Beschleunigung der Anwendung von Kernmethoden

Adaptive Minimierung der Support Vektor Menge

vorgelegt von
Volker Rölke

Diplomarbeit

Mai 2001

Lehrstuhl für Kognitive Systeme
des Instituts für Informatik und Praktische Mathematik
der Technischen Fakultät der
Christian-Albrechts-Universität zu Kiel
Prof. Dr. G. Sommer
Betreuer
Dr. J. Pauli

for Bird

ZUSAMMENFASSUNG

Diese Arbeit beschäftigt sich mit der Beschleunigung der Anwendungsphase von Kernmethoden wie Support Vektor Maschinen oder Kernel PCA. Bei einer Anwendung handelt es sich um die Berechnung einer gewichteten Summe von Skalarprodukten mit bereits ermittelten Vektoren, die i.A. Support Vektoren genannt werden. Im Gegensatz zu einigen bisherigen Arbeiten setzen die hier entwickelten Algorithmen erst nach der Lernphase der jeweiligen Kernmethode an. Dadurch sind die Verfahren von der Kernmethode unabhängig einsetzbar. Desweiteren wird von der bisher üblichen Zielstellung die Menge der Support Vektoren statisch zu verringern Abstand genommen. Der neue Ansatz ermöglicht eine adaptive Anpassung der Support Vektor Menge an das aktuelle Eingabeelement. Es wird erst in der Anwendungsphase entschieden, ob das Skalarprodukt mit einem bestimmten Support Vektor zur Berechnung der gewichteten Summe notwendig ist. Diese Entscheidung geschieht unter der Hinzunahme von Vorwissen über die Relationen der Support Vektoren untereinander. Die eigentliche Menge der Support Vektoren wird dabei nicht manipuliert und die Kernfunktion bleibt frei wählbar. Die drei neuen Algorithmen, welche in dieser Arbeit entwickelt werden, sind auf realen Text- und Bilddatensätzen getestet und ermöglichen für die nicht modellhaften Anwendungen eine Zeitersparnis von bis zu 60% .

INHALTSVERZEICHNIS

NOTATION

$\mathbb{N}$	Natürliche Zahlen
$\mathbb{R}$	Reelle Zahlen
P	Wahrscheinlichkeitsverteilung
ϵ, η	kleine reelle Zahlen, $\epsilon, \eta \geq 0$
r	Radius, $r > 0$
S	Eine dem Lernverfahren zu Grunde liegende Funktionenfamilie
Λ	Eine zu S gehörende Parametermenge
α	Parametervektor, Element aus Λ
h	VC-Dimension
ϕ	Konfidenz
c	Kostenfunktion in Ω
R	Risiko
R_{emp}	empirisches Risiko
α_i	Parameter aus α
f	Die zu lernende Funktion, Element aus S
Ω	Eingabemenge von f
N	Dimension von Ω
Υ	Ausgabemenge von f
$\mathbf{x}, \mathbf{a}$	Elemente aus Ω
y	Element aus Υ
L	Anzahl der Trainingsdaten
N_S	Anzahl der Support Vektoren
E	Hyperebene
$\mathbf{\Psi}$	Normalenvektor von E
b	Reeller Parameter von E

A	Reziproke Dicke von E, $A \geq 0$
v	Eigenvektor
λ	Eigenwert
O	Kovarianzmatrix
M	Kompakte Menge
ψ Eigenfunktion	
C	Kostengewichtung, $C > 0$
ξ	Kostenvariable, $0 \leq \xi \leq C$
F	Merkmalsraum
N_F	Dimension von F
$\bar{\mathbf{x}}$	Element aus F
d	Grad des Momentenraumes
Φ	Transformation von Ω nach F
k	Kernfunktion für das Skalarprodukt in F
K	Auswertungsmatrix von k auf den SVen
τ	Zu minimierender Term
K_{max}	Konstante zur Abschätzung von k
d	Distanzfunktion in Ω (Innere Kernfunktion)
g	Äußere Kernfunktion, angewendet auf d
V	Menge der transformierten SVen
F_S	Der durch V aufgespannte Unterraum von Ω
$\mathbf{x}_{perp}$	Orthogonaler Anteil eines Vektors
$\mathbf{x}_{\vert\vert}$	Paralleler (lin. abh.) Anteil eines Vektors
$\Phi(\mathbf{x}_i)_\perp$	Orthogonaler Support Vektor
$\widetilde{\Phi(\mathbf{x}_i)}$	Orthonormaler Support Vektor
B	Menge der orthonormalisierten SVen, Basis von V
N_B	Anzahl der Elemente von B, Dimension von F_S
ϱ	Länge eines SV
$\tilde{\alpha}_i$	Koeffizient der Darstellung von Ψ in B
Θ	Ergebnis des Skalarproduktes von $\Phi(\mathbf{x})$ mit einem Element aus B, Koeffizient der Darstellung von $\Phi(\mathbf{x})$ in B
$sum_{(i,n)}$	Abkürzung der Summe (1.1) von i bis n
$\widetilde{sum}_{(i,n)}$	Abkürzung der Summe des Orthonormalverfahrens
q	Anzahl der zur Summation verwendeten Elemente von B
q_{max}	Effizienzgrenze von q
t	Zeitbedarf eines Verfahrens
$\#$ SV	Normierte Anzahl der Support Vektoren
VC-DIM	Normierte Schätzung der VC-Dimension

1. EINLEITUNG

1.1 Aufbau der Arbeit

Nachdem in diesem Abschnitt der Aufbau der Arbeit erläutert wird, folgt in dem zweiten Teil der Einleitung die Erläuterung der Motivation des Themas und die Zielstellung für die Neuentwicklung. Das erste Kapitel schließt dann mit der Einbettung in die Liste bisheriger Veröffentlichungen.

In Kapitel 2 werden die Grundlagen von Kernmethoden erläutert und die bekanntesten Vertreter so weit vorgestellt, wie es zum Verständnis der neuen Verfahren nötig ist. Im darauf folgenden Kapitel 3 werden die neuen adaptiven Algorithmen zur zeitlichen Effizienzsteigerung hergeleitet. Die Ergebnisse der Experimente dieser Methoden sind in Kapitel 4 zusammenfassend illustriert. Zum Schluß werden im letzten Kapitel 5 die wichtigsten Aussagen dieser Arbeit rekapituliert um dann noch offen gebliebene Aspekte zu diskutieren.

Im Anhang befinden sich die in C++ implementierten Algorithmen sowie die Tabellen der genauen Meßdaten.

1.2 Motivation und Zielstellung

Seit der Einführung von Support Vektor Maschinen (SVM) mit nichtlinearen Kernen ([20]) hat die Forschung große Fortschritte im Bereich der Kernmethoden gemacht. Verschiedene Verfahren zur Klassifikation, Regression und Merkmalsextraktion werden bereits intensiv in Gebieten wie dem Data-Mining verwendet. In dieser Arbeit wird ein neuer Ansatz zur Optimierung der zeitlichen Effizienz der Anwendungen solcher Verfahren vorgestellt. Die Berechnung eines *Reduced Set* [2] erzielt bereits sehr hohe Beschleunigungsraten. Hier soll eine neue Verfahrensklasse vorgestellt werden, die auch in komplexen Situationen, welche die Anwendung bisheriger state-of-the-art Techniken ausschließt, durchgeführt werden kann.
Bisherige Arbeiten haben ausschließlich eine *statische* Reduktion von Sup-

port Vektoren (SVen) zum Ziel. Im Gegensatz dazu soll hier die Menge der benutzten Support Vektoren *adaptiv* zum jeweiligen Eingabevektor reduziert werden. Die Entscheidung über die Relevanz eines SV geschieht also nicht mehr in der Trainingsphase sondern während der Laufzeit in der Anwendungsphase.

Auffallend ist, daß die Anwendungsvorschriften der meisten Kernmethoden, wie später in den Gleichungen (2.58),(2.59) sowie (2.72) zu sehen ist, in der Form

$$f_{\{\mathbf{\Psi},b\}}(\mathbf{x}) = \left(\sum_{i=1}^{N_S} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \right) - b \tag{1.1}$$

geschrieben werden können. Dabei handelt es sich um die Summe über die mit reellen Konstanten α_i gewichteten Auswertungen einer Kernfunktion auf dem Eingangsvektor $\mathbf{x}$ und fixen Vektoren $\mathbf{x}_i$. Die Konstanten α_i und die Menge der Vektoren $\{\mathbf{x}_i \ : \ i = 1..N\}$ werden in der Lernphase der jeweiligen Kernmethode bestimmt. Die Anwendung einer Kernmethode besteht also stets hauptsächlich aus der Berechnung von (1.1).
Selbstverständlich ist die Geschwindigkeit der Anwendung für einen effizienten Einsatz des jeweiligen Verfahrens ausschlaggebend. Für die Berechnung der Kernfunktion wird i.A. eine Distanzfunktion im Eingaberaum Ω angewendet. Die direkte Berechnung von Gleichung (1.1) benötigt normalerweise mindestens eine Laufzeit in $O(N_S \cdot N)$. Eine Beschleunigung kann durch die Reduktion der Dimension N des Eingaberaumes Ω oder durch die Verwendung weniger SVen N_S erreicht werden.

Eine Dimensionsreduktion kann sehr gute Ergebnisse erzielen ([22]), ist jedoch nicht für jede Aufgabe geeignet ([8], [10]). In dieser Arbeit wird versucht, die Anzahl der SVen zu verringern, so daß ein Teil der teuren Berechnungen der Kernfunktion k in Gleichung (1.1) eingespart werden können.
Es wird dabei vermieden, die Anzahl der SVen *statisch* zu verringern. Vielmehr wird die Entscheidung, welche SVen zur Berechnung von (1.1) beisteuern sollen, erst in der Anwendungsphase und somit *adaptiv* getroffen. Aus dieser Idee entwickelt sich eine ganze Familie neuer *adaptiver Verfahren* zur Beschleunigung der Anwendungsphase.

Weil die Verfahren erst nach der Lernphase einsetzen, sind sie von der jeweiligen Kernmethode unabhängig. Sie können für sämtliche Berechnungen der Form (1.1) angewendet werden und sind somit zur *Effizienzsteigerung von Support Vektor Maschinen, Kernel PCA* und anderen Kernmethoden

geeigent. Einer der Ansätze muß lediglich Einschränkungen auf die Art der verwendeten Kernfunktion vornehmen.
In den Verfahren wird kein a priori Wissen verwendet. Es werden weder Annahmen über eventuelle Beschaffenheiten der Eingangsdaten gemacht, noch werden neue Heuristiken verwendet.
Die Datensätze moderner Kernmethoden werden immer komplexer und umfangreicher. Zur Berechnung der Kernfunktion werden schon häufig komplizierte Algorithmen durchgeführt. Der klassische Datenvektor und sein Skalarprodukt taucht in neuen Anwendungen zur Bild- und Textanalyse kaum noch auf [10]. Die hier entwickelten Verfahren nehmen auf diese Begebenheiten Rücksicht, als daß die Ursprungsdaten in keinster Weise manipuliert werden. So können die Support Vektoren oder deren Repräsentationen unverändert in der Datenbank bleiben, auf die u.U. ohnehin nur lesend zugegriffen werden kann. Das Verfahren kann auf Daten in einer beliebigen Repräsentation arbeiten, solange mindestens die Kernfunktion gegeben ist.
Das Ergebnis der neuen Verfahren soll die Berechnung von (1.1) optimal approximieren.

1.3 Literarische Einbettung

Obwohl die Theorie der Kernmethoden als alternative Berechnungsvorschrift eines Skalarproduktes bereits seit Jahrzehnten bekannt ist, wurde das Verfahren erst seit 1995 zum Betrachtungsobjekt zahlreicher Forschungsarbeiten in dem Bereich des maschinellen Lernens. Vapnik legte mit seinen Arbeiten [20], [6] den Grundstein zu einer neuen Lernmaschine, der *Support Vektor Maschine* (SVM) zur Klassifikation bzw. Regression. Sie realisiert die Umsetzung der bereits früher entwickelten *Strukturellen Risikominimierung* [21]. Dieses Verfahren steht im Gegensatz zur *empirischen Risikominimierung* und kann einen minimalen mittleren Fehler auf einem Testdatensatz garantieren. Die Strukturelle Risikominimierung minimiert den tatsächlich zu erwartenden Fehler einer gelernten Funktion bei der Anwendung auf einem Testdatensatz. Das Verfahren ist sehr komplex, weil es eine komplizierte Struktur auf dem erlaubten lernbaren Funktionenraum voraussetzt. Daher ist die SVM in seiner ursprünglichen Version lediglich zum Erlernen einer linearen Funktion geeignet. Erst durch den Einsatz einer Kernfunktion und der damit verbundenen nichtlinearen Erweiterung des Modells können komplexe reale Probleme gelöst werden. Die Vorteile der so entwickelten SVM liegen in der potentiellen Bewältigung des Zeitproblems beim Rechnen in hochdimensionalen Räumen sowie der äußerst variablen Transformation, welche der Kernfunktion zu Grunde liegt.

Durch den Einsatz von Kernfunktionen in anderen Bereichen des maschinellen Lernens sind weitere Kernmethoden wie die *Kernel Principal Component Analysis* (Kernel-PCA) nach Schölkopf ([13], S.79ff) oder das Verfahren der *Novelty Detection* (vgl. [5]) im Bereich des unüberwachten Lernens entwickelt worden. Die Kernel-PCA erlaubt die Durchführung einer Hauptkomponentenanalyse entlang einer nichtlinearen Projektionslinie. Eine Novelty Detection bestimmt zunächst eine minimale Hülle um die Vektoren eines Datensatzes. Darauf basierend wird entschieden, ob es sich bei einem neuen Vektor um einen artfremden oder ähnlichen handelt. Diese Methode wird beispielsweise zur Alarmierung bei einer rapiden Änderung der Größen eines Systems verwendet. Desweiteren ist die Technik der SVM durch die Verarbeitung von Wahrscheinlichkeitsaussagen von Tipping in die *Relevance Vektor Maschine* (RVM) umformuliert ([19]) worden. Die Menge der Support Vektoren wird hier so gewählt, daß sie möglichst repräsentativ für die gesamte Datenmenge ist. Resultierend daraus werden bevorzugt Clusterzentren zur Klassifikation herangezogen. Durch die Repräsentativität ist es zusätzlich möglich neben der Klassifikationsentscheidung noch eine Wahrscheinlichkeitsaussage über die Korrektheit des Ergebnisses zu treffen.

Der Umgang mit großen Datenmengen hoher Dimensionalität birgt jedoch selbstverständlich immer noch Probleme, insbesondere ist die Zeitperformance ein wichtiger Gesichtspunkt. Typischerweise wird im maschinellen Lernen mit Bild- oder Textrepräsentationen mit hunderten oder gar tausenden von Dimensionen gearbeitet. Eine klassische Dimensionsreduktion ist nicht immer empfehlenswert ([8]) und kostet wertvolle Rechenzeit, so daß Lernverfahren für gewöhnlich mit hochdimensionalen Rohdaten umgehen müssen.

Viele Arbeiten haben sich bereits mit effizienzsteigernden Verfahren für den *Lernalgorithmus der Trainingsphase* beschäftigt ([18]S.73ff, [9], [7], [1], [11]). Den meisten dieser Arbeiten liegt dieselbe Idee zu Grunde. Ein Vektor, der im Endeffekt kein Support Vektor (SV) wird, hat auf die gelernte Funktion einer SVM keinen Einfluß. Ein solches Element der Trainingsmenge sollte möglichst früh erkannt und aus dem weiteren Lernprozeß ausgeschlossen werden.

Die *praktische Anwendung* einer Kernmethode besteht in der Regel aus der sehr häufigen Auswertung der fertig gelernten Funktion (1.1). Die Dauer einer solchen Anwendung kann typischerweise mehrere Zehntelsekunden betragen. Für den Einsatz auf einer großen Testdatenmenge wie dem Seitenangebot im Internet ist also eine Effizienzsteigerung der Anwendungsphase von großer Bedeutung.

Durch die Art der mathematischen Herleitung des Lernverfahrens einer SVM nach Vapnik besitzt eine fertig gelernte SVM unter bestimmten Umständen redundante Support Vektoren. Vektoren der Trainingsmenge, die von der gelernten Funktion nicht korrekt klassifiziert werden können, werden auf jeden Fall in die Berechnung von (1.1) übernommen. Diese Support Vektoren sind häufig nicht zwingend zur Berechnung der gelernten Funktion nötig und somit redundant. Diese Situation tritt immer dann auf, wenn die Trainingsmenge im Merkmalsraum nicht linear trennbar ist. Osuna und Girosi stellen in Ihrer Arbeit [12] zwei Methoden vor, um die Auswertungen der Kernfunktion mit gerade diesen redundanten Support Vektoren in der Berechnung von Gleichung (1.1) zu vermeiden. Der erste Ansatz benutzt eine zweite SVM zur Regression des Ergebnisses der gelernten Funktion mit einer wählbaren Genauigkeit. Für die zweite Methode wird eine Reformulierung des Lernproblems durchgeführt. Von der sonst üblichen Lösung über ein Lagrange-Polynom wird Abstand genommen, so daß die Multiplikatoren der besagten Support Vektoren nicht mehr auf die Kostenkonstante C gezwungen werden. Beide Verfahren liefern gute bis sehr gute Ergebnisse für die Menge der stark nichtseparierbaren Probleme.

In [15] zeigen Schölkopf und Smola eine neue Lernmethode für SVMs, die eine externe Kontrolle der Anzahl der Support Vektoren zuläßt. Durch einen Parameter ν wird die bisher freie Kostenvariable C bzw. ϵ der Kostenfunktion zur Fehlerbestrafung im klassischen Klassifikations- bzw. Regressionsansatz jetzt automatisch kontrolliert um die gewünschte Anzahl von Support Vektoren zu erhalten.

Der vermutlich bekannteste und am meisten benutzte Ansatz der *Reduced Set Methode* wurde von Burges in [2] entwickelt. Ausgegangen wird von einer fertig gelernten SVM. Die neue Support Vektor Menge mit einer vorher festgelegten Größe wird mit einem Gradientenabstiegsverfahren auf der Menge aller möglichen Support Vektoren optimiert. Für eine gegebene Menge von neuen Support Vektoren läßt sich nun die optimale Darstellung des Gewichtsvektors $\mathbf{\Psi}$ in einem Merkmalsraum bestimmen. Damit liefert die Abweichung der neuen Darstellung gegenüber der alten ein Optimierungskriterium für das Gradientenverfahren. Die neuen Vektoren werden künstlich generiert und sind weder Support Vektoren der bisherigen Funktionsdarstellung noch sind es überhaupt Elemente der Trainingsmenge. Wäre die nichtlineare Transformation Φ, die der jeweils verwendeten Kernfunktion zu Grunde liegt, bijektiv, so würde dieses Verfahren potentiell ein fehlerfreies Ergebnis mit nur *einem* Support Vektor liefern. Der Support Vektor wäre dann das Urbild des Gewichtsvektors $\mathbf{\Psi}$. Obwohl Φ in der Regel nicht bijektiv ist und zudem die Gefahr besteht, nur ein lokales Minimum zu finden, liefert das Verfahren bereits mit wenigen Support Vektoren eine gute Approximation der ursprünglichen Funktion ([4]).

Diese Methode kann durch die Kombination mit der Ausnutzung von Vorwissen über den betreffenden Datensatz noch verbessert werden. Die Arbeiten [14] und [16] zur Modellierung und Verarbeitung von Invarianzen der Daten fließen in die Entwicklung einer optimierten SVM von Burges und Schölkopf ein ([4]). Nach dem ersten Training der herkömmlichen SVM werden auf Grund von Invarianzannahmen künstliche Trainingselemente generiert. Die sog. *Virtuellen Support Vektoren* werden der ursprünglichen Trainingsmenge zugefügt. Mit dieser erweiterten Menge wird ein erneutes Training durchgeführt und die Menge der neuen Support Vektoren als Grundlage für das Reduced Set Verfahren benutzt.

Neuere Entwicklungen in der Arbeit mit Kernmethoden ([10]) zeigen jedoch zwei entscheidende Nachteile des Verfahrens auf. Zum einen müssen die Support Vektoren der SVM als konkrete Daten*vektoren* vorliegen, um einen Gradientenabstieg durchführen zu können. Komplexe Kernfunktionen arbeiten jedoch nur noch selten mit einer vektoriellen Repräsentation der als Support Vektoren ausgewählten Elemente der Trainingsmenge. Wenn diese Repräsentation überhaupt vektorieller Natur ist, kann ein Zugriff auf die einzelnen Dimensionen immer noch Probleme beherbergen. Die Einträge einer Wörterliste sind u.U. nicht stets in der gleichen Reihenfolge und schwer einer Dimension zuzuordnen. Der zweite und gravierendere Nachteil resultiert aus der Tatsache, daß ein Gradientenabstiegsverfahren auf einer neuen Support Vektor Menge durchgeführt werden soll. Es muß also die Möglichkeit gegeben sein, daß ein künstliches Element der Eingabemenge generiert werden und nach mathematischen Vorschriften verändert werden kann. Zum einen kann es dabei Probleme geben, überhaupt auf der Eingabemenge zu arbeiten, wenn sie sich etwa in einer Datenbank mit schreibgeschütztem Zugriff befindet. Auf der anderen Seite gibt es Kernfunktionen, die keine vektorielle Darstellung benutzen ([10]) und die Eingabemenge somit nicht vektorieller Natur ist. Bereits der Vergleich von Matrizen und Tensoren jedoch auch Objektrepräsentationen in Graphen und reine Textdaten beinhalten diese Problematik. Eine gezielte Manipulation der Daten wie bei einem Gradientenabstiegsverfahren kann bei solchen Problemstellungen nicht mehr realisiert werden. Ein weiteres prinzipielles Problem ist die feste Wahl der Anzahl der Support Vektoren. Durch die Wahl wird in der Regel ein systematischer Fehler in die Rechnung einbezogen, der im Nachhinein nicht zu beheben ist.

Die in dieser Arbeit vorgestellten Verfahren umgehen die angesprochenen Probleme weitestgehend und öffnet den Bereich der Echtzeitanwendungen für eine Reihe moderner Kernmethoden. Es handelt sich um adaptive Ansätze, mit denen die Anzahl der zu berechnenden Kernfunktionsauswertungen erst in Abhängigkeit vom jeweiligen Eingabeelement bestimmt wird. Das daraus resultierende dynamische Verhalten bietet einen optimalen Rechnungsverlauf für jeden Einzelfall und muß nicht wie bisher global auf der Trainingsmenge optimiert werden. Der zugelassene Fehler ist stets frei wählbar und wird nicht in der Lernphase systematisch einbezogen. Die Darstellung der Support Vektoren wird nicht verändert, so daß auch Kernfunktionen behandelt werden können, die auf nichtvektoriellen Elementen arbeiten. Der Lernschritt besteht hauptsächlich in der *Extraktion von Beziehungen* der Support Vektoren untereinander. Das Wissen über die Konstellation der Support Vektoren im Merkmalsraum wird dann in der Anwendungsphase ausgenutzt.

2. GRUNDLAGEN

Zusammenfassung

In diesem Kapitel werden zunächst einige Begrifflichkeiten geklärt und dann die mathematischen Grundlagen von Kernmethoden erläutert. Darauf aufbauend sind die z.Zt. bekanntesten Vertreter dieser Familie einzeln beschrieben um die Motivation der Arbeit zu rechtfertigen. Dieses Kapitel ist nicht als vollständige Einführung in die Theorie der Kernmethoden zu verstehen. Die einzelnen Aspekte werden lediglich so weit ausgeführt, wie es zum Verstehen der weiteren Arbeit nötig ist. Hierbei wird besonders auf die Anwendung der Support Vektor Maschine zur binären Klassifikation eingegangen, weil diese Standardmethode zur Durchführung von Effizienzmessungen exemplarisch implementiert wird.

2.1 Begriffe

In dieser Arbeit wird versucht, so nah wie möglich an der üblichen Namensgebung zu bleiben. Da die Vielfältigkeit des Einsatzes von Kernmethoden sich jedoch erst seit kurzem zu entwickeln begann, gibt es einige Begriffe, die mehrdeutig benutzt werden. Die folgenden Begriffsdefinitionen sollen helfen Mißverständnisse zu vermeiden.
Bei einer Kernmethode handelt es sich um ein Lernverfahren, das auf einem Skalarprodukt eines hochdimensionalen *Merkmalsraumes* arbeitet. Dieses Skalarprodukt wird durch eine *Kernfunktion* berechnet.
Eine Kernel Methode ist unterteilt in eine *Lernphase* und eine *Anwendungsphase*. Anwendungsphase (online) berechnet das Ergebnis der fertig gelernten Methode für einen beliebigen Eingabevektor $\mathbf{x}$. Die Lernphase (offline) bestimmt den Parametervektor α der Funktionenfamilie (2.1) sowie die Menge der Support Vektoren als Untermenge der Trainingsdatensatzes, welche die Ergebnisse einer Anwendung bestimmen. Die zwei Phasen können alternieren, so daß ein permanenter Lernprozeß stattfindet.
Als Support Vektor $\mathbf{x}_i$ wird in dieser Arbeit ein Vektor der Trainingsmenge

bezeichnet, der durch die Auswahl eines nicht weiter spezifizierten Lernverfahrens zur Anwendung dieses Verfahrens notwendig ist, also insbesondere zur Berechnung von (1.1) herangezogen werden muß. Es werden sowohl die herkömmlichen Support Vektoren der Support Vektor Maschinen, sei es zur Klassifizierung oder Regression, als auch die zur Berechnung anderer Kernmethoden notwendigen Vektoren der Trainingsmenge Support Vektoren genannt.
Support Vektor wird im folgenden mit SV abgekürzt.

2.2 Mathematische Grundlagen

In diesem Abschnitt sollen zuerst kurz die für diese Arbeit interessanten Aspekte solcher Lernprobleme dargestellt werden, die in den Beispielen aus Abschnitt 2.3 und den Implementationen behandelt werden. Anschließend sind die mathematischen Grundlagen von Kernmethoden so weit erklärt, wie es zum Verstehen des weiterführenden Kapitels 3 nötig ist.

2.2.1 Funktionen Lernen aus Beispielen

Problembeschreibung

Bei dem Lernen aus Beispielen betrachtet man eine Funktionenfamilie

$$S = \{f_\alpha : \alpha \in \Lambda\}, \; f_\alpha : \Omega \to \Upsilon \tag{2.1}$$

sowie eine Menge von Beispieldatensätzen

$$Z = \{(\mathbf{x}_i, y_i) \,| i \in \{1..L\}, \; \mathbf{x}_i \in \Omega, \; y_i \in \Upsilon\}, \quad L \in \mathbb{N}. \tag{2.2}$$

α bezeichnet hierbei einen Parametervektor der Funktion f, welcher durch die Menge Λ eingeschränkt wird. Ω ist die Ein- und Υ ist die Ausgabemenge der zu lernenden Funktion. Es wird angenommen, daß die Beispieldaten jeweils von einer unbekannten, jedoch festen Wahrscheinlichkeitsverteilung $P(\mathbf{x}, y)$ generiert sind. Als Lernproblem wird nun die Bestimmung des Parametervektors α bezeichnet, so daß die resultierende Funktion f_α der aus $P(\mathbf{x}, y)$ beschriebenen Verteilung möglichst ähnlich wird. Zu diesem Zweck wird eine *Kostenfunktion*

$$c : \Omega \times \Upsilon \times \Upsilon \to \mathbb{R} \tag{2.3}$$

eingeführt, die eine Abweichung von $f_\alpha(\mathbf{x}_i)$ zu y_i in Υ bestraft. Der Einfachheit halber wird für c meistens die euklidsche Norm

$$c(\mathbf{x}, f_\alpha(\mathbf{x}), y) = |f_\alpha(\mathbf{x}) - y| \tag{2.4}$$

benutzt.
Somit kann man nun das *Risiko*

$$R(\alpha) = \int \frac{1}{2}|f_\alpha(\mathbf{x}) - y| dP(\mathbf{x}, y) \tag{2.5}$$

betrachten, welches den Erwartungswert des Fehlers bei der Anwendung des gelernten Verfahrens angibt. Das Risiko beschreibt den zu erwartenden Fehler auf einer von der Trainingsmenge i.A. disjunkten Testmenge, der jedoch dieselbe Wahrscheinlichkeitsverteilung P zu Grunde liegt. Hat eine Funktion f_α ein geringes Risiko, so schreibt man ihr eine gute *Generalisierungsfähigkeit* zu, d.h. die Funktion liefert einen geringen Fehler auf einer beliebigen Testdatenmenge.
Diesen Fehler gilt es nun zu minimieren. Da $P(\mathbf{x}, y)$ jedoch i.A. unbekannt ist, läßt sich dies Problem nicht analytisch lösen. Ein einfacher Ansatz wäre es, anstelle des tatsächlichen Risikos aus Gleichung (2.5) das *empirische Risiko*

$$R_{emp}(\alpha) = \frac{1}{2L} \sum_{i=1}^{L} |f_\alpha(\mathbf{x}_i) - y_i| \tag{2.6}$$

über Z zu betrachten. Für viele Funktionenfamilien läßt sich das empirische Risiko jedoch für eine endliche Anzahl von Trainingsdaten stets eliminieren. Das minimieren des empirischen Risikos hat demnach nicht zwingend zur Folge, daß das tatsächliche Risiko ebenfalls klein ist. Wenn das empirische Risiko wesentlich kleiner als das tatsächliche ist, spricht man auch von einem *Overfitting* der gelernten Funktion. f_α hat sich in diesem Fall zu sehr an die Trainingsdaten angepaßt und die *Generalisierungsfähigkeit* ist verloren gegangen.

Strukturelle Risikominimierung

Um trotz der Nichtberechenbarkeit ein maximales Risiko gemäß Gleichung (2.5) garantieren zu können, werden statt dessen obere Grenzen des Risikos gesucht und diese dann minimiert. Vapnik hat ein solches Verfahren zur *Strukturellen Risikominimierung* entwickelt ([20]). Grundlage dafür ist die

Einführung eines Maßes der *Kapazität* einer Funktionenfamilie, welche ihre Generalisierungsmöglichkeit beschreibt. Gemessen werden kann die Kapazität mit der *VC(Vapnik-Chervonenkis)-Dimension*[1] h. Die VC-Dimension eines binären Klassifikators gibt die maximale Anzahl u unterschiedlicher Elemente an, die von ihm in alle 2^u unterschiedlichen Klasseneinteilungen zugeordnet werden können. Je mehr Elemente das sind, um so besser kann sich ein Klassifikator an die Trainingsdaten anpassen und um so größer ist seine Kapazität. Mit wachsender Kapazität sinkt für gewöhnlich die *Generalisierungsfähigkeit* der gelernten Funktion. Die Notwendigkeit des Ausgleichs zwischen der Kapazität einer Funktion und ihrer Fähigkeit zu generalisieren wird durch das *Bias-Variance Dilemma* formuliert. Ist für eine Funktionenfamilie die VC-Dimension h bekannt, so gilt folgender

Satz : Wähle ein $0 < \eta \leq 1$, so gilt für jedes $\alpha \in \Lambda$, $L > h$ mit einer Wahrscheinlichkeit von $1 - \eta$ die Grenze

$$R(\alpha) \leq R_{emp}(\alpha) + \phi\left(\frac{h}{L}, \frac{\log(\eta)}{L}\right). \tag{2.7}$$

Dabei beschreibt $R_{emp}(\alpha)$ das empirische Risiko aus Gleichung (2.6) und der Term ϕ die *Konfidenz* [2] zur Kontrolle der Kapazität durch

$$\phi\left(\frac{h}{L}, \frac{\log(\eta)}{L}\right) = \sqrt{\frac{h(\log\left(\frac{2L}{h}\right) + 1) - \log\left(\frac{\eta}{4}\right)}{L}}. \tag{2.8}$$

Anhand der Gleichung (2.7) kann man nun erkennen, daß das Minimieren dieser Grenze durch zwei Faktoren bestimmt wird .

Zum einen muß das empirische Risiko $R_{emp}(\alpha)$ auf den Trainingsdaten klein sein. Dies wird durch die richtige Wahl von α zu einer gegebenen Funktionenfamilie erreicht.

Zum anderen soll auch die Kapazität der gewählten Funktionenfamilie klein sein. Die Kapazität wird hier durch den Konfidenzterm (2.8) gemessen. Der Konfidenzterm kann durch die Wahl einer Funktionenfamilie mit geringer VC-Dimension und somit kleiner Kapazität und guter Generalisierungsfähigkeit minimiert werden. Eine solche Wahl nennt man *Kapazitätskontrolle.*

[1] Für eine Herleitung und genauere Erklärung von h siehe z.B. [18]. Für den Fall der Hyperebenen wird h in Kapitel 2.2.4 erläutert.

[2] Dieser Term beschreibt eigentlich die *Variabilität* der gelernten Funktion. Der Name *Konfidenz* ist in Anlehnung an bisherige Veröffentlichungen gewählt.

Diese beiden Faktoren müssen parallel betrachtet werden, denn die Minimierung nur einer der beiden Terme garantiert kein gutes Ergebnis.
Die strukturelle Risikominimierung hat dafür folgende Vorgehensweise :
Teile zunächst die zu betrachtende Funktionenfamilie S aus Gleichung (2.1) in eine Struktur von Untermengen

$$S_1 \cdots S_k \; ; k \leq \infty \; ; \; S_i = \{f_\alpha \, : \, \alpha \in \Lambda_i\}, \tag{2.9}$$

so daß

$$S_1 \subset S_2 \subset \cdots \subset S_k \subset S. \tag{2.10}$$

Die Einschränkung der Funktionenmenge von S_i auf S_{i-1} kann durch eine Restriktion der erlaubten Parametermenge Λ_i erfolgen[3]. Der Mengenaufbau ist in Abbildung 2.1 verdeutlicht.

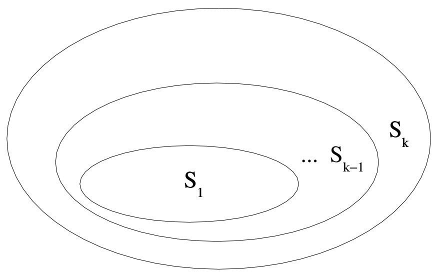

Abb. 2.1: *Die Struktur der Untermengen $S_i \subset S$ ermöglicht eine strukturelle Risikominimierung.*

Für die VC-Dimensionen der Mengen S_i gilt

$$h(S_1) \leq h(S_2) \leq \cdots \leq h(S_k) \leq h(S). \tag{2.11}$$

Für jede dieser Funktionsmengen S_i läßt sich nun ein optimaler Parametervektor α mit minimalem empirischen Risiko ermitteln. Desweiteren ist für jedes S_i die VC-Dimension h und somit auch der Konfidenzterm ϕ bekannt, so daß mit Hilfe von Gleichung (2.7) die obere Schranke für das tatsächliche

[3] Für den Fall der Hyperebenen zur binären Klassifikation wird eine solche Unterteilung in Kapitel 2.2.4 vorgestellt.

Risiko berechnet werden kann. Das Zusammenspiel dieser drei Größen in Abhängigkeit der gewählten Untermenge S_i ist in Abbildung 2.2 verdeutlicht. Als optimales f_α wird nun die Funktion f_{α^*} gewählt, die das kleinste Risiko nach Gleichung (2.7) garantieren kann.

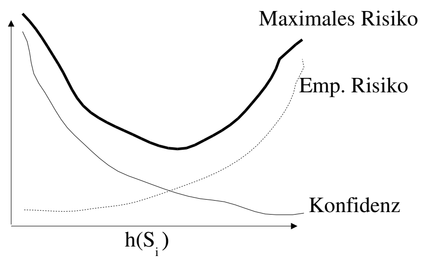

Abb. 2.2: *Die Grenze des maximalen Risikos auf einer Testdatenmenge ist von dem empirischen Risiko der Trainingsmenge und der durch den Konfidenzterm beschränkten Generalisierungsfähigkeit der gewählten Funktionenfamilie abhängig. Diese Grenze gilt es zu minimieren, um eine strukturelle Risikominimierung durchzuführen.*

2.2.2 Regression

Bei der *Regression* handelt es sich um das Lernen einer reellen Funktion anhand von Beispielen. Die Funktionenfamilie gemäß (2.1) wird dabei meist auf eine Funktionsform festgelegt [4].
Häufig wird lediglich der eindimensionale reelle Fall betrachtet, so daß sich die Menge (2.1) zu

$$\{f_\alpha : \alpha \in \Lambda\}, \; f_\alpha : \mathbb{R} \to \mathbb{R} \tag{2.12}$$

vereinfacht. α ist von der Art der gewählten Funktionenfamilie abhängig [5] .

2.2.3 Binäre Klassifikation

Die binäre Klassifikation unterteilt einen Datensatz auf Grund eines Entscheidungskriteriums in zwei disjunkte Mengen. Sie läßt sich als Lernproblem

[4] Bei der *linaren Regression* werden z.B. ausschließlich Hyperebenen betrachtet.
[5] Z.B. hat ein Polynom n-ten Grades einen n+1 dimensionalen Parametervektor.

darstellen, indem man die Ausgabemenge Υ der Gleichungen (2.1) und (2.2) als $\Upsilon = \{-1, 1\} \subset \mathbf{Z}$ annimmt.[6] Jede Zahl in Υ steht nun stellvertretend für eine der beiden Mengen. Die Gesamtheit der Elemente aus Ω, die zu einer dieser Mengen zugeordnet werden, bezeichnet man als *Klasse*, ihr zugeordneter Stellvertreter in Υ heißt *Klassenlabel* oder einfach nur *Label*. Als *Klassifikation* bezeichnet man nun die Zuordnung eines beliebigen Elementes $\mathbf{x} \in \Omega$ zu einem Klassenlabel durch f_α.
Häufig wird der N-dimensionale reelle Fall betrachtet, bei dem sich die Funktionenfamilie (2.1) für die reelle binäre Klassifikation als

$$\{f_\alpha \,:\, \alpha \in \Lambda\}\,,\, f_\alpha : \mathbb{R}^N \to \{-1, 1\} \tag{2.13}$$

schreiben läßt.

2.2.4 Lineare Trennbarkeit

Problemdarstellung

Eine Datenmenge im N-dimensionalen Raum, deren binäre Klasseneinteilung durch eine N-1 dimensionale Hyperebene bestimmt werden kann, heißt linear trennbar[7].
Eine Hyperebene E im N-dimensionalen reellen Raum kann durch einen Normalenvektor $\mathbf{\Psi} \in \mathbb{R}^N$ wie folgt beschrieben werden :

$$E = \left\{\mathbf{x} \in \mathbb{R}^N \;:\; \mathbf{x} \cdot \mathbf{\Psi} - b = 0\right\}. \tag{2.14}$$

Diese Hyperebene teilt $\mathbb{R}^N$ in zwei Mengen. Das Vorzeichen der Zuordnungsvorschrift in Gleichung (2.14) gibt dazu an, auf welcher Seite der Ebene sich ein bestimmtes $\mathbf{x} \in \mathbb{R}^N$ befindet. Die Funktionenfamilie der binären Klassifikation aus (2.13) wird zum Erlernen einer linearen Trennung also auf Funktionen der Form

$$\left\{\operatorname{sgn}\left(f_{\{\mathbf{\Psi},b\}}\right) \;:\; \{\mathbf{\Psi}, b\} \in \mathbb{R}^N \times \mathbb{R}\right\}\;,\; \operatorname{sgn}\left(f_{\{\mathbf{\Psi},b\}}\right) : \mathbb{R}^N \to \{\pm 1\} \tag{2.15}$$

mit der Berechnungsvorschrift

[6] Die Zahlen $-1, 1$ werden zur einfacheren mathematischen Beschreibung der binären Klassifikation gewählt.
[7] Zur Überprüfung der Trennbarkeit müssen also sämtliche Labels bekannt sein.

$$f_{\{\mathbf{\Psi},b\}}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{\Psi} - b \tag{2.16}$$

eingeschränkt. Bei der sgn Funktion handelt es sich um eine Funktion, die für eine reelle Zahl ihr Vorzeichen ergibt.

$$\mathrm{sgn} \,:\, \mathbb{R} \to \{\pm 1\} \;;\; \mathrm{sgn}(x) = \begin{cases} +1 \text{ für } x \geq 0 \\ -1 \text{ für } x < 0 \end{cases} \tag{2.17}$$

Eine Struktur auf der Menge der Hyperebenen

Um eine Risikominimierung nach Kapitel 2.2.1 durchzuführen benötigen wir nun die VC-Dimension h der Funktionenfamilie gemäß Gleichung (2.16). Für das Problem der binären Klassifikation ist die VC-Dimension h einer Funktionenfamilie wie folgt definiert :

Definition : Für eine Funktionenfamilie gemäß Gleichung (2.13) ist die VC-Dimension h gleich der maximalen Anzahl von verschiedenen Punkten aus Ω, die durch eine Funktion f_α in alle 2^h möglichen Klasseneinteilungen klassifiziert werden können.

Satz : Für die Familie der Hyperebenen im $\mathbb{R}^N$ gilt

$$h = N + 1 \tag{2.18}$$

Den Beweis entnehme man [3], Seite 5.
Da jedoch häufig sehr hochdimensionale Probleme behandelt werden sollen hilft diese Grenze i.A. nicht viel. Aus diesem Grund wird eine Hyperebene mit einer bestimmten *Dicke* (engl. *Margin*, auch *Marginalität*) eingeführt. Es wird gefordert, daß sich kein Punkt der Trainingsmenge innerhalb einer bestimmten Umgebung von E befindet.
Erreicht werden kann dies, indem man für die Entscheidungsfunktion (2.16) zusätzlich verlangt, daß

$$\min_{i=1..L} |\mathbf{\Psi} \cdot \mathbf{x}_i - b| \; = 1. \tag{2.19}$$

Der Abstand eines Punktes von dieser Ebene ist nun lediglich von dem Normalenvektor $\mathbf{\Psi}$ abhängig. Gleichung (2.19) definiert zwei neue Ebenen parallel zu E mit jeweils gleichem Normalenvektor $\mathbf{\Psi}$ und einem Abstand von $1/|\mathbf{\Psi}|$ zu E. Zwischen diesen beiden Ebenen liegen keine Punkte der Trainingsmenge, also kann man sich E auf die Dicke $2/|\mathbf{\Psi}|$ ausgedehnt vorstellen. Die Dicke einer Ebene ist in Abbildung 2.3 illustriert.

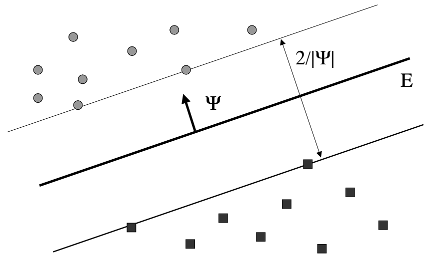

Abb. 2.3: *Die Ebene E hat die Dicke $\frac{2}{|\mathbf{\Psi}|}$. Kein Punkt der Trainingsmenge liegt innerhalb der durch dir Dicke gegebenen Grenzen. Je kleiner* $\mathbf{\Psi}$ *um so dicker die Ebene.*

Eine solche Ebene hat schon anschaulicherweise eine geringere Kapazität, da sie Punktmengen mit einem kleineren Abstand als $2/|\mathbf{\Psi}|$ zwischen zwei Punkten unterschiedlicher Klassen nicht trennen kann. Die Dicke der Ebene schränkt die Kapazität natürlich nur dann ein, wenn man das Verhältnis zu der Ausdehnung der Trainingsdatenmenge mit in Betracht zieht. Vergrößert man die Dicke der Ebene und streckt Ω im gleichen Verhältnis, so ist natürlich keine Verbesserung der Situation aufgetreten. Eine Maximierung der Ebenenausdehnung $2/|\Psi|$ entspricht einer Minimierung der Länge des Gewichtsvektors $|\Psi|$. Dieser Sachverhalt kommt wie folgt zu tragen :

Satz : Sei r der Radius der kleinsten Kugel $K_r(\mathbf{a}) = \{\mathbf{x} \in \Omega : |\mathbf{x}-\mathbf{a}| < r\}$ mit $\mathbf{a} \in \Omega$, die alle Elemente $\mathbf{x_1} \cdots \mathbf{x_L}$ enthält, dann hat die Funktionsfamilie

$$\left\{\mathrm{sgn}(f_{\{\mathbf{\Psi},b\}})\ :\ b \in \mathbb{R}; \mathbf{\Psi} \in \Omega; |\mathbf{\Psi}| \leq A\right\} \tag{2.20}$$

gemäß den Gleichungen (2.15) und (2.16) eine VC-Dimension

$$h < r^2 A^2 + 1. \tag{2.21}$$

Für zwei beliebige Hyperebenenfamilien mit $0 \leq A_1 \leq A_2$ ist also $h_1 \leq h_2$ ([13], Seite 35f), weil ein größere Mindestdicke der Ebene $2/A_1 \geq 2/A_2$ gefordert wird. Somit läßt sich auf der Familie dieser speziellen Hyperebenen eine Struktur gemäß Gleichung (2.9) einführen, indem man sich Untermengen mit Hyperebenen wachsender Mindestdicke konstruiert. Eine große Ebenenausdehnung wird wiederum durch eine geringe Länge des Gewichtsvektors $|\Psi|$ ausgedrückt.

$$S_i = \{f_{\{\mathbf{\Psi},b\}}\ :\ |\mathbf{\Psi}| \leq A_i\}\ , \forall S_j, S_i\ :\ j < i \Rightarrow A_j < A_i \tag{2.22}$$

Die Obermenge S mit Hyperebenen beliebig kleiner Dicke und $A = \infty$ hat nach Gleichung (2.18) die VC-Dimension $h = N + 1$, weil in ihr auch die ausdehnungslosen Hyperebenen enthalten sind. Die VC-Dimension der geordneten Untermengen S_i läßt sich mit der Ungleichung (2.21) abschätzen und ist kleiner als die von S. Für S_1 kann für jede Trainingsmenge eine maximale Dicke $2/A_1$ festgelegt werden, deren Betrachtung von Interesse ist. Eine Betrachtung von Trennungsebenen mit einer Dicke, die den maximalen Abstand zweier Trainingspunkte überschreitet, ist nicht sinnvoll.
Auf dieser Struktur läßt sich nun das Verfahren der strukturellen Risikominimierung durchführen. Aufgabe des Lernverfahrens ist die Ermittlung der passenden Parameter $\{\mathbf{\Psi}, b\}$ zur Minimierung des empirischen Risikos R_{emp} bei einer maximalen Dicke $2/|\mathbf{\Psi}|$ der Klassifikationsebenen. Für jede Untermenge S_i kann der das empirische Risiko minimierende Klassifikator bestimmt werden, um anschließend die Risikoschranken aus Gleichung (2.7) aller Klassifikatoren zu vergleichen. Der Klassifikator mit der geringsten Risikoschranke wird durch die strukturelle Risikominimierung ausgewählt.

2.2.5 Der Merkmalsraum

Aus der Signalverarbeitung ist bekannt, daß aus einem Vektor viele verschiedene *Merkmale* extrahiert werden können. Neben den eigentlichen Koeffizienten sind häufig eine Vielzahl weiterer Merkmale wie *Korrelationen* oder *Momente* von Bedeutung und sollten in die jeweilige Betrachtung mit einfließen. Aus diesem Grund kann es sinnvoll sein, einen Vektor $\mathbf{x}$ in einen i.A. höherdimensionalen *Merkmalsraum* F mit einer *nichtlinearen Transformation* Φ zu überführen. In diesem Merkmalsraum stehen die gewünschten Merkmale als zusätzliche Dimensionen im transformierten Vektor zur Verfügung.

$$\Phi : \Omega \rightarrow F \ , \tag{2.23}$$

für den reellen Fall also

$$\Phi : \Omega = \mathbb{R}^N \rightarrow \mathbb{R}^{N_F} = F \tag{2.24}$$

Der Vektor $\Phi(\mathbf{x})$ beinhaltet nun i.A. mehr Information als $\mathbf{x}$.

Eine solche Transformation birgt neben dem Informationsgewinn jedoch das Problem der hohen Dimensionalität. Für sinnvolle Anwendungen wird die Dimension des Merkmalsraumes oft derart groß, daß eine echte Durchführung von Φ nicht realisierbar ist, geschweige denn eine Rechnung in F durchgeführt werden kann.

Beispiel
Für die Transformation aus $\mathbb{R}^2$ in den geordneten Momentenraum 2. Ordnung[8] lautet die Transformation

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^4 \ ; \ \Phi(\mathbf{x}) = ({x_1}^2, x_1 x_2, x_2 x_1, {x_2}^2)^T \tag{2.25}$$

Der Momentenraum d-ter Ordnung hat für einen Eingabevektor $\mathbf{x} \in \mathbb{R}^N$ eine Dimension von

$$N_F = \frac{(N+d-1)!}{d!(N-1)!} \ , \tag{2.26}$$

d.h. bereits ein Bild mit 16×16 Pixeln ergibt bei einer Transformation in den Momentenraum 5. Ordnung einen 10^{10} dimensionalen Merkmalsvektor.

2.2.6 Kernfunktionen

Kernfunktionen erlauben es, das in Kapitel 2.2.5 beschriebene Problem der hohen Dimensionalität zu umgehen. Eine Kernfunktion k ermöglicht eine einfache Berechnung des Skalarproduktes im Merkmalsraum ohne die konkrete Durchführung der Transformation Φ. Das Skalarprodukt von F wird also im Eingaberaum Ω bestimmt.

$$k : \Omega^2 \rightarrow \mathbb{R} \ , \ k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) \tag{2.27}$$

Ein bestimmtes Merkmal muß nicht unbedingt explizit als Dimension berechnet werden, sondern kann je nach Art der Kernfunktion indirekt berechnet werden. Desweiteren erlaubt die Verwendung einer Kernfunktion die Verarbeitung nichtvektorieller Elemente. Das Ergebnis der Kernfunktion kann auf beliebige Weise berechnet werden und muß lediglich gewissen Bedingungen entsprechen, um als Skalarprodukt im Merkmalsraum *interpretiert* werden zu können. Die Berechnung kann also auch durch effiziente Algorithmen erfolgen, ohne auf spezielle Dimensionen zugreifen zu müssen. Es ist z.B. möglich zwei Texte durch einen effizienten Wortvergleich basierend auf einem schnellen Suchalgorithmus zu implementieren und das Ergebnis als Skalarprodukt in einem geeigneten Hilbertraum zu deuten [10].

[8] Die zweite und dritte Komponente dieses Vektors sind zwar stets gleich, sie vereinfachen jedoch die weitere mathematische Betrachtung. Die Dimension des Merkmalsraumes ist somit effektiv nur drei.

Für das **Beispiel** aus Kapitel 2.2.5 heißt das

$$\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) = {x_1}^2{y_1}^2 + {x_2}^2{y_2}^2 + 2x_1x_2y_1y_2 = (\mathbf{x} \cdot \mathbf{y})^2 \tag{2.28}$$

Das Skalarprodukt des Merkmalsraumes $F = \mathbb{R}^4$ läßt sich also als Quadrat des Skalarproduktes in $\Omega = \mathbb{R}^2$ berechnen. Eine ähnliche Kernfunktion läßt sich auch für den Momentenraum d-ter Ordnung und den Raum der Momente mit *maximal* d-ter Ordnung angeben.

Das *Mercer Theorem* formuliert, welche Funktionen als Kernfunktionen zugelassen sind und wie Merkmalsräume und Kernfunktionen zusammenhängen.
Für den Fall eines endlichen Datensatzes gilt der anschauliche

Satz : Sei k eine reelle symmetrische Funktion auf einem endlichen Eingaberaum

$$k : \{\mathbf{x}_1, .., \mathbf{x}_L\ ; \forall i : \mathbf{x}_i \in \Omega\}^2 \to \mathbb{R} \tag{2.29}$$

und K die zugehörige $L \times L$ Matrix mit den Einträgen

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x_j}), \tag{2.30}$$

dann ist k eine ein Skalarprodukt darstellende Kernfunktion genau dann, wenn K positiv semi-definit ist (vgl. [5]).

Genereller gilt für eine kompakte Menge $M \subset \mathbb{R}^N$ der

Satz (Mercer's Theorem) : Ist $k(\mathbf{x}, \mathbf{y})$ ein kontinuierlicher, symmetrischer Kern eines positiven Integraloperators T mit

$$(Tf)(\mathbf{y}) = \int_M k(\mathbf{x}, \mathbf{y}) l(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{2.31}$$

und

$$\int_{M \times M} k(\mathbf{x}, \mathbf{y}) l(\mathbf{x}) l(\mathbf{y}) \mathrm{d}\mathbf{x} \mathrm{d}\mathbf{y} \geq 0 \tag{2.32}$$

für jedes $l \in L_2(M)$, dann läßt sich k durch eine einheitlich konvergente Reihe aus Eigenfunktionen ψ_i und positiven Eigenwerten λ_i von T ausdrücken.

$$k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}), \tag{2.33}$$

wobei N_F die Anzahl der positiven Eigenwerte von T und somit die Dimension des zugehörigen Merkmalsraumes ist (vgl. z.B. [3]).
Aus Gleichung (2.33) läßt sich sofort die Konstruktion einer passenden Transformation Φ ablesen.

$$\Phi(\mathbf{x}) = \left(\sqrt{\lambda_1}\psi_1(\mathbf{x}), .., \sqrt{\lambda_{N_F}}\psi_{N_F}(\mathbf{x})\right) \tag{2.34}$$

Zu beachten ist hier besonders, daß es sich potentiell um einen ∞-dimensionalen Merkmalsraum F handeln kann. Bei einem durch ein solches Φ gebildeten Merkmalsraum F handelt es sich um einen *Reproduzierenden Kernel Hilbert Raum.*
Es gibt eine Vielzahl von Funktionen, die das Mercer Theorem erfüllen und somit als Kernfunktion in Frage kommen. Einfache Anwendungen konzentrieren sich häufig auf Kernfunktionen der folgenden Arten :

- Radiale Exponentialfunktionen

  $$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{|\mathbf{x}-\mathbf{y}|}{2\sigma^2}\right) \tag{2.35}$$

  bzw.

  $$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{|\mathbf{x}-\mathbf{y}|^2}{2\sigma^2}\right), \tag{2.36}$$

- Signoid Funktionen

  $$k(\mathbf{x}, \mathbf{y}) = \tanh\left(\kappa(\mathbf{x}\cdot\mathbf{y}) + \Theta\right) \tag{2.37}$$

  für einen eingeschränkten Bereich von Θ.

- Polynomiale Funktionen

  $$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^d \tag{2.38}$$

  bzw.

  $$k(\mathbf{x}, \mathbf{y}) = ((\mathbf{x}\cdot\mathbf{y}) + 1)^d. \tag{2.39}$$

In modernen Anwendungen verbergen sich jedoch häufig die Durchführungen komplizierter Algorithmen in der Kernfunktionsauswertung. Jede Funktion, die das Mercer Theorem erfüllt, ist eine Kernfunktion des Skalarproduktes eines konstruierbaren Hilbertraumes. Wenn eine Funktion als Kernfunktion erkannt ist, sagt das jedoch noch nichts über die Art des dazugehörigen

Hilbertraumes aus [9]. Anders herum kann ebenfalls nicht für jeden beliebigen Hilbertraum eine zum Eingaberaum passende Kernfunktion konstruiert werden. Die Wahl der Kernfunktion ist jedoch entscheidend für das Verhalten und die Güte des Lernverfahrens. Neuere Arbeiten versuchen, eine den Trainingsdaten angepaßte Kernfunktion zu generieren, indem Vorwissen wie Invarianzen verarbeitet werden [16].

2.3 Kernmethoden

In diesem Kapitel werden einige Kernmethoden vorgestellt, für die die in Kapitel 3 entwickelten adaptiven Ansätze anwendbar sind. Hier soll *keine* detaillierte Beschreibung der einzelnen Methoden erfolgen. Insbesondere wird die Beschreibung der Lernphase auf solche Aspekte eingeschränkt, die zum Verstehen der Bedeutung des zu lernenden Parametervektors α wichtig sind. Zum einen würde eine zu detaillierte Einführung von dem eigentlichen Thema ablenken, da diese Verfahren bereits sehr gut in den Referenzen beschrieben sind. Zum anderen soll gerade hervorgehoben werden, daß die in Kapitel 3 entwickelten Ansätze von der Kernmethode, auf die sie angewendet werden, unabhängig sind.

2.3.1 Lineare Support Vektor Maschinen

Die Support Vektor Maschine ist ein von Vapnik entwickeltes Verfahren, das eine strukturelle Risikominimierung auf der in Kapitel 2.2.4 eingeführten Struktur der Hyperebenen realisiert ([20], [6]).

SVM's zur Klassifikation

Von der gesuchten Ebene wird eine optimale Trennung der Trainingsdaten bei minimaler VC-Dimension gefordert.
Die korrekte Klassifikation der Trainingsdaten wird durch die Forderungen

$$y_i \cdot (\mathbf{x}_i \cdot \mathbf{\Psi} - b) \geq 1 \ \ \forall i \in \{1..L\} \tag{2.40}$$

[9] Zu diesem Problem zeigt Smola in seiner Arbeit [18], S.39ff den Zusammenhang zwischen Kernfunktionen und *Green'schen Funktionen* sowie *Regularisierungoperatoren*. Diese Zusammenhänge werden ausgenutzt, um ausgehend von einer Kernfunktion den zugehörigen Regularisierungsoperator zu bestimmen und somit mehr über den beschriebenen Hilbertraum zu erfahren. Für einen gegebenen Regularisierungsoperator läßt sich auf umgekehrtem Weg die passende Kernfunktion bestimmen.

ausgedrückt. Um gleichzeitig die Kapazität der Funktion so gering wie möglich zu halten,muß der Ausdruck

$$\tau(\mathbf{\Psi}) = \frac{1}{2}|\mathbf{\Psi}|^2 \tag{2.41}$$

minimiert werden, um eine maximale Dicke und somit minimale VC-Dimension zu erhalten. Die Minimierung des Ausdrucks (2.41) unter Berücksichtigung von 2.40 führt zu dem Lagrangian

$$L(\mathbf{\Psi}, b, \alpha) = \frac{1}{2}|\mathbf{\Psi}|^2 - \sum_{i=1}^{L} \alpha_i (y_i((\mathbf{x}_i \cdot \mathbf{\Psi}) - b) - 1) \tag{2.42}$$

mit den Lagrange Multiplikatoren $\alpha_i \geq 0$. L muß nun in Bezug auf α maximiert, sowie für $\mathbf{\Psi}$ und b minimiert werden [10]. Die notwendigen Bedingungen

$$\frac{\partial}{\partial b} L(\mathbf{\Psi}, b, \alpha) \;=\; 0 \tag{2.43}$$

und

$$\frac{\partial}{\partial \mathbf{\Psi}} L(\mathbf{\Psi}, b, \alpha) \;=\; 0 \tag{2.44}$$

für den gesuchten Sattelpunkt ergeben

$$\sum_{i=1}^{L} \alpha_i y_i \;=\; 0 \tag{2.45}$$

sowie

$$\mathbf{\Psi} = \sum_{i=1}^{L} \alpha_i y_i \mathbf{x}_i. \tag{2.46}$$

Da bei realen Problemen jedoch meist keine echte lineare Trennung gefunden werden kann, werden die Variablen ξ_i zur Bestrafung einer Fehlklassifikation von $\mathbf{x}_i$ eingeführt. Anstelle der Bedingungen in (2.40) treten nun die Forderungen

$$y_i \cdot (\mathbf{x}_i \cdot \mathbf{\Psi} - b) \geq 1 - \xi_i \tag{2.47}$$

$$\xi_i \geq 0 \;\; \forall i \in \{1..L\} \tag{2.48}$$

[10] Der Name α ist hier absichtlich konform mit den Gleichungen (1.1) und (2.1) gewählt, da diese $\alpha_\mathbf{i}$ später Parameter der Anwendungsfunktion f sind.

für eine möglichst korrekte Klassifizierung der Trainingsdaten. Statt des Ausdrucks (2.41) soll die gesuchte Ebene nun den Term

$$\tau(\mathbf{\Psi}, \xi) = \frac{1}{2}|\mathbf{\Psi}|^2 + C\sum_{i=1}^{L}\xi_i \tag{2.49}$$

unter Erfüllung der Bedingungen (2.47) minimieren. Ein kleines $|\mathbf{\Psi}|$ bewirkt eine große Dicke $2/|\mathbf{\Psi}|$ der Hyperebene und garantiert somit eine minimale VC-Dimension h. Der Koeffizient C ist frei zu wählen und gewichtet den Einfluß der Bestrafungsvariablen ξ_i. C bestimmt also das Verhältnis zwischen empirischem Risiko R_{emp} und der Konfidenz ϕ aus der Schranke (2.7) für den Einfluß auf das Lernverfahren.
Die Lösung wird über das *duale Problem* gefunden. In dessen Formulierung wird ausschließlich mit Skalarprodukten gearbeitet wodurch die spätere Substitution mit einer Kernfunktion möglich wird. Man erhält dabei wiederum eine Lösung in der Form

$$\mathbf{\Psi} = \sum_{i=1}^{L}\alpha_i y_i \mathbf{x}_i \tag{2.50}$$

mit

$$\sum_{i=1}^{L}\alpha_i y_i \ = \ 0 \ \ ; 0 \leq \alpha_i \leq C \forall i. \tag{2.51}$$

Die Anwendung des gefundenen Klassifikators geschieht nun durch die Multiplikation eines neuen Eingabevektors $\mathbf{x}$ mit dem Gewichtsvektor $\mathbf{\Psi}$.

$$\mathbf{x} \cdot \mathbf{\Psi} = \sum_{i=1}^{L}\alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} \tag{2.52}$$

Sowohl in der Lern- als auch in der Anwendungsphase werden Elemente des Eingaberaumes lediglich in Skalarprodukten verwendet, so daß das gesamte Verfahren durch die Substitution mit einer Kernfunktion in einen nichtlinear abgebildeten Merkmalsraum überführt werden kann. Es soll noch erwähnt werden, daß genau die $\alpha_i \neq 0$ sind, für die in der Forderung (2.40) bzw. (2.47) gerade die Gleichheit erfüllt ist. Ein Vektor $\mathbf{x}_i$ mit $\alpha_i \neq 0$ ist ein SV für diese SV Maschine, da nur diese Vektoren für die Berechnung von (2.16) und somit für die Anwendung der Methode verwendet werden.
Außerdem ist $\alpha_i = C$ gdw. $\xi_i \neq 0$, also genau die von der gefundenen Ebene *falsch klassifizierten* SVen $\mathbf{x}_i$ werden mit dem Kostenfaktor C gewichtet. Solche SV werden im weiteren *fehlklassifizierte Support Vektoren* genannt.

Die letzten beiden Aussagen können zusätzlich genutzt werden, um aus den so gewonnenen Geichungen den fehlenden Parameter b zu berechnen.

SVM's zur Regression

Um die strukturelle Risikominimierung auf der Menge der Hyperebenen zur Regression anzuwenden, müssen die Forderungen (2.47) zu

$$\begin{aligned} y_i - \mathbf{\Psi} \cdot \mathbf{x}_i + b &\leq \epsilon + \xi_i \\ (\mathbf{\Psi} \cdot \mathbf{x}_i - b) - y_i &\leq \epsilon + \hat{\xi}_i \\ \hat{\xi}_i, \xi_i &\geq 0 \;\forall i \end{aligned} \tag{2.53}$$

umformuliert werden. Der Fehler wird außerhalb einer *unempfindlichen* ϵ Umgebung um y_i in Abhängigkeit von ξ_i und $\hat{\xi}_i$ bestraft[11]. Die Kostenfunktion ist in Abbildung 2.4 dargestellt.

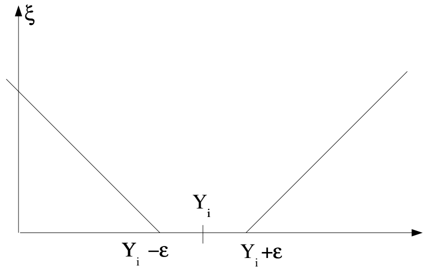

Abb. 2.4: *Kostenfunktion zur Regression. Um einen unempfindlichen ϵ-Bereich wird ein Fehler gegenüber y_i mit ξ bestraft.*

Zu minimieren ist nun analog zum Ausdruck (2.49)

$$\tau(\mathbf{\Psi}, \xi, \hat{\xi}) = |\mathbf{\Psi}|^2 + C \sum_{i=1}^{L} (\xi_i + \hat{\xi}_i). \tag{2.54}$$

Bei der Lösung über die *duale Formulierung* der aus den Forderungen (2.53) und (2.54) resultierenden Lagrange Funktion mit Hilfe von *Standardmethoden*

[11] Zur Bestrafung kann auch eine beliebige andere konvexe Kostenfunktion benutzt werden.

der quadratischen Optimierung ([6])) ergibt sich wiederum

$$\mathbf{\Psi} = \sum_{i=1}^{L} \alpha_i y_i \mathbf{x}_i \tag{2.55}$$

mit

$$\alpha_i \geq 0 \forall i. \tag{2.56}$$

2.3.2 Nichtlineare Support Vektor Maschinen

Jedes Verfahren, das ausschließlich auf Skalarprodukten in dem betrachteten Raum arbeitet, kann durch die Verwendung von Kernfunktionen erweitert werden. Das Lernverfahren der SVM muß dafür kaum verändert werden. Die Verwendung der Kernfunktion ist äquivalent mit einer nichtlinearen Vorverarbeitung der Daten, nur daß die in Kapitel 2.2.6 aufgeführten Vorteile wirksam sind.
Anstelle der Ergebnisse (2.46), (2.50) bzw. (2.55) erhält man nun einen einheitlichen Vektor

$$\bar{\mathbf{\Psi}} = \sum_{i=1}^{L} \alpha_i y_i \Phi(\mathbf{x}_i), \tag{2.57}$$

so daß eine Anwendung von (2.16) sich zu

$$f_{\{\mathbf{\Psi},b\}}(\mathbf{x}) = \Phi(\mathbf{x}) \cdot \bar{\mathbf{\Psi}} - b \tag{2.58}$$

transformiert. Dabei wird die Anwendung von allen Kernmethoden durch das Produkt

$$\Phi(\mathbf{x}) \cdot \mathbf{\Psi} = \sum_{i=1}^{L} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}). \tag{2.59}$$

und der anschließenden Subtraktion von b berechnet.

In den folgenden Kapiteln sollen für die Bildung von $\mathbf{\Psi}$ nur noch Vektoren mit $\alpha_i \neq 0$ verwendet werden. Die Summen gemäß den Gleichungen (2.46), (2.50), (2.55) und (2.59) werden *nur noch über die Support Vektoren* gebildet. Damit ändert sich die Summationsgrenze von der Anzahl der Trainingsdaten L zur Anzahl der SVen N_S. Desweiteren wird ab hier ausschließlich im Merkmalsraum gearbeitet, so daß die Markierung der Elemente von F durch den Überstrich fallen gelassen wird. Insbesondere wird der Gewichtsvektor in $\bar{\mathbf{\Psi}}$ nur noch durch $\mathbf{\Psi}$ abgekürzt. Die Lagrange-Multiplikatoren α_i werden mit den Klassenlabels y_i multipliziert, so daß die Anwendung (2.59) in die Form von Gleichung (1.1) gebracht wird.

2.3.3 Kernel Principal Component Analysis

Das Verfahren der *Principal Component Analysis* (PCA) ist eine altbewährte Methode, um möglichst aussagekräftige Merkmale aus einer Datenmenge zu extrahieren. Dieses Verfahren läßt sich durch die Benutzung von Kernfunktionen nichtlinear erweitern ([13]S.79ff, [18]S.98ff). Die Verwendung einer Kernfunktion erlaubt die Berechnung von *Hauptkomponenten entlang einer Kurve* statt einer Geraden im linearen Fall. Es handelt sich um einen Ansatz des nicht überwachten Lernens, so daß sich die Eingabemenge im Vergleich zur Menge (2.2) auf

$$X = \{\mathbf{x}_i | i \in \{1..L\}\,,\ \mathbf{x}_i \in \Omega\}\ ,\quad L \in \mathbb{N}. \tag{2.60}$$

reduziert. Anstelle der herkömmlichen Betrachtung der geschätzten Kovarianzmatrix

$$O = \frac{1}{L}\sum_{i=1}^{L} \mathbf{x}_i \cdot \mathbf{x}_i \tag{2.61}$$

wird in der Kernel PCA unter Benutzung einer nichtlinearen Transformation gemäß der Vorschrift (2.24) der Ausdruck

$$\bar{O} = \frac{1}{L}\sum_{i=1}^{L} \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_i)^T \tag{2.62}$$

betrachtet. Eine größtmögliche Dekorrelation der Merkmale erreicht man durch eine Diagonalisierung dieser Matrix. Um eine Diagonalisierung durchführen zu können, muß die Eigenwertgleichung

$$\lambda \bar{\mathbf{v}} = \bar{O}\bar{\mathbf{v}} \tag{2.63}$$

mit Eigenvektoren $\bar{\mathbf{v}} \in F, \bar{\mathbf{v}} \neq 0$ gelöst werden.
Da für alle Lösungen $\bar{\mathbf{v}}$ von Gleichung (2.63) mit $\lambda \neq 0$

$$\bar{\mathbf{v}} \in \mathrm{span}\{\Phi(\mathbf{x}_1), .., \Phi(\mathbf{x}_L)\} \tag{2.64}$$

gilt, lassen sie sich als Linearkombination

$$\bar{\mathbf{v}} = \sum_{i=0}^{L} \alpha_i \Phi(\mathbf{x}_i) \tag{2.65}$$

schreiben. Desweiteren läßt sich Gleichung (2.63) auf jeden transformierten Vektor projezieren, wodurch man das Gleichungssystem

$$\lambda(\Phi(\mathbf{x_k}) \cdot \bar{\mathbf{v}}) = (\Phi(\mathbf{x_k}) \cdot \bar{O}\bar{\mathbf{v}}) \;; k \in \{1..L\} \tag{2.66}$$

erhält. Setzt man nun (2.65) in (2.66) ein, so erhält man

$$\lambda \sum_{i=1}^{L} (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_i)) = \frac{1}{L} \sum_{i=1}^{L} \alpha_i \left(\Phi(\mathbf{x}_k) \cdot \sum_{j=1}^{L} \Phi(\mathbf{x}_j)(\Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i)) \right) \tag{2.67}$$

für jedes $k \in \{1..L\}$.
Benutzt man die abkürzende quadratische $L \times L$ Matrix nach Gleichung (2.30)

$$K_{ij} = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)), \tag{2.68}$$

so erhält man anstelle von (2.67)

$$L\lambda K\alpha = K^2\alpha, \tag{2.69}$$

wobei α für den Vektor $(\alpha_1 ... \alpha_L)^T$ steht. Die Eigenvektoren $\bar{\mathbf{v}}$ werden durch die Forderung

$$\bar{\mathbf{v}}^2 = 1 \; \forall k \in \{1..L\} \tag{2.70}$$

normalisiert. Gleichung (2.69) kann mit der Forderung (2.70) über das Eigenwertproblem

$$L\lambda\alpha = K\alpha \tag{2.71}$$

gelöst werden (Vergl. [13]S.80ff)
Eine Anwendung des Verfahrens besteht nun aus der Projektion des Transformierten Eingabevektors auf den normierten k-ten Eigenvektor durch

$$(\Phi(\mathbf{x}) \cdot \bar{\mathbf{v}}^k) = \sum_{i=1}^{L} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{L} \alpha_i^k k(\mathbf{x}_i, \mathbf{x}). \tag{2.72}$$

Im folgenden wird als Anwendung dieses Verfahrens lediglich die Projektion auf *einen* der k Eigenvektoren $\bar{\mathbf{v}}$ betrachtet[12]. Die Anwendung (2.72) der Projektion auf eine Hauptkomponentenkurve hat ebenfalls die Form von Gleichung (1.1), wenn man den reellen Anteil $b = 0$ wählt. Die neuen Verfahren zur Beschleunigung lassen sich demnach auch für die Anwendungen dieser Kernmethode benutzen. Als Support Vektor dieser Anwendung werden die Elemente $\mathbf{x}_i$ bezeichnet, deren $\alpha_i \neq 0$ sind.

[12] Die gleichzeitige Betrachtung von mehreren Projektionen ähnelt der Betrachtung von einem Multiklassen-Klassifikator, der aus mehreren binären Klassifizierern zusammengesetzt ist.

2.3.4 Andere Kernmethoden

Kernmethoden werden inzwischen in vielen verschiedenen Forschungsgebieten untersucht und angewendet. Es gibt noch weitere Methoden wie z.B. die *Novelty Detection*, ein Verfahren zur Erkennung von abnormalen Eingabedaten.

Ein anderes Verfahren bildet die *Relevance Vector Machine* (RVM). Das Verfahren der SVMs ist zu einer Vektor Maschine umgewandelt worden, die anstelle der Randvektoren ähnlich zu einem RBF-Netzwerk stellvertretende Vektoren aus der Trainingsmenge sucht [19]. Durch die Wahl von möglichst repräsentativen Vektoren werden bei einer geeigneten Verteilung der Trainingsdaten wesentlich weniger Support Vektoren als von einer SVM benötigt. Zusätzlich kann neben dem Klassifikationsergebnis eine Wahrscheinlichkeitsaussage über die Korrektheit der Entscheidung gefällt werden. Die funktionale Darstellung einer Anwendung der Relevance Vector Machine sowie der Novelty Detection entspricht ebenfalls Gleichung (1.1).

Prinzipiell läßt sich jedes Verfahren, das ausschließlich mit dem Skalarprodukt in dem untersuchten Raum arbeitet, durch eine *Kernsubstitution* zu einer Kernmethode umformulieren. Vermutlich befindet sich die Forschung erst am Beginn der vollständigen Ausschöpfung der Möglichkeiten.

In diesem Kapitel wurde gezeigt, daß die Anwendungen aller behandelten Kernmethoden durch eine einheitliche Formel (1.1) dargestellt werden kann. Basierend auf dieser Rechenvorschrift werden nun im folgenden Kapitel Verfahren für eine schnellere Berechnung aller Kernmethodenanwendungen hergeleitet.

3. ADAPTIVE ANSÄTZE

Zusammenfassung

In diesem Kapitel werden die verschiedenen neuen Ansätze zur schnelleren Anwendung einer Kernmethode vorgestellt. Es wird jeweils kurz die Idee erläutert und dann die mathematische Umsetzung beschrieben. Der Pseudocode der einzelnen Verfahren befindet sich im Anhang. Es handelt sich um zwei 'ad hoc' Verfahren, die ein grundlegendes Verständnis schaffen sollen und später zu einem Vergleich mit dem dritten und effizientesten Algorithmus herangezogen werden sollen. Die Verfahren sind auf beliebigen Kernmethoden anwendbar, insofern ihre Anwendung gemäß der Rechenvorschrift (1.1) gebildet werden kann.

3.1 Abschätzen der Restsumme

3.1.1 Der Ansatz

Grundsätzlich liegt allen Verfahren die Idee zu Grunde, daß man die Summe (1.1) nicht ganz ausrechnet, sondern nur bestimmte Summanden zum Endergebnis führen. Mit einer geeigneten *Vorsortierung* der Summanden werden also o.B.d.A. genau die letzten Summanden vernachlässigt. Die Entscheidung darüber, bei welchem Summanden das Ergebnis genügend exakt berechnet ist, kann über eine Abschätzung der Restsumme geschehen.

Zur vereinfachenden Schreibweise sei

$$sum_{(j,k)} = \left(\sum_{i=j}^{k} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \right) \tag{3.1}$$

eine Abkürzung für die Summation vom j'ten bis zum k'ten Element. Diese Summe kann sowohl für auf- als auch absteigende k und j trivialerweise rekursiv aus den vorherigen durch einfache Addition bzw. Subtraktion eines Summanden berechnet werden. Somit wird Gleichung (1.1) zu

$$f_{\{\mathbf{\Psi},b\}}(\mathbf{x}) = sum_{(1,N_S)} - b, \tag{3.2}$$

oder für ein $q \in \{1..N_S\}$ ist

$$f_{\{\mathbf{\Psi},b\}}(\mathbf{x}) = (sum_{(1,q)} - b) + sum_{(q+1,N_S)}. \tag{3.3}$$

f wird nun durch das jeweils bereits berechnete $sum_{(1,q)} - b$ approximiert.

Wenn die verwendete Kernfunktion k durch eine reelle Zahl K_{max}

$$\max_{i \in \{1..N_S\};\, \mathbf{x} \in \Omega} |k(\mathbf{x}, \mathbf{x}_i)| \leq K_{max} \tag{3.4}$$

beschränkt ist, gilt die Abschätzung

$$sum_{(q+1,N_S)} \leq \sum_{i=q+1}^{N_S} |\alpha_i| K_{max} = (N_S - q) K_{max} \cdot \sum_{i=q+1}^{N_S} |\alpha_i|. \tag{3.5}$$

Wird nun für ein $p \in [0..1]$ die Bedingung

$$|(sum_{(1,q)} - b)| \cdot p > (N_S - q) K_{max} \sum_{i=q+1}^{N_S} |\alpha_i| \tag{3.6}$$

als Abbruchbedingung der Summenbildung über j verwendet, so kann *garantiert* werden, daß der relative Fehler maximal p Prozent beträgt.

Je größer nun die erlaubte Toleranz p ist, um so schneller wird das Verfahren arbeiten.

Beachtet werden sollte hier noch, daß die Berechnung der rechten Seite in (3.6) bereits offline vor der Anwendung des Verfahrens geschehen kann und somit keine Zeit in Anspruch nimmt.

Betrachtet man dieses Verfahren im Merkmalsraum F, so wird durch die Ungleichung (3.4) lediglich das Skalarprodukt in F durch K_{max} gemäß

$$|k(\mathbf{x}, \mathbf{x}_i)| = |\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)| \leq K_{max} \tag{3.7}$$

beschränkt. Ein geeigentes K_{max} kann durch die Abschätzung nach Cauchy-Schwartz

$$|\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)| \leq |\Phi(\mathbf{x})| \cdot |\Phi(\mathbf{x}_i)| = \sqrt{k(\mathbf{x}, \mathbf{x})}\sqrt{k(\mathbf{x_i}, \mathbf{x_i})} \tag{3.8}$$

gefunden werden.

Der Betrag eines Vektors in F kann durch die Beziehung

$$|\Phi(\mathbf{x})|^2 = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) \tag{3.9}$$

berechnet werden.

Beispiel : Eine geeignete Kernfunktion für eine solche Abschätzung ist z.B. die Exponentialfunktion. Eine Abschätzung der e Funktion durch K_{max} in Ω oder nach Ungleichung (3.8) in F liefert jeweils die gleiche Grenze mit $K_{max} = 1$. Andere Kernfunktionen lassen sich ebenfalls durch einen geeigneten Eingaberaum Ω beschränken[1].

3.1.2 Summenabbruch für die binäre Klassifikation

Für den Fall der binären Klassifikation mit einer SV Maschine wird als Anwendungsergebnis lediglich das Vorzeichen des in Gleichung (1.1) berechneten Wertes ermittelt. Die Summation kann abgebrochen werden, sobald sich das bereits gefundene Vorzeichen durch die Restsumme nicht mehr verändern kann. Die Abschätzung der Restsumme darf also nicht größer als der bereits berechnete Wert sein. Die Ungleichung (3.6) ändert sich für diesen Fall zu

$$|(sum_{(1,q)} - b)| > (N_S - q)K_{max} \sum_{i=q+1}^{N_S} |\alpha_i|. \tag{3.10}$$

bzw. zu

$$|(sum_{(1,q)} - b)| > (N_S - q)|\Phi(\mathbf{x})| \cdot \sum_{i=q+1}^{N_S} |\alpha_i||\Phi(\mathbf{x}_i)|. \tag{3.11}$$

bei einer Betrachtung in F gemäß der Abschätzung (3.8).

3.1.3 Umsortierung der Support Vektoren

Damit die Bedingungen (3.6) bzw. (3.10) mit einem möglichst kleinen q zutreffen und somit möglichst wenige Berechnungen der Kernfunktion benötigt werden, lohnt es sich die SVen nach der Lernphase neu zu sortiern. Um die größten Summanden der Summe (1.1) zuerst zu berechnen, werden die SVen

[1] Die Gruppe der Polynomialfunktionen ist auf dem Bereich $\{\mathbf{x} \in \Omega; |\mathbf{x}| \leq 1\}$ auf $K_{max} = 1$ beschränkt.

nach den Beträgen $|\alpha_i||\Phi(\mathbf{x}_i)|$ sortiert. Somit wächst $sum_{(1,q)}$ und die Abschätzung von $sum(q+1, N_S)$ verkleinert sich rapider mit wachsendem q als in der vorherigen zufälligen Verteilung.

3.1.4 Implementationsaspekte

Es wäre ebenfalls denkbar, eine adaptive Ordnung in die SVen zu bringen. Dabei wird stets in Abhängigkeit von den bereits berechneten Auswertungen der Kernfunktion entschieden, welcher Summand aus der Summe (1.1) als nächstes berechnet werden soll. Es wird dabei keine feste Reihenfolge in die Support Vektor Menge gebracht, sondern der gesamte Baum der möglichen Umsortierungen offen gehalten. Die Wahl des statistisch besten SV soll so ein größtmögliches Ergebnis erzielen. Diese Methode hat den Nachteil, daß neben des zusätzlichen Speicheraufwandes für die exponentielle Anzahl der Entscheidungsmatrizen auch ein nicht unerheblicher Overhead in der Anwendungsphase notwendig wird. Der Speicheraufwand macht diesen Ansatz schnell unrealisierbar und die aufwendige Entscheidung führt u.U. dazu, daß trotz einem vorzeitigen Summationsabbruch bei einem $q < N_S$ keine Beschleunigung der Berechnung auftritt. Wie später in der Versuchsauswertung klar wird, tritt dieses Problem sogar bereits bei der einfachen Version dieses Verfahrens auf.

3.2 Abschätzung mit Hilfe des Abstandsmaßes

Die grundlegende Idee dieses Ansatzes liegt in der Tatsache, daß viele Kernfunktionen wie etwa die Exponentialfunktion für große Abstände zwischen den verglichenen Vektoren zu einem Grenzwert konvergieren. Liegen zwei Support Vektoren weit auseinander, so liefert bei lediglich lokal sensitiven Kernfunktionen nur einer der beiden einen nennenswerten Anteil zur Summe (1.1) und die Berechnung der zweiten Kernfunktionsauswertung kann vernachlässigt werden. Durch das Bestimmen der Lage der Support Vektoren untereinander und der Berechneten Distanz eines Support Vektors zu dem Eingabeelement können somit u.U. Kernauswertungen mit anderen Support Vektoren vermieden und durch den Grenzwert approximiert werden.

3.2.1 Der Ansatz

Kernfunktionen lassen sich stets durch eine Funktionsverkettung gemäß

$$k(\mathbf{x}, \mathbf{y}) = g(d(\mathbf{x}, \mathbf{y})) \tag{3.12}$$

darstellen[2]. Dabei ist g eine reelle Funktion

$$g : \mathbb{R} \to \mathbb{R} \tag{3.13}$$

und d eine Funktion im Eingaberaum mit

$$d : \Omega^2 \to \mathbb{R}. \tag{3.14}$$

Handelt es sich bei g nun um eine konvergente Funktion mit

$$\lim_{z \to \infty} g(z) = g_\infty; \quad g_\infty \in \mathbb{R} \tag{3.15}$$

so kann für ein gewünschtes $\epsilon > 0$ ein $z_\epsilon \in \mathbb{R} > 0$ gefunden werden, so daß

$$\forall z \in \mathbb{R}; z \geq z_\epsilon : |g(z) - g_\infty| \leq \epsilon. \tag{3.16}$$

Ist d weiterhin eine Distanzfunktion in Ω, so kann auf d die Dreiecksungleichung angewendet werden. Für zwei Support Vektoren $\mathbf{x}_i$ und $\mathbf{x}_j$ gilt somit

$$\forall \mathbf{x} \in \Omega : d(\mathbf{x}, \mathbf{x}_j) \geq d(\mathbf{x}_i, \mathbf{x}_j) - d(\mathbf{x}, \mathbf{x}_i), \tag{3.17}$$

d.h. die Distanz von $\mathbf{x}$ zu einem der beiden SVen kann durch die Distanz zum anderen und die Distanz zwischen den SVen abgeschätzt werden. Ist nun bereits die rechte Seite der Ungleichung (3.17) größer als z_ϵ, so kann das Ergebnis von g durch g_∞ mit einem maximalen absoluten Fehler von ϵ abgeschätzt werden. In Abbildung 3.1 wird dieser Zusammenhang graphisch wiedergegeben.

Mit dieser Abschätzung wird das folgende Verfahren entwickelt :
Für jeden in der Summe (1.1) zu behandelnden SV $\mathbf{x}_i$ kann versucht werden, das Ergebnis eines noch nicht berechneten Terms $k(\mathbf{x}, \mathbf{x}_j)$ aus dem bereits berechneten $d(\mathbf{x}, \mathbf{x}_i)$ mit Hilfe der Abschätzung (3.17) und einer ϵ-Genauigkeit zu approximieren. Jeder SV, dessen Distanzabschätzung zu $\mathbf{x}$ den Wert z_ϵ überschreitet wird in der noch folgenden Summation übersprungen und seine Kernauswertung mit g_∞ approximiert.

[2] Diese Darstellung läßt sich stets trivial erreichen, wenn man für g die Identität wählt.

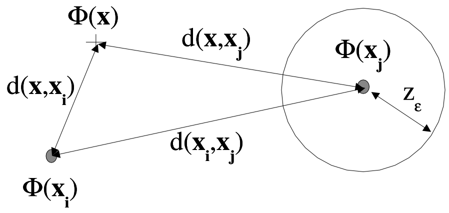

Abb. 3.1: *Der Wert von $d(\mathbf{x}, \mathbf{x}_j)$ wird mit der Ungleichung (3.17) gegen z_ϵ abgeschätzt. Außerhalb der Kugel um $\Phi(\mathbf{x}_j)$ mit Radius z_ϵ wird $d(\mathbf{x}, \mathbf{x}_j)$ mit g_∞ approximiert.*

Das ϵ kann dabei entweder fest gewählt werden oder abhängig von dem im i'ten Schritt erzielten Zwischenergebnis $sum_{(1,i)} - b$ adaptiv ermittelt werden. So kann ein maximaler absoluter bzw. relativer Fehler gut kontrolliert werden.

Beispiel : Die Exponentialfunktion nach Gleichung (2.35) kann durch die euklidsche Distanz d in Ω und die reelle Exponentialfunktion

$$z = d(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| \tag{3.18}$$

$$g(z) = \exp^{-\frac{z}{2\sigma^2}} \tag{3.19}$$

dargestellt werden. Desweiteren ist

$$\lim_{z \to \infty} g(z) = 0 = g_\infty \tag{3.20}$$

und für ein $\epsilon > 0$ gilt die Abschätzung (3.16) für jedes $z \geq z_\epsilon$ mit

$$z_\epsilon = 2\sigma^2 \ln \frac{1}{\epsilon}. \tag{3.21}$$

Dieses Verfahren bietet keine Möglichkeit der Kontrolle des Gesamtfehlers, weil in jeder Abschätzung ein maximaler Fehler zugelassen ist. Der Fehler kann nur durch die Limitierung der Anzahl der durchgeführten Abschätzungen erreicht werden. Durch die Verwendung einer ausschließlich absoluten Fehlerschranke birgt die Berechnung in dieser Form gerade bei kleinen Ergebnissen das Risiko eines großen relativen Fehlers.

3.2.2 Umsortierung der Support Vektoren

Auch bei diesem Verfahren macht es Sinn, wenn man die Menge der Support Vektoren vor der Anwendungsphase nach einem bestimmten Schema umsortiert. Um möglichst viele SVen bereits sehr früh in der Summenbildung (1.1) ausschließen zu können, bietet sich hier eine Vorsortierung nach der maximalen Distanzsumme

$$d_{\mathbf{x}_i} = \sum_{j=1}^{N_S} d(\mathbf{x}_i, \mathbf{x}_j) \tag{3.22}$$

zwischen den SVen an. Durch die maximale Distanzsumme wird ein Vektor ausgewählt, der möglichst weit von *allen* anderen entfernt ist, so daß möglichst viele Kernauswertungen durch die Abschätzung (3.16) in einem Schritt approximiert werden können. Der SV mit der größten Distanzsumme steht am weitesten außerhalb der Vektorschar und verspricht somit am ehesten, daß nach der Abschätzung (3.17) die Auswertung der Kernfunktion mit anderen Support Vektoren durch den Granzwert g_∞ ersetzt werden können.

3.2.3 Implementationsaspekte

Die Distanzen zwischen den Support Vektoren untereinander können bereits vor der Anwendungsphase bestimmt und in einer *Distanzmatrix* gespeichert werden.

Bei einer adaptiven Festlegung von z_ϵ in Abhängigkeit eines sich ändernden ϵs bietet sich das Anlegen einer Tabelle für die möglichen z_ϵ an, um in der Anwendungsphase Zeit zu sparen. Denkbar ist z.B. das Tolerieren eines maximalen prozentualen Fehlers p im jeweiligen Summationsschritt, so daß im Schritt q

$$\epsilon_q = p \cdot (sum_{(1,q)} - b) \tag{3.23}$$

gewählt wird. Das Berechnen der passenden z_ϵ kann für eine Quantisierung von $sum_{(1,q)}$ vor der Anwendungsphase geschehen, so daß der Wert ohne die Auswertung von (3.23) direkt aus einer Matrix gelesen werden kann.

3.3 Orthonormalisierung der Menge der Support Vektoren

3.3.1 Ansatz

Die grundlegende Idee des Verfahrens liegt darin, die Methodik aus Abschnitt 3.1 zu optimieren. Die Summation (1.1) soll nach möglichst wenigen Kernauswertungen abbrechen können um so eine geringe Rechenzeit zu erreichen. Außerdem ist eine genauere Abschätzung der Restsumme als in den Ungleichungen (3.7) bzw. (3.8) wünschenswert. Aus dem Verfahren nach Abschnitt 3.2 wird die Idee der Extraktion von Vorwissen über die Support Vektoren übernommen. Ähnlich wie die dort berechnete Distanzmatrix werden hier die Skalarprodukte der transformierten Support Vektoren vor und nach einer durchgeführten Orthonormalisierung im Merkmalsraum berechnet und in jeweils einer Matrix gespeichert. Von jeglichen Einschränkungen auf die Kernfunktion wird Abstand genommen.

In den folgenden Unterkapiteln wird gezeigt, daß die herkömmliche Berechnung Redundanz beinhalten kann. Eine konsequente Beseitigung dieser Redundanz führt zu einer orthonormalen Basis der Support Vektoren, einer neuen Berechnungsvorschrift für die Summe (1.1) und einer genaueren Restsummenabschätzung.

3.3.2 Eine graphische Interpretation der Menge der Support Vektoren

In Kapitel 2.2.6 wurde verdeutlicht, daß die Auswertung einer Kernfunktion k nach Mercer gerade der Berechnung des Skalarproduktes in einem Merkmalsraum F entspricht. Formuliert man die Summe (1.1) in F, so erhält man

$$\begin{aligned} f_{\{\mathbf{\Psi},b\}}(\mathbf{x}) &= \left(\sum_{i=1}^{N_S} \alpha_i k(\mathbf{x}, \mathbf{x}_i)\right) - b \\ &= \left(\sum_{i=1}^{N_S} \alpha_i \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)\right) - b \\ &= \left(\Phi(\mathbf{x}) \cdot \sum_{i=1}^{N_S} \alpha_i \Phi(\mathbf{x}_i)\right) - b \end{aligned}$$

$$= (\Phi(\mathbf{x}) \cdot \mathbf{\Psi}) - b. \qquad (3.24)$$

Von dieser Schreibweise lassen sich folgende Tatsachen ableiten :

- Die Support Vektoren bilden eine Vektorenmenge oder Vektorschar V im Merkmalsraum F.

$$V = \{\Phi(\mathbf{x}_i)\, ; i = 1..N_S\} \qquad (3.25)$$

- V spannt somit einen Unterraum F_S von F auf.

$$F_S = span\{V\} \qquad (3.26)$$

- V hat genau N_S Elemente und somit hat F_S maximal N_S Dimensionen.
- Bei der Berechnung von $\Phi(\mathbf{x}) \cdot \mathbf{\Psi}$ bzw. von $sum_{(1,N_S)}$ nach Gleichung (3.1) handelt es sich um eine Projektion von $\Phi(\mathbf{x})$ auf den Gewichtsvektor $\mathbf{\Psi}$ und somit in den Unterraum F_S.
- Die Anteile von $\Phi(\mathbf{x})$, die nicht in F_S liegen, sind für das Ergebnis der Summe (3.24) irrelevant.

Es ist offensichtlich, daß die Berechnung der Summe (3.24) Redundanz enthält, wenn zwei transformierte Support Vektoren aus S *linear abhängig* sind. I. A. sind die Elemente aus S jedoch linear unabhangig in F, aber *nicht orthogonal* zueinander. Jeder Vektor $\Phi(\mathbf{x})$ läßt sich in Vektoranteile unterteilen, die parallel bzw. orthogonal zu den anderen transformierten Support Vektoren sind. Betrachtet man die Kernauswertung weiter als Skalarmultiplikation, so wird klar, daß bei jeder Auswertung die Multiplikationen mit den parallelen Vektoranteilen redundant sind, da sie durch eine einfache Modifikation der Vorfaktoren α der anderen Support Vektoren überflüssig werden.

Beispiel : Betrachtet man eine Menge aus zwei SVen

$$V = \{\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2)\} \qquad (3.27)$$

mit dem zugehörigen Gewichtsvektor $\alpha = (\alpha_1, \alpha_2)^t$, so läßt sich der zweite Vektor durch die Projektion

$$\Phi(\mathbf{x}_2)_{||} = \left(\Phi(\mathbf{x}_2) \cdot \frac{\Phi(\mathbf{x}_1))}{|\Phi(\mathbf{x}_1)|}\right) \frac{\Phi(\mathbf{x}_1)}{|\Phi(\mathbf{x}_1)|} = K_{2,1} \frac{\Phi(\mathbf{x}_1)}{|\Phi(\mathbf{x}_1)|^2} \qquad (3.28)$$

und die Abkürzung

$$\Phi(\mathbf{x}_2)_\perp = \Phi(\mathbf{x}_2) - \Phi(\mathbf{x}_2)_{||} \tag{3.29}$$

in einen parallelen und orthogonalen Vektoranteil bezüglich des ersten unterteilen. K kürzt hier wiederum die Anwendung der Kernfunktion auf zwei SVen nach Gleichung (2.30) ab.

$$\Phi(\mathbf{x}_2) = \Phi(\mathbf{x}_2)_{||} + \Phi(\mathbf{x}_2)_\perp \tag{3.30}$$

Eine Skalarmultiplikation eines neuen Elementes $\Phi(\mathbf{x})$ mit $\Phi(\mathbf{x}_2)$

$$\begin{aligned} \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_2) &= \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x_2})_\perp + \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x_2})_{||} \\ &= \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x_2})_\perp + \beta\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_1) \end{aligned} \tag{3.31}$$

mit einem geeigenten $\beta \in \mathbb{R}$ enthält somit Redundanz, da die Multiplikation mit $\Phi(\mathbf{x}_2)_{||}$ nach einer bereits durchgeführten Multiplikation mit $\Phi(\mathbf{x}_1)$ überflüssig ist. Die Berechnung der Projektion auf den orthogonalen Anteil ist zwar unumgänglich, jedoch können die Vorteile der entstandenen linearen Unabhängigkeit der Ergebnisse zur besseren Restsummenabschätzung genutzt werden.

3.3.3 Die orthonormale Basis

Es ist wünschenswert die Redundanz in der Summenbildung (1.1) aus folgenden Gründen zu beseitigen.

- Bei jeder Kernauswertung soll die *größtmögliche Information* aus dem Ergebnis gezogen werden.
- Die *orthogonalen* Support Vektoren werden mit zunehmenden Fortschreiten in der Summation kleiner, so daß der Hauptteil der Summe (3.24) durch die ersten Support Vektoren bestimmt wird.
- Eine *orthonormale Basis* B von $F_S = span\{V\}$ kann darauf aufbauend hergeleitet werden, in dem eventuell vorhandene linear abhängige Elemente aus S verschwinden. B hat somit eventuell weniger Elemente als S und die Summationsgrenze N_S in (3.24) wird kleiner.

- Ein solches orthonormales System erlaubt eine wesentlich bessere Restsummenabschätzung als in der Approximation (3.8).

Es sollte hier erwähnt werden, daß die nun zu diskutierenden neuen Basiselemente in F nie konkret berechnet werden können. Es handelt sich um *imaginäre* Elemente[3], für die ausschließlich ein Algorithmus zur Ermittlung ihres Skalarproduktes mit einem anderen Element aus F hergeleitet werden kann. Diese Berechnung stützt sich wiederum lediglich auf das Skalarprodukt mit Elementen der ursprünglichen Support Vektor Menge V und ist mit Hilfe der Kernfunktion berechenbar.

Ein solches System läßt sich mit Hilfe des erweiterten Schmidt'schen Orthonormalisierungsverfahrens finden. Dieses Verfahren orthonormalisiert eine endliche Vektorschar und eliminiert dabei linear abhängige Vektoren. Es benutzt ausschließlich das Skalarprodukt und kann somit im Merkmalsraum F mit Kenntnis über die Kernfunktion k ausgeführt werden.

Die Orthonormalisierung

Die Orthonormalisierung besteht hauptsächlich aus der Bestimmung der Matrizen K und $\tilde{K}$, in denen die Anwendung der Kernfunktion auf die Support Vektoren und deren Orthonormalisierungen abgespeichert werden. Desweiteren werden in ϱ_i die Längen der *Orthogonal*vektoren abgespeichert. Diese Berechnungen geschehen bereits vor der Anwendungsphase und die Ergebnisse verändern sich für eine fertig gelernte Kernmethode nicht mehr.

Es ist V nach Gleichung (3.25) die Menge der transformierten Support Vektoren und somit eine Vektorschar in F. Es gelte o.B.d.A.

$$\forall i = 1..N_S \ : \ |\Phi(\mathbf{x}_i)| \neq 0. \tag{3.32}$$

Die *konkrete Berechnung* der orthonormalen Basis B wird *induktiv* aufgebaut.

Induktionsanker : Für i=1 sei das Element von B_1

$$\Phi\widetilde{(\mathbf{x}_1)} = \frac{\Phi(\mathbf{x}_1)_\perp}{|\Phi(\mathbf{x}_1)_\perp|} \tag{3.33}$$

[3] Dieser Name hat natürlich nichts mit der imaginären Einheit zu tun. Er wurde gewählt, um keine Verwirrung mit den *Virtuellen Support Vektoren* nach [4] zu schaffen.

wobei hier trivialerweise

$$\Phi(\mathbf{x}_1)_\perp = \Phi(\mathbf{x}_1) \tag{3.34}$$

gilt. Die Basis in diesem Schritt ist

$$B_1 = \left\{ \widetilde{\Phi(\mathbf{x}_1)} \right\}. \tag{3.35}$$

Die Anzahl der linear *abhängigen* Vektoren in V wird in u gezählt. Für $i = 1$ ist stets $u = 0$. Der Index des neuen Basiselementes in B_1 ist $\tilde{i}_1 = 1$. Es kann nun jedes Skalarprodukt mit $\widetilde{\Phi(\mathbf{x}_1)}$ mit der Kernfunktion durch

$$\widetilde{\Phi(\mathbf{x}_1)} \cdot \Phi(\mathbf{x}) = \frac{\Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x})}{|\Phi(\mathbf{x}_1)|} = \frac{k(\mathbf{x}_1, \mathbf{x})}{\sqrt{k(\mathbf{x}_1, \mathbf{x}_1)}} \tag{3.36}$$

berechnet werden. Die Multiplikation eines neuen Basisvektors mit einem Element aus V sei durch

$$\tilde{K}_{i,j} = \widetilde{\Phi(\mathbf{x}_i)} \cdot \Phi(\mathbf{x}_j) \tag{3.37}$$

abgekürzt[4]. An dieser Stelle ist somit die erste Zeile der Matrix $\tilde{K}$ mit $i = 1$ durch die Anwendung der Formel (3.36) auf jeden Support Vektor in V berechenbar. Desweiteren sei die Länge des orthogonalen Vektors durch

$$\varrho_1 = |\Phi(\mathbf{x}_1)_\perp| = \sqrt{k(\mathbf{x}_1, \mathbf{x}_1)} = \sqrt{K_{1,1}} \tag{3.38}$$

bezeichnet. Gleichung (3.36) liest sich somit als

$$\widetilde{\Phi(\mathbf{x}_1)} \cdot \Phi(\mathbf{x}) = \varrho_1^{-1} \Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}) \tag{3.39}$$

Induktionsannahme : Bestehe die Menge

$$B_i = \left\{ \widetilde{\Phi(\mathbf{x}_1)}, ..., \widetilde{\Phi(\mathbf{x}_{\tilde{i}_{(i-u)}})} \right\} \tag{3.40}$$

aus orthonormalen Elementen, für die

$$span\{B\} = span\{\Phi(\mathbf{x}_1), ..., \Phi(\mathbf{x}_i)\} \tag{3.41}$$

gilt. Die Anzahl der linear abhängigen Vektoren u in B_i ist gleich der Differenz der Anzahl der Elemente aus B_i und der Zahl i.

[4] Man beachte, daß $\tilde{K}$ *nicht symmetrisch* ist, weil der orthonormierte Vektor stets durch den ersten Index festgelegt wird. Die Werte mit i>j sind Null.

Induktionsschritt auf B_{i+1} : Zur Bildung der Menge B_{i+1} wird zuerst der potentiell neue *Orthogonalvektor*

$$\begin{aligned}\Phi(\mathbf{x}_{i+1})_\perp &= \Phi(\mathbf{x}_{i+1}) - \Phi(\mathbf{x}_{i+1})_{||}\\ &= \Phi(\mathbf{x}_{i+1}) - \sum_{n=1}^{i-u} \left(\Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) \cdot \Phi(\mathbf{x}_{i+1})\right)\ \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}})\\ &= \Phi(\mathbf{x}_{i+1}) - \sum_{n=1}^{i-u} \tilde{K}_{\tilde{i}_n,i+1}\Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}})\end{aligned} \tag{3.42}$$

bestimmt. Von dem ursprünglichen Support Vektor werden die parallelen Anteile vektoriell abgezogen. Da es sich bei B um eine *orthonormale* Basis handelt, werden die Koeffizienten der Summe in (3.42) durch eine Skalarmultiplikation des Support Vektors $\Phi(\mathbf{x}_{i+1})$ mit dem Orthonormalvektor $\Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}})$ ermittelt. Die Länge des neuen *orthogonalen* Vektors $\Phi(\mathbf{x}_{i+1})_\perp$ wird durch

$$\begin{aligned}\varrho_{i+1} = |\Phi(\mathbf{x}_{i+1})_\perp| &= \sqrt{\Phi(\mathbf{x}_{i+1})_\perp{}^2}\\ &= \sqrt{K_{i+1,i+1} - \sum_{n=1}^{i-j} \tilde{K}^2_{\tilde{i}_n,i+1}}\end{aligned} \tag{3.43}$$

beschrieben. Gleichung (3.43) ergibt sich durch das Quadrieren von (3.42) unter der Beachtung der Orthonormalität von B_i.

- Ist $\varrho_{i+1} = 0$, so handelte es sich bei $\Phi(\mathbf{x}_{i+1})$ um ein Element, das bereits durch die Basis B_i dargestellt werden konnte und ist somit unerheblich für die weitere Betrachtung. Die Anzahl der linear abhängigen Elemente wird mit $u = u + 1$ erhöht. Die Basis bleibt unverändert $B_{i+1} = B_i$. Die entsprechende Zeile in $\tilde{K}$ wird auf 0 gesetzt.

- Für den regulären Fall daß $\varrho_{i+1} > 0$ wird wie folgt verfahren :

  Das neue Element von B hat den Index $\tilde{i}_{i+1-u} = i + 1$ und wird durch

  $$\Phi(\widetilde{\mathbf{x}_{i+1}}) = \varrho_{i+1}^{-1}\Phi(\mathbf{x}_{i+1})_\perp \tag{3.44}$$

  bestimmt. Gleichungen (3.42) und (3.44) lassen sich zu

  $$\Phi(\widetilde{\mathbf{x}_{i+1}}) = \varrho_{i+1}^{-1}\left(\Phi(\mathbf{x}_{i+1}) - \sum_{n=1}^{i-u} \tilde{K}_{\tilde{i}_n,i+1}\Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}})\right) \tag{3.45}$$

zusammenfassen. Damit ist

$$B_{i+1} = B_i \cup \{\Phi(\widetilde{\mathbf{x}_{i+1}})\}. \tag{3.46}$$

Die im nächsten Schritt und zur Anwendung nötigen Werte der neuen Zeile von $\tilde{K}$ werden durch die Bildung des Skalarproduktes zwischen dem neuen Basiselement und einem transformierten Support Vektor mit

$$\begin{aligned}
\tilde{K}_{i+1,r} = \Phi(\widetilde{\mathbf{x}_{i+1}}) \cdot \Phi(\mathbf{x}_r) &= \varrho_{i+1}^{-1} \Phi(\mathbf{x}_{i+1})_\perp \cdot \Phi(\mathbf{x}_r) \\
&= \varrho_{i+1}^{-1} \Phi(\mathbf{x}_r) \left(\Phi(\mathbf{x}_{i+1}) - \sum_{n=1}^{i-u} \tilde{K}_{\tilde{i}_n, i+1} \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) \right) \\
&= \varrho_{i+1}^{-1} \left(K_{i+1,r} - \sum_{n=1}^{i-u} \tilde{K}_{\tilde{i}_n, i+1} \tilde{K}_{\tilde{i}_n, r} \right)
\end{aligned} \tag{3.47}$$

berechnet. Das Skalarprodukt eines beliebigen neuen Eingabevektors mit diesem neuen Basisvektor kann nun analog mit der Formel

$$\begin{aligned}
&\Phi(\mathbf{x}) \cdot \Phi(\widetilde{\mathbf{x}_{\tilde{i}_{i+1-u}}}) \\
&= \varrho_{i+1}^{-1} \Phi(\mathbf{x}) \cdot \left(\Phi(\mathbf{x}_{i+1}) - \sum_{n=1}^{i-u} \tilde{K}_{\tilde{i}_n, i+1} \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) \right) \\
&= \varrho_{i+1}^{-1} \left(k(\mathbf{x}_{i+1}, \mathbf{x}) - \sum_{n=1}^{i-u} \tilde{K}_{\tilde{i}_n, i+1} \left(\Phi(\mathbf{x}) \cdot \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) \right) \right)
\end{aligned} \tag{3.48}$$

ermittelt werden. Man beachte, daß sämtliche Werte, die in der Summe von (3.48) verwendet werden bereits berechnet sind, insbesondere ist das Skalarprodukt bereits durch die induktive Berechnungsformel bekannt. Die Hauptaufgabe besteht in der Ermittlung der Kernauswertung.

Durch dieses Verfahren erhält man eine *imaginäre orthonormale Basis* $B = B_{N_S}$ von F_S aus

$$N_B = N_S - j \leq N_S \tag{3.49}$$

Elementen. Konkret berechnet werden im Schritt der Orthonormalisierung die Matrizen K und $\tilde{K}$ sowie die Vektorlängen ϱ_i. Dieser Schritt geschieht vor der Anwendungsphase sobald die jeweilige Kernmethode eine Anwendungsvorschrift zur Verfügung stellt.

3.3.4 Die neue Darstellung des Gewichtsvektors $\boldsymbol{\Psi}$

Die Anwendung des jeweils betrachteten Kernverfahrens soll nun auf der Grundlage der neuen Basis geschehen. Insbesondere ist die Berechnung von $\Phi(\mathbf{x}) \cdot \boldsymbol{\Psi}$ in der neuen Basis zu bestimmen. Dazu wird die Darstellung von $\boldsymbol{\Psi}$ in B benötigt, welche durch

$$\boldsymbol{\Psi} = \sum_{n=1}^{N_S} \alpha_n \Phi(\mathbf{x}_n) = \sum_{n=1}^{N_B} \tilde{\alpha}_n \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) \tag{3.50}$$

festgelegt ist. Da es sich bei B um eine Orthonormalbasis handelt, lassen sich die Koeffizienten $\tilde{\alpha}$ durch

$$\begin{aligned}\tilde{\alpha}_t &= \boldsymbol{\Psi} \cdot \Phi(\widetilde{\mathbf{x}_{\tilde{i}_t}}) \\ &= \left(\sum_{n=1}^{N_S} \alpha_n \Phi(\mathbf{x}_n)\right) \cdot \Phi(\widetilde{\mathbf{x}_{\tilde{i}_t}}) \\ &= \sum_{n=1}^{N_S} \alpha_n \left(\Phi(\mathbf{x}_n) \cdot \Phi(\widetilde{\mathbf{x}_{\tilde{i}_t}})\right) \\ &= \sum_{n=1}^{N_S} \alpha_n \tilde{K}_{t,n}\end{aligned} \tag{3.51}$$

mit Hilfe der Matrix $\tilde{K}$ nach Gleichung (3.37) berechnen. Dieser Schritt kann wie die Orthonormalisierung vor der Anwendungsphase durchgeführt werden.

3.3.5 Die neue Darstellung des Eingabeelementes $\Phi(\mathbf{x})$

Analog zu $\boldsymbol{\Psi}$ läßt sich auch $\Phi(\mathbf{x})$ in der neuen Basis B darstellen. Es ist

$$\Phi(\mathbf{x}) = \sum_{n=1}^{N_B} (\Phi(\mathbf{x}) \cdot \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}))\Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) = \sum_{n=1}^{N_B} \Theta_n \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) \tag{3.52}$$

wobei Θ_n die Multiplikation des neuen transformierten Eingabeelementes mit einem transformierten Support Vektor abkürzt. Diese Werte müssen in der Anwendungsphase bestimmt werden und bedeuten einen prinzipiellen Mehraufwand, der noch kompensiert werden muß um eine Effizienzsteigerung zu erreichen.

3.3.6 Die neue Anwendungsvorschrift : Berechnung des Skalarproduktes in F_S

Die Berechnung des Skalarproduktes $\Phi(\mathbf{x}) \cdot \mathbf{\Psi}$ mit einem beliebigen Element $\Phi(\mathbf{x})$ kann nun in F_S auf der Grundlage von B berechnet werden. Es ist

$$\Phi(\mathbf{x}) \cdot \mathbf{\Psi} = \Phi(\mathbf{x}) \cdot \sum_{n=1}^{N_B} \tilde{\alpha}_n \widetilde{\Phi(\mathbf{x}_{\tilde{i}_n})} = \sum_{n=1}^{N_B} \tilde{\alpha}_n \left(\widetilde{\Phi(\mathbf{x}_{\tilde{i}_n})} \cdot \Phi(\mathbf{x}) \right), \tag{3.53}$$

wobei die Summe nach dem induktiven Aufbau von B sukzessiv berechnet werden kann. Die Skalarprodukte ergeben sich dazu aus den Gleichungen (3.36) und (3.48) aus Abschnitt 3.3.3. Es soll hier die Schreibweise

$$\widetilde{sum}_{(j,q)} = \sum_{n=j}^{q} \tilde{\alpha}_n \Phi(\mathbf{x}) \cdot \widetilde{\Phi(\mathbf{x}_{\tilde{i}_n})} \tag{3.54}$$

ähnlich der Abkürzung (3.1) benutzt werden.

Zuerst wird die Skalarmultiplikation mit dem ersten Element aus B mit

$$\Theta_1 = \widetilde{\Phi(\mathbf{x}_{\tilde{i}_1})} \cdot \Phi(\mathbf{x}) = \frac{k(\mathbf{x}_1, \mathbf{x})}{\sqrt{K_{1,1}}} \tag{3.55}$$

bestimmt, so daß

$$\widetilde{sum}_{(1,1)} = \tilde{\alpha}_1 \Theta_1 \tag{3.56}$$

ist. Allgemein läßt sich durch die rekursive Berechnung von Θ_k mit $1 < k \leq N_B$ nach Gleichung (3.45) gemäß

$$\begin{aligned} \Theta_q &= \widetilde{\Phi(\mathbf{x}_{\tilde{i}_q})} \cdot \Phi(\mathbf{x}) \\ &= \varrho_q^{-1} \Phi(\mathbf{x}) \cdot \left(\Phi(\mathbf{x}_q) - \sum_{n=1}^{q-1} \tilde{K}_{\tilde{i}_n,q} \widetilde{\Phi(\mathbf{x}_{\tilde{i}_n})} \right) \\ &= \varrho_q^{-1} \left(k(\mathbf{x}, \mathbf{x}_q) - \sum_{n=1}^{q-1} \tilde{K}_{\tilde{i}_n,q} \Theta_n \right) \end{aligned} \tag{3.57}$$

die Summe $\widetilde{sum}_{(1,N_B)}$ sukzessiv berechnen.

$$\widetilde{sum}_{(1,q)} = \widetilde{sum}_{(1,q-1)} + \tilde{\alpha}_q \Theta_q \tag{3.58}$$

Das Skalarprodukt wird schließlich durch

$$\Phi(\mathbf{x}) \cdot \mathbf{\Psi} = \widetilde{sum}_{(1,N_B)} \tag{3.59}$$

bestimmt.

3.3.7 Abbruch der Summation

Die Reformulierung (3.53) in B hat einige Vorteile gegenüber der Summe (1.1). Die einzelnen Summanden sind geometrisch voneinander unabhängig, so daß bei einer sukzessiven Berechnung durch $\widetilde{sum}$ nach jedem Summationsschritt q folgende Tatsachen bekannt sind :

- Das Skalarprodukt $\Phi(\mathbf{x}) \cdot \mathbf{\Psi}$ ist bisher durch die Summe $\widetilde{sum}_{(1,q)}$ angenähert berechnet. Die Restsumme ist $\widetilde{sum}_{(q+1,N_B)}$.
- Die bisher noch nicht behandelten $N_B - q$ Elemente von B spannen einen $(N_B - q)$-dimensionalen Unterraum von F_S auf.
- Der noch nicht behandelte Anteil von $\mathbf{\Psi}$ ist
$$\mathbf{\Psi}_q = \sum_{n=q+1}^{N_B} \tilde{\alpha}_n \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}) \tag{3.60}$$
und hat eine euklidsche Norm
$$|\mathbf{\Psi}_q| = \sqrt{\sum_{n=q+1}^{N_B} \tilde{\alpha}_n^2}. \tag{3.61}$$
- Der noch nicht behandelte Anteil von $\Phi(\mathbf{x})$ ist
$$\Phi(\mathbf{x})_q = \Phi(\mathbf{x}) - \sum_{n=1}^{q} \Theta_n \Phi(\widetilde{\mathbf{x}_{\tilde{i}_n}}). \tag{3.62}$$
Von dem ursprünglichen Vektor werden die bereits behandelten Anteile abgezogen. Wegen der linearen Unabhängigkeit haben diese Anteile keinen weiteren Einfluß auf das Ergebnis der Restsumme. Die euklidsche Norm dieses noch relevanten Vektoranteils ist
$$|\Phi(\mathbf{x})_q| = \sqrt{k(\mathbf{x},\mathbf{x}) - \sum_{n=1}^{q} {\Theta_n}^2}. \tag{3.63}$$

Mit diesem Wissen kann die Restsumme durch

$$|\widetilde{sum}_{(q+1,N_B)}| = |\Phi(\mathbf{x})_\mathbf{q} \cdot \mathbf{\Psi}_\mathbf{q}| \leq |\mathbf{\Phi}(\mathbf{x})_\mathbf{q}| \cdot |\mathbf{\Psi}_\mathbf{q}| \tag{3.64}$$

abgeschätzt werden. Diese Abschätzung ist wesentlich genauer als die des Verfahrens zur Ausnutzung der einfachen Restsummenabschätzung gemäß Kapitel 3.1. Im Gegensatz zu der aus den Gleichungen (3.5) und (3.8) resultierenden Grenzen ist die hier benutzte Grenze wesentlich genauer. Es wird lediglich eine Anwendung der Cauchy-Schwartz'schen Ungleichung durchgeführt, anstatt sie wie in Gl. (3.5) in jedem Summanden zu benutzen. Darüber hinaus war es vorher nicht möglich einen bereits behandelten Teil von $\Phi(\mathbf{x})$ zu definieren, geschweige denn ihn in der Abschätzung berücksichtigen zu können. Desweiteren trägt die redundanzfreie Darstellung von $\mathbf{\Psi}$ in B zur besseren Abschätzung bei. Wie auch in Abbildung 3.2 zu erkennen ist sind die Vektoranteile in der neuen Darstellung kleiner. Die Abschätzung läuft nicht wie bisher über die gesamte Länge des Support Vektors sondern beschränkt sich auf seinen linear unabhängigen, orthogonalen Anteil.

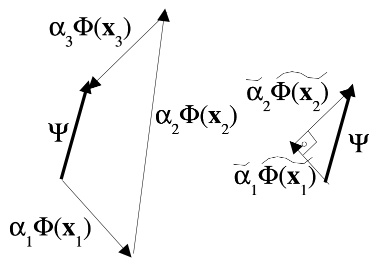

Abb. 3.2: *Die neue orthonormale Darstellung von Ψ auf der rechten Seite läßt sich wesentlich besser Abschätzen als in der redundanten Darstellung links. Der linear abhängige Support Vektor $\Phi(\mathbf{x_3})$ fällt ganz heraus und von $\Phi(\mathbf{x_3})$ bleibt lediglich der normierte zu $\Phi(\mathbf{x_1})$ orthogonale Anteil übrig.*

Analog zu Abschnitt 3.2 kann auch hier die Summation unter bestimmten Voraussetzungen bei einem $q < N_B$ abgebrochen werden. Soll ein maximaler relativer Fehler von p Prozent erreicht werden, so kann die Bedingung

$$|\Phi(\mathbf{x})_q||\mathbf{\Psi}_q| \leq p \cdot \left(|\widetilde{sum}_{(1,q)}| - b\right) \tag{3.65}$$

nach jedem Summationsschritt für steigendes q überprüft werden. Für den Fall der binären Klassifikation genügt wiederum die Bedingung

$$|\Phi(\mathbf{x})_q||\mathbf{\Psi}_q| < \left(|\widetilde{sum}_{(1,q)}| - b\right) \tag{3.66}$$

zum Summationsabbruch. Ist die Abbruchbedingung (3.66) erfüllt, so kann sich das Vorzeichen des Endergebnisses der Summe nicht mehr verändern. Die Restsumme hat dann keinen Einfluß mehr auf die endgültige Klassenzuteilung des Verfahrens und das Ergebnis steht fest.

Bemerkung : Selbst nach einer Summation bis $q = N_B$ kann $|\Phi(\mathbf{x})_{N_B}| \gg 0$ sein. Es handelt sich lediglich um den Betrag des Vektors $\Phi(\mathbf{x})$, von dem seine Projektion in den Unterraum F_S abgezogen ist. $|\Phi(\mathbf{x})_{N_B}|$ beschreibt also die Länge des Vektoranteils von $\Phi(\mathbf{x})$, der *nicht* in dem betrachteten Unterraum dargestellt werden kann.

3.3.8 Umsortierung der Support Vektoren

Der approximative Wert $\widetilde{sum}_{(1,q)}$ wächst mit steigendem q schneller an, wenn die Orthonormalvektoren mit maximalem $\tilde{\alpha}$ am Anfang der Summe stehen. Dafür wird die Reihenfolge der SVen während der Bestimmung der neuen Basis verändert. In jedem Schritt zur Berechnung des neuen Elementes von B wird für jeden SV, der noch nicht durch B dargestellt werden kann, das zugehörige $\tilde{\alpha}$ berechnet. In B wird jedoch nur der orthonormale Vektor mit maximalem $\tilde{\alpha}$ übernommen. D.h. die SVen mit dem größten Anteil an $\mathbf{\Psi}$ kommen in der Summe an den Anfang.

Im weiteren wird auf die Unterscheidung zwischen N_S und N_B verzichtet. Ein linear abhängiger Support Vektor wird als orthonormaler Support Vektor mit Koeffizient $\tilde{\alpha} = 0$ angenommen. Somit steht er stets am Ende der Summe und wird durch die Abschätzung zu 0 immer vernachlässigt. Praktisch kommt bei den betrachteten hochdimensionalen Problemen so gut wie keine lineare Abhängigkeit in der Menge der Support Vektoren vor.

3.3.9 Laufzeitbetrachtung

Grundsätzlich hat die sukzessive Berechnung des Skalarproduktes $\Phi(\mathbf{x}) \cdot \mathbf{\Psi}$ nach dem neuen Verfahren eine ungünstigere Laufzeit als mit der herkömmlichen Berechnung der Summe (1.1). Das alte Verfahren wertet die Kernfunktion für jeden SV aus und kommt somit zu einer Laufzeit in $O(N \cdot N_S)$. Läuft das neue Verfahren ohne vorzeitigen Summationsabbruch durch, so liegt die Laufzeit in $O(N \cdot N_S + N_S^2)$. Die aufwendige Berechnung der neuen Darstellung des Eingabevektors durch die Koeffizienten Θ_i bei jedem Summationsschritt ist zu den Kernauswertungen hinzugekommen. Betrachtet man die Verteilung der Rechenzeit genauer, so läßt sich erkennen, daß nach dem q-ten Summationsschritt im herkömmlichen Verfahren $N \cdot q$ Schleifendurchläufe unternommen werden mußten, wohingegen das Orthonormalverfahren auf $N \cdot q + \sum_{i=1}^{q} i = N \cdot q + \frac{(q+1)q}{2}$ Schleifendurchläufe kommt. Der erste Teil dieser Summe beschreibt die Laufzeit für die q nötigen Kernauswertungen und der zweite Teil berechnet den wachsenden Mehraufwand für die Berechnung der Θ. Diese grobe Abschätzung läßt bereits erkennen, daß dieses Verfahren nur dann einen Geschwindigkeitsvorteil bewirken kann, wenn die Summation bei einem $q < N_S$ abbrechen darf. In Abbildung 3.3 ist der Verlauf der Laufzeit abhängig von q im Vergleich zu der bisher konstanten Laufzeit der einfachen Summenbildung nach Gleichung (1.1) aufgetragen.

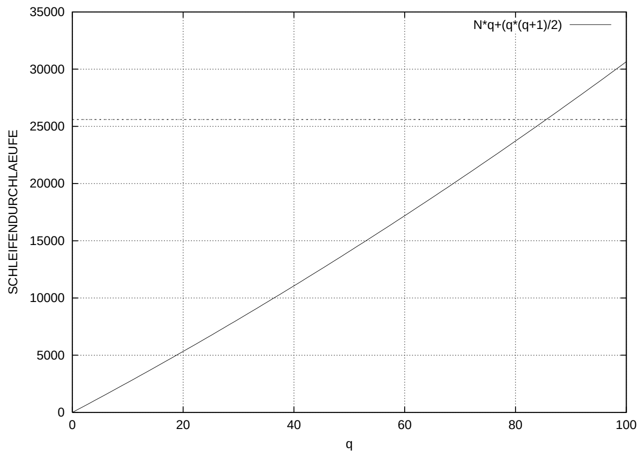

Abb. 3.3: *Zeitabschätzung des Orthonormalverfahrens gegenüber der herkömmlichen Summation. Bei einer Dimension des Eingaberaumes* $N = 256$ *und einem Klassifizierer mit* $N_S = 100$ *Support Vektoren dürfen im Schnitt nicht mehr als* $q_{max} = 86$ *orthonormale SVen benutzt werden. Die konstante Grenze* $N \cdot N_S = 25600$ *ist als horizontale Linie eingezeichnet.*

Genauer gesagt muß bei dem Summationsabbruch mindestens die Bedingung

$$\frac{q^2 + q}{2} < N \cdot (N_S - q) \tag{3.67}$$

erfüllt sein, um eine Beschleunigung der Berechnung zu erreichen. Die rechte Seite der Ungleichung beschreibt den gesparten Aufwand für die Berechnung der Kernauswertungen und die linke Seite bildet der Term für den zusätzlichen Aufwand zur Berechnung der Θ_i. Die Ungleichung kann nur erfüllt sein, wenn ein geringer Overhead durch möglichst kleines q einer großen Zeitersparnis gegenüber steht. Zu jedem Klassifikator kann ein spezifischer Wert q_{max} berechnet werden, der die maximale Anzahl von Kernauswertungen angibt, bei denen noch ein Geschwindigkeitsvorteil zu erwarten ist. Dieser Grenzwert für q ergibt sich durch die Auflösung der Gleichheit in (3.67) zu

$$q_{max} = \sqrt{2N \cdot N_S + \left(\frac{2N+1}{2}\right)^2} - \frac{2N+1}{2}. \tag{3.68}$$

In den Abbildungen 3.4 und 3.5 ist der Verlauf verschiedener q_{max}-Kurven aufgetragen, wobei jeweils eine der entscheidenden Größen N bzw. N_S konstant gelassen wird. Daraus wird deutlich,daß das Verfahren besonders für Problemstellungen in hochdimensionalen Eingaberäumen Ω mit $N \gg N_S$ konzipiert ist. Bei hochdimensionalen Problemen bringt bereits die Einsparung von wenigen Kernauswertungen einen erheblichen Geschwindigkeitsvorteil.

Die Identität im Graphen 3.4 beschreibt den Idealfall, in dem selbst bei der Verwendung aller Support Vektoren noch eine Beschleunigung der Berechnung erfolgt. Diese Gerade kann durch den Mehraufwand natürlich nicht erreicht werden, jedoch wird sie für hohe Dimensionen N des Eingaberaumes recht gut angenähert. Auffallend ist es, daß sich bei einer Zunahme der Support Vektoren die Kurve weiter von der Ideallinie entfernt. Je mehr Support Vektoren vorhanden sind, um so größer muß auch der *Prozentsatz* der eingesparten Kernauswertungen sein. Dies wird durch das quadratische Verhalten des Mehraufwandes zur Berechnung der Θ_i verursacht.

Abbildung 3.5 zeigt das Verhalten von Gleichung (3.68) in Abhängigkeit von der Dimension N. Im unteren Wertebereich steigt die Kurve zunächst stark an um sich daraufhin dem Idealwert der jeweils fixen Anzahl von Support Vektoren N_S anzunähern. Für hohe Dimensionen lohnt sich die einzelne Einsparung einer Kernauswertung mehr, so daß in der schnelleren Berechnung insgesamt mehr SVen zur Auswertung der Anwendung verwendet werden dürfen.

3.3.10 Implementationsaspekte

Auch in diesem Verfahren ist eine adaptive Ordnung der SVen anstelle der in Kapitel 3.3.8 vorgeschlagenen Methode denkbar. Es könnte für jede Reihen-

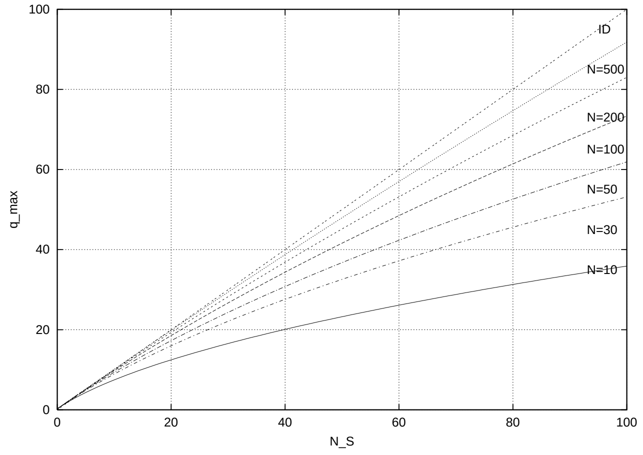

Abb. 3.4: *Gleichung (3.68) für $N_S = [0..100]$ aufgetragen. Die Abhängigkeit der maximal erlaubten Anzahl orthonormaler SVen q_{max} von der Anzahl der Support Vektoren N_S ist deutlich zu erkennen. Je größer die N_S, desto weiter entfernt sich die Kurve von der unerreichbaren Idealen der Identität, wo $q_{max} = N_S$ bereits eine Beschleunigung versprechen würde. Je mehr Support Vektoren verwendet werden, um so größer muß auch der* Prozentsatz *der eingesparten Kernauswertungen sein.*

folge der SVen eine eigene Basis B generiert werden. In der Anwendungsphase ist es dann möglich, in Abhängigkeit von den bereits berechneten Summanden den statistisch besten Basisvektor zur Berechnung des nächsten Summanden zu bestimmen. Dieses Verfahren würde ein früheres Abbrechen der Summation versprechen. Leider ist das Verfahren bereits für eine moderate Anzahl von SVen aus Speichergründen nicht realisierbar. Für N_S SVen werden alleine schon $N_S!$ Speicherstellen für die verschiedenen $\tilde{\alpha}$ benötigt, desweiteren muß für jede der gespeicherten $N_S!$ Reihenfolgen die zugehörige Matrix $\tilde{K}$ verwaltet werden. Es ist somit klar, daß dieser Ansatz physikalisch nicht durchführbar ist. Selbst die abgewandelte Version, in der nach jeder Berechnung eines Summanden lediglich zwischen *zwei* Nachfolgern entschieden wird, ist für reale Anwendungen mit einem Speicheraufwand von $2^{(N_S-1)}$ nicht realisierbar.

Das Verfahren beherbergt die Gefahr einer *numerischen Instabilität.* Der orthogonale Anteil eines SVs kann u.U. sehr klein werden. Da zur Normierung der Basis in Gleichung (3.45) jedoch durch seine Länge ϱ geteilt wird, kann es

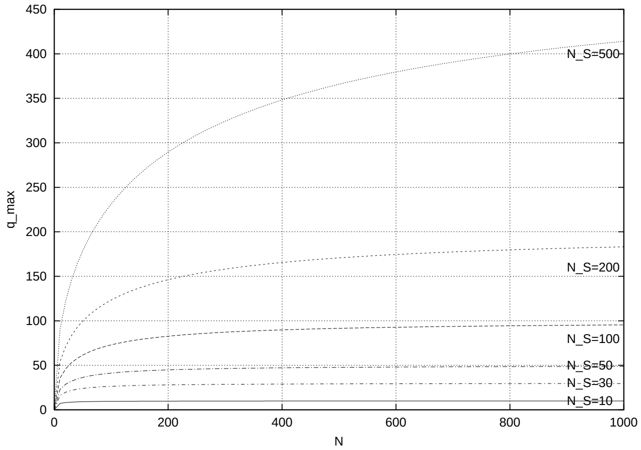

Abb. 3.5: *Gleichung (3.68) für die Dimension $N = [0..1000]$ aufgetragen. Für hohe Dimensionen müssen nur weniger Kernauswertungen in der Rechnung eingespart werden, um einen Geschwindigkeitsvorteil zu erzielen.*

hier zu numerischen Komplikationen kommen. Durch einen geeigneten Threshold ϱ_ϵ bei der Prüfung $\varrho \leq \varrho_\epsilon$ auf lineare Abhängigkeit im Induktionsschritt kann dieses Problem jedoch weitestgehend umgangen werden.

4. AUSWERTUNG DER EXPERIMENTE

4.1 Verwendete Daten

Die Experimente basieren auf sechs Datensätzen zur binären Klassifikation. Es handelt sich um fünf Textdatensätze und einen Bilddatensatz, deren Klassifikation per Hand vorgenommen ist. Einige wichtige Größen der Datensätze sind in Tabelle 4.1.2 zusammengefaßt.

4.1.1 Textdatensätze

Die Textdatensätze bestehen aus einzelnen HTML-Seiten, die durch eine zufällige Auswahl aus dem Internet geladen wurden. Jeder Trainingsdatensatz ist künstlich durch ein weiteres Auswahlverfahren auf Trainingsdatensätze geringerer Größe gebracht, um so mehr Experimente durchführen zu können. In 10%-Schritten sind die Trainingsdatensätze auf 10% bis 100% ihrer ursprünglichen Größe dezimiert, so daß effektiv 60 Trainingsdatensätze zur Verfügung stehen. Bei der Unterteilung ist das Verhältnis zwischen den Anzahlen der jeweiligen Klassenelementen in allen zehn Trainingsdatensätzen weitestgehend konstant. Die einzelnen zusammengehörenden zehn Mengen sind bewußt nicht disjunkt gewählt worden, um einen aussagekräftigen Verlauf bestimmter Größen wie die Anzahl der SVen N_S und die geschätzte VC-Dimension verfolgen zu können. Eine Meßreihe besteht aus den Meßwerten dieser jeweils zehn Untermengen. Die Werte einer Meßreihe sind in den Graphen verbunden dargestellt. Die verwendete Distanz basiert auf Worthistogrammen, in denen die Häufigkeit bestimmter Schlüsselwörter in einem Text gezählt werden.

4.1.2 Bilddatensatz

Der Bilddatensatz besteht aus einer zufälligen Auswahl aus den im Internet verfügbaren Bildern. Er dient der Unterscheidung zwischen Abbildern von realen Szenen wie Photos und künstlich generierten Bildern wie Grafiken. Aus diesem Datensatz wurden ebenfalls nach dem selben Prinzip wie bei den Textdatensätzen zehn verschiedene neue Datensätze generiert. Das verwendete Distanzmaß basiert auf einem Vergleich im Fourier-Spektrum.

Name	Abkürzung	Art	#Training	#Test	N
Englisch	E	Text	659	773	4801
Deutsch	D	Text	561	347	3307
Deutsch/Englisch	Text	DE	971	1119	4930
Sachgebiet	P	Text	1181	1251	3529
Bilder/Grafiken	BG	Bilder	160	1166	N/A

Abb. 4.1: *Zusammenfassende Darstellung der Datensätze. Die Elemente von BG haben keine konstante Dimension.*

4.2 Durchgeführte Messungen

Jeder Datensatz wurde einmal mit einem linearen und mehreren exponentiellen Kernfunktionen trainiert. Von den Klassifizierern mit einem exponentiellen Kern wurde lediglich derjenige mit dem geringsten Klassifikationsfehler auf dem Testdatensatz zur Messung ausgewählt. Gemessen wurde jeweils der absolute Zeitbedarf der direkten Summenberechnung der Vorschrift (1.1) und der Zeitbedarf der neuen Klassifikationsmethoden mit einem vorzeitigen Summationsabbruch. Die neuen Verfahren wurden so eingestellt, daß es stets zum gleichen Klassifikationsergebnis kam. Im Vergleich zum herkömmlichen Verfahren hat also keines der Verfahren ein abweichendes Ergebnis erzielt. Die Messungen wurden auf einem herkömmlichen PC mit 500Mhz durchgeführt. Jeder einzelne Zeitmeßwert ergibt sich aus der Klassifikation des gesamten Testdatensatzes und ist somit ein gemitteltes Ergebnis.

4.2.1 Die orthonormale Darstellung von Ψ

Die Geschwindigkeit des Orthonormalisierungsverfahrens ist stark von der verbesserten Darstellung des Gewichtsvektors $\mathbf{\Psi}$ in B abhängig. Durch die

Vermeidung der redundanten orthogonalen Anteile in den Support Vektoren ist gemäß Abschnitt 3.3.4 eine günstigere Zerlegung von $\mathbf{\Psi}$ gegeben. Die Abschätzung der Restsumme ist sowohl in der einfachen Form laut Kapitel 3.1 sowie in der orthonormierten Betrachtung in 3.3.7 stark von der Größe der Koeffizienten α_i bzw. $\tilde{\alpha_i}$ abhängig. In Abbildung 4.2 sind diese Koeffizienten für einen Klassifikator aufgetragen. Es wird deutlich, daß die Koeffizienten $\tilde{\alpha_i}$ der orthonormierten Darstellung wesentlich kleiner sind als in der ursprünglichen durch die Support Vektor Maschine bestimmten Zusammensetzung.

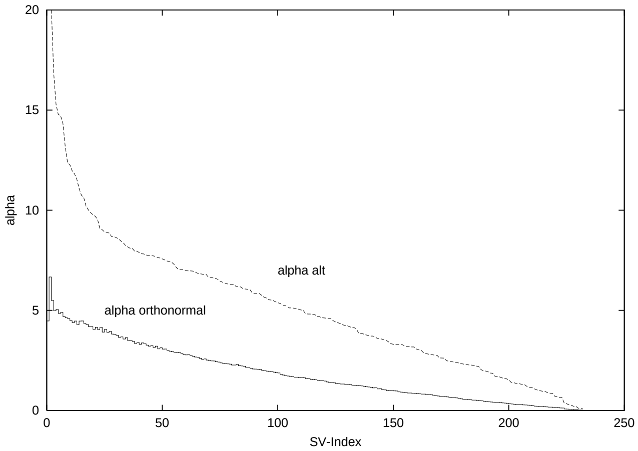

Abb. 4.2: *Aufgetragen sind die Koeffizienten α (alpha alt) der ursprünglichen Darstellung und die Koeffizienten $\tilde{\alpha}$ (alpha orthonormal) der orthonormalen Zerlegung von $\mathbf{\Psi}$ eines Klassifikators des Deutsch-Datensatzes.*

4.2.2 Statistische Verteilung der Zeitersparnis

Wie in Kapitel 3.3.9 bereits theoretisch erläutert wurde, ist gerade das Orthonormalisierungsverfahren nicht in jedem Fall schneller als die gewöhnliche Summenberechnung. Eine Zeitersparnis ergibt sich meistens erst im Mittel von vielen Anwendungen. Abbildung 4.3 gibt eine exemplarische Übersicht, wie die Zeitersparnis von ca. 60% bei der Anwendung des Deutsch-Klassifikators auf einem der Testdatensätze unter der Verwendung einer linearen Kernfunktion zustande kommt. Aufgetragen ist das Histogramm über die Anzahl der tatsächlich berechneten Kernauswertungen für einen Testlauf mit allen 347 Testelementen.

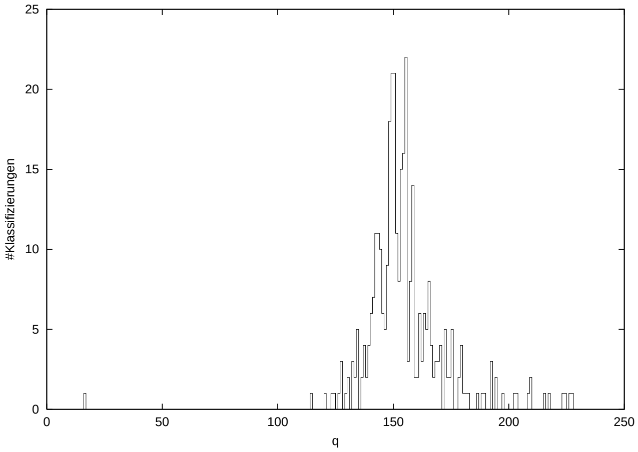

Abb. 4.3: *Histogramm über die Anzahl der verwendeten Kernauswertungen bei einem Testlauf auf dem Datensatz Deutsch. Alle Werte liegen unter dem theoretischen $q_{max} = 225$, so daß eine hohe Beschleunigungsrate erzielt werden kann.*

Im Gegensatz dazu ist in der Abbildung 4.4 das Histogramm eines Testlaufes aufgetragen, bei dem das Orthonormalisierungsverfahren keinen Geschwindigkeitsvorteil erreichen kann. Die Anwendung ist in dem Fall des Deutsch/Englisch Klassifikators ca. 10% langsamer als die herkömmliche Berechnung der Summe (1.1). Dieser Trainingsdatensatz hat die meisten Elemente und erzielt die meisten Support Vektoren aller betrachteten Meßreihen nach dem Training. Der Schwerpunkt im Historamm liegt zwar immer noch deutlich weiter links als der theoretisch berechnete Wert q_{max}, jedoch steigt der Mehraufwand in der Berechnung mit wachsender Anzahl von berechneten Kernauswertungen q quadratisch an. Obwohl nur eine Minderheit der Testdaten mehr als q_{max} Kernauswertungen benötigen, fällt ihr Geschwindigkeitsnachteil deutlich gravierender ins Gewicht als die von den übrigen Klassifikationen erzielte Beschleunigung.

4.2.3 Zeitmessungen

In Kapitel 3.3.9 wurde bereits erwähnt, daß das vorgestellte Verfahren nicht unter allen Umständen eine beschleunigte Anwendung zur Folge haben muß. Aus den Meßdaten soll hier nun in erster Linie analysiert werden, ob es charakteristische Größen einer SV Maschine gibt, die den Erfolg des Verfahrens

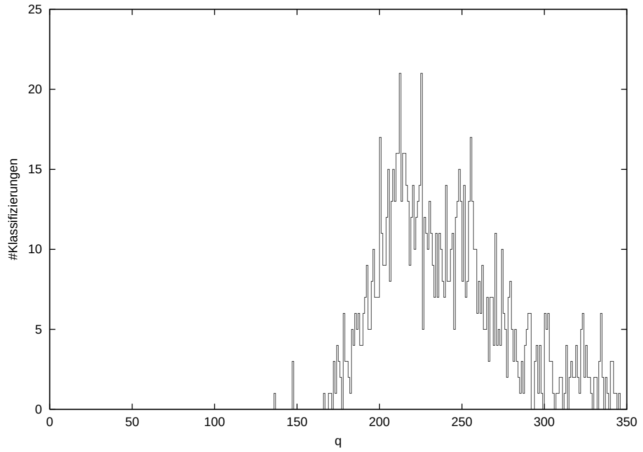

Abb. 4.4: *Histogramm über die Anzahl der verwendeten Kernauswertungen bei einem Testlauf auf dem Datensatz Deutsch-Englisch. Die Klassifikationen mit mehr als $q_{max} = 321$ fallen durch Ihren mit q quadratisch wachsenden Mehraufwand so sehr ins Gewicht, daß im Mittel keine Beschleunigung erreicht wird.*

beeinflussen. Die betrachteten Größen lassen sich in drei Kategorien einteilen.

- *Datensatzspezifische* Größen wie Dimension des Eingaberaumes N und Anzahl der Trainingsdaten.

- Eine *modellspezifische* Größe des Klassifikators liefert die geschätzte VC-Dimension.

- Die Anzahl der Support Vektoren N_S wird sowohl von der Art des Datensatzes als auch von dem gewählten Klassifikator bestimmt und bietet somit eine *übergreifende* Größe zwischen den zwei Komponenten.

Um Plattformunabhängige Meßwerte zu erhalten ist der Zeitgewinn als Quotient $\frac{t_{neu}}{t_{alt}}$ angegeben. Ein Verhältnis $\frac{t_{neu}}{t_{alt}} < 1$ bedeutet somit eine Effizienzverbesserung mit dem jeweils untersuchten neuen Verfahren im Vergleich zu der direkten Berechnung der Summe (1.1). Werte größer als eins drücken eine Verlangsamung aus. Wenn nicht anders angegeben beziehen sich die Messungen auf einen Klassifikator mit einer exponentiellen Kernfunktion nach Gleichung (2.35). Das Kürzel 'L' am Ende der Datensatzabkürzung gibt die Verwendung einer linearen Kernfunktion an, d.h. es wird das Skalarprodukt im Eingaberaum verwendet[1]. Das Kürzel 'DEL' bedeutet z.B., daß die Meßreihe aus dem Deutsch-Englisch Datensatz unter der Verwendung eines linearen Kernes resultiert. Die exakten Ergebnisse der Experimente sind im Anhang unter B zu finden.

Einfluß der SV-Anzahl

In den Graphen 4.5,4.6 und 4.7 sind die Beschleunigungsraten der drei Verfahren gegen die Anzahl der SVen des Klassifikators aufgetragen. Um die Verläufe besser vergleichen zu können, sind die Anzahlen der SVen auf einer normierten Skala aufgetragen. Aus einer Meßreihe über die jeweils zehn Unterdatensätze bezeichnen N_{min} und N_{max} die jeweils kleinste und größte Anzahl von Support Vektoren N_S. Die Anzahl der Support Vektoren N_S wird daruaf basierend durch die Formel

$$\#SV = \frac{N_S - N_{min}}{N_{max} - N_{min}} \tag{4.1}$$

auf den Bereich [0..1] normiert. Über dieser normierten Skala sind die gemessenen relativen Beschleunigungen $\frac{t_{neu}}{t_{alt}}$ aufgetragen. Abbildung 4.5 zeigt die Beschleunigungsrate für das auf der Restsummenabschätzung basierenden Verfahren aus Abschnitt 3.1 in Abhängigkeit von der normierten Anzahl der Support Vektoren gemäß Gl. (4.1).

Die Kurve im Graphen 4.5 läßt keine eindeutige Tendenz erkennen. Eine generelle Effizienzverbesserung mit steigender Anzahl der SVen ist anzunehmen, weil das Verfahren so gut wie keinen Mehraufwand in der Anwendungsphase benötigt und bei mehr SVen die Wahrscheinlichkeit einfach höher ist, daß einige Kernauswertungen eingespart werden können. Die Zeiteinsparungen liegen im einstelligen Prozentbereich.

Die Abbildung 4.6 zeigt den Verlauf der relativen Laufzeit gegen die normierte Anzahl von SVen für das Verfahren zur Ausnutzung der Distanzschätzung aus Abschnitt 3.2.

Die Werte liegen praktisch alle über eins und zeigen, daß dieses Verfahren nicht zur Geschwindigkeitsverbesserung für die betrachteten Problemstellungen geeignet ist. Mit wachsender Anzahl der SVen steigt auch der Zeitbedarf. Der Mehraufwand zur Distanzabschätzung gemäß Ungleichung (3.17)

[1] Gerade Textdaten haben häufig die Eigenschaft, daß ihre Klassifikation in verschiedene Kategorien bereits linear Trennbar ist.

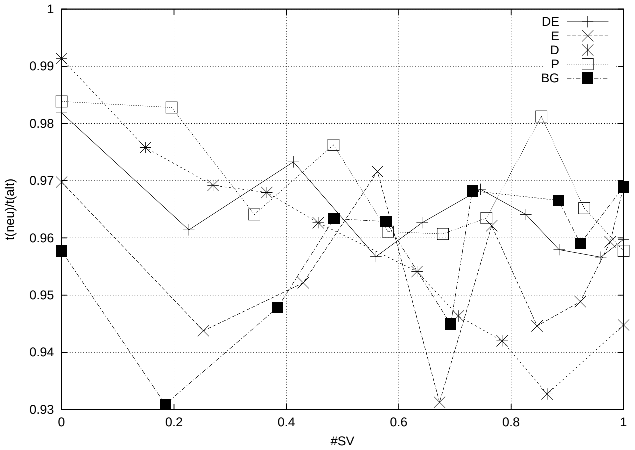

Abb. 4.5: *Beschleunigungsrate* $\frac{t_{neu}}{t_{alt}}$ *der Meßergebnisse des auf der Restsummenabschätzung basierenden Verfahrens gegen die nach gemäß Gl. (4.1) normierten Anzahl der SVen* N_S *aufgetragen.*

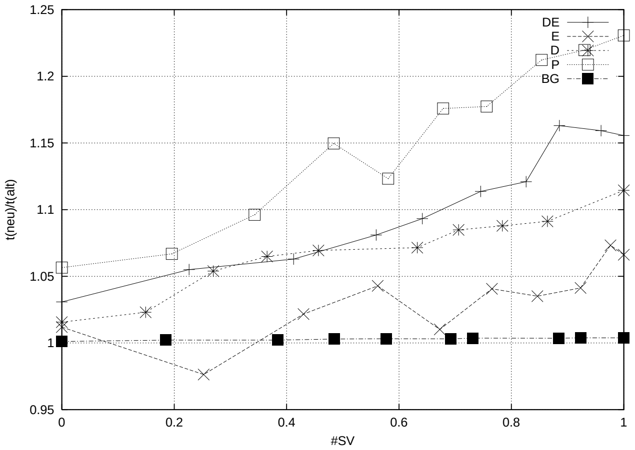

Abb. 4.6: *Beschleunigungsrate des auf der Distanzschätzung basierenden Verfahrens gegen die gemäß Gl. (4.1) normierte Anzahl der SVen* N_S *aufgetragen.*

in jedem Summationsschritt verlangt mehr Zeit als durch die vermiedenen Kernauswertungen eingespart werden kann.

In Abbildung 4.7 ist die relative Laufzeit des Orthonormalisierungsverfahrens gegen die normierte Achse der Anzahl der Support Vektoren aufgetragen.

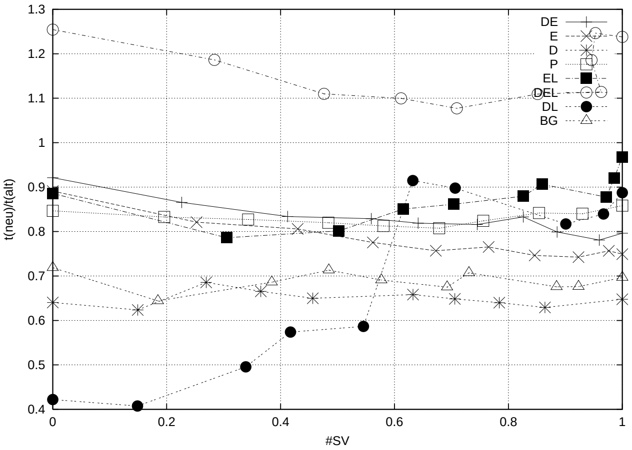

Abb. 4.7: *Beschleunigungsrate der Orthonormalisierung gegen die Anzahl der SVen N_S aufgetragen. Die letzten drei Meßreihen entstanden durch die Verwendung einer linearen Kernfunktion (Abk. L am Ende, z.B. DL=Deutsch Linear).*

Fast alle Werte des Graphen 4.7 liegen unter eins, d.h. das Orthonormalisierungsverfahren liefert in so gut wie allen Fällen eine Beschleunigung. Die Einsparungen sind mit bis zu 60% deutlich höher als die der anderen zwei Verfahren. Der in Sektion 3.3.9 diskutierte negative Einfluß der Anzahl der Support Vektoren auf das Laufzeitverhalten kann nicht festgestellt werden. Das Verfahren spart in den meisten Fällen genügend Kernauswertungen ein, um den wachsenden Mehraufwand zu kompensieren. Bis auf einige Ausreißer zeigen die Kurven einen tendenziell konstanten Verlauf. Eine Veränderung der Anzahl der Support Vektoren bei ansonsten gleichbleibenden Bedingungen wie die Wahl der Kernfunktion schränkt den Zeitgewinn des Orthonormalisierungsverfahrens nicht ein.

Einfluß der Dimension N

Für das Orthonormalisierungsverfahren ist in Gleichung (3.68) der Zusammenhang zwischen der zu erwartenden Beschleunigung und den Größen der SV Anzahl N_S sowie der Dimension N beschrieben. Die Dimension hat bei zunehmender Größe einen positiven Einfluß auf die zu erwartende Beschleunigung. Im letzten Paragraphen ist gezeigt, daß der theoretisch zu erwartende negative Einfluß der Support Vektor Anzahl nicht aufgetreten ist. Das Zusammenspiel der beiden Werte kann über ihrem Quotienten $\frac{N}{N_S}$ beobachtet werden. Trotz des großen Wertebereiches der beiden Größen kann ihr Quotient in einem überschaubaren Bereich dargestellt werden.

Abbildung 4.8 zeigt die relative Beschleunigung $\frac{t_{neu}}{t_{alt}}$ des auf der Restsummenabschätzung nach Abschnitt 3.1 basierenden Verfahrens gegen den Quotienten $\frac{N}{N_S}$ aufgetragen.

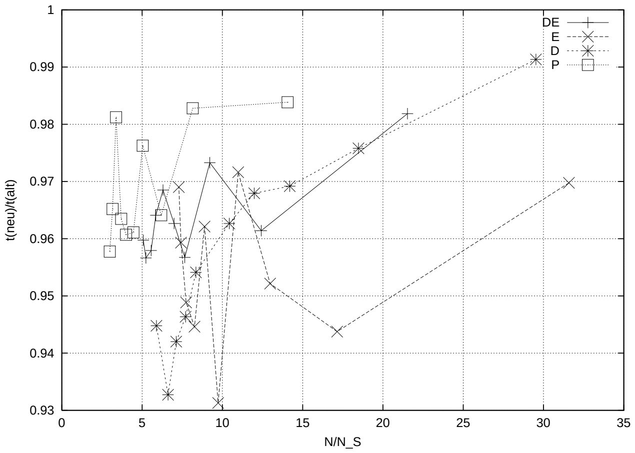

Abb. 4.8: *Die relative Beschleunigung $\frac{t_{neu}}{t_{alt}}$ des auf der Restsummenabschätzung basierenden Verfahrens gegen den Quotienten $\frac{N}{N_S}$ aufgetragen.*

Die Kurven des Graphen 4.8 zeigen keinen eindeutigen Verlauf, die Werte alternieren bei einer Variation des Quotienten $\frac{N}{N_S}$.

Die Abbildung 4.9 stellt den Effizienzgewinn des aus der Distanzabschätzung resultierenden Verfahrens gegen den Wert von $\frac{N}{N_S}$ dar.

Man kann einen eindeutigen Verlauf aller Kurven im Graphen 4.9 ablesen. Das Verfahren zur Ausnutzung der Distanzabschätzung nach Abschnitt 3.2 verbessert sein Laufzeitverhalten eindeutig bei wachsender Dimension und weniger Support Vektoren. Die Einsparung einer Kernauswertung erzielt bei hoher Dimension N einen größeren Geschwindigkeitsvorteil. Außerdem verringert eine niedrigere Anzahl von Support Vektoren N_S auch die Anzahl der zusätzlichen Vergleiche und somit den Mehraufwand des Verfahrens. Die Beschleunigungsrate fällt jedoch nicht unter eins und das Verfahren erzielt keinen Effizienzgewinn.

Der Verlauf der relativen Beschleunigung des Orthonormalisierungsverfahrens gemäß Kapitel 3.3 ist in Abbildung 4.10 gegen den Quotienten $\frac{N}{N_S}$

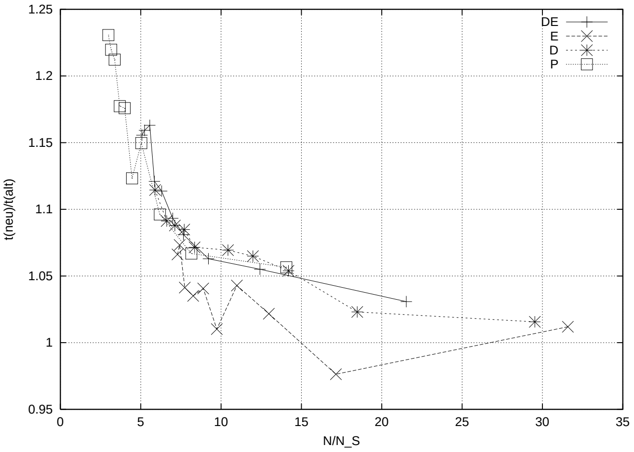

***Abb. 4.9:** Die relative Beschleunigung $\frac{t_{neu}}{t_{alt}}$ des auf der Distanzabschätzung basierenden Verfahrens gegen den Quotienten $\frac{N}{N_S}$ aufgetragen.*

aufgetragen.

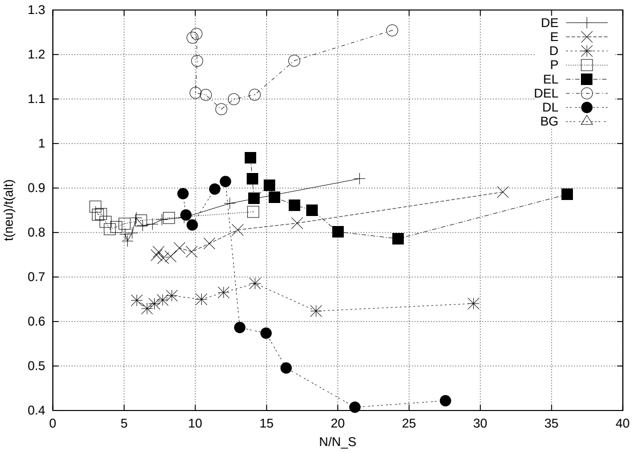

***Abb. 4.10:** Die relative Beschleunigung $\frac{t_{neu}}{t_{alt}}$ des Orthonormalisierungsverfahrens gegen den Quotienten $\frac{N}{N_S}$ aufgetragen.*

Es läßt sich auch hier eine prinzipiell gleiche Form aller im Graphen 4.10 aufgetragenen Kurven erkennen. Nach einem recht hohen Wert im niedrigen Bereich von $\frac{N}{N_S}$ durchläuft die Meßreihe ein Minmum, um dann nach rechts wieder leicht anzusteigen. Die Meßreihen E und D des Englisch- bzw.

Deutsch-Klassifikators mit exponentieller Kernfunktion beinhalten vermutlich nicht genügend Meßwerte im niedrigen Wertebereich von $\frac{N}{N_S}$ um den Anstieg nach links hin darstellen zu können. Dieser Verlauf kommt wie folgt zustande. Zunächst zeigt sich der Einfluß der wachsenden Dimension sehr stark, so daß der Kurvenverlauf absinkt und zu einer guten Beschleunigungsrate führt. Die prinzipiell positive Abnahme der Anzahl der Support Vektoren nach rechts hin verursacht dann aber doch eine Verschlechterung des Zeitaufwandes, weil bei zu wenigen SVen die Wahrscheinlichkeit sinkt, daß überhaupt Kernauswertungen eingespart werden können. In einem hochdimensionalen Raum F wird mit wenigen Support Vektoren nur noch ein kleiner Unterraum F_S dargestellt (vgl. Kap. 3.3), so daß die Wahrscheinlichkeit sinkt, daß ein neuer Eingabevektor $\Phi(\mathbf{x})$ in diesem Unterraum dargestellt werden kann. Ein großer Anteil von $\Phi(\mathbf{x})$ im Unterraum F_S ist jedoch die Voraussetzung für einen frühen Abbruch der Summation gemäß Abschnitt 3.3.7.

Einfluß der geschätzen VC-Dimension

Analog zu Kapitel 4.2.3 sind hier in den Graphen 4.11, 4.12 und 4.13 die erzielten Beschleunigungsraten $\frac{t_{neu}}{t_{alt}}$ gegen die nach Gleichung (2.21) aus Kapitel 2.2.4 geschätzte VC-Dimension VC des Klassifikators aufgetragen. Die stark zwischen den Klassifikatoren verschiedener Datensätze differierenden Werte sind nach demselben Prinzip wie in der Umrechnung (4.1) auf das in der jeweiligen Meßreihe Maximum VC_{max} und Minimum VC_{min} gemäß

$$VC - DIM = \frac{VC - VC_{min}}{VC_{max} - VC_{min}} \tag{4.2}$$

normiert, so daß eine Skala im Intervall [0..1] abgedeckt wird.

Sowohl das Verfahren der Ausnutzung der Restsummenabschätzung als auch das auf der Distanzschätzung basierende Verfahren läßt keinen analytischen Zusammenhang zwischen der relativen Beschleunigung und der geschätzten VC-Dimension erkennen.

In Abbildung 4.13 ist der Verlauf der relativen Geschwindigkeit des Orthonormalisierungsverfahrens gegen die normierte Schätzung der VC-Dimension aufgetragen. Die Änderung der VC-Dimension im Verlauf einer Meßreihe sind auf die Variation der *Dicke* der Klassifikationsebene laut Kapitel 2.2.4 zurückzuführen. Das Orthonormalisierungsverfahren hat ein generell konstantes Zeitverhalten gegenüber leichten Änderungen der VC-Dimension eines

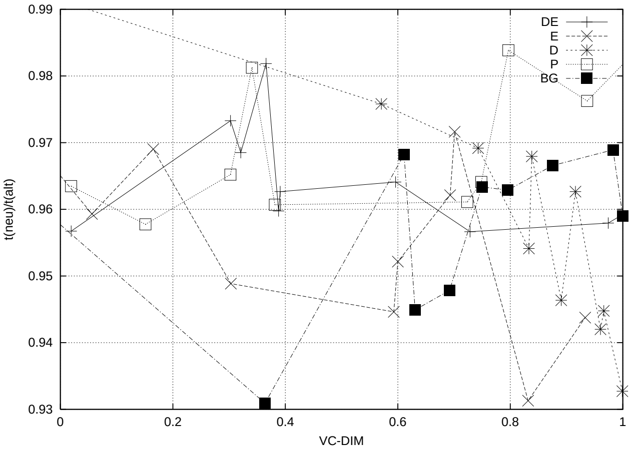

Abb. 4.11: *Beschleunigungsrate $\frac{q_{neu}}{q_{alt}}$ des auf der Restsummenabschätzung gemäß Abschnitt 3.1 basierenden Verfahrens gegen die normierte Schätzung der VC-Dimension nach Gleichung (2.21) aufgetragen.*

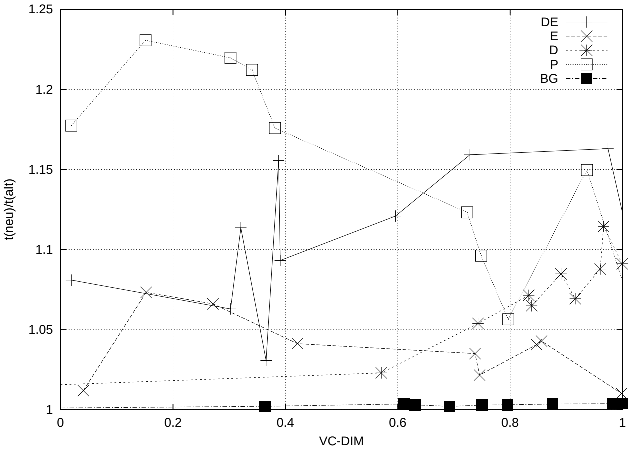

Abb. 4.12: *Beschleunigungsrate $\frac{q_{neu}}{q_{alt}}$ des auf der Distanzabschätzung gemäß 3.2 basierenden Verfahrens gegen die normierte Schätzung der VC-Dimension nach Gleichung (2.21) aufgetragen.*

Klassifikators. Erst ein dramatischer Eingriff in die äußeren Bedingungen des Verfahrens wie durch die Wahl einer anderen Kernfunktion bewirkt eine starke Änderung des Effizienzgewinns. So liegt die gesamte Meßreihe des Deutsch-Englisch Klassifikators mit einem exponentiellen Kern (DE) unterhalb eins und somit im effizienzsteigernden Bereich. Durch die Wahl einer linearen Kernfunktion verschlechtert sich das Zeitverhalten, so daß die gesamte Meßreihe (DEL) für diese Wahl keine Zeiteinsparung mehr zuläßt.

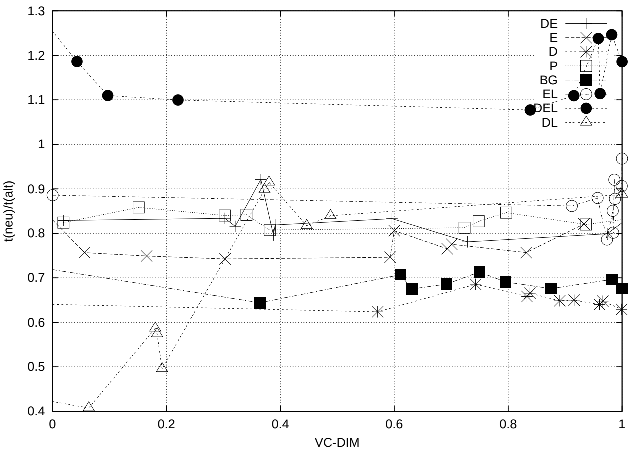

Abb. 4.13: *Beschleunigungsrate der Orthonormalisierung gegen die geschätzte VC-Dimension aufgetragen.*

4.3 Zusammenfassung der Ergebnisse

Das Verfahren der Distanzschätzung hat sich als ungeeignet herausgestellt.

Die Restsummenabschätzung bietet eine dem Programmieraufwand angemessene Effizienzverbesserung im einstelligen Prozentbereich. Bei einer hochdimensionalen Problemstellung kann dies bereits einen erheblicheen Geschwindigkeitsvorteil ausmachen.

Das Orthonormalisierungsverfahren hat die überzeugendsten Ergebnisse geliefert. Bis auf eine Ausnahme resultiert seine Anwendung in einer erheblichen Effizienzsteigerung des Klassifizierers.

Über den Quotienten aus der Dimension des Problems N und der Anzahl der Support Vektoren N_S ist eine Prognose der Effizienz des Orthonormalisierungsverfahrens möglich. Letztendlich wird der Beschleunigungsfaktor jedoch von der dem Datensatz zugrundeliegenden Wahrscheinlichkeitsverteilung sowie der gewählten Kernfunktion bestimmt. Diese beiden Faktoren bestimmen die Geometrie des im Merkmalsraum F aufgespannten Unterraumes F_S, dessen Repräsentativität für den gesamten Datensatz für eine Laufzeitverbesserung von entscheidender Bedeutung ist (vgl. Abschnitt 3.3.7). Idealerweise muß aus den SVen ein Unterraum aufgespannt werden, in dem die Gesamtheit des Datensatzes repräsentiert wird. In diesem Fall kann eine wie am Schluß von Kapitel 3.3.7 diskutierte Situation, daß kein vorzeitiger Summationsabbruch erfolgen kann, nicht eintreten und es werden in der Regel nur $q < N_S$ Kernauswertungen benötigt.

5. FAZIT UND AUSBLICK

Das Verfahren zur Orthonormalisierung der Menge der Support Vektoren führt als einziges der drei untersuchten Algorithmen zu einer erheblich effizienteren Berechnung der Anwendung von Kernmethoden.
Das Orthonormalisierungsverfahren läßt sich auf die Kernmethoden der Support Vektor Maschinen nach Kapitel 2.3.1 sowie der Kernel-PCA nach Kapitel 2.3.3 anwenden. Dabei dürfen beliebige Kernfunktionen gemäß Kapitel 2.2.6 benutzt werden. Das Verfahren ist im Lernprozeß *parameterfrei* und läßt sich in der Anwendungsphase leicht über die direkte prozentuale Fehlerkonstante p wie in Sektion 3.3.7 beschrieben adaptiv regeln. Es kann mit vielen bereits bekannten Verfahren zur Beschleunigung der Anwendungsphase wie dem Ansatz der 'Reduced Set Methode' kombiniert werden. Weil die Support Vektoren nicht manipuliert werden müssen, ist der Einsatz auch dann möglich, wenn andere Verfahren auf Grund eines nicht zu realisierenden Zugriffs auf die einzelnen Dimensionen der Trainingselemente nicht angewendet werden können. Somit wird die Verarbeitung von nichtvektoriellen Elementen wie Tensoren, Texten oder Graphen möglich.

Die Herleitungen in dieser Arbeit basieren zum größten Teil auf gut bekannten Grundlagen der analytischen Geometrie und schöpft diese sicherlich noch nicht gänzlich aus. Eine weitere Arbeit in dieser Richtung ist somit sinnvoll und vielversprechend.

Zum Schluß sollen noch folgende Denkanstöße zur Weiterarbeit in diesem Feld animieren.

- In dieser Arbeit sind lediglich Experimente mit einer Support Vektor Maschine zur Klassifikation durchgeführt. Das Verfahren funktioniert theoretisch jedoch mit allen bisher bekannten Kernmethoden, die eine Anwendungsvorschrift gemäß Gleichung (1.1) besitzen.

- Es wurde keine konkrete Größe gefunden, die das Geschwindigkeitsverhalten des Verfahrens für einen bestimmten Klassifikator prognostizieren kann. Der Quotient $\frac{t_{neu}}{t_{alt}}$ aus Kapitel 4.2.3 gibt lediglich einen prinzipiellen Verlauf an.

- Es ist denkbar, daß das Orthonormalisierungsverfahren in der sequentiellen Ausführung mit anderen Beschleunigungsverfahren noch einen weiteren Geschwindigkeitsvorteil erzielen kann. Möglich ist jedoch auch, daß von der optimierten Support Vektor Menge keine Kernauswertungen mehr eingespart werden können.

- In der Untersuchung der Meßergebnisse in Kapitel 4 ist der starke Einfluß der Geometrie der transformierten Support Vektoren auf das Laufzeitverhalten des Orthonormalisierungsverfahrens aufgefallen. Der aufgespannte Unterraum F_S muß möglichst repräsentativ für den gesamten Datensatz sein. Die Kernmethode der Relevance Vektor Maschine konzentriert sich in ihrem Lernverfahren speziell auf das Finden von repräsentativen Support Vektoren und verspricht somit eine gute Grundlage zur Anwendung des Verfahrens zu bieten.

- Die Fehlerbehandlung des Algorithmus ist lediglich durch die Festsetzung einer maximalen relativen Fehlerschranke erfolgt. Interessant wäre darüber hinaus eine neue Formulierung der Abbruchbedingung in Termen des *PAC-Lernens* (Probably Approximately Correct).

ANHANG

A. IMPLEMENTIERUNGEN

Diese Arbeit beschäftigt sich mit der Effizienzsteigerung von Support Vektor Anwendungen. Die Geschwindigkeit eines Verfahrens ist hauptsächlich von der Art der gewählten Implementierung abhängig. Um einen direkten Einblick in die neuen Methoden zu gewährleisten und eine Weiterarbeit mit ihnen zu erleichtern sind die wesentlichen Programmteile des Quallcodes in die Arbeit intergriert.
In diesem Kapitel werden die Implementierungen der drei in Kapitel 3 erläuterten Verfahren in C++ vorgestellt. Zu jedem Verfahren gibt es eine Klasse, in der die Routinen zur Offline-Berechnung und die Berechnung einer Anwendung enthalten sind. Die entprechenden Funktionsnamen sind 'offline' sowie 'score'. Um das Verständnis zu erleichtern ist an einigen Stellen der Quellcode so abgeändert, daß der eigentliche Algorithmus besser hervortritt. Jedes der Verfahren geht davon aus, daß bereits ein gelernter Klassifizierer vorliegt. Die Trainingsvektoren sind in `_data` gespeichert und die benötigten SVen können über das Indexfeld `_snum` angesprochen werden. Die zu den SVen zugehörigen Klassenlabels und Lagrange-Multiplikatoren sind in den Feldern `label` bzw. `_alpha` vorliegend.

A.1 Abschätzen der Restsumme

```
#define prec 1.e-8

class CSVMAlternateOrderClassifier :
 public CSVMSingleFastClassifier
{
public :
CSVMAlternateOrderClassifier();
bool sortAlpha();
// Alphas nach Label und Groesse sortieren
bool getAlphaSum();
// Summe ueber sortierte alphas berechnen
bool multiplyAlpha();
// alphas mit ihren labels multiplizieren
bool offline();
TSVMFloat score(const TSVMDataClass& D);

private :
vector<TSVMFloat> _PosAlphaSum, _NegAlphaSum;
// Summe der noch nicht behandelten Alphas
// zur Fehlerabschaetzung
vector<TSVMFloat> _alpha; // Lagrange-Multipliers
vector<int>_snum; // Index des SV in _data
unsigned int _csv; // Anzahl der SV
};



// Die zu den SV gehoerenden alphas werden nach der
// Groesse sortiert.
// Parallel werden auch die Eintraege von snum
// neu sortiert.

bool CSVMAlternateOrderClassifier::sortAlpha()
{
unsigned int i;
sort_vector<TSVMFloat, int> Alpha;
```

```
// _alpha in Alpha kopieren um sortieren zu koennen
for (i=0; i<_csv; i++)
Alpha.insert(Alpha.end(), _alpha[i]);

Alpha.parallel_sort(_snum);

// und zurueck nach _alpha

for (i=0; i<_csv; i++)
_alpha[i] = Alpha[i];

return true;
};


bool CSVMAlternateOrderClassifier::getAlphaSum()
{
_PosAlphaSum.resize(_csv);
_NegAlphaSum.resize(_csv);
int i = 0;
TSVMFloat PosSum = 0.0;
TSVMFloat NegSum = 0.0;
for (i=_csv-1; i>=0; i--)
{
_PosAlphaSum.insert(_PosAlphaSum.begin(),PosSum);
// Summe der Alphas mit label 1
_NegAlphaSum.insert(_NegAlphaSum.begin(), NegSum);
// Summe der Alphas mit label -1

if (_classifier->label(_snum[i]) == 1)
{
PosSum -= _alpha[i];
// Negativ gezaehlt, um '-' im Vergleich online zu sparen
}
else
{
NegSum += _alpha[i];
```

```
}
}
return true;
};


// Die alphas werden mit den Klassenlabels multipliziert,
// da das Verfahren nur auf den
// alpha's arbeitet
bool CSVMAlternateOrderClassifier::multiplyAlpha()
{
unsigned int i;
for (i=0; i<_alpha.size(); i++)
_alpha[i] *= _classifier->label(_snum[i]);
return true;
}


bool CSVMAlternateOrderClassifier::offline()
{
unsigned int i;
if (!sortAlpha()) return -1;
// sortieren
if (!getAlphaSum()) return -1;
// Summen berechnen
if (!multiplyAlpha()) return -1;
// und alphas mit ihren labels multiplizieren
return (true);
};


// Berechnung der Anwendung
TSVMFloat
CSVMAlternateOrderClassifier::score( const TSVMDataClass& D)
{
TSVMFloat result = 0;
result -=_classifier->b;
bool fixed_class = false;
```

```
unsigned int i;

for(i=0;(i<_csv && !fixed_class);i++)
{
result +=_alpha[i] * k(_data(_snum[i]), D) );
// k auf SV nummer i und D angewendet
if ( (result > _NegAlphaSum[i]) || (result < _PosAlphaSum[i]))
 fixed_class = true;
};

return result;
}
```

A.2 Abschätzen mit Hilfe des Abstandsmaßes

```
#define prec 1.e-8

class CSVMDistanceClassifier : public CSVMSingleFastClassifier
{
public :
CSVMDistanceClassifier();
bool offline();
TSVMFloat score(const TSVMDataClass& D);


private :
void _init();
// initialisierung
bool multiplyAlpha();
// alphas mit ihren labels multiplizieren
bool getDistanceMatrix();
// Distanzen zwischen den SV ermitteln
bool DistanceSort();
// alpha und snum nach Distanzsumme sortieren
bool getMaxIndex();
// Matrix maxIndex berechnen

vector<TSVMFloat> _alpha;
// Lagrange-Multipliers
vector<int>_snum;
// Nummer des SV in _data
unsigned int _csv;
// Anzahl der SV
CLaUpTriMatrix<TSVMFloat> _dist;
// Distanzen zwischen den SVen
CLaMatrix<int> _maxIndex;
// Index der SV nach Abstand zu SV i geordnet
CLaVector<bool> _skip;
// Flag, ob ein SV uebersprungen werden soll
TSVMFloat _malSigma;
// ab wie viel mal Sigma Abstand soll ein
```

```
// SV ignoriert werden ? (z.B. 4..6)
// epsilon = _sigma * _malSigma
TSVMFloat _sigma;
// Parameter aus RBFKernel uebernehmen

};

void
CSVMDistanceClassifier::_init()
{
_dist.Resize(0);
_malSigma = 4.0;
};


CSVMDistanceClassifier::CSVMDistanceClassifier()
{
_init();
}


// Nach Berechnung der Distanzmatrix _dist fuer
// die neue Reihenfolge wird hier die
// Auswertungsreihenfolge festgelegt
// (nach maximalem Abstand)

bool CSVMDistanceClassifier::getMaxIndex()
{
unsigned int i=0, j=0;
_maxIndex.Resize(_csv,_csv);
vector<int> index;
sort_vector<TSVMFloat, int> alphaZeile;
for (i=0; i<_csv; i++)
{
alphaZeile.push_back(0.0);
index.push_back(0);
};
```

```
for (i=0; i<_csv; i++)
// Fuer jeden SV
{
for (j=0; j<_csv; j++)
// index und alphaZeile initialisieren
{
alphaZeile[j] =  _dist(i,j);
index[j] = j;
};
// und sortieren
// cout << " Sort " << i << endl;
alphaZeile.parallel_sort(index);
for (j=0; j<_csv; j++)
// und die neue Reihenfolge merken
{
_maxIndex(i,j) = index[j];
}
};
cout << _maxIndex << endl;
return true;
};
```

```
// Fuellt _dist mit den Distanzen zwischen den SVen

bool CSVMDistanceClassifier::getDistanceMatrix()
{

unsigned int i,j;
TSVMFloat distance;
_dist.Resize(_csv);
for (i=0; i<_csv; i++)
{
for (j=i; j<_csv; j++)
{
```

```
distance = _data(_snum[i]).DistanceTo(_data(_snum[j]))
_dist(i,j) = distance;
}
};

return true;

};


// snum und alpha werden abhaengig von der groessten
// Summe der Distanzen in _dist
// sortiert

bool CSVMDistanceClassifier::DistanceSort()
{
sort_vector<TSVMFloat, int> distSum;
TSVMFloat Sum = 0;
vector<int> index;
vector<TSVMFloat> alphaCopy;
vector<int> snumCopy;
unsigned int i = 0,j = 0;

// index initialisieren, Distanz-Summen Ausrechnen
for (i=0; i<_csv; i++)
{
index.insert(index.begin(),i);
Sum = 0;
for (j=0; j<_csv; j++)
{
Sum += _dist(i,j);
};
distSum.insert(distSum.begin(), Sum);
};

// index parallel zu distSum sortieren
distSum.parallel_sort(index);
```

```
// _snum und alpha kopieren
snumCopy = _snum;
alphaCopy = _alpha;

// _snum und _alpha nach index umsortieren

for (i=0; i<_csv; i++)
{
_snum[i] = snumCopy[index[i]];
_alpha[i] = alphaCopy[index[i]];
};
return true;
};


bool CSVMDistanceClassifier::multiplyAlpha()
{
unsigned int i;
for (i=0; i<_alpha.size(); i++)
_alpha[i] *= _classifier->label(_snum[i]);
return true;
}

bool CSVMDistanceClassifier::offline()
{
_sigma =_kernel->_sigma;   //Nur fuer RBF-Kernel
int i;
_skip.Resize(_csv);
if (!getDistanceMatrix()) return -1;
if (!DistanceSort()) return -1;

// Anstatt geschickter Vertauschungen in
// DistanceSort hier einfach
// fuer die neue Reihenfolge nochmal berechnen
// -ist ja offline ;o)
if (!getDistanceMatrix()) return -1;
if (!getMaxIndex()) return -1;
if (!multiplyAlpha()) return -1;
```

```
return (true);
};


TSVMFloat
CSVMDistanceClassifier::score( const TSVMDataClass& D)
{
TSVMFloat distance = 0;
TSVMFloat result = 0;
result -=_b;
unsigned int i,j,next;
bool skipThis = false;
bool stop = false;
_skip = false;

for(i=0 ; !stop ; )
{
distance = _data(_snum[i]).DistanceTo(D);
result +=_alpha[i] * k(distance);
// Kernelfunktion k auf bereits
// berechnete Distanz anwenden
n_dist ++;
skipThis = true;
_skip[i] = true;

next = i;
for (j=0; ((skipThis) && (j<_csv-1)); j++)
{
if (!_skip[_maxIndex(i,j)])
{
if ( _dist(i,_maxIndex(i,j)) - distance >= _malSigma * _sigma )
{
_skip[_maxIndex(i,j)] = true;
}
else
{
skipThis = false;
next = _maxIndex(i,j);
```

```
};
}
};
stop = (next == i);
i = next;

};

return result;
}
```

A.3 Orthonormalisierungsverfahren

Die Namensgebung im Programm bezieht sich wie folgt auf die in der Herleitung benutzten Symbole:

K	K
$\tilde{K}$	`_Lambda`
ϱ	`_rho`
α	`_alpha`
$\tilde{\alpha}$	`_lda`
$\lvert\mathbf{\Psi}_q\rvert^2$	`_psilength`
$\lvert\Phi(\mathbf{x})_q\rvert^2$	`dX`
Ω	`_Omega`
$\{1..N_S\}$	`_snum`
$\tilde{i}$	`_dataOrder`

```
#define prec 1.e-8

class CSVMOrthogonalClassifier :
 public CSVMSingleFastClassifier
{
public :
virtual bool offline();
virtual TSVMFloat score( const TSVMDataClass& D);
inline unsigned int csv(void){ return _order_size; };

private :
CLaSimpleMatrix<TSVMFloat> _Lambda;
// (i,j) : Produkt neuer Basis
// vektor i * alter j
vector<TSVMFloat> _rho;
// Laenge der neuen Basisvektoren
vector<TSVMFloat> _lda;
// neue Darstellung des Gewichtsvektors
vector<TSVMFloat> _psilength;
  // Laenge des noch nicht behandelten
// Gewichtsvektors in Summationsschritt k
unsigned int _order_size;
// Groesse von _dataOrder
```

```
vector<int> _dataOrder;
// neue Reihenfolge in der Bewertung
// die tatsaechliche Position des SV in _data
vector<TSVMFloat> _Omega;
// Vektor zur online-Berechnung
// Phi(x) * Neuem SV
int _csv;
// Anzahl der Support Vektoren
vector<int> snum;
// Index von SV i in _data
vector<TSVMFloat> alpha;
// Lagrange-Multiplier der SVen
};


bool CSVMOrthogonalClassifier::offline()
{
int i,j,k,l;
vector<int> order;
// neue Reihenfolge in der Bewertung
// abweichend von snum, gibt den Eintrag in snum an


vector<bool> used;
// used : flag fuer jeden alten SV, ob er in der
// neuen Darstellung bereits untergebracht ist

for (i=0; i<_csv; i++) used.push_back(false);


CLaSimpleMatrix<TSVMFloat> Lambda_single(_csv, _csv);
// Lambda_single speichert Lambda[i,j] fuer ein
// festes i fuer alle uebrigen SVen in einem Durchlauf

Lambda_single = static_cast<TSVMFloat> (0);


CLaSimpleMatrix<TSVMFloat> Lambda(_csv, _csv);
```

```
// Lambda speichert die spaeteren Werte fuer _Lambda,
// jedoch sind in der Matrix noch spaeter irrelevante Eintraege

Lambda = static_cast<TSVMFloat> (0);

// rho speichert die verschiedenen rhos in einem Durchlauf

vector<TSVMFloat> rho;
// rho speichert die verschiedenen rhos in einem Durchlauf

for (i=0; i<_csv; i++) rho.push_back(0.0);

// lda speichert die verschiedenen lda in einem Durchlauf

vector<TSVMFloat> lda;
for (i=0; i<_csv; i++) lda.push_back(0.0);

for (l=0; l<_csv; l++)
// Uebernimm pro Schleifendurchlauf mind. einen Vektor in
// die neue Struktur
{
for(i=0; i<_classifier->csv; i++)
{

if (!used[i])
// Fuer jeden noch uebrigen Vektor werden die noetigen
// Werte von rho, lda und Lambda berechnet
// um dann den besten folgenden zu finden
{
rho[i] = K(snum[i], snum[i]);  // Matrix K auslesen

// berechne rho = laenge des neuen Stellvertreters
// rho = K(i,i)
for (k=0; k<(order.size()); k++)
// alle bisher uebernommenen Anteile von SV i abziehen
{
rho[i] -= sqr(Lambda(order[k],i));
};
```

```
if (rho[i]<prec)
// Der Vektor hat keinen Beitrag in der neuen Darstellung
{
rho[i] = 0;
used[i] = true;
}
else
{
rho[i] = sqrt(rho[i]);
// Wurzel aus rho ist der eigentlich gesuchte Wert


// Jetzt die zugehoerigen Werte fuer Lambda berechnen,
// zuerst aber in Lambda_single zwischenspeichern
// Lambda(i,j) = Neuer Vektor i * alter Vektor j

for (j=0;j<_csv;j++)
{
TSVMFloat result = 0.0;
// Lambda(i,j) = L(i,j)
result = K(snum[i], snum[j]);
for (k=0; k< (order.size()); k++)
// und die bisher uebernommenen Anteile abziehen
{
result -= Lambda(order[k],j) * Lambda(order[k],i);
};
result /= rho[i];
if (fabs(result) > prec)
Lambda_single(i,j) = result;
else
Lambda_single(i,j) = 0.0;
};
};
// Neues lda fuer die verschiedenen Kandidaten berechnen

lda[i] = 0.0;
for (j=0; j<_csv; j++)
```

```
{
lda[i] += label(snum[j]) * alpha[snum[j]]
* Lambda_single(i,j);
};
}; // ende if !used[i]
}; // ende for i

int max_index = -1;
// suche das maximale Produkt aus rho und lda zwischen
// den uebrigen Kandidaten aus den gerade berechneten Werten
TSVMFloat max_lda = 0.0;
for (i=0; i<_csv; i++)
{
if (!used[i])
{
if (fabs(lda[i]) > max_lda)
{
max_index = i;
max_lda = fabs(lda[i]);
};
};
};
if (max_index != -1)
{
// Die Daten fuer den SV merken,
// der den groessten lda Wert ergiebt, d.h. mit
// dem groessten Anteil an PSI
order.insert(order.end(),max_index);
_dataOrder.insert(_dataOrder.end(),snum[max_index]);
_lda.insert(_lda.end(),lda[max_index]);
_rho.insert(_rho.end(),rho[max_index]);
_Omega.insert(_Omega.end(),0.0);
for(j=0; j<_csv; j++)
{
Lambda(max_index, j) = Lambda_single(max_index,j);
};
used[max_index] = true;
}
```

```
else
{
// cout << " Keine sinnvollen Vektoren mehr da " << endl;
};


}; // ende for l



// Hier ist die Orthonormalisierung abgeschlossen

// Berechnung von _psilength als Quadratsumme der neuen lambdas

for (i=0; i<(_lda.size());i++)
{
_psilength.push_back(0.0);
};
TSVMFloat sqrsum = 0.0;
for (i=(_lda.size()-2); i>=0; i--)
{
sqrsum += sqr(_lda[i+1]);
if (sqrsum > prec)
{
_psilength[i] = (sqrsum);
}
else
_psilength[i] = 0.0;
};

// Den relevanten Inhalt von Lambda nach _Lambda kopieren

_Lambda.Resize(order.size(), order.size());
for (i=0; i < (order.size()); i++)
{
for (j=0; j < (order.size()); j++)
{
_Lambda(i,j) = Lambda(order[i], order[j]);
```

```
};
};

_order_size = order.size();

return true;
};



TSVMFloat
CSVMOrthogonalClassifier::score(const TSVMDataClass& D)
{
TSVMFloat dX = k(D, D);
//dX quadrat : So viel wurde von x noch nicht dargestellt
TSVMFloat result;
bool fixed_class = false;
// Das Vorzeichen des Ergebnisses kann
// sich nicht mehr veraendern ?
unsigned int i,j;
TSVMFloat sum = -_classifier->b;
// Summe lda[i] * Omega[i], erst mal b abziehen

for (i=0; (i<_order_size) && (!fixed_class); i++)
{
result =  k(_data(_dataOrder[i]), D);
// result = K(SVi,D)
// alter SV !
for (j=0; j<i; j++)
{
result -= _Lambda(j,i) * _Omega[j];
};
result = result / _rho[i];
// durch rho teilen
sum += _lda[i] * result;
// summe += lambdai * k(D,neuer SVi)
dX -= result * result;
// dX quadrat -= Omega[i] quadrat
```

```
if ( sum*sum > (dX * _psilength[i] ))
{
 fixed_class = true;
// Rest der Summe wird das Vorzeichen nicht mehr
// veraendern, also ist die Klasse festgelegt
}
else
{
_Omega[i] = result;
// altes Omega merken
};

};
return sum;


}
```

B. MEßERGEBNISSE

In einer Tabelle sind jeweils die Ergebnisse einer Meßreihe über die zehn Unterdatensätze eingetragen.
In den Tabellenköpfen werden folgende Abkürzungen benutzt :

t(Original)	Zeitbedarf der Summenbildung nach Gleichung (1.1).
t(Restsumme)	Zeitbedarf des auf der Restsummenabschätzung basierenden Verfahrens gemäß Abschnitt 3.1 in der Anwendungsphase
t(Orthonormal)	Zeitbedarf des Orthonormalisierungsverfahrens in der Anwendungsphase
t(Abstand)	Zeitbedarf des auf der Distanzschätzung basierenden Verfahrens gemäß Abschnitt 3.2 in der Anwendungsphase

Die Zeitangaben sind alle in Sekunden gemessen und beziehen sich jeweils auf die Anwendung für den gesamten Testdatensatz. Weitere Angaben über den verwendeten Datensatz sind dem Kapitel 4 zu entnehmen, speziell der Tabelle in Abbildung 4.1.2.

Anzahl SV	VC-Dim	t(Original)	t(Restsumme)	t(Orthogonal)	t(Abstand)
152	1.34445	27.1575	26.3362	24.2005	27.4822
280	1.46198	52.4343	49.4855	43.0539	51.1943
370	1.42201	63.6839	60.6368	51.3278	65.0639
437	1.43413	74.3728	72.2625	57.663	77.5626
493	1.44979	84.9765	79.1383	64.3134	85.8432
540	1.43317	92.2343	88.7407	70.5873	95.9821
581	1.42113	98.0594	92.6287	73.1905	101.499
620	1.38637	103.697	98.3936	76.9988	107.977
647	1.35675	105.377	101.089	79.7185	113.099
659	1.36983	108.636	105.271	81.3728	115.823

Abb. B.1: *Die Meßergebnisse des Englisch-Datensatzes unter Verwendung einer exponentiellen Kernfunktion.*

Anzahl SV	VC-Dim	t(Original)	t(Orthogonal)
133	17.3319	1.28301	1.13652
198	29.2841	2.55071	2.00516
240	29.4184	2.99479	2.40019
264	29.4091	3.20423	2.72559
283	28.5259	3.42317	2.94933
309	29.0828	3.72739	3.27829
316	29.6016	3.68119	3.33664
340	29.4566	3.7989	3.3334
343	29.4436	3.66471	3.37326
346	29.6082	3.66565	3.54841

Abb. B.2: *Die Meßergebnisse des Englisch-Datensatzes unter Verwendung einer linearen Kernfunktion.*

Anzahl SV	VC-Dim	t(Original)	t(Restsumme)	t(Orthogonal)	t(Abstand)
112	2.22961	3.85433	3.82091	2.46791	3.91469
179	3.76521	6.10781	5.96004	3.80874	6.24849
233	4.22811	7.92035	7.67622	5.43074	8.34743
276	4.48479	9.34217	9.04269	6.21626	9.94858
317	4.69338	10.7721	10.3698	7.001	11.5194
396	4.4708	13.4474	12.8304	8.84989	14.4087
429	4.62539	14.5644	13.7831	9.44314	15.8001
464	4.81338	15.5638	14.6613	9.95871	16.9316
500	4.918	16.8006	15.6702	10.5731	18.3331
561	4.82926	18.7141	17.6806	12.1217	20.8568

Abb. B.3: *Die Meßergebnisse des Deutsch-Datensatzes unter Verwendung einer exponentiellen Kernfunktion.*

Anzahl SV	VC-Dim	t(Original)	t(Orthogonal)
120	9.33544	0.26863	0.11338
156	17.2103	0.349326	0.142328
202	33.1335	0.400583	0.198533
221	32.0654	0.421621	0.24199
252	31.6621	0.480346	0.281804
273	56.3832	0.485234	0.443833
291	55.3963	0.540208	0.484915
338	64.5703	0.658296	0.537798
354	69.7184	0.717046	0.601907
362	133.068	0.726939	0.645024

Abb. B.4: *Die Meßergebnisse des Deutsch-Datensatzes unter Verwendung einer linearen Kernfunktion.*

Anzahl SV	VC-Dim	t(Original)	t(Restsumme)	t(Orthogonal)	t(Abstand)
229	2.38876	38.5993	37.899	35.553	39.7869
397	2.57182	70.7773	68.0456	61.2613	74.6737
535	2.37168	93.7139	91.2109	78.135	99.6137
644	2.29517	112.879	107.994	93.5736	122.02
705	2.39554	120.383	115.888	98.5781	131.608
782	2.3766	135.06	130.807	110.191	150.413
842	2.45094	143.899	138.733	119.87	161.302
886	2.55298	153.921	147.447	123.006	179.013
941	2.48668	158.676	151.796	123.9	183.94
971	2.39473	166.77	160.058	132.795	192.717

Abb. B.5: *Die Meßergebnisse des Deutsch-Englisch-Datensatzes unter Verwendung einer exponentiellen Kernfunktion.*

Anzahl SV	VC-Dim	t(Original)	t(Orthogonal)
207	11.4393	1.52294	1.91035
291	14.0702	2.92405	3.46822
348	17.3825	4.06141	4.50724
388	24.9355	4.68934	5.15637
417	62.8296	5.11315	5.50846
459	67.5222	5.55498	6.16263
492	70.3523	5.6918	6.34065
487	72.7075	5.38409	6.38472
489	71.5952	5.37422	6.69906
503	70.1509	5.56608	6.89074

Abb. B.6: *Die Meßergebnisse des Deutsch-Englisch-Datensatzes unter Verwendung einer linearen Kernfunktion.*

Anzahl SV	VC-Dim	t(Original)	t(Restsumme)	t(Orthogonal)	t(Abstand)
251	1.65118	36.6899	36.0975	31.0622	38.7635
433	1.70295	59.4841	58.461	49.5403	63.4594
570	1.6396	76.2409	73.504	63.0656	83.5747
701	1.68477	90.1864	88.0451	73.9452	103.681
791	1.63362	103.726	99.6912	84.2803	116.513
882	1.55154	114.979	110.459	92.8587	135.2
954	1.46457	124.235	119.698	102.395	146.273
1045	1.54173	134.122	131.605	112.879	162.587
1116	1.53257	144.237	139.217	121.179	175.926
1181	1.4963	150.956	144.578	129.551	185.778

Abb. B.7: *Die Meßergebnisse des Sachgebiets-Datensatzes unter Verwendung einer exponentiellen Kernfunktion.*

Anzahl SV	VC-Dim	t(Original)	t(Restsumme)	t(Orthogonal)	t(Abstand)
30	4.05783	30.5104	29.2202	21.9276	30.5451
54	6.82165	63.0795	58.7217	40.6273	63.2165
80	9.31179	100.591	95.3441	69.0095	100.808
93	9.74825	114.765	110.561	81.8354	115.103
105	10.0968	130.507	125.661	90.1565	130.918
120	8.84717	151.339	143.004	102.151	151.806
125	8.69758	157.532	152.525	111.402	158.106
145	10.7013	181.626	175.552	122.739	182.28
150	11.6494	186.761	179.105	126.264	187.48
160	11.5178	204.333	197.984	142.323	205.118

Abb. B.8: *Die Meßergebnisse des Bilder-Grafiken-Datensatzes unter der Verwendung einer exponentiellen Kernfunktion.*

Literatur

[1] Bradley, P., Mangasarian, O., Massive Data Discrimination via Linear Support Vector Machines. *Mathematical Programming Technical Report 98-05*, University of Wisconsin Madison, 1998.

[2] Burges, C.J.C., Simplified Support Vector Decision Rules. 13th International Conference on Machine Learning 1996, pp. 71-77.

[3] Burges, C.J.C., A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[4] Burges, C.J.C., Schölkopf, B., Improving the Accuracy and Speed of Support Vector Machines. *Advances in Neural Information Processing Systems 9*, pp. 375-381, Camebridge, MA. 1997, MIT Press.

[5] Campbell, C., An Introduction to Kernel Methods. *Radial Basis Function Networks : Design and Applications*, Springer Verlag, Berlin, 2000.

[6] Cortes, C., Vapnik, V., Support Vector Networks. *Machine Learning* 20 (1995), pp. 273-297.

[7] Joachims, T., Making Large-Scale SVM Learning Practical. *Advances in Kernel Methods – Support Vector Learning*, pp. 169-184, Camebridge, MA, 1999. MIT Press.

[8] Joachims, T., Text Categorization with Support Vector Machines : Learning with many Relevant Features. *European Conference on Machine Learning (ECML)*, 1998.

[9] Joachims, T., Transductive Inference for Text Classification using Support Vector Machines. *International Conference on Machine Learning (ICML)*, 1999.

[10] Lodhi, H., Shawe-Taylor, J., Cristianini, N., TextClassification using String Kernels. *NeuroColt2 Technical Report Series NC-TR-2000-079*, University of London, 2000.

[11] Osuna, E., Freund, R., Girosi, F., Support Vector Machines : Training and Applications. Massachusetts Institute of Technology, 1996.

[12] Osuna, E., Girosi, F., Reducing the run-time complexity in Support Vector Machines. *ICPR'98*, Brisbane, Australia, 1998.

[13] Schölkopf, B., Support Vector Learning. R. Oldenbourg Verlag, München, 1997.

[14] Schölkopf, B., Burges, C.J.C., Vapnik, V., Incorporating Invariances in Support Vector Learning Machines. *In Proceedings ICANN'96 – International Conference on Artificial Neural Networks*, Springer Verlag, Berlin, 1996.

[15] Schölkopf, B., Knirsch, P., Smola, A., Burges, C.J.C., Fast Approximation of Support Vector Kernel Expansions, and an Interpretation of Clustering as Approximation in Feature Spaces. *Mustererkennung 1998 – 20. DAGM-Symposium*, Informatik Aktuell, pp. 124-132, Springer Verlag Berlin, 1998.

[16] Schölkopf, B., Simard, P., Smola, A., Vapnik, V., Prior Knowledge in Support Vector Kernels. *Advances in Neural Information Processing Systems*, vol. 10, pp. 640-646, Camebridge, MA, MIT Press 1998.

[17] Schölkopf, B., Smola, A., Williamson, R., Bartlett, P., New Support Vector Algorithms. *Technical Report NC-TR-98-027, NeuroColt2*, University of London, UK, 1998b. submitted to *Neural Computation*.

[18] Smola, A., Learning with Kernels. Technische Universität Berlin, 1998.

[19] Tipping, M.E., The Relevance Vector Machine. *Advances in Neural Information Processing Systems 12*, Camebridge, Mass, MIT Press, 2000.

[20] Vapnik, V., The Support Vector Method. *International Conference on Artificial Neural Networks 1997 (ICANN)* , pp. 263-271.

[21] Vapnik, V., The Nature of Statistical Learning Theory. Springer Verlag, 1995, New York.

[22] Yang, Y., Pedersen, J.O., A Comparative Study on Feature Selection in Text Categorization. *International Conference on Machine Learning (ICML)*, 1997.

An dieser Stelle möchte ich mich gerne bei Herrn Dr. J. Pauli und Herrn C. Werner für die gute und stets sehr unmittelbare Betreuung meiner Arbeit bedanken. Ich wurde während dieser Arbeit freundlicherweise von der Cobion AG, Kassel (www.cobion.com) unterstützt. Dank gilt auch allen meine Freunden - einfach so.

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Kiel, den