Aktivitätsdiagramm 11 aufgezeigte Struktur.

In Diagramm 11 sind einige Akteure bereits als Rahmen verdeutlicht. Bei diesen Akteuren handelt es sich um Klassen. Sie werden in Kapitel 3.3 weiter erläutert.

### 3 Softwaretechnische Umsetzung



**Abbildung 10:** Verschiedene Kombinationsmöglichkeiten der beiden Optimierungen. Eine „Update“-Aktivität entspricht einem einzigen Iterationsschritt.



**Abbildung 11:** Aktivitätsdiagramm des Trainings von Support Vektor Machines. Die Rahmen verdeutlichen, welcher Akteur für die jeweiligen Aktivitäten zuständig ist. An Stelle des MinimizerInterface können verschiedene Implementierungen eingesetzt werden.

## 3.2 Benutzung

Das Programm läuft lediglich in der Kommandozeile. Da das Haupteinsatzgebiet innerhalb einer automatisierten Prozesses ist ist eine Benutzeroberfläche nicht notwendig. Um dem Benutzer Einflussmöglichkeiten zu geben kann es jedoch mit verschiedenen Schlüsselwörtern gestartet werden, welche das Verhalten verändern. So kann zum Beispiel der Modus (Training oder Vorhersage) gewählt werden. Außerdem müssen beim Start der Pfad und Name der Dateien angegeben werden aus denen das Programm weitere Angaben bezieht sowie die Informationen über die Punkte erhält.

Wird das Programm ohne Argumente gestartet, erhält der Benutzer einen Hinweis auf den fehlerhafte Umgang sowie einen Hilfe-Text.

### 3.2.1 Training

Soll ein Training durchgeführt werden startet der Benutzer das Programm mit den entsprechenden Argumenten. Die Angaben beinhalten:

- Dateipfad und -name der Datenbank
- Dateipfad und -name in der die Informationen des trainierten Modells abgespeichert werden
- ob Schleifen parallelisiert werden sollen (betrifft vor allem die Matrizen)
- Dateipfad und -name der Konfigurationsdatei mit weiteren Optionen

Innerhalb der Konfigurationsdatei können weitere Einstellungen verwendet werden. Dazu zählt die Auswahl der Korrelations-/Kernelfunktion, des Kriging- oder des SVM-Algorithmus und weitere. Auch die Angabe, ob die Hyperparameter verbessert werden sollen oder nicht, kann hier gemacht werden. Die verschiedenen Einstellungen, die für die Tests in Kapitel 4 benötigt wurden, wurden so dem Programm mitgeteilt.

Wird das Programm gestartet ist das Ziel des Trainings die Bereitstellung aller Informationen, die für die Berechnung von Vorhersagen notwendig ist. In diesem Fall sind das der  $\alpha$ -Vektor, der  $\theta$ -Vektor und  $b$ . Sind diese Informationen gefunden, bilden sie gemeinsam das Modell des Klassifikators und werden ausgeschrieben, um sie später für Vorhersagen nutzen zu können. Auch die gewählten Einstellungen werden in die Ergebnisdatei geschrieben um sie gegebenenfalls rekonstruieren zu können.

#### 3.2.2 Vorhersage

Für eine Vorhersage muss das Programm mit anderen Parametern gestartet werden. Da alle notwendigen Einstellungen und Informationen in der oben genannten Ergebnisdatei stehen, ist keine weitere Konfigurationsdatei notwendig.

- Dateipfad und -name der Ergebnisdatei, die beim Training erstellt wurde
- Dateipfad und -name der Datei in der die Punkte liegen, deren Klassenzugehörigkeit geschätzt werden soll. In dieser Datei können beliebig viele Punkte vorhanden sein. So wird nur ein Programmaufruf benötigt, um eine Vielzahl von Punkten vorherzusagen.
- Dateipfad und -name der Datei, in die die Ergebnisse geschrieben werden sollen

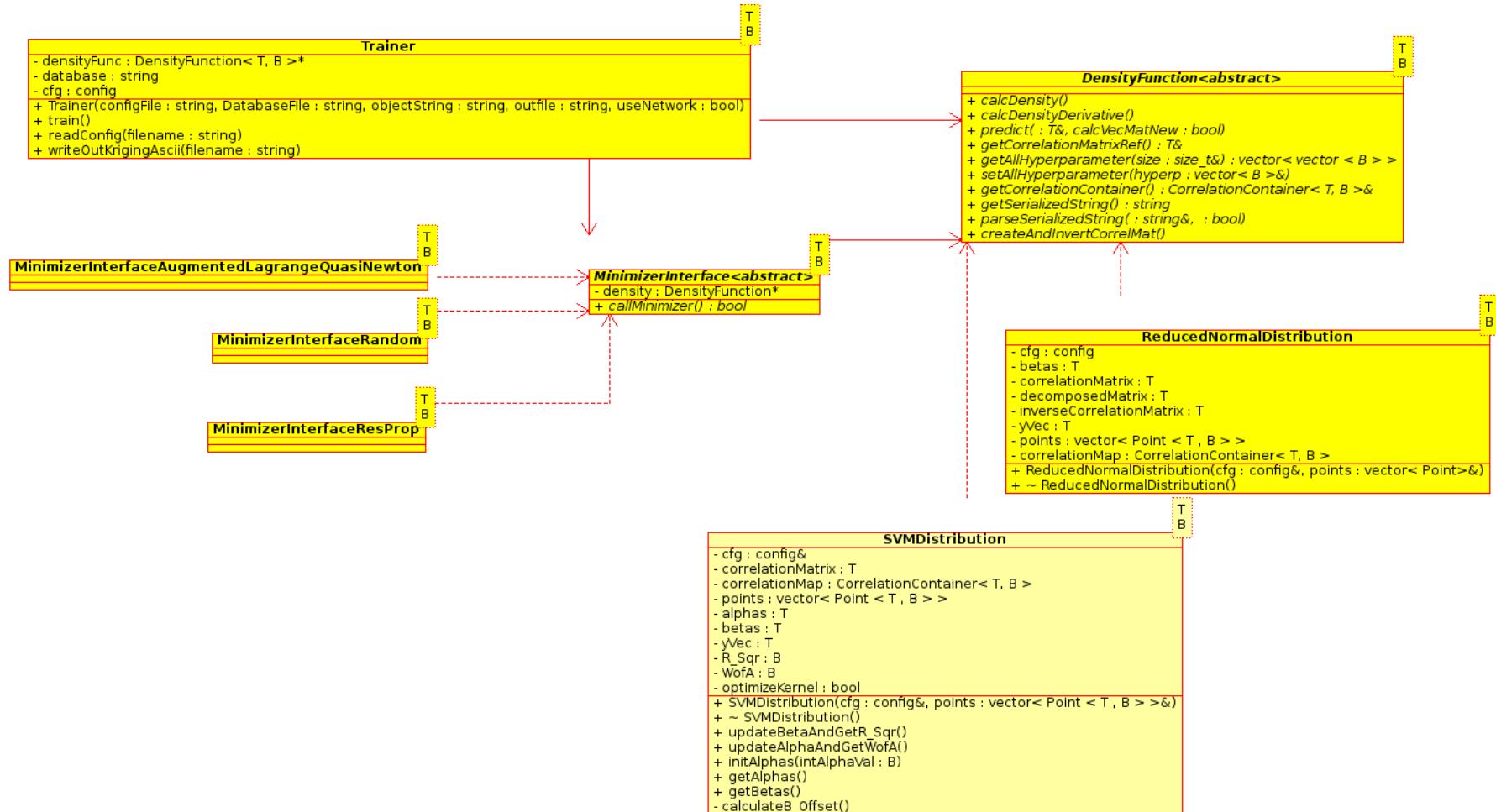
#### 3.3 Programmstruktur und -ablauf

Die Umsetzung des Projekts erfolgte innerhalb des bereits bestehenden Kriging-Projektes. Die Rahmenbedingungen der Entwicklung waren dadurch bereits von Beginn an gegeben.

Als Programmiersprache findet C++ Verwendung. C++ kombiniert Objektorientierung mit hoher Performance. Daher ist es in den Ingenieurwissenschaften weit verbreitet, so auch in dem Institut für Antriebstechnik. Als IDE wird Eclipse eingesetzt. Für gängige Methoden wie die Verwaltung einer Logdatei oder der Benutzung von regulären Ausdrücken werden Boost Libraries eingesetzt.

Der benutzte Computer ist so leistungsstark ausgelegt das nicht all zu umfangreiche Strömungssimulationen auch lokal durchgeführt werden können. Er verfügt über 2 Intel Xeon E5-2620 Prozessoren. Diese haben jeweils 6 Kerne, die auf 2GHz getaktet sind. Durch das von Intel verwendete Hyperthreading ist es jedem Prozessor möglich, 12 Threads parallel abzuarbeiten. Bei Strömungssimulationen wird oft viel Arbeitsspeicher eingesetzt. Daher besitzt der Rechner 32 GB RAM. Als Betriebssystem wird Debian 7.5 „Wheezy“ eingesetzt.

Das bereits bestehende Projekt hat einen Großteil der Programmarchitektur bereits definiert. Der Großteil des SVM-Algorithmus konnte in ergänzenden Klassen abgehandelt werden, ohne tief in bereits bestehende Strukturen eingreifen zu müssen. Da nur wenige architektonische Überlegungen durchgeführt werden mussten ist der Fokus dieses Kapitels auf der Bestandsaufnahme des fertigen Programms. Es soll eher die Funktionsweise erläutern als die Schritte, die zu diesem Design geführt haben.



**Abbildung 12:** Klassendiagramm der im Training eingesetzten Klassen. Durch die Verwendung von Interfaces lässt sich auch während der Laufzeit leicht zwischen verschiedenen Implementierungen wählen. So kann man z.B. zwischen verschiedenen Ansätzen zur Minimierung der Zielfunktion wählen.  
 Als zentrale Klasse steuert der Trainer die DensityFunction sowie den gewählten Minimierer, welcher von ihm die Instanz der DensityFunction mitgeteilt bekommt.  
 Desweitern hält der Trainer alle notwendigen Informationen in einer Struktur config bereit.

#### 3.3.1 Verwendete Klassen

##### Trainer.hpp

Mithilfe des Schlüsselwortes `--train` lässt sich bei Programmaufruf in der Kommandozeile der Trainingsmodus aktivieren. Das Ziel des Trainingsmodus ist das Finden der Entscheidungsfunktion  $f(x)$  mit der daraufhin Vorhersagen getroffen werden können. Dabei müssen in erster Linie  $\{\alpha_0 \dots \alpha_l\}$  sowie  $b$  gefunden werden. Des weiteren soll durch verändern der Hyperparameter  $\{\theta_0 \dots \theta_d\}$  ein möglichst guter Kernel gefunden werden. Dazu wird der Term  $R^2\|\mathbf{w}\|$  minimiert (vgl. Kapitel 2.1.5).

Ist das Schlüsselwort enthalten, instantiiert die `main`-Funktion ein Objekt der Trainer-Klasse. Dies ist dementsprechend die zentrale Klasse während des Trainings (vgl. Abb. 12). Er liest die Datenbank der Trainingspunkte aus und interpretiert die Konfigurationsdatei.

Der Trainer besitzt Instanzen von zwei Interfaces<sup>1</sup>. Abhängig von den getroffenen Optionen wählt er bei der Instantiierung zwischen verschiedenen Implementierungen. Die Wahl des Interfaces der `DensityFunction` ist dafür verantwortlich, ob der Kriging- oder der SVM-Algorithmus durchgeführt wird. Soll Kriging gestartet werden, verwendet der Trainer die `ReducedNormalDistribution`, sonst die `SVMDistribution`. Das andere Interface erlaubt die Benutzung von verschiedenen Minimierungsalgorithmen.

Der Ablauf des Trainings und die Interaktionen zwischen dem Trainer und den anderen Klassen ist in dem Aktivitätsdiagramm 11 dargestellt.

##### 3.3.2 Kriging.hpp

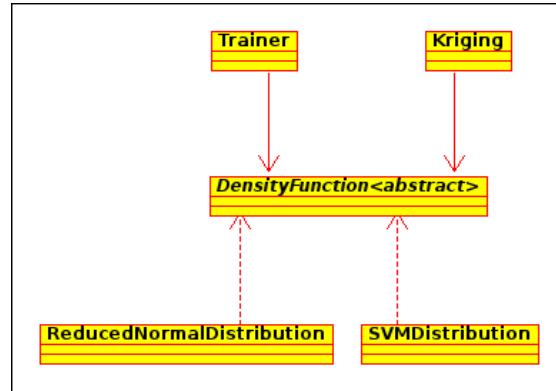
Im Vorhersagemode kommt die Kriging-Klasse zum Einsatz. Die Klasse wird nicht nur beim Kriging-Algorithmus eingesetzt, der Name ist lediglich historisch bedingt.

Die Kriging-Klasse ließt zunächst die vorherzusagenden Punkte ein und berechnet für diese den  $\mathbf{k}$ -Vektor (vgl. Kapitel 3.1). Da sich die Entscheidungsfunktionen zwischen Kriging und SVM leicht unterscheiden, verfügt auch die Kriging-Klasse über eine Instanz der `DensityFunktion`. Diese Instanz gleicht der Instanz des Trainings, da sie mithilfe der Ergebnisdatei aus dem Training instanziert wird. Mithilfe der `DensityFunktion` wird die

---

<sup>1</sup>C++ besitzt keine formal definierten Interfaces. Spricht man im C++ Umfeld von Interfaces handelt es sich meist um abstrakte Klassen, die lediglich virtuelle Funktionen beinhaltet

Entscheidungsfunktion ausgerechnet.



**Abbildung 13:** Die Beziehung zwischen Trainings- und Vorhersagemodus kann über die verwendeten Klassen dargestellt werden. Die Trainer-Klasse wird nur beim Training eingesetzt, die Kriging-Klasse nur bei der Vorhersage. Durch das Verwenden von einer identischen Instanz der DensityFunction wird sicher gestellt, dass sich das Training und die Vorhersage auf das gleiche Modell beziehen. Diese identische Instanz wird bei der Vorhersage durch das Einlesen der Ergebnisdatei des Trainings erzeugt.

#### OpenMPMatrix.hpp

Die OpenMPMatrix-Klasse ist die für die Darstellung von Matrizen und Vektoren verwendete Klasse. Sie wurde im Rahmen des Kriging-Projektes entwickelt und speziell auf dessen Anforderungen zugeschnitten. So sind zum Beispiel die Matrixoperationen mithilfe von OpenMP parallelisiert. Um den Zugriff weiter zu beschleunigen wird die gesamte Matrix linear im Speicher abgelegt.

Bei der Programmierung mit dynamischen Datenstrukturen wie der OpenMPMatrix sollte man darauf achten, dort wo es möglich ist mit Zeigern und Referenzen zu arbeiten. Unnötige Kopiervorgänge können so verhindert werden.

#### DensityFunction.hpp

Die abstrakte Klasse der DensityFunction ist eine Verallgemeinerung um verschiedene Implementierungen zu erlauben. Die Aufgaben, die diese Klassen gemeinsam haben, ist das

### 3 Softwaretechnische Umsetzung

---

Bereitstellen des zu minimierenden Terms sowie dessen Ableitungen. Des weiteren müssen sie die Entscheidungsfunktion errechnen können.

Die Art und Weise wie dieser Zustand kommt ist den konkreten Implementierungen überlassen. Die Nutzung der `DensityFunction`-Klasse als Interface ist es geschuldet, dass das Einbinden der Support Vector Machine ohne große Eingriffe in bestehenden Code möglich war. Es musste lediglich eine neue Implementierung des Interfaces hinzugefügt werden.

#### **ReducedNormalDistribution**

Die `ReducedNormalDistribution` war ursprünglich die einzige Umsetzung der `DensityFunction`. Sie liefert den so genannten Likelihood-Wert für die aktuell eingestellten Hyperparameter zurück, den es zu optimieren gilt. Leider ist das Berechnen dieses Wertes in der Regel recht teuer, da hier große Matrix-Operationen nötig sind. Wird sie eingesetzt wird ein Modell auf Grundlage des Kriging-Algorithmus erstellt. In dem Aktivitätsdiagramm 11 wird sie in der linken Verzweigung dargestellt. Da Kriging selber kein Teil der Arbeit war wird dort nicht näher darauf eingegangen.

#### **SVMDistribution**

Die `SVMDistribution` ist die Implementierung der `DensityFunction` für die Support Vector Machines. Ein Hauptteil des in diesem Projekt entstandenen Quellcodes befindet sich innerhalb dieser Klasse.

Auch hier ist die Hauptaufgabe das Zurückliefern eines zu Optimierenden Wertes. Dabei handelt es sich um die Abschätzung der Fehlerquote aus 2.1.5. Um das Verhältnis zu berechnen müssen jedoch zunächst alle  $\alpha_i, \beta_i, i \in \{1, \dots, l\}$  bestimmt werden. Die Ermittlung der  $\alpha_i$  kommt dabei bereits der Erstellung eines SVM-Modells gleich.

Als Implementierung des `DensityFunction`-Interfaces muss die `SVMDistribution` einige Funktionen umsetzen, auf die die Trainer- und die Minimierer-Klasse sowie die Kriging-Klasse zugreifen können. Des Weiteren verfügt sie über einige als privat deklarierte Funktionen, die Aufgaben übernehmen die lediglich intern Verwendung finden.

Die interessantesten dieser Funktionen verdienen einer genaueren Betrachtung.

- **Normierung der Parameter und Abbildung auf die zwei Klassen** Ehe eine Support Vector Machine erstellt werden kann, müssen einige Anpassungen an den Trainingsindividuen vorgenommen werden, die der Trainer noch nicht vorgenommen hat. Im Kriging betrifft die Vorhersage eine Reelle Zahl, daher ist im gesamten Projekt der zu findende Wert als double abgespeichert.

In der Initialisierung wird sichergestellt, dass es nur zwei unterschiedliche Einträge gibt. Sind diese nicht  $\{-1, +1\}$ , werden sie darauf abgebildet.

Die Parameter der Member sind sehr unterschiedlicher Natur. Einige befinden sich zwischen sehr engen Grenzen, andere können über mehrere Größenordnungen variieren. Um den Einfluss dieser unterschiedlichen Skalierungen zu verringern werden die Parameter so normiert, dass ihr arithmetisches Mittel über die Menge an Trainingsmembern 0 und ihre Standardabweichung 1 beträgt.

*Anmerkung:* Bei dieser Normierung fließt die Klassenzugehörigkeit der einzelnen Member nicht mit ein. Man streckt/staucht lediglich die Dimensionen des Parameterraums damit die Trainingssamples in alle Richtungen ähnlich verteilt sind. Dadurch, dass die Verteilung aller Parameter zueinander nach der Normierung gleich ist, ist auch die Abhängigkeit der Klassenzugehörigkeit bei einer gleichen Veränderung der Parameter ähnlich. Wie groß die Abhängigkeit jedoch tatsächlich ist, kann durch diese Normierung nicht bestimmt werden. Daher ist die Optimierung des  $\theta$ -Vectors immer noch sinnvoll. Durch die Normierung ist jedoch dafür gesorgt, dass die einzelnen Einträge in  $\theta$  näher aneinander liegen. Ein Startpunkt der Optimierung, mit  $\theta_1 = \dots = \theta_d$  führt daher schneller zu einer Lösung.

- **Initialisierung** Die Initialisierung stellt sicher, das schon zu Beginn der Optimierung der Hyperparameter  $\theta$  dort sinnvolle Einträge vorliegen. Dazu sind verschiedene Möglichkeiten implementiert. Während der späteren Anwendung innerhalb von AutoOpti stellt `restart` eine sinnvolle Lösung dar. Dazu werden die  $\theta_i$  Einträge des letzten SVM-Modells übernommen.

Eine anderen Möglichkeit ist die Initialisierung eines  $\theta$ -Vektors mit gleichen Einträgen. Mit welchem Wert alle Einträge belegt werden, kann einerseits von Nutzer angegeben werden. Eine andere Option löst diese Aufgabe naiv durch ausprobieren. Für die Auswahl wird hier das Qualitätskriterium für Kernel aus Kapitel 2.1.5 genutzt.

- **Berechnung der Kernelmatrix** Das Erstellen der Kernelmatrix ist innerhalb des Kriging-Algorithmus sehr komplex. Da in dessen Weiterentwicklung, dem Cokriging, auch Ableitungen

### 3 Softwaretechnische Umsetzung

---

der Punkte vorkommen, werden innerhalb einer Kernelmatrix verschiedene Kernelfunktionen eingesetzt. Diese werden von einem Container verwaltet. Der Container kommt bei der Berechnung zum Einsatz. Im SVM- sowie im Kriging-Algorithmus ist ihre Berechnung eines der rechenintensivsten Aufgaben, daher wurden hier bereits Anstrengungen unternommen, sie möglichst effizient zu gestalten. Sie ist Parallelisiert, außerdem wird dort die Symmetrie der Matrix ausgenutzt.

Um die Geschwindigkeit noch weiter zu erhöhen sind einige der Berechnungen sogar auf die Grafikkarte ausgelagert.

- **Berechnung der Alphas und Betas** Die eigentliche Aufgabe eines SVM-Algorithmus, das Finden des  $\alpha$ -Vektors, wird mithilfe des Gradient-Projection Ansatzes gelöst. Die Implementierung befindet sich nicht innerhalb der SVMDistribution-Klasse. Sie ist als globale Funktion implementiert. Damit die möglicherweise große Kernelmatrix nicht auf den Stack kopiert werden muss wird bei dem Aufruf der `solveQPwithGP` genannten Funktion mit Referenzen gearbeitet.

- **Berechnung der Zielfunktion und deren Ableitungen** Sind  $\alpha$  und  $\beta$  gefunden ist die Berechnung der Zielfunktion mithilfe der Kernelmatrix leicht möglich.

Für die Ableitung muss jedoch die Matrix nach jedem Kernelparameter erfolgen. An dieser Stelle liegt daher ein großes Potential für eine Verbesserung der Performance, da sie sich leicht parallelisieren ließe.

## 4 Tests

Um zu überprüfen, wie gut das Projekt die Erwartungen erfüllt, wurden verschiedene Tests erdacht und durchgeführt. Dabei wurde so vorgegangen, dass zunächst nicht von der selbständigen Auswahl des Kernels Gebrauch gemacht wurde. In den verschiedenen Tests kam diese Fähigkeit erst nach und nach zum Einsatz. Dadurch kann gut beobachtet werden, wie die Eigenschaften des Programms sich mit höherem Grad der Selbständigkeit verändern. Werden die Parameter vorher festgelegt geschieht das durch einen Grid Search. Bei einem Grid Search werden viele verschiedenen Einstellungen ausprobiert. Es ist die klassische Methode, ein Support-Vector-Machine zu kalibrieren.

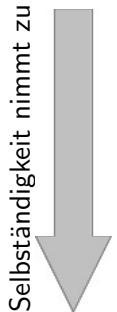
Durch die steigende Selbständigkeit lässt sich zum Beispiel die Frage beantworten, ob auch die selbständige Suche die gleichen Kernelparameter findet, die im Grid Search die Besten Ergebnisse erzielt haben.

Da er für komplexe Fälle die besten Ergebnisse liefert wurde in allen Test der Radial-Basis-Function-Kernel benutzt. Die Form der bereits in Kapitel 2.1 erwähnten Kernel-Funktion ist

$$K_{\theta}(\mathbf{x}, \tilde{\mathbf{x}}) = e^{\sum_i \theta_i (x_i - \tilde{x}_i)^2}$$

In einer solchen Form ist  $\theta_i$  direkt für den Einfluss des  $i$ -ten Parameters verantwortlich.

Die Tests wurden in vier Kategorien aufgeteilt.



	Voreingestellt	selbständig gefunden
Test 1	C $\theta$ (für alle Parameter gleich)	$\alpha$
Test 2	C $\theta$ (für alle Parameter gleich)	$\alpha$
Test 3	C	$\alpha, \beta$ $\theta$ (für alle Parameter gleich)
Test 4	C	$\alpha, \beta$ $\theta$ (wird für jeden Parameter angepasst)

Für jeden Test werden seine Eigenschaften und die Ergebnisse erläutert sowie ein Fazit gezogen.

## 4.1 Test-Setting

### 4.1.1 Testdaten

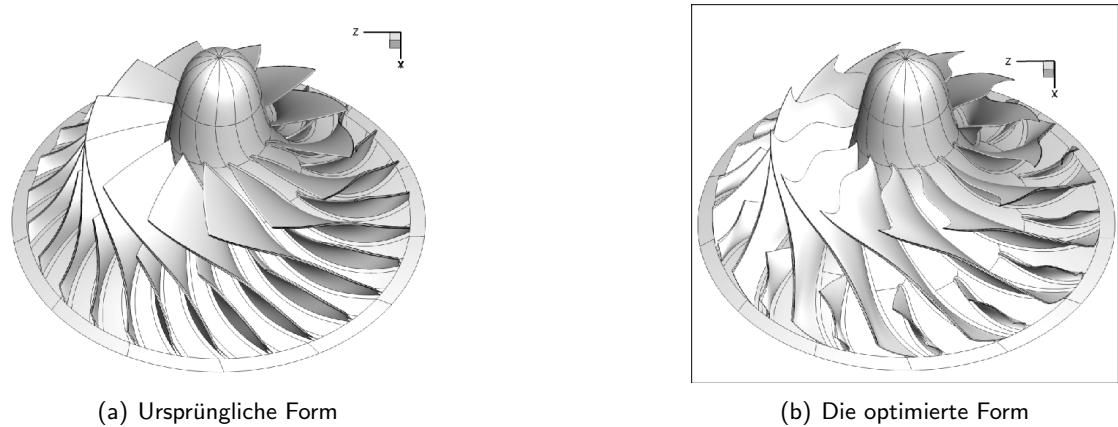
Die Testdaten wurden einem realen Fall entnommen. Dabei handelt es sich um die Optimierung eines Radialverdichters. Ausgehend von der Form eines existierenden Verdichters wurde eine neue Form mit verbesserten Eigenschaften gesucht. Die Optimierung konnte erfolgreich die aerodynamischen Eigenschaften beeinflussen, sodass ein höherer Wirkungsgrad erzielt wird. Dabei mussten strukturmechanische Grenzen eingehalten werden (vgl. Abb. 14). Für Testzwecke wurde bereits ein Exemplar des Radialverdichters gefertigt (vgl. Abb. 15).[VAR14]

Die zur Verfügung stehende Datenbank enthielt 2340 Member mit 53 Dimensionen. Bei den 53 Parametern handelte es sich um Angaben wie einer Dickenverteilung, Staffelungswinkel usw. Bei 754 Member konnte die Simulation erfolgreich durchgeführt werden, bei 1586 lag ein Programmabbruch vor. Die erfolgreiche Berechnung eines Members dauerte circa acht Stunden, die gesamte Optimierung lief auf mehreren Knoten des Institutsclusters für mehrere Wochen.

Da am Anfang einer Optimierung die Member zufällig gewählt werden bzw. weniger Informationen vorliegen, liegt die Dichte an Programmabrüchen dort höher. Die Information der zeitlichen Verteilung floss jedoch nicht in den Test mit ein. Damit die durchgeführte Kreuzvalidierung belastbare Ergebnisse liefert, ist es nötig das sich die einzelnen Teilmengen

## 4 Tests

---



**Abbildung 14:** Die Form des Verdichters vor und nach der Optimierung *Bildquelle: [VAR14]*



**Abbildung 15:** Für weitere Tests am Prüfstand wurde die berechnete Form aus einem Aluminiumblock gefräst *Bildquelle: [VAR14]*

zueinander gleich verhalten. Die Zuordnung der Member in die Teilmengen wurde daher unabhängig von der Reihenfolge zufällig durchgeführt.

Die Testdatenbank ist die gleiche, die bereits für die Arbeit im Praxismodul III verwendet wurde [Sch14].

## 4.2 Testskript

Um die Tests effektiv durchführen zu können wurde ein Skript erstellt, das die notwendigen Schritte automatisiert durchführt. Aufgrund der Schnelligkeit in der Programmierung wurde hierbei Python verwendet. Rechenintensive Operationen kommen innerhalb des Skriptes nicht vor. Pythons größter Nachteil, die relativ schlechte Performance, kommt daher nicht zu tragen.

Das Mittel der Wahl für die Berechnung einer Fehlerquote ist die Kreuzvalidierung (vgl. Kapitel 2.1.5). Da die Trainingsmember nicht gleichzeitig Testmember sind, verhindert man eine Verfälschung des Ergebnis und man erhält eine Fehlerquote die gut der tatsächlichen entspricht. Daher wird auch in diesem Skript die Kreuzvalidierung eingesetzt. Aufgrund der meist relativ kurzen Rechenzeit des SVM-Algorithmus von einigen wenigen Sekunden kann hier leicht eine 5-fache Kreuzvalidierung eingesetzt werden. Mit 2340 Membern verfügt die Testdatenbank über viele Elemente, sodass das Training leicht mit mehreren Hundert Membern durchgeführt werden kann. Wird in einem Test davon abgewichen wird darauf hingewiesen.

Im Rahmen des Praxismoduls III wurde bereits ein Python Skript für die Durchführung einer Kreuzvalidierung erstellt (vgl. [Sch14]). Für die Berechnung des SVM-Modells konnte dort eine Bibliothek verwendet werden. Das war innerhalb dieses Projektes nicht mehr möglich. Das Script musste daher bearbeitet werden um das entwickelte C++ Programm mit den gewünschten Einstellungen aufrufen zu können.

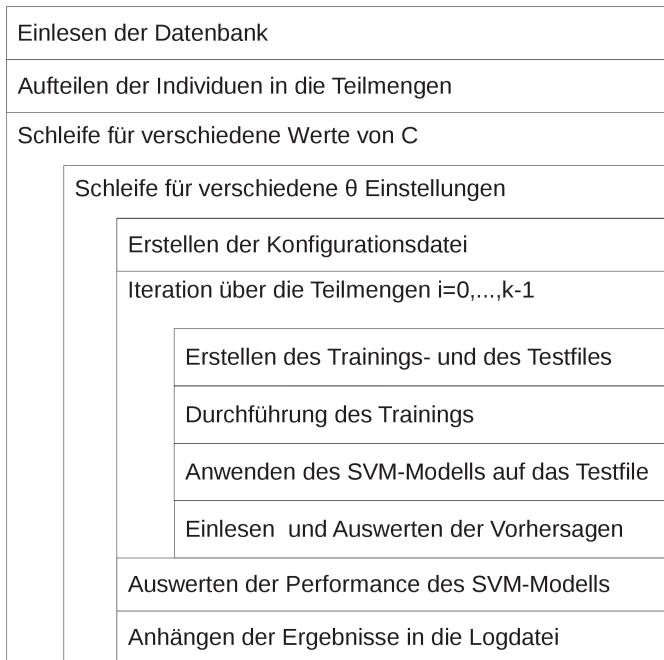
Die Struktur des Skriptes ist in Abbildung 16 dargestellt. Das Testskript kann grob in mehrere Bereiche aufgeteilt werden.

### 4.2.1 Einlesen der Datenbank

Die Daten liegen in einer ASCII-Datei vor. Eine Zeile beinhaltet dabei die Informationen eines Individuums. Abgesehen von den Parametern des Members ist auch sein Eintrag der Zielfunktion von Interesse. Dieser Wert wurde von der Prozesskette berechnet (vgl. Kapitel 1.4). Lag jedoch ein Abbruch der Prozesskette vor, beinhaltet der Eintrag einen Fehlercode  $> 10000$ . Da das Ergebnis der erfolgreich berechneten Zielfunktion solche Werte nicht erreichen kann wird dieser Eintrag genutzt, um die Klassenzugehörigkeit des Members zu

## 4 Tests

---



**Abbildung 16:** Der Ablauf des Skriptes, dargestellt als Struktogramm

bestimmen. Ist er  $> 10000$  teilt ihm das Skript die Klasse 0 (Prozesskettenabbruch) zu, sonst die Klasse 1 (erfolgreich).

### 4.2.2 Aufteilen in die Teilmengen der Kreuzvalidierung

Der Nutzer kann dem Skript beim Start in der Kommandozeile zwei Parameter übergeben, die die Größe und die Anzahl der benötigten Teilmengen bestimmen. Die erste Angabe stellt die Anzahl  $k$  der Teilmengen dar, die bei der Kreuzvalidierung genutzt werden. Die zweite Angabe betrifft die Anzahl  $l$  der Individuen, die bei jedem Trainieren des SVM-Modells eingesetzt werden.

Pro Training werden  $k-1$  Teilmengen eingesetzt. Die Größe einer Teilmenge ergibt sich daher aus  $\frac{l}{k-1}$ . Die Datenbank muss also mindestens  $\frac{l \cdot k}{k-1}$  Einträge enthalten. Um vergleichbare Ergebnisse zu produzieren soll das Training immer mit der gleichen Anzahl an Trainingsmembern durchgeführt werden. Zusätzlich verfügbare Member werden daher nicht verwendet und gelöscht.

Wie bereits in 4.1.1 bemerkt findet die Zuordnung auf die Teilmengen zufällig statt. Eine

Zuteilung auf Basis der Reihenfolge würde das Ergebnis verzerrten.

### 4.2.3 Parametereinstellungen

Dem SVM-Algorithmus müssen einige Werte vorgegeben werden. Da der Algorithmus die Werte aus einer Konfigurationsdatei einliest kann das Skript diese nicht direkt übergeben. So wird zunächst eine temporäre Konfigurationsdatei erstellt.

Um verschiedene Werte automatisch testen zu können befindet sich an diesem Ort eine Schleife, die das restliche Skript mit verschiedenen Einstellungen starten kann.

### 4.2.4 Durchführen der Kreuzvalidierung

Der SVM-Algorithmus muss für jede Einstellung  $k$ -mal eingesetzt werden. Dazu wird wieder eine temporäre Datei angelegt, die alle Trainingsmember enthält. Das sind die Member aller Teilmengen außer der aktuell untersuchten.

Ein Terminalaufruf für den Start des SVM-Algorithmus kann in Python über `os.system()` erfolgen.

Das gleiche gilt für die Auswertung der Testmember, die dafür zunächst in eine weitere Datei geschrieben werden. Sein Ergebnis schreibt der SVM-Algorithmus in ein weitere Datei.

Bei der Auswertung wird für jeden Member der aktuellen Testteilmenge gespeichert, ob es richtig erkannt wurde. Dazu wird zunächst die oben genannte Datei von Skript eingelesen und mit der tatsächlichen Klassenzugehörigkeit verglichen. Nach  $k$  Durchläufen war jedes Member einmal Teil der Testmenge.

Es werden verschieden Werte ermittelt und für die spätere Betrachtung in ein Logdatei geschrieben. Dazu zählt neben der Zeit, die für die Durchführung des Trainingsalgorithmus und für die Vorhersage benötigt werden, auch die Fehlerquote. Für die Zuordnung der ermittelten Werte zu den benutzten Einstellungen und den Eigenschaften der benutzten Trainingsmenge werden auch diese ausgeschrieben.

Die Logdatei beinhaltet pro Zeile:

Trainingsgröße:	Die Anzahl der Member, die pro Training zum Einsatz kamen.
Kategorie 0 (in %):	Bei schlechter Auswahl der $\theta$ kann es vorkommen, dass die Vorhersage immer 0 ist (die häufiger vorkommende Klasse). In diesem Fall ist der Anteil der Kategorie 0 gleich dem Anteil der korrekt bestimmten Member.
Trainingsdauer (in s):	Die Zeit, die das Training durchschnittlich benötigte. Gemittelt aus den $k$ Durchführungen.
Vorhersagendauer (in s/Sample)	Die Zeit, die ein Sample brauchte um vorhergesagt zu werden.

Optional, wenn sie fest vom Benutzer eingestellt wurden, kommt zusätzlich hinzu:

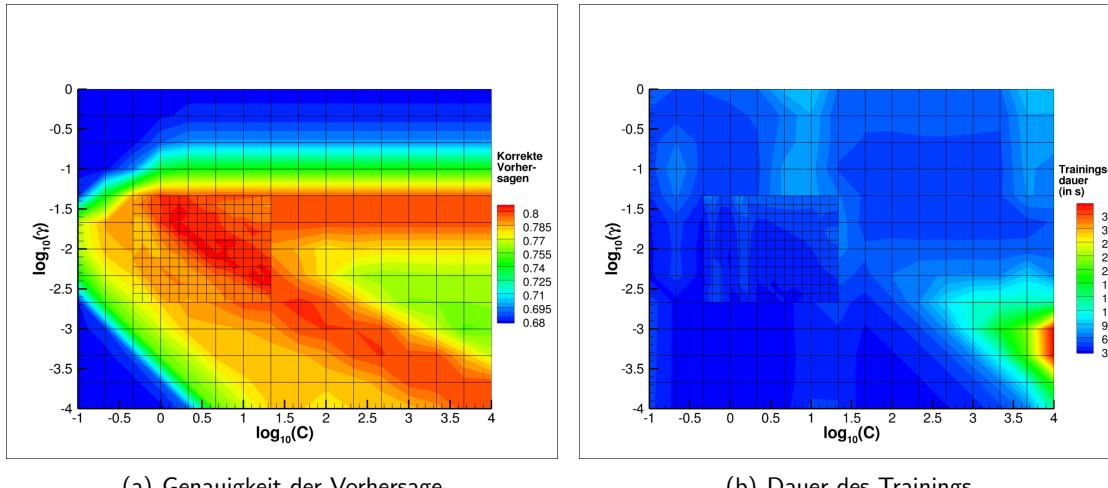
- C: Der Parameter, mit dem man die Eigenschaften der Soft-Margin beeinflussen kann.
- $\theta$ : Ein einziger Wert, der für alle  $\theta_1, \dots, \theta_l$  angewandt wird.

### 4.3 Test 1

#### 4.3.1 Beschreibung

In Test 1 wird möglichst viel von Nutzer eingestellt. Dadurch lassen sich die Ergebnisse leicht mit dem Resultat der gängigen SVM-Bibliotheken vergleichen. Die Tests mit der Bibliothek libSVM wurden bereits außerhalb dieser Arbeit durchgeführt [Sch14]. Die besten Ergebnisse lagen dabei bei einer Zuverlässigkeit der Vorhersage von  $\approx 80\%$ . Diese konnten in einem Bereich um  $C = 10^{\frac{1}{3}} = 2.154$  und  $y$  bzw.  $\theta_1 = \dots = \theta_l = 10^{-\frac{5}{3}} = 0.021$  erzielt werden (vgl. Abb. 17).

Die Tests der SVM-Bibliothek wurden mit 1872 Mustern und einer 5-fachen Kreuzvalidierung durchgeführt. Um die besser vergleichen zu können, wurden die gleichen Randbedingungen für die Tests dieses Projektes gewählt.



**Abbildung 17:** Testergebnisse mit der Bibliothek libSVM. Die Knoten des Gitters sind hierbei die Messpunkte *Bildquelle*: [Sch14]

### 4.3.2 Beobachtungen

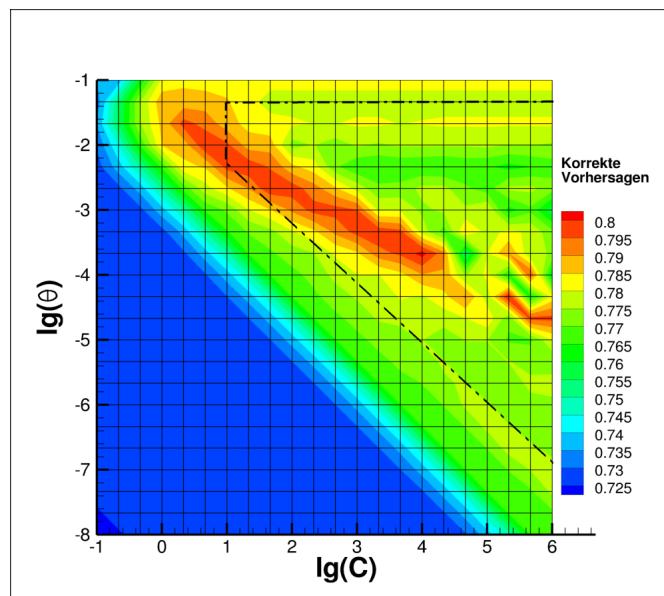
Vergleicht man Abbildung 17(a) mit den Ergebnissen 18(a) der Eigenentwicklung kann man, trotz anderem Bildausschnitt, ein Pattern wiedererkennen. Das gesamte Pattern scheint jedoch in Richtung der x-Achse verschoben. So beginnt der Peak erst bei  $C > 10^{\frac{1}{2}}$ . Mit einem Trefferquote von  $\approx 80\%$  liefert er jedoch gute Vorhersagen wie die Bibliothek.

Um möglichst gut Einstellungen auszuwählen sollte man nicht nur die Qualität der Vorhersagen betrachten. Auch die Dauer des Trainings spielt hier eine Rolle. Das diese auch von den  $\theta$  und  $C$  Einstellungen abhängt verdeutlicht das zweite Diagramm in Abbildung 18. Während der Algorithmus für viele Einstellungen nur kurze Zeit von  $< 20\text{s}$  benötigt, erhebt sich bei großen  $C$  ein Plateau in dem  $\approx 70\text{s}$  lang gerechnet wird. Zeichnet man die recht steilen Flanken dieses Plateaus in Diagramm (a) ein erkennt man, dass ein großer Teil des Peaks auf diesem unerwünschten Plateau liegt. Als gute Einstellung für ein Training mit dieser Datenbank kommen sie daher nicht in Frage, da eine gleiche Qualität der Vorhersage auch mit weniger Rechenzeit durchgeführt werden kann. Anbieten würde sich zum Beispiel  $C = 10^{\frac{1}{3}} \approx 2.1544$  und  $\theta_1 = \dots = \theta_l = 10^{-\frac{5}{3}} \approx 0.0215$

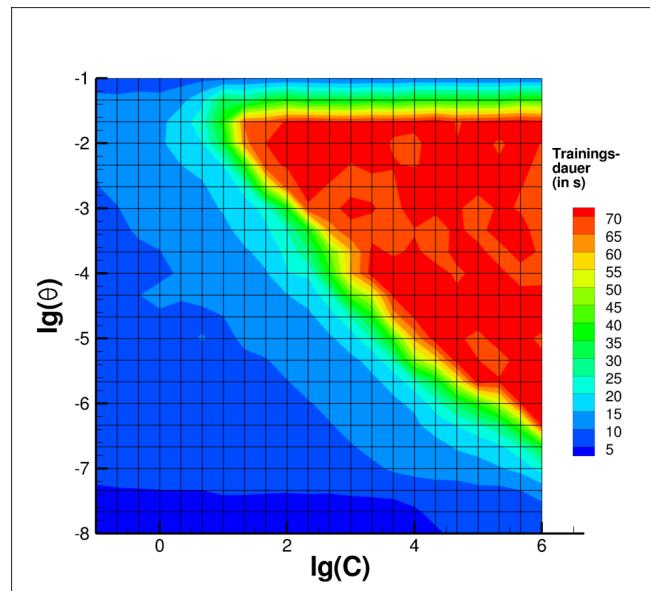
Da die Tests der Bibliothek und des Projektes auf verschiedenen Maschinen durchgeführt wurden, lassen sich die Rechenzeiten nicht direkt vergleichen. In der Bibliothek besteht jedoch

## 4 Tests

---



(a) Untersuchung der Qualität der Vorhersage



(b) Untersuchung des Aufwandes des Trainings

**Abbildung 18:** Testergebnisse des Projektes mit fest eingestellten Parametern. Die schwarze Linie in (a) verdeutlicht die Umrisse des Plateaus aus (b)

keine so große Abhängigkeit zu den gewählten Einstellungen (vgl. Abb. 17(b)).

### 4.3.3 Fazit

Test 1 hat gezeigt das mit dem in dieser Arbeit entstandenen Programm ein ähnlich gutes Ergebnis erzielen kann wie mit einer verbreiteten Bibliothek. Wieso sich das Pattern dabei auf der  $C$ -Achse zu höheren Werten verschoben hat wurde nicht untersucht. Da die Diagrammachsen logarithmisch dargestellt werden entspricht das einem konstanten Faktor. Es ist denkbar das dieser Faktor bei der Ausführung der Bibliothek auftritt. Da  $C$  aber eine dimensionslose Größe ohne direkten Bezug zu Eigenschaften der Trainingsmenge handelt, verändert sich für einen Benutzer nichts. Es macht lediglich den Vergleich zwischen libSVM und dem erstellten Programm schwieriger.

Die Unterschiedliche Form des Trainingsdauer-Diagramms ist vermutlich auf verschiedene Ansätze für das Lösen des Quadratischen Problems zu begründen.

## 4.4 Test 2

### 4.4.1 Beschreibung

Wie in Test 1 sind auch in Test 2 die Einstellungen bereits zu Beginn festgelegt. Untersucht werden soll ob die Qualität der Vorhersage in Abhängigkeit mit der Anzahl der Trainingsmember steigt. Auch die Entwicklung der benötigten Laufzeit ist dabei interessant.

Die Einstellungen  $C = 10^{\frac{1}{3}} \approx 2.1544$  und  $\theta_1 = \dots = \theta_l = 10^{-\frac{5}{3}} \approx 0.0215$  wurden dafür aus Test 1 übernommen.

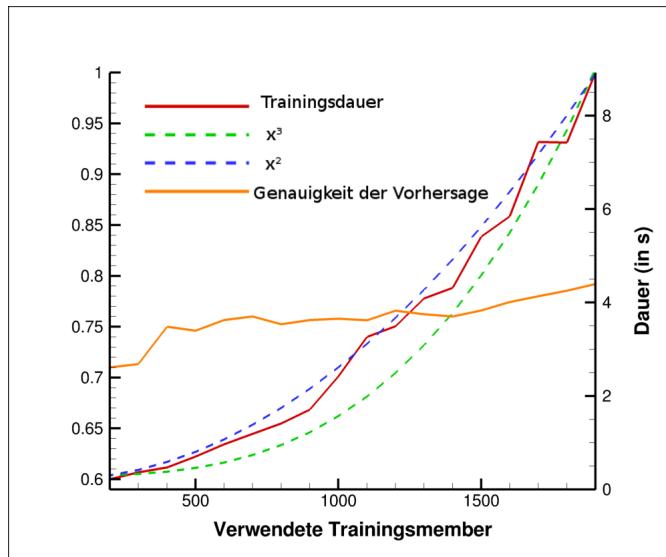
Für diesen Test musste von dem Ablauf auf Abbildung 16 abgewichen und das Testskript leicht angepasst werden

### 4.4.2 Beobachtungen

Bereits mit wenigen Hundert Membern in der Trainingsmenge erreicht die Support Vector Machine eine Trefferquote von  $> 75\%$  (vgl. Abb. 19). Steigert man die Anzahl weiter, verbessert sich die Anzahl der korrekten Vorhersagen weiter leicht.

Die Dauer des Trainings nimmt dagegen deutlich zu. Dabei verläuft der Anstieg weder exakt

kubisch (grün) noch quadratisch (blau).



**Abbildung 19:** Die Ergebnisse von SupportVektor Machines bei variierender Größe der Trainingsmenge

### 4.4.3 Fazit

Mit der relativ kleinen Datenbank war es nicht möglich weitere Aussagen über den Aufwand der SVM zu fällen. Wie auch in der Dokumentation von libSVM [CL02] angegeben, scheint er im Bereich zwischen  $O(l^2 \times d)$  und  $O(l^3 \times d)$  zu liegen.

Die generelle Aussage das das Hinzufügen von Trainingsdaten immer zu einer Verbesserung des erstellten Modells führt, konnte in diesem Test im Trend gezeigt werden. Der Gewinn ist jedoch nicht immer so groß, das es den vielfachen Aufwand rechtfertigt. Auf Basis des Ergebnis, das auch mit der 1000 Membern fast das Optimum erreicht wurde, wurde in den weiteren Test darauf verzichtet, alle Member der Datenbank mit einzubeziehen.

## 4.5 Test 3

### 4.5.1 Beschreibung

Test 3 ist der erste Test in dem das Qualitätskriterium  $\frac{R^2}{M^2}$  eingesetzt wird. Es wird von dem Programm dazu verwendet einen optimalen Wert für  $\theta_1 = \dots = \theta_d$  zu finden.  $C$  wird weiterhin von dem Benutzer gewählt.

Dieses Experiment zeigt ob das Kriterium eine gute Möglichkeit ist, den Kernel einzustellen. Die besten Ergebnisse sind dann zu erwarten, wenn das Programm selber den Wert für  $\theta$  findet, der auch in Test 1 bei der Kreuzvalidierung die besten Vorhersagen gefunden hat.

### 4.5.2 Beobachtung

In Abbildung 20 ist die Qualität der Vorhersage in Abhängigkeit des Bestrafungsfaktors  $C$  dargestellt. Es zeigt sich deutlich, dass für  $C < 10^{\frac{1}{2}} \approx 3.163$  keine guten Vorhersagen getroffen werden konnten. Der Grund dazu wird offensichtlich, wenn man die Ergebnisse von Test 1 beachtet. Auch hier konnten, unabhängig von dem gewählten  $\theta$ , keine guten Vorhersagen in dem Bereich mit  $C < 10^{\frac{1}{2}}$  getroffen werden (vgl. Abb. 18(a)).

Wie auch in Test 1 steigt die erreichbare Trefferquote in Bereichen mit einem höheren  $C$  jedoch an, sodass hier gute Ergebnisse erzielt werden können. Interessant ist hierbei der Wert des gefundenen  $\theta$  in Abhängigkeit von dem voreingestellten  $C$ . Vergleicht man den sich daraus ergebenden Pfad in Abbildung 21 mit dem im Hintergrund dargestellten Ergebnis aus Test 1 wird deutlich, dass die gefundenen  $\theta$  zunächst zwar gut den Peak finden, im weiteren Verlauf sich aber von ihm entfernen.

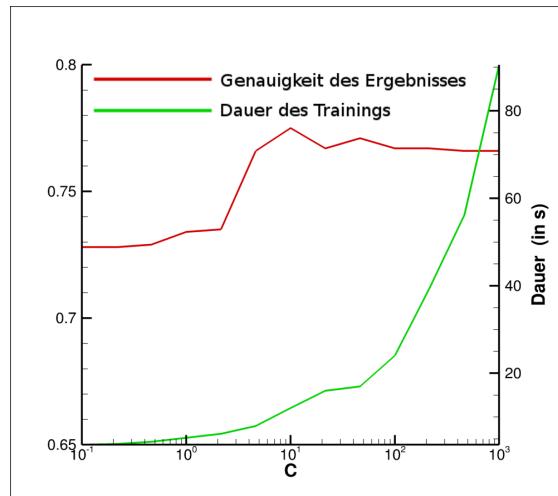
### 4.5.3 Fazit

Test 3 zeigt das das Qualitätskriterium eine gute Näherung ist. Für große  $C$ -Werte nimmt die Genauigkeit jedoch ab, lediglich am Anfang liegt der gefundene  $\theta$ -Wert genau auf dem Peak.

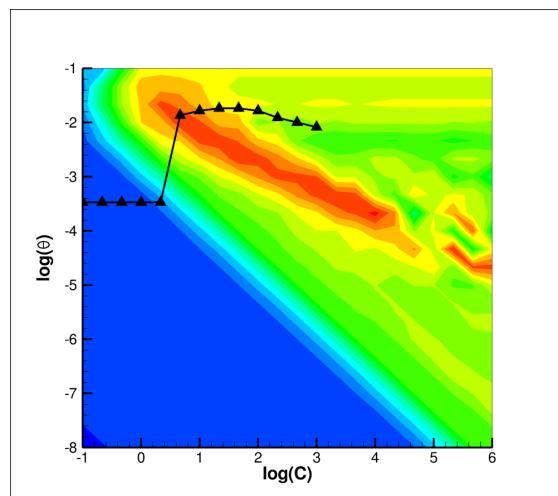
Der Einsatz des Kriteriums hat sich dennoch bewährt. Die Rechenzeit ist zwar größer als bei der Erstellung der klassischen SVM aus Test 1. Der Nachteil ist aber nicht so groß als das es langsamer ist als der rechenintensive Grid-Search, der dort notwendig ist.

## 4 Tests

---



**Abbildung 20:** Test 2: Die Qualität der Vorhersage und die Trainingsdauer in Abhängigkeit des voreingestellten  $C$



**Abbildung 21:** Test 3: Der nach dem Kriterium aus Kapitel 2.1.6 automatisch ausgewählte Wert für  $\theta$  (schwarz). Im Hintergrund befindet sich die Darstellung der Ergebnisse aus Test 1. Für kleine Werte von  $C$  können noch keine sinnvolle  $\theta$ -Werte gefunden werden. Nach einer Steigerung wird der Peak zunächst sehr genau getroffen, der Pfad driftet dann jedoch aus dem guten Bereich ab.

## 4.6 Test 4

### 4.6.1 Beschreibung

Der Einfluss verschiedener Parameter auf die Klassenzugehörigkeit kann sehr unterschiedlich sein. Ein klassischer Kernel mit einem einzigen Parameter  $\gamma$  (bzw.  $\theta_1 = \dots = \theta_d$ ) berücksichtigt das jedoch nicht. Durch den Einsatz des leicht zu berechnendem Kriteriums  $\frac{R^2}{M^2}$  kann auch ein Kernel mit einem hohen Freiheitsgrad angepasst werden.

in Test 4 werden die Optionen des Programms so eingestellt, dass es automatisiert einen möglichst guten  $\theta$ -Vektor findet. Der Kernel wird dadurch so eingestellt, dass ein Optimum des Kriteriums gefunden wird.

Der Faktor  $C$  des Bestrafungsterms wird auch in diesem Test von Nutzer bestimmt. Es wurden verschiedene  $C$  genutzt, um den Einfluss dieses Terms zu untersuchen.

Da das Training teilweise viel Zeit in Anspruch nahm, wurde in diesem Training von der 5-fachen Kreuzvalidierung abgewichen und stattdessen eine 2-fache Kreuzvalidierung eingesetzt. Die Größe der Trainingsmenge ist 800 Member.

### 4.6.2 Beobachtung

Die Diagramme in Abbildung 22 zeigen den Verlauf der Qualität und der Laufzeit in Abhängigkeit von  $C$ . Sie verdeutlichen eindrücklich, dass in der Optimierung durch numerische Methoden Probleme in der Konvergenz auftreten können.

Es wurden zwei verschiedene Algorithmen getestet, um den optimalen Kernel zu finden. Diagramm (a) stellt die Ergebnisse dar, die mit dem Quasi-Newton Verfahren erzielt wurden. Es zeigt sich, dass nicht immer ein sinnvoller  $\theta$ -Vektor gefunden werden konnte. In solchen Fällen konvergierte  $\theta_i \quad \forall i \in \{1, \dots, l\} \rightarrow 0$  und die Vorhersage war immer die häufiger auftretende Klasse. Der Grund für dieses Verhalten ließ sich nicht lokalisieren. Das Quasi-Newton-Verfahren ist jedoch für seine numerische Instabilität bekannt.

Konnte ein Minimum erfolgreich gefunden werden, waren die Ergebnisse jedoch sehr gut. Mit einer Trefferquote von  $\approx 83\%$  war die Vorhersage besser als bei der Nutzung eines  $\theta$ -Vektors mit gleichen Einträgen. Die Einträge der einzelnen  $\theta_i$  erstreckten sich über den gesamten Bereich zwischen 0 und 1.

Für hohe Werte von  $C$  änderte sich das trainierte Modell überraschenderweise nicht mehr. Es

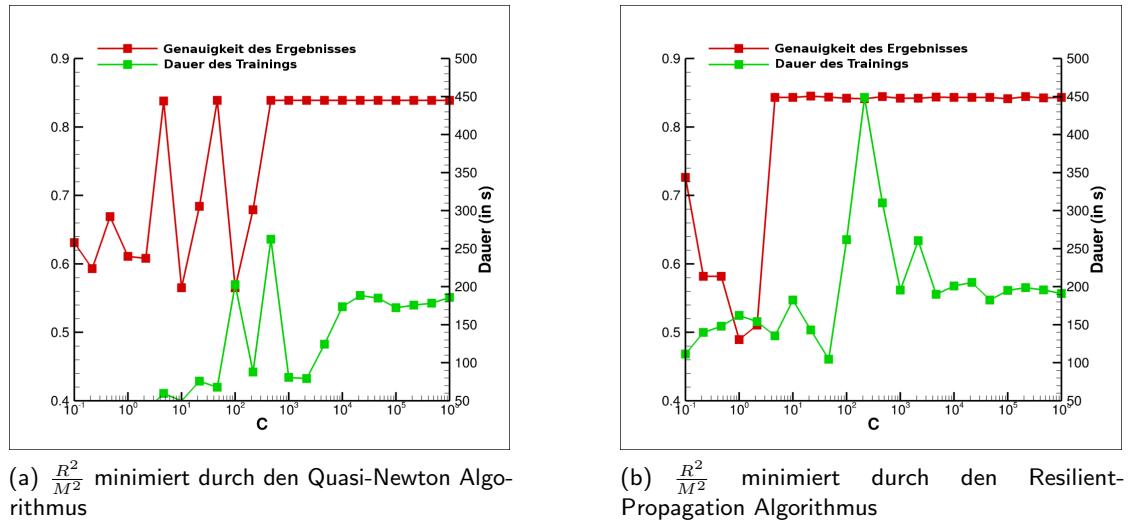
## 4 Tests

---

wurde immer der gleiche  $\theta$ -Vektor gefunden. Es gab auch in  $\alpha$  keinen Eintrag mit  $\alpha_i = C$ . Das hätte Trainingsmembern entsprochen, die selber nicht auf der richtigen Seite der trennenden Hyperebene liegen. Vor allem in komplexen Problemstellungen erhöht das normalerweise das Abstraktionsvermögen und verbessert die Vorhersage.

Es wurde ein weiterer Test mit dem Resilient-Propagation Algorithmus durchgeführt (b). Diese Untersuchung soll zeigen, ob die Unstetigkeiten hier nicht mehr auftreten. Der Resilient-Propagation Algorithmus ist in der Regel numerisch stabiler als Quasi-Newton.

Die extremen Schwankungen benachbarter  $C$ -Werte trat hier nicht auf. Insgesamt scheint das Ergebnis stabiler als bei Benutzung des Quasi-Newton Algorithmus. Kleinere  $C$ -Werte sorgen schon zuverlässig für ein gutes Ergebnis von  $\approx 84\%$ . Es treten jedoch auch hier kaum bis keine  $\alpha$  Einträge mit  $\alpha_i = C$  auf.



**Abbildung 22:** Testergebnisse für einen  $\theta$ -Vektor in dem die Einträge unabhängig voneinander sind. (a) und (b) verwenden verschiedene Ansätze für die Optimierung

### 4.6.3 Fazit

Test 4 bestätigt die Vermutung das die Verwendung von einem Kernel mit einem hohen Freiheitsgrad zu einem besseren Ergebnis führen kann. Das gewählte Kriterium von  $\frac{R^2}{M^2}$  ist geeignet, einen solch komplexen Kernel zu kalibrieren.

## 4 Tests

---

Überraschend ist das Verhalten bei großen  $C$ -Werten. Obwohl der Soft-Margin Ansatz genutzt wird, wird eine Hyperebene gefunden die die Klassen genau trennen kann. In der Literatur ist der Soft-Margin Ansatz als Mittel der Wahl für komplexe Sachverhalte beschrieben. Für klassische Kernel konnte in Test 1 und 2 bestätigt werden, dass die Qualität der SVM bei  $C \rightarrow \infty$  nach erreichen eines Hochpunkts wieder abnimmt. Wenn jedoch schon bei dem Hochpunkt  $\alpha_i < C \quad \forall i \in \{1, \dots, l\}$  gilt, ändert eine weitere Erhöhung von  $C$  das trainierte Modell nicht.

Ob das bei komplexen Kernel immer gilt bedarf weiterer Untersuchungen.

Test 4 zeigt eindrucksvoll, das durch das hinzufügen eines weiteren Optimierers auch das Modell komplexer werden lässt. Um immer gute Ergebnisse zu liefern müssen dann auch Wechselwirkungen zwischen dem Optimierungsproblem und den verwendeten Minimiereralgorithmen untersucht und beachtet werden.

## 5 Fazit und Ausblick

Ziel dieser Arbeit war es, einen geeigneten Klassifikator für den Einsatz innerhalb von AutoOpti zu finden und zu implementieren. Schließlich sollte validiert werden, wie gut die Vorhersagen und die Performance des Algorithmus sind.

Ausgewählt und recherchiert wurde hierzu der Support Vector Machine-Algorithmus. Bei der Recherche wurde ein Qualitätskriterium ermittelt, das eine weitere Verbesserung des SVM-Algorithmus erlaubt. Dieser verbesserte Algorithmus konnte erfolgreich in das bestehende Projekt eingebunden werden.

Die Tests konnten das Ergebnis validieren. Sie haben jedoch auch Raum für Verbesserungen offenbart, die es weiter zu untersuchen gilt.

So sind mit einem komplexen RBF-Kernel mit hohem Freiheitsgrad wie erwartet bessere Vorhersagen möglich als mit dem klassischen Ansatz. Eine zuverlässige Konvergenz ist jedoch nur mir dem stabileren, jedoch langsameren Resilient-Propagation-Algorithmus möglich. Der effizientere Quasi-Newton-Algorithmus hat, abhängig von dem verwendeten Bestrafungsterm-Faktor  $C$ , nicht immer ein Optimum gefunden.

Überraschend ist die Eigenschaft des komplexen Kernels, das ein mit ihm erstelltes Modell auf die Möglichkeit der Soft-Margin verzichtet. Er erlaubt das finden einer Trennebene, die die Klassen „sauber“ trennt und in dem gewählten Beispiel trotzdem gute Ergebnisse liefert. Dies widerspricht der Literatur, die einer saubereren Trennung den Hang zum Overfitting nachsagt.

Muss der ideale Kernel erst gefunden werden ist die in dieser Arbeit entwickelte Umsetzung zur Zeit noch langsamer als das Kriging-Verfahren. Der Vorteil des SVM-Algorithmus ist jedoch, das nicht immer der ideale Kernel gefunden werden muss. So waren selbst mit dem klassischen Kernel bereits gute Ergebnisse von  $\approx 80\%$  bei meist sehr kurzen Laufzeiten möglich. Dadurch ist es denkbar, bei einer Optimierung den idealen Kernel lediglich zu Beginn zu finden, um ihn danach weiter zu verwenden.

Das zukünftige Ziel ist es, den SVM-Klassifikator als Teil von AutoOpti zu etablieren.

## 5 Fazit und Ausblick

---

Die Möglichkeit, aus dem Abstand zur Hyperebene die Wahrscheinlichkeit der Klassenzugehörigkeit abzuschätzen wurde noch nicht implementiert. Erst dadurch wird jedoch eine sinnvolle Einbindung in eine laufende Optimierung möglich.

Zunächst gilt es jedoch die gefundenen Effekte weiter zu analysieren. Dazu gehört die Untersuchung von  $C$  auf die Konvergenz mit dem Quasi-Newton Verfahren. Außerdem sollte geprüft werden, ob der gefundene Effekt eines komplexen Kernels immer gilt. Liefert dieser für hohe  $C$  immer gute Ergebnisse, konnte auf den Einsatz der Soft-Margin sogar verzichtet werden.

Des Weiteren ließe sich die Geschwindigkeit weiter optimieren. Es wurden zum Beispiel noch nicht alle Möglichkeiten der Parallelisierung genutzt, hier besteht ein großes Optimierungspotential. Theoretisch ist es für große Datensätze möglich, auch mit der Optimierung des Kernels schneller als der Kriging-Algorithmus zu werden.

Der Algorithmus konnte somit noch nicht in einer laufenden Optimierung eingesetzt werden. Den Vorteil, den ein solcher Einsatz mit sich bringt, gilt es zu untersuchen und zu dokumentieren.

# Literaturverzeichnis

- [CL02] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines – documentation. <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>, 2002.
- [CST00] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 edition, 2000.
- [CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.
- [CVBM02] Olivier Chapelle, Vapnik Vladimir, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–159, 2002.
- [Har08] Lothar Harzheim. *Strukturoptimierung*. Verlag Harry Deutsch, 2008.
- [HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2009.
- [Mar03] Florian Markowetz. Klassifikation mit supporting vector machines. [http://lectures.molgen.mpg.de/statistik03/docs/Kapitel\\_16.pdf](http://lectures.molgen.mpg.de/statistik03/docs/Kapitel_16.pdf), 2003.
- [Nie83] Heinrich Niemann. *Klassifikation von Mustern*. Springer-Verlag, 1983.
- [NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 1999.
- [Pla99] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*. MIT Press, 1999.
- [PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes, The Art of Scientific Computing. Third Edition*. Cambridge University Press, 2007.

- [Sch13] Andreas Schmitz. Entwicklung eines objektorientierten und parallelisierten Gradient Enhanced Kriging Ersatzmodells, 2013.
- [Sch14] Timo Schumacher. Bericht zum Modul Praxis III, 2014.
- [SS02] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.
- [TRA12] Abteilung für Numerische Methoden. [http://www.dlr.de/at/DesktopDefault.aspx/tabcid-1519/](http://www.dlr.de/at/desktopdefault.aspx/tabcid-1519/), 2012.
- [Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.
- [VAR14] Christian Voß, Marcel Aulich, and Till Raitor. Metamodel assisted aeromechanical optimization of a centrifugal compressor. *15th International Symposium on Transport Phenomena and Dynamics of Rotating Machinery. ISROMAC-15*, 2014.
- [VC00] Vladimir Vapnik and Olivier Chapelle. Bounds on error expectation for support vector machines. In *Neural Computation*, volume 12, pages 2013–2036. MIT Press Journals, 2000.
- [Yil07] Emre Alper Yildirim. Two algorithms for the minimum enclosing ball problem. [http://www.optimization-online.org/DB\\_FILE/2007/05/1654.pdf](http://www.optimization-online.org/DB_FILE/2007/05/1654.pdf), 2007.