Hochschule Neubrandenburg
University of Applied Sciences

Hochschule Neubrandenburg
Fachbereich Landschaftsarchitektur, Geoinformatik,
Geodäsie und Bauingenieurwesen

- Masterarbeit -

im Mastertudiengang

Geodäsie und Geoinformatik

Entwicklung von effizienten Klassifikatoren zur Gesichtserkennung

vorgelegt von: Ulrike Blank

Zum Erlangen des akademischen Grades

Master of Engineering (M.Eng.)

Erstprüfer: Prof. Dr. Gerd Teschke

Zweitprüfer: Dr. Martin Nitschke

Bearbeitungszeitraum: 15.07.2010 - 17.01.2011

urn:nbn:de:gbv:519-thesis2010-0148-4

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Neubrandenburg, den 17. Januar 2011

Danksagung

Ich möchte mich an dieser Stelle bei Herrn Prof. Dr. Gerd Teschke für seine intensive Betreuung bedanken. Besonderer Dank gilt meiner Familie, die mich während des gesamten Zeitraums unterstützt hat.

Kurzfassung

Für die Erkennung von Gesichtern in Bildern wurden Support-Vektor-Maschinen (SVM) bereits mehrfach erfolgreich eingesetzt. SVMn zeichnen sich vor allem durch die gute Generalsierung und Klassifizierung aus. Bei den herkömmlichen Methoden sind die Rechenzeiten für die Klassifizierung sehr hoch. Diese Arbeit beschreibt die Entwicklung von effizienten Klassifikatoren zur Gesichtserkennung. Dazu wurden zwei Verfahren näher betrachtet - die Wavelet - Support-Vektor-Maschine (W-SVM) und die reduzierte Support-Vektor-Maschine (RVM). Bei der W-SVM wird eine Beschleunigung durch die Reduktion der Rechenschritte erzielt. Die Laufzeit während der Klassifzierungsphase hängt von der Anzahl der Support-Vektoren ab. Bei der RVM wird eine Näherung an die SVM bestimmt, die mit viel weniger Vektoren für die Klassifizierung auskommt. Beide Verfahren wurden in MATLAB implementiert und erfolgreich auf einer kleinen vorgegebenen SVM angewandt. Die Berechnungszeit konnte gegenüber der Original-SVM deutlich verkürzt werden bei gleichbleibender Genauigkeit.

Abstract

Support vector machines (SVM) were already used successfully several times for the recognition of faces in images. SVMs are characterized above all by the good generalization and classification. In the conventional methods, the computing time for classification are very high. This master thesis describes the development of efficient classifiers for face detection. These two methods have been examined more closely - the wavelet - support vector machine (W-SVM) and the reduced support vector machine (RVM). In the W-SVM is achieved by accelerating the reduction of the computational steps. The run time during the classification phase depends on the number of support vectors. In the RVM an approximation is determined at the SVM, which works with much less vectors for the classification. Both methods were implemented in MATLAB and applied successfully on a small set SVM. The computation time was reduced significantly compared with the original SVM while maintaining accuracy.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Gegenwärtig ist die automatisierte Erkennung von Gesichtern ein sehr aktuelles Forschungsfeld, auf dem in den letzten Jahren einige Fortschritte erzielt wurden. Im Wesentlichen bestehen die Schwierigkeiten bei der Gesichtserkennung in der Variabilität der Abbildung der Gesichter auf Grund der zahlreichen Möglichkeiten der Darstellung, u.a. durch Größenskala und Blickrichtung. Wichtige Aspekte bei der Klassifikation von Gesichtern sind die Genauigkeit, mit der man klassifiziert, und die Geschwindigkeit. Support-Vektor-Maschinen (SVM) kommen in verschiedenen Bereichen der Mustererkennung und Klassifikation zum Einsatz, so auch bei der Gesichtserkennung. Neben der Gesichtserkennung finden sie u.a. Anwendung in der Medizin [3] (Erkennung von Tumoren), Bioinformatik [5] (Klassifizierung von Proteinen), Klassifikation von Textdokumenten [7] und Schrifterkennung [2].

Die Idee der Support-Vektor-Maschine (SVM) geht auf eine Arbeit von Vapnik und Chervonenkis [13] von 1974 zurück. Der große Durchbruch gelang aber erst Mitte der 1990er Jahre. Seither wurden zahlreiche Weiterentwicklungen und Modifikationen der SVM veröffentlicht. Die Aufgabe einer SVM besteht darin, alle Objekte einer bestimmten Klasse auf einem beliebigen Bild zu erkennen und deren Position auszugeben. Mit Hilfe eines Suchfenster wird jeder Ausschnitt durch die SVM klassifiziert. Das Suchfenster hat in der Regel eine Größe von 20x20 Pixel und wird über das zu untersuchende Bild verschoben. Um Gesichter im Vorder- als auch im Hintergrund zu erkennen, wird das Bild in unterschiedlichen Skalierungsstufen durchlaufen. Da so mehrere zehntausend Ausschnitte pro Bild klassifiziert werden müssen, ist der derzeitig genutzte Algorithmus für eine Echtzeit-Gesichtsdetektion ungeeignet. Auch im Hinblick auf die spätere Nutzung der SVM für Bildsequenzen ist die Performance des Algorithmus nicht ausreichend. Die Auswertung

einer Sequenze dauert zu lange, als dass man Gesichter verfolgen könnte.

Im Bereich der Gesichtserkennung hat sich das Verfahren von Viola und Jones [8] als besonders effizient erwiesen. Es ist gegenwärtig ein international anerkanntes Verfahren, das auf einer leicht modifizerten Variante eines Adaboost Klassifikators basiert. Trotz seiner guten Klassifikationsgüte und Effizienz hat dieses Verfahren auch Nachteile [10], zum Beispiel sind die verwendeten Filter in ihrer Anzahl begrenzt und müssen manuell ausgewählt werden.

Das Ziel dieser Arbeit ist es Klassifikatoren zu entwickeln, mit denen eine beschleunigte Klassifikation möglich ist. Die Klassifikatoren sollen automatisch erstellt werden. Parameter, wie zum Beispiel die verwendete Kernfunktion, sollen unverändert bleiben. Die Güte der Klassifikation sollte aber nicht wesentlich schlechter sein als beim herkömmlichen Algorithmus. Dazu werden im Kapitel 2 zunächst die Gundlagen einer Support-Vektor-Maschine erläutert. Anschließend werden in Kapitel 3 und Kapitel 4 zwei Verfahren zur Beschleunigung beschrieben, implementiert und auf einer zur Verfügung gestellteten SVM getestet. Im Kapitel 5 wird eine Zusammenfassung der Arbeit und ein Ausblick für weitere Schritte gegeben.

Kapitel 2

Support-Vektor-Maschine

In diesem Kapitel werden Grundlagen der binären Klassifikation mit Support-Vektor-Maschinen vorgestellt. Zunächst wird im ersten Abschnitt das Verfahren der linearen Trennung betrachtet. Im folgenden Abschnitt wird das beschriebe Verfahren auf die nicht-lineare Trennung übertragen und der Einsatz einer Kernfunktion erläutert.

2.1 Lineare Trennung

2.1.1 Separabler Fall

Die Grundlage für den Bau einer Support-Vektor-Maschine (SVM) bildet eine Trainingsmenge X (2.1). Jedes Objekt x_i wird durch einen Vektor im Vektorraum repräsentiert und es ist jeweils bekannt, zu welcher Klasse y_i es gehört. Für binäre Klassifikationen gilt:

$$X = \{(x_i, y_i)\}_{i=1,\dots,m},\ x_i \in \mathbb{R}^N,\ y_i \in \{\pm 1\} \tag{2.1}$$

Anhand der Trainingsmenge X bestimmt der SVM-Trainingsalgorithmus eine Hyperebene, die X (bestmöglich) in positive und negative Beispiele trennt (siehe Abb. 2.1). Bei der Berechnung der Hyperebene werden nur die nächstliegenden Vektoren (Support-Vektoren) berücksichtigt. Vektoren, die weiter entfernt von der Ebene liegen, haben keinen Einfluss auf die Lage und Position der Ebene, da sie sozusagen hinter einer Front anderer Vektoren „versteckt“ sind. Eine Hyperebene ist definiert durch:

$$H = H(\omega, b) = \{x | \langle \omega, x \rangle + b = 0\}$$

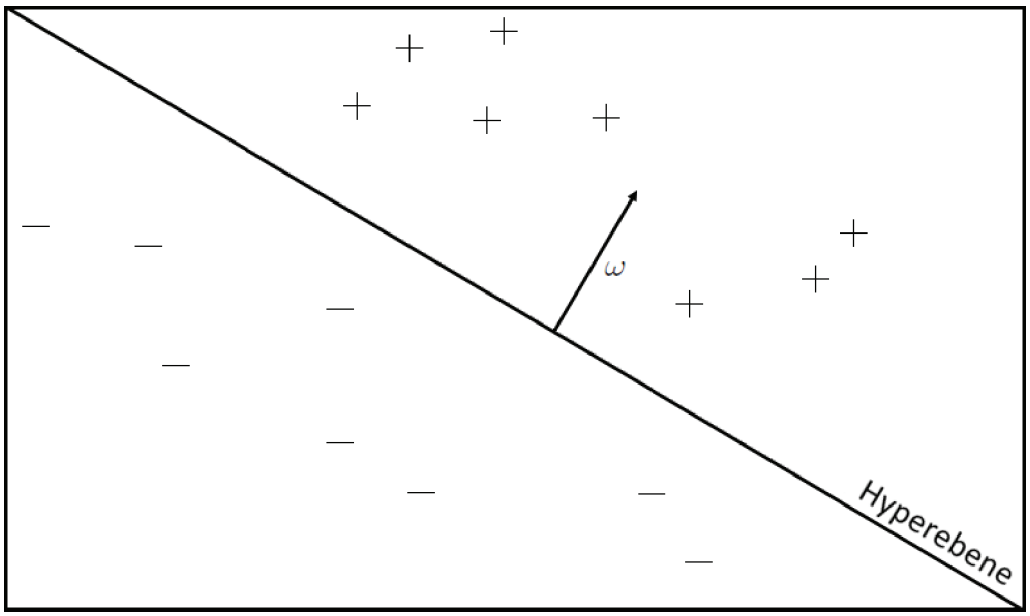

Abbildung 2.1: Lineare Trennung von Punkten zweier Klassen durch eine Hyperebene

Die Hyperebene teilt den Eingaberaum in zwei Halbräume:

$$\langle \omega, x_i \rangle + b > 0, \quad \text{für} \quad y_i = 1 \tag{2.2}$$

$$\langle \omega, x_i \rangle + b < 0, \quad \text{für} \quad y_i = -1 \tag{2.3}$$

Für die Lage der Hyperebene existieren unendlich viele Möglichkeiten (s. Abb. 2.2). Um trotzdem eine gute Generalisierungsfähigkeit der SVM zu gewährleisten und Overfitting zu vermeiden, wird eine Hyperebene mit maximalem Abstand zu den nächsten Trainingsdaten gesucht (s. Abb. 2.3). Um die Hyperebene eindeutig beschreiben zu können, wird (ω, b) relativ zur Trainingsmenge X skaliert, so dass gilt:

$$\min_{x_i \in X} |\langle \omega, x_i \rangle + b| = 1 \tag{2.4}$$

Eine so skalierte Ebene wird auch als kanonische Hyperebene bezeichnet.

Der maximale Rand der Hyperebene ergibt sich aus der Zusammenfassung der Ungleichungen (2.2) und (2.3) zu einer Ungleichung:

$$y_i(\langle \omega, x_i \rangle + b) > 0 \quad \forall i = 1, \ldots, m$$

Aufgrund der vorhergehenden Skalierung (2.4) erhält man die kanonische Form der Ungleichung:

$$y_i(\langle \omega, x_i \rangle + b) \geq 1 \quad \forall i = 1, \ldots, m$$

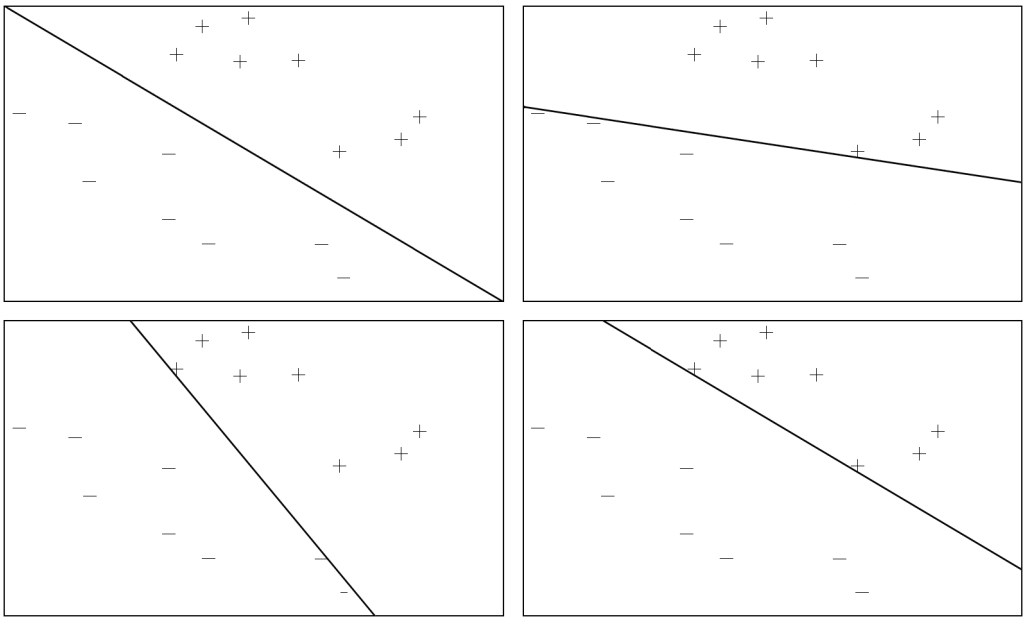

Abbildung 2.2: Unterschiedlich trennende Hyperebenen

Der Abstand $d(H, x_i)$ eines Objektes x_i der Klasse y_i zu einer Hyperebene H lässt sich berechnen mit:

$$d(H, x_i) = y_i \left(\left\langle \frac{\omega}{\|\omega\|}, x_i \right\rangle + \frac{b}{\|\omega\|} \right)$$

Für die Trainingsdaten $(x_1, +1)$ und $(x_2, -1)$, die sich nahe an der Hyperebene befinden, gilt:

$$\langle \omega, x_1 \rangle + b = +1 \quad \text{und} \quad \langle \omega, x_2 \rangle + b = -1$$

$$\Rightarrow \left\langle \frac{\omega}{\|\omega\|}, x_1 \right\rangle + \frac{b}{\|\omega\|} = \frac{1}{\|\omega\|} \quad \text{und} \quad \left\langle \frac{\omega}{\|\omega\|}, x_2 \right\rangle + \frac{b}{\|\omega\|} = \frac{-1}{\|\omega\|}$$

$$\Rightarrow \left\langle \frac{\omega}{\|\omega\|}, (x_1 - x_2) \right\rangle = \frac{2}{\|\omega\|}$$

Der Rand der trennenden Hyperebene beträgt somit: $\frac{2}{\|\omega\|}$. Um eine Hyperebene mit maximalem Rand zu erhalten, bedeutet also $\|\omega\|$ zu minimieren, dies ist gleichbedeutend mit der Minimierung von $\|\omega\|^2$. Für die korrekte Trennung der Trainingsdaten durch die Hyperebene wird noch eine Nebenbedingung (2.6) eingeführt.

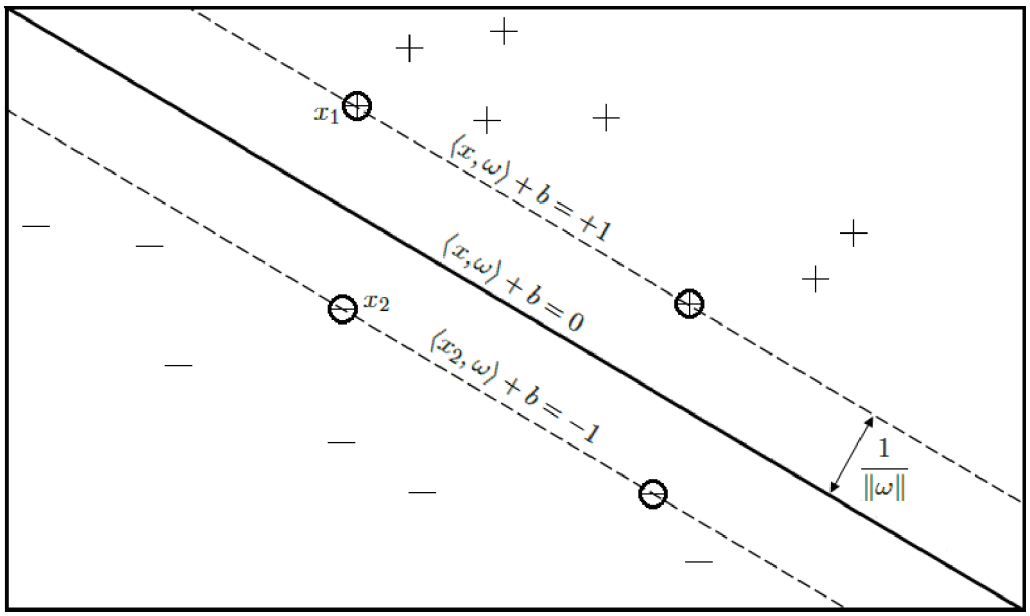

Abbildung 2.3: Hyperebene mit maximalem Rand

Zusammengefasst ergibt sich folgendes primäres Optimierungsproblem:

$$\min \qquad \tfrac{1}{2}\|\omega\|^2 \tag{2.5}$$

$$\text{unter der Bedingung} \quad y_i\left(\langle \omega, x_i \rangle + b\right) \geq 1, \quad \text{für } i = 1, \ldots, m \tag{2.6}$$

Das Optimierungsproblem lässt sich mit Hilfe von nicht-negativen Lagrange-Multiplikatoren ($\alpha_i \geq 0$) lösen. Mit den Lagrange-Multiplikatoren lassen sich Extrema von Funktionen unter Nebenbedingungen finden. Der Einsatz dieser Methode hat im Wesentlichen zwei Gründe [1]. Zum einen wird die Bedingung (2.6) durch die Bedingungen an die Lagrange-Multiplikatoren ersetzt, die viel einfacher zu handhaben sind. Zum anderen treten die Trainigsdaten nur noch in Form von Skalarprodukten auf. Dies ist von entscheidener Bedeutung bei der Übertragung der Vorgehensweise auf den nicht-linearen Fall (siehe Abschnitt 2.2). Die Formulierung des Problems (2.5) lässt sich in folgende Lagrange-Funktion überführen:

$$L(\omega, b, \alpha) = \frac{1}{2}\|\omega\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i\left(\langle \omega, x_i \rangle + b\right) - 1\right) \quad \text{mit } \alpha_i \geq 0 \tag{2.7}$$

Diese Funktion wird in Bezug auf ω und b minimiert und in Bezug auf α_i maximiert, d.h. die Lösung des Optimierungsproblems entspricht dem Sattelpunkt der Lagrange-Funktion. Aufgrund der Konvexität des Problems ist jedes lokale Minimum gleichzeitig ein globales Minimum [9].

Für den Sattelpunkt von $L(\omega, b, \alpha)$ gilt:

$$\frac{\partial}{\partial \omega} L(\omega, b, \alpha) = \omega - \sum_{i=1}^{m} \alpha_i y_i x_i = 0$$

$$\frac{\partial}{\partial b} L(\omega, b, \alpha) = \sum_{i=1}^{m} \alpha_i y_i = 0$$

Das Einsetzen der Sattelpunkt-Bedingungen in die Gleichung (2.7) liefert das duale Optimierungsproblem:

$$\max \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{2.8}$$

$$\text{unter den Nebenbedingungen} \qquad \alpha_i \geq 0 \qquad \text{und} \qquad \sum_{i=1}^{m} \alpha_i y_i = 0$$

Dieses Problem lässt sich jetzt effizient mit Standardverfahren der quadratischen Optimierung lösen, da die Zielfunktion nur noch von α_i abhängig ist. Trainingsdaten x_i. für die gilt: $\alpha_i > 0$, heißen Support-Vektoren. Mit den α_i lässt sich der Normalenvektor ω mit der Formel:

$$\omega = \sum_{i=1}^{m_{sv}} \alpha_i y_i x_i \tag{2.9}$$

berechnen, wobei m_{sv} die Anzahl der Support-Vektoren ist. Der Parameter b lässt sich einfach aus den Karush-Kuhn-Tucker (KKT) Bedingungen der Optimierungstheorie [1][6] berechnen[1]:

$$\alpha_i \left(y_i \left(\langle \omega, x_i \rangle + b \right) - 1 \right) = 0, \quad \forall i = 1, \ldots, m_{sv}$$

$$\Rightarrow b = y_i - \langle \omega, x_i \rangle, \qquad \forall \alpha_i > 0 \tag{2.10}$$

Somit sind alle Parameter der optimal trennenden Hyperebene gefunden. Zusammen mit (2.9) erhält man die Entscheidungsfunktion des binären Klassifikators:

$$g(x) = \langle \omega, x \rangle + b = \sum_{i=1}^{m_{sv}} \alpha_i y_i \langle x_i, x \rangle + b$$

Die Klassenzugehörigkeit neuer Daten wird durch das Vorzeichen der Entscheidungsfunktion bestimmt:

$$f(x_{neu}) = \operatorname{sgn} g(x) = \operatorname{sgn} \left(\sum_{i=1}^{m_{sv}} \alpha_i y_i \langle x_i, x_{neu} \rangle + b \right) \tag{2.11}$$

[1] Numerisch sicherer ist die Bildung des Mittelwertes für b aus allen Gleichungen mit $\alpha_i \neq 0$ [1].

2.1.2 Nicht-separabler Fall

Für linear nicht-separierbare Daten liefert das Optimierungsproblem keine gültige Lösung, da die Zielfunktion immer weiter anwächst. Um die Daten trotzdem durch eine Hyperebene optimal trennen zu können, werden Fehler zugelassen. Zu diesem Zweck werden sog. *Slack Variables* (Schlupfvariablen) $\xi_i, i = 1,\ldots,m$ eingeführt. Die *Slack Variables* erlauben Fehlklassifikationen von Trainingsdaten, die auf der falschen Seite der Hyperebene liegen (s. Abb. 2.4):

$$\begin{aligned} \langle \omega, x_i \rangle + b &\geq +1 - \xi_i\,, \quad \text{für } y_i = +1 \\ \langle \omega, x_i \rangle + b &\leq -1 + \xi_i\,, \quad \text{für } y_i = -1 \\ \xi_i &\geq 0\,, \quad \forall i \end{aligned}$$

Ein Fehler tritt auf, wenn $\xi_i > 1$. Die Summe der Fehler ξ_i ist somit eine obere Grenze für die Anzahl der Fehlklassifizierungen. Um die Breite des Randes zu maximieren, aber gleichzeitig die Anzahl der Fehlklassifikationen klein zu halten, wird die Zielfunktion des Optimierungsproblems um einen zusätzlichen Kostenausdruck erweitert:

$$\begin{aligned} &\min && \tfrac{1}{2}\|\omega\|^2 + C\textstyle\sum_{i=1}^{m} \xi_i \\ &\text{unter der Bedingung} && y_i(\langle \vec{\omega}, \vec{x_i} \rangle + b) \geq 1 - \xi_i, \text{ für } i = 1,\ldots,m \end{aligned} \tag{2.12}$$

Der Parameter C ist eine frei wählbare Konstante. Je größer C ist, desto stärker werden die Fehler bestraft.

Bei (2.12) handelt es sich wiederum um ein konvexes Optimierungsproblem. Die dazugehörige Lagrange-Funktion lautet:

$$L(\omega, b, \xi, \alpha, \mu) = \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i (y_i(\langle x_i, \omega \rangle + b) - 1 + \xi_i) - \sum_{i=1}^{m} \mu_i \xi_i$$

und die KKT-Bedingungen [1] liefern:

$$\begin{aligned} \alpha_i + \beta_i &= C \\ \alpha_i (y_i(\langle x_i, \omega \rangle + b) - 1 + \xi_i) &= 0 \\ \mu_i \xi_i &= 0 \end{aligned}$$

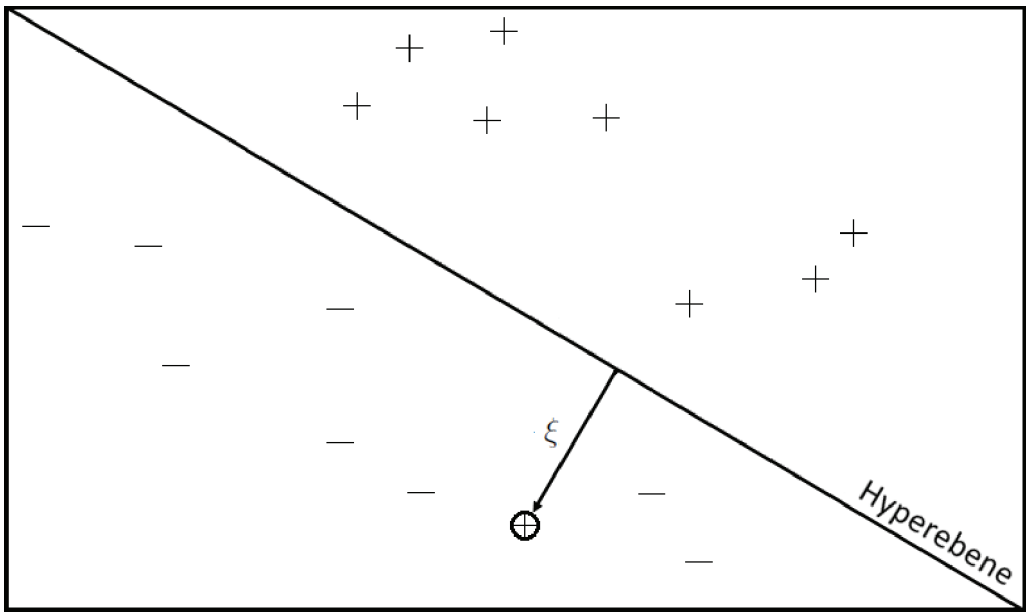

Abbildung 2.4: Trennende Hyperebene für den nicht-separablen Fall

Analog zum Abschnitt 2.1.1 erhält man das duale Optimierungsproblem:

$$\max_{\alpha} \quad W(\alpha) \quad = \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \qquad (2.13)$$

$$\text{unter den Nebenbedingungen} \quad 0 \leq \alpha_i \leq C \quad \text{und} \quad \sum \alpha_i y_i = 0, \forall i = 1, \ldots, m$$

Das duale Problem (2.13) bleibt im Vergleich zu (2.8) somit unverändert, nur die Nebenbedingungen haben sich geändert. Die Variablen α_i sind nach oben durch den Parameter C beschränkt. Aus den KKT-Bedingungen folgt unmittelbar, dass für $\alpha_i < C$ die Schlupfvariable $\xi_i = 0$ ist. Fehlklassifikationen treten daher nur bei x_i mit $\alpha_i = C$ auf.

Der Normalenvektor ω ergibt sich wiederum aus (2.9) und der Bias b aus (2.10) (vgl. die KKT-Bedingungen [1]). Die Entscheidungsfunktion bleibt unverändert:

$$f(x_{neu}) = sgn\left(\sum_{i=1}^{m_{sv}} \alpha_i y_i \langle x_i, x_{neu} \rangle + b\right)$$

2.2 Nicht-lineare Trennung

Die im vorhergehenden Abschnitt beschriebene SVM eignet sich nur für lineare Daten. In den meisten Fällen sind die Daten jedoch nicht-linear, wie zum Beispiel bei der Gesichtserkennung. Das beschriebene Konzept der SVM lässt sich jedoch auf nicht-lineare Daten erweitern.

Die Trainingsdaten x_i tauchen sowohl im dualen Optimierungsproblem (2.8,2.13) als auch in der Entscheidungsfunktion (2.11) nur in Skalarprodukten auf. Diesen Umstand macht man sich zunutze. Die Daten werden in einem höherdimensionalen Raum F abgebildet, in dem die Trennung durch eine Hyperebene leichter ist (s. Abb. 2.5). Die Abbildung geschieht mit einer sog. mapping Funktion Φ:

$$\Phi : \mathbb{R}^d \longrightarrow F$$

$$x \longmapsto \Phi(x)$$

Eine lineare Trennung im Raum F entspricht bei geeigneter Wahl der Abbildung Φ einer nicht-linearen Trennung im Eingaberaum X.

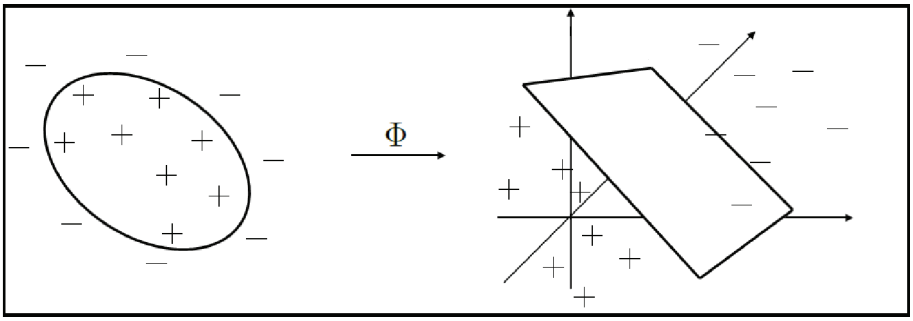

Abbildung 2.5: Transformation in einen Raum höherer Dimension

Bei F handelt es sich um einen Raum („Feature Space“) mit beliebiger (auch unendlicher) Dimension. Durch die Abbildung hängt der Trainingsalgorithmus nur vom Skalarprodukt $\langle \Phi(x), \Phi(x_i) \rangle$ der Daten in F ab. Die Berechnung des Skalarproduktes kann sehr schwierig oder gar unmöglich sein, wenn die Dimension von F zu groß ist. An dieser Stelle wird der sog. Kern-Trick angewendet. Anstatt des Skalarproduktes kommt eine Kernfunktion K zum Einsatz, die sich in F wie ein Skalarprodukt verhält:

$$K(x,y) = \langle \Phi(x), \Phi(y) \rangle$$

Der Kern wird benutzt, um alle Berechnungen in F implizit durchzuführen. Somit ist es nicht notwendig, die Abbildung Φ zu kennen. Ist eine symmetrische Funktion K gegeben, so gibt das Mercer Theorem eine notwendige und hinreichende Bedingung, mit der geprüft werden kann, ob K eine Kernfunktion ist.

Theorem 1 *Wenn μ ein endliches Maß und $K \in L_\infty(X)$ ein symmetrischer Kern, so dass der Integraloperator $T_K : L_2(X) \to L_2(X)$,*

$$T_K f(x) = \int_X K(x,y) f(y) \, d\mu(y)$$

positiv definit ist. Sei $\Phi_j \in L_2(X)$ die normalisierte Eigenfunktion von T_K mit den Eigenwerten $\lambda_j > 0$ und normalisiert, so dass $\left\|\Phi_j\right\|_{L_2(X)} = 1$ für alle $j \in \mathbb{Z}$.
Dann gilt:

1. $\left\{\lambda_j\right\}_j \in l_1$,
2. $\Phi_j \in L_\infty$,
3. $K(x,y) = \sum_{j\in\mathbb{Z}} \lambda_j \Phi_j(x) \Phi_j(y)$ *für fast alle* (x,y)

Aus 3. folgt, dass K dem Skalarprodukt in F entspricht, das heißt $K(x,y) = \langle \Phi(x), (y) \rangle_{l_2}$ mit:

$$\begin{aligned} \Phi : X &\longrightarrow & F \\ x &\longmapsto & \left\{\sqrt{\lambda}_j \Phi_j(x)\right\}_j. \end{aligned}$$

Durch die Konstruktion geeigneter Kernfunktionen kann man mit komplexen hochdimensionalen Merkmalsräumen arbeiten. Die Wahl der Kernfunktion ist somit für das Klassifikationsverfahren ausschlaggebend. Häufig verwendete Kernfunktionen sind:

- Linear: $K(x,x_i) = \langle x, x_i \rangle$
- Polynome: $K(x,x_i) = (\gamma \langle x, x_i \rangle + c_0)^d$ mit $\gamma, c_o \geq 0$ und $d \in N$
- Sigmoid: $K(x,x_i) = \tanh(\gamma \langle x, x_i \rangle + c_0)$ mit $\gamma > 0$ und $c_0 < 0$
- radiale Basisfunktion: $K(x,x_i) = \exp\left(-\gamma \cdot \|x - x_i\|^2\right)$ mit $\gamma > 0$

Im Bereich der Objekterkennung in Bildern hat sich vor allem die radiale Basisfunktion nach Gauß am besten bewährt. Die Ableitung des Kerns läßt sich wieder in Abhängigkeit vom Kern selbst ausdrücken:

$$\frac{\partial}{\partial x_i} K(x, x_i) = 2\gamma(x - x_i) K(x, x_i)$$

Um eine nicht-lineare Klassifikation durchzuführen, bedarf es demnach keiner algorithmischen Änderung des linearen Klassifikators. Das Skalarprodukt der Vektoren wird lediglich durch eine geeignete Kernfunktion ersetzt. Analog zum linearen Fall ergibt sich somit folgende Entscheidungsfunktion:

$$f(x_{neu}) = sgn\left(\sum y_i \alpha_i K(x_i, x_{neu}) + b\right) \tag{2.14}$$

Kapitel 3

Wavelet - Support-Vektor-Maschine

In diesem Kapitel wird ein Verfahren zur Beschleunigung der Klassifikationsphase von einer Support-Vektor-Maschine beschrieben. Die Beschleunigung wird durch Reduktion der Rechenoperationen erzielt. Die Idee dazu stammt aus [10]. Im ersten Abschnitt werden die theoretischen Grundlagen des Verfahrens erläutert. Der nächste Abschnitt beinhaltet die einzelenen Schritte für die Erstellung einer W-SVM. Der veränderte Algorithmus für die Klassifizierung wird im Abschnitt 3 beschrieben. Die Ergebnisse der Anwendung des Verfahrens werden im letzten Abschnitt präsentiert.

3.1 Theoretische Grundlagen

Bei der Klassifizierung eines neuen Patches nimmt die Berechnung des Kerns die meiste Zeit in Anspruch. Im Fall des Gauß-Kerns $K(x,x_i) = exp\left(-\gamma\|x-x_i\|^2\right)$ ist die Berechnung der Norm besonders zeitintensiv:

$$\|x-x_i\|^2 = x^t x - 2x_i^t x + x_i^t x_i \tag{3.1}$$

Bei einer Patch und Support-Vektor Größe von 20x20 Pixel sind jeweils 1200 Rechenschritte pro Support-Vektor notwendig. Zur Beschleunigung des Rechenvorgangs und somit Reduzierung des zeitlichen Aufwands wird der Term $x_i^t x_i$ bereits vor der Klassifizierung berechnet und gespeichert, da er nur von den Support-Vektoren abhängt und somit konstant ist. Für eine effiziente Berechnung von $2x_i^t x$ wird für x_i eine optimale Näherung u_i mit blockartiger Struktur (d.h. Rechtecke mit konstanten Grauwerten) ermittelt. Mit Hilfe der Näherung und eines Integralbildes lässt sich der

Term wie folgt berechnen:

$$2x^t u_i = 2\sum_{k=1}^{D} x_k u_{i,k} = 2\sum_{r=1}^{R_i} v_{i,r} \sum_{j=1}^{D_r} x_j,$$

mit $D =$ Dimension der Vektoren (zum Beispiel 400 bei einer Patchgröße von 20x20), $R_i =$ Anzahl der Rechtecke von u_i, $v_{i,r} =$ Grauwert zum Rechteck r und $x_j =$ Grauwert von x innerhalb des r-ten Rechtecks.

Integralbild

Die Summe aller Grauwerte innerhalb eines Rechtecks $\sum_{j=1}^{D_r} x_j$ wird effizient mit einem Integralbild [8] berechnet. Der Wert des Integralbildes ii am Punkt (x,y) ergibt sich aus der Summe aller Grauwerte des Eingabebildes i, die sich links und oberhalb des Punktes (x,y) befinden, einschließlich von Punkt (x,y) selbst:

$$ii(x,y) = \sum_{a=1}^{x} \sum_{b=1}^{y} i(a,b)$$

Diese Formel ist aber nur suboptimal, da für die Berechnung des Integralbildes einzelne Punkte mehrfach aus dem Eingabebild entnommen werden. Mit den folgenden Rekursionsgleichungen lässt sich das Integralbild in einem einzigen Durchlauf über das Eingabebild berechnen:

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \end{aligned}$$

$s(x,y)$ ist die kumulative Zeilensumme und es gilt: $s(x,-1) = 0$ und $ii(-1,y) = 0$. Das Integralbild ist jeweils um eine Dimension größer als das Eingabebild, um so auch Rechtecke am oberen und linken Rand eines Bildes berechnen zu können. Die zusätzliche Spalte und Zeile besteht jeweils nur aus Nullen und wird links und oberhalb des Integralbildes eingefügt.

Der Vorteil von Integralbildern ist, dass sich die Summe der Grauwerte des Eingabebildes innerhalb eines beliebigen Rechteckes in konstanter Zeit berechnen lässt, da nur vier Speicherzugriffe notwendig sind (s. Abb. 3.1). Die Berechnung ist unabhängig von der Position und Größe des Rechtecks:

$$\sum_{\substack{x_1 < a \leq x_4, \\ y_1 < b \leq y_4}} i(a,b) = ii(x_4,y_4) - ii(x_2,y_2) - ii(x_3,y_3) + ii(x_1,y_1)$$

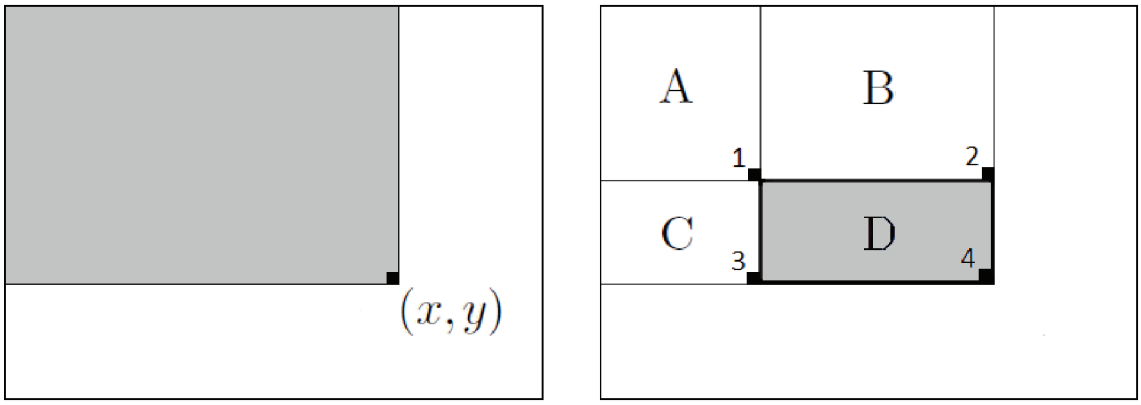

Abbildung 3.1: links: Der Wert an der Stelle $ii(x,y)$ ergibt sich aus der Summe aller Grauwerte, die sich links und oberhalb des Punktes $i(x,y)$ befinden. rechts: Berechnung der Summe aller Grauwerte in einem Rechteck: $D = ii(x_4,y_4) - ii(x_2,y_2) - ii(x_3,y_3) + ii(x_1,y_1)$.

Mit Hilfe eines Integralbildes lässt sich nun der Term $2x^t u_i$ effizient berechnen mit vier Rechenoperationen pro Rechteck.

$$2x^t u_i = 2\sum_{r=1}^{R_i} v_{i,r} \cdot (ii(x_4,y_4) - ii(x_2,y_2) - ii(x_3,y_3) + ii(x_1,y_1)) \tag{3.2}$$

Durch Anwendung der Integralbildmethode lässt sich die Entscheidungsfunktion jetzt wie folgt berechnen:

$$\begin{aligned} f(x) &= \operatorname{sgn}\left(\sum_{i=1}^{m_{sv}} y_i \alpha_i \exp\left(-\gamma\left(\|x\|^2 - 2x^t u_i + \|u_i\|^2\right)\right) + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{m_{sv}} y_i \alpha_i \exp\left(-\gamma\|u_i\|^2\right) \exp\left(-\gamma\|x\|^2\right) \exp\left(2\gamma x^t u_i\right) + b\right) \\ &= \operatorname{sgn}\left(\exp\left(-\gamma\|x\|^2\right) \cdot wsv \cdot \exp(xu) + b\right) \end{aligned} \tag{3.3}$$

mit $wsv = \left(y_i \alpha_i \exp\left(-\gamma\|u_i\|^2\right)\right)_{i=1,\ldots,m_{sv}}$, $G = (2\gamma v_{i,r})_{i=1,\ldots,m_{sv},\ r=1,\ldots,R_i}$ und
$xu = \left(\sum_{r=1}^{R_i} G_{i,r}\left(ii(x_{r,4,i}, y_{r,4,i}) - ii(x_{r,2,i}, y_{r,2,i}) - ii(x_{r,3,i}, y_{r,3,i}) + ii(x_{r,1,i}, y_{r,1,i})\right)\right)^t_{i=1,\ldots,m_{sv}}$. Für eine effiziente Berechnung wird der Vektor wsv und die Matrix g bereits vor der Klassifizierung bestimmt und gespeichert.

3.2 Vorverarbeitung der Support-Vektoren

3.2.1 Wavelet-Shrinkage

Um die Vorteile der Integralbildmethode ausnutzen zu können, müssen die Support-Vektoren eine blockartige Struktur aufweisen. D.h., es ist eine Näherung u gesucht, die viel weniger Rechteckregionen mit konstanten Grauwerten besitzt als der gegebene Support-Vektor x. Eine solche Näherung lässt sich mit Hilfe von Wavelet-Shrinkage [12] bestimmen. Diese Technik stammt aus dem Bereich Signal-Denoising. Durch Entfernen des „Rauschens“ wird eine Näherung an das ursprüngliche Signal bzw. Bild erzeugt, ohne dabei wichtige Strukturen, wie zum Beispiel Kanten, zu verlieren.

Das Optimierungsproblem lässt sich mathematisch wie folgt beschreiben:

$$\min_u \left\{ \underbrace{\|x-u\|^2}_{Distanz} + 2\alpha \underbrace{|u|}_{Rauschen} \right\}$$

Die Eliminierung des Rauschens aus den Wavelet-Koeffizienten geschieht in drei Schritten[10]:

- Wavelet Transformation des Eingabebildes:

$$x = \sum_{\lambda \in \Lambda} x_\lambda \Psi_\lambda \quad \rightarrow \quad x_\lambda = \langle x, \Psi_\lambda \rangle, \qquad \{\Psi_\lambda\} \text{ Wavelet-Basis}, \quad \lambda = (i,j,k)$$

- Anwendung einer Soft-Shrinkage Funktion mit dem Schwellenwert α auf die Wavelet-Koeffizienten x_λ:

$$u_\lambda = S_\alpha(x_\lambda) = \operatorname{sgn}(x_\lambda) \max\{|x_\lambda| - \alpha, 0\}$$

- Rücktransformation:

$$u = \sum_{\lambda \in \Lambda} u_\lambda \Psi_\lambda = \sum_{\lambda \in \Lambda} S_\alpha(x_\lambda) \Psi_\lambda = W^{-1} S_\alpha(Wx)$$

  Das W steht für die jeweilige Wavelet-Transformation.

Um die Rechteckstruktur zu erzeugen, wird als Wavelet-Basis das Haar-Wavelet verwendet:

$$\Psi(x) = \begin{cases} 1 & 0 \le x < 1/2 \\ -1 & 1/2 \le x < 1 \\ 0 & \text{sonst} \end{cases}$$

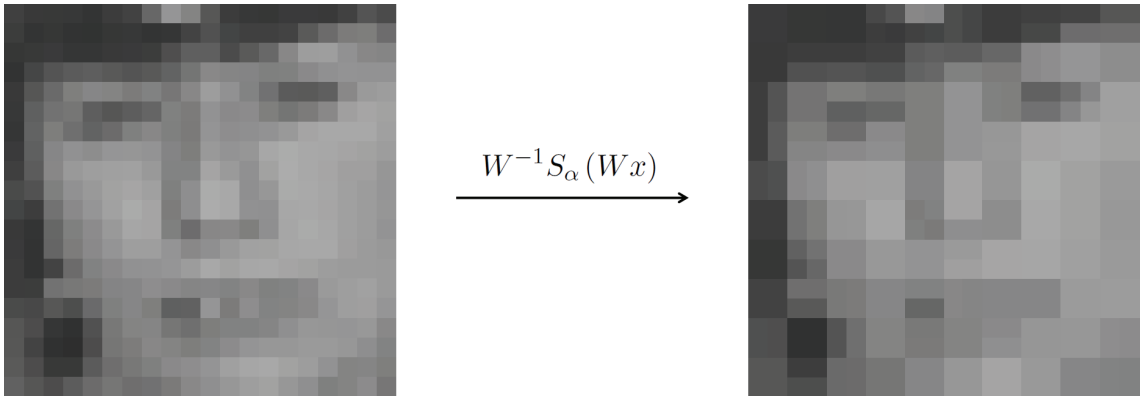

Abbildung 3.2: links: Orginal Support-Vektor, rechts: mit Wavelet-Shrinkage bearbeiteter Support-Vektor ($\alpha = 0.1$)

In dieser Arbeit werden Support-Vektoren, die mit Wavelet-Shrinkage bearbeitet wurden, als Wavelet-Support-Vektoren (W-SV) bezeichnet. Das Ergebnis einer Wavelet-Shrinkage Operation ist in der Abbildung 3.2 dargestellt. Im rechten Bild ist die gewünschte Block-struktur deutlich zu erkennen.
Die Anzahl der Rechtecke in einem W-SV hängt vom Schwellwert α ab. Je größer α ist, desto weniger Rechtecke sind im Vergleich zum Original Support-Vektor vorhanden. Die korrekte Wahl von α ist für eine effiziente und richtige Klassifizierung von entscheidener Bedeutung. Ist α zu groß, wird (3.3) schnell berechnet, aber auch viel falsch klassifiziert. Für ein zu kleines α wird zwar der Großteil richtig klassifiert, die Berechnung dauert aber auch entsprechend lange (siehe Abschnitt 3.4).

3.2.2 Rechtecke finden und speichern

Für die Berechnung von (3.2) werden die Koordinaten der Punkte 1 bis 4 benötigt (siehe Abb. 3.1). Zu diesem Zweck werden die W-SV in Rechtecke mit konstanten Grauwerten zerlegt. Die Anzahl der Rechtecke sollte dabei minimal sein. Aus den Koordinaten der linken-oberen Ecke (x_1, y_1) und rechten-unteren Ecke (x_2, y_2) eines Rechtecks ergeben sich die Koordinaten der Punkte 1 bis 4 wie folgt:

- Punkt 1 $(x, y) = (x_1, y_1)$
- Punkt 2 $(x, y) = (x_2 + 1, y_1)$
- Punkt 3 $(x, y) = (x_1, y_2 + 1)$

- Punkt 4 $(x,y) = (x_2+1, y_2+1)$

Anstelle der (x,y) - Koordinaten der Punkte werden die dazugehörigen Indizes gespeichert. In MATLAB kann so schneller auf den Inhalt einer Matrix zugegriffen werden. Die Suche nach den Rechtecken ist im Algorithmus 1 dargestellt.

Algorithmus 1 Funktion: Liste(W_SV)

Eingabe: W_SV ... mit Wavelet-Shrinkage bearbeitete Support-Vektoren

1: **for** $n = 1, \ldots$, Anzahl der W_SV **do**
2: $gw \leftarrow$ alle vorkommenden Grauwerte in W_SV_n ermitteln
3: $m = 1$
4: **for** $i = 1, \ldots$, Länge des Vektors gw **do**
5: **while** gw_i in W_SV_n vorkommt **do**
6: $[x,y] \leftarrow$ alle (x,y) - Koordinaten für die gilt: $W\_SV_n == gw_i$ ermitteln
7: **for** $j = 1, \ldots$, Länge des Vektors x **do**
8: **if** Rechteck $W\_SV_n(\ (x_1,y_1)\ ,\ (x_j,y_j)\ ) == gw_i$ **then**
9: $k = j$
10: **end if**
11: **end for**
12: $W\_SV_n(\ (x_1,y_1)\ ,\ (x_k,y_k)\ ) = -1$ ◁ Rechteck als gefunden markieren
13: $ind = [\ (x_1,y_1)\ ,\ (x_k+1,y_1)\ ,\ (x_1,y_k+1)\ ,\ (x_k+1,y_k+1)\ ]$
14: $ind \leftarrow$ Umrechnung der Koordinaten von (x,y) zu Index
15: $liste_{m,1-5,n} = [\ ind_1\ ,\ ind_2\ ,\ ind_3\ ,\ ind_4\ ,\ gw_i\ ]$
16: $m = m+1$
17: **end while**
18: **end for**
19: **end for**

Die Überprüfung in Zeile 8 ist notwendig, da einige Grauwerte in mehreren Regionen in einem Bild auftreten und einige Regionen mit konstantem Grauwert erst in mehrere Rechtecke zerlegt werden müssen (siehe Abb. 3.3). Als Ergebnis liefert der Algorithmus eine mx5xn Matrix (siehe Tabelle 3.1). In den Spalten eins bis vier ist jeweils der Index der Punkte 1 bis 4 und in der fünften Spalte der Grauwert eines Rechtecks vermerkt.

Punkt 1	Punkt 2	Punkt 3	Punkt 4	Grauwert
361	362	382	383	0,4000
382	383	403	404	0,4000
340	341	361	362	0,4039
339	340	381	382	0,4039
403	404	424	425	0,4039
381	382	402	403	0,4039
319	320	340	341	0,4078
402	403	423	424	0,4078
298	299	319	320	0,4156
383	384	425	426	0,4196
340	319	339	318	0,4196
384	363	383	362	0,4235
363	342	362	341	0,4274
319	298	318	297	0,4274
299	278	298	277	0,4313
342	321	341	320	0,4392

Tabelle 3.1: Auszug aus einer mit Algorithmus 1 erzeugten Liste

3.3 Klassifizierung

Für die Klassifizierung eines Patches sind jeweils zwei Schritte notwendig:

1. Erzeugen eines Integralbildes *ii* aus dem zu klassifizierenden Patch

2. Berechnen der Entscheidungsfunktion (3.3).

Die Schritte werden entsprechend der Anzahl der zu klassifizierenden Patches wiederholt. Ein Integralbild *ii* kann in MATLAB effizient mit der Funktion `cumsum` bestimmt werden. Die Funktion wird zweimal hintereinander ausgeführt und zwar entlang der ersten und zweiten Dimension. Alle benötigten Parameter werden im Vorfeld in einer Datei gespeichert und vor der Klassifizierung geladen. Die Datei enthält folgende Werte:

- Parameter: b - Bias, γ - Skalierungsfaktor für den Gauß-Kern, w - Gewichtsvektor
- wsv - $\left(y_i\alpha_i \exp\left(-\gamma\|u_i\|^2\right)\right)_{i=1,\ldots,m_{sv}}$ (vorab berechnet)
- Matrix der Grauwerte G - $(2\gamma v_{i,r})_{i=1,\ldots,m_{sv},\, r=1,\ldots,R_i}$ (vorab berechnet)
- Matrizen L_1, L_2, L_3 und L_4 mit den jeweiligen Koordinaten der Punkte 1 bis 4

Die Parameter b, γ und w werden aus einer vorgegeben SVM entnommen. Die Matrizen L_1 , L_2 , L_3 und L_4 und G ergeben sich aus der im Abschnitt 3.2.2 erstellten Liste.

3.4 Tests und Ergebnisse

Das beschriebene Verfahren wurde in MATALB[1] implemtiert und getestet. Zunächst wurde getestet, welchen zeitlichen Vorteil die Berechnung des Skalarproduktes mit der Integralbildmethode bringt. Die Abbildung 3.4 zeigt die benötigte Zeit für die Berechnung am Beispiel eines Patches und 100 Support-Vektoren. Je größer die Anzahl der Rechtecke in den Support-Vektoren ist, desto mehr Zeit wird für die Berechnung benötigt. Als Vergleich wurde das Skalarprodukt mit der dafür vorgesehenden MATLAB Funktion `dot` und mit $x^t y$ berechnet. Gegenüber der `dot`-Funktion ist die Integralbildmethode immer schneller, selbst wenn die Support-Vektoren aus 400 Rechtecken bestehen. Im Vergleich zu $x^t y$ dürfen in einem Support-Vektor maximal 100 Rechtecke auftreten, um mindestens genauso schnell zu sein. Bei einer Vektorlänge von 400 sind 799 Rechenschritte für die Berechnung von $\langle x, y\rangle$ notwendig. Theoretisch könnten nach (3.2) somit maximal 200 Rechtecke pro Support-Vektor auftreten, um in etwa die gleiche Zeit zu erhalten wie für $x^t y$. Die Berechnung des Integralbildes und der Zugriff darauf verlangsamen aber die Berechnung. Tests haben gezeigt, dass ab ca. 40 Support-Vektoren die Berechnung des Integralbildes aber kaum noch Einfluß auf die Gesamtzeit der Berechnung hat. Bei der Berechnung des Skalarproduktes ist die Integralbildmethode somit nur mit weniger als 100 Rechtecken pro Support-Vektor und mit mehr als 40 Vektoren schneller als $x^t y$.

Im nächsten Schritt wurde die Integralbildmethode auf eine zur Verfügung gestellte Support-Vektor-Maschine angewendet. Die SVM enthält insgesamt 135 Support-Vektoren, davon sind 47 Vektoren Gesichter und 88 Vektoren nicht Gesichter, mit einer Größe von jeweils 20x20 Pixel. Als

[1] MATLAB Version: 7.8.0.347 (R2009a), 64-bit (win64); PC: Intel Core i7 CPU 920, 6 GB RAM, Windows 7 (64-bit)

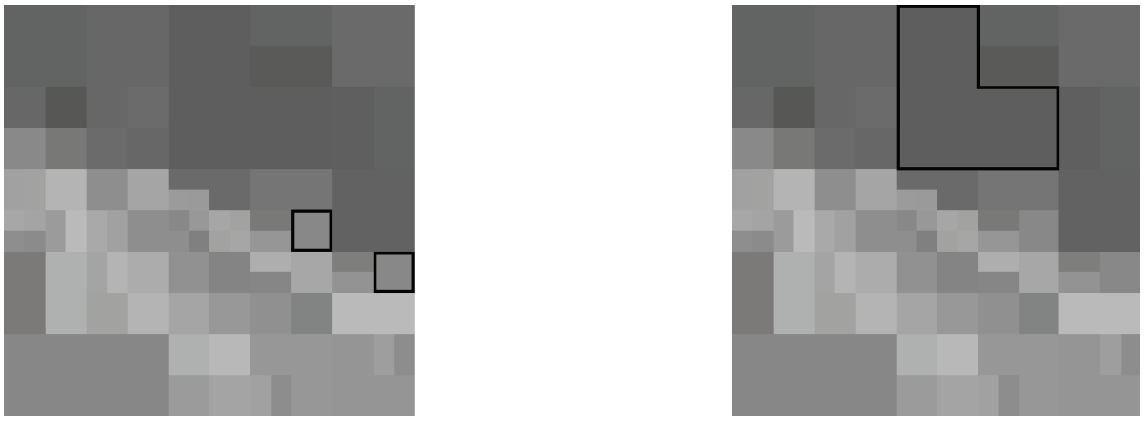

Abbildung 3.3: links: gleicher Grauwert in verschiedenen Regionen des Bildes (schwarz umrandet), rechts: Region mit konstantem Grauwert (schwarz umrandet)

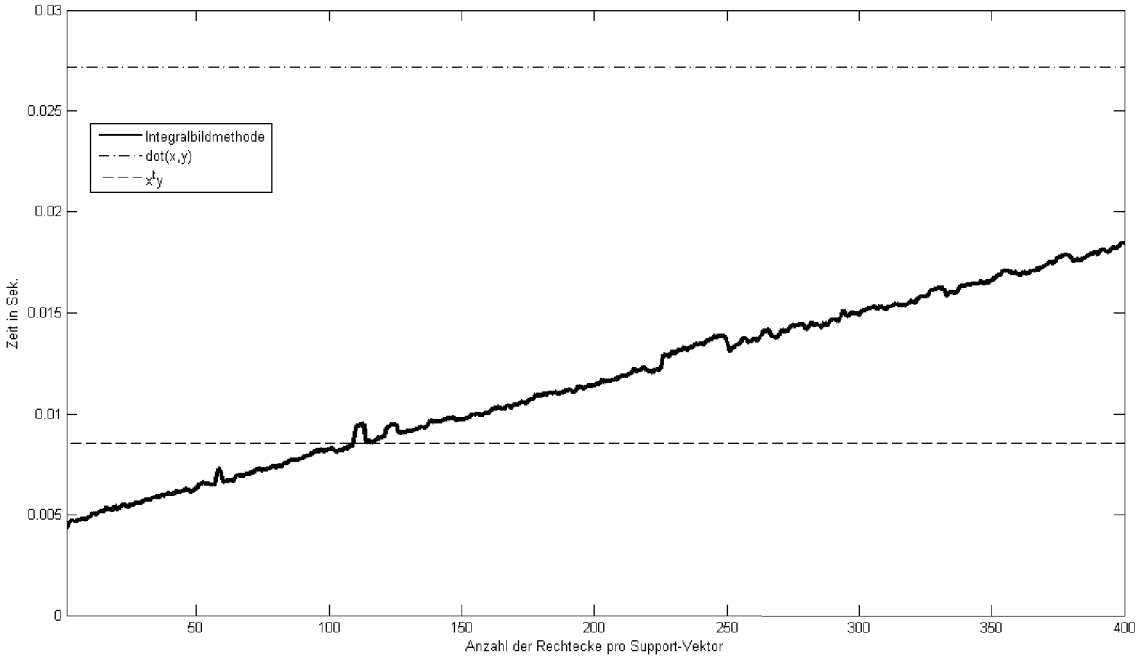

Abbildung 3.4: Zeiten für die Berechnung des Skalarproduktes

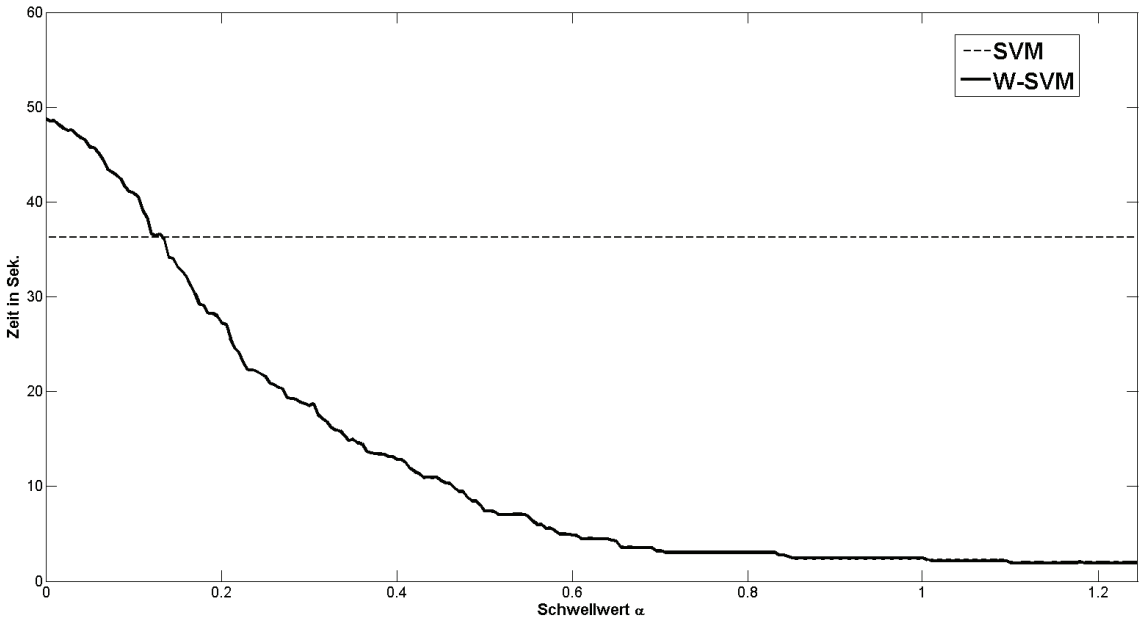

Abbildung 3.5: Klassifizierungsgeschwindigkeiten in Abhängigkeit von den Schwellwerten α

Testdatensatz wurden die Trainingsdaten der SVM verwendet. Der Datensatz besteht insgesamt aus 23.469 Bildern, mit 3.194 positiv und 20.275 negativ Beispielen für Gesichter. Die Klassifizierung wurde mit unterschiedlichen Schwellwerten α für die Erzeugung der Wavelet-Vektoren durchgeführt. Die benötigte Zeit für die Klassifizierung ist in der Abbildung 3.5 dargestellt. In der Tabelle 3.2 sind die dazugehörigen Genauigkeiten, Erkennungsraten und Falschalarmraten (nicht-Gesichter als Gesichter erkannt) aufgeführt. Der Schwellwert $\alpha = 0,00$ entspricht der Original SVM. Ab einem Schwellwert von 0,13 ist die Berechnung der Klassenzugehörigkeit mit der W-SVM in etwa genauso schnell wie mit herkömmlichen Verfahren. Bei diesem Schwellwert wurden insgesamt 11.157 Rechtecke erzeugt, d.h. ca. 82 Rechtecke pro Vektor. Aus der Tabelle 3.2 ist ersichtlich, dass dieser Schwellwert aber ungeeignet ist, da die Erkennungsrate mit weniger als 15% ziemlich gering ist. Die Genauigkeit ist zwar noch relativ hoch, aber nur weil die Mehrheit des Testdatensatzes aus nicht-Gesichtern besteht und je größer der Schwellwert ist, desto mehr Patches werden als nicht-Gesicht klassifiziert.

Um zu ermitteln, welche Schritte bei der Klassifizierung besonders viel Zeit in Anspruch nehmen, wurde der Progammcode mit dem MATLAB - Profiler untersucht. Das Ergebnis ist in Abbildung 3.6 dargestellt. Die Abbildung zeigt die benötigte Zeit zum einen für einzelne Rechenoperationen (Zeile 34 - 41) und zum anderen für die Durchführung der Berechung in einem Schritt (Zeile 44

Schwellwert	Rechtecke insgesamt	Genauigkeit	Erkennungsrate	Falschalarmrate
0,00	51.250	0,990	0,992	0,063
0,01	45.999	0,992	0,992	0,053
0,02	39.511	0,993	0,990	0,045
0,03	34.062	0,993	0,984	0,033
0,04	29.735	0,993	0,972	0,027
0,05	26.199	0,988	0,938	0,023
0,06	23.186	0,983	0,897	0,019
0,07	20.648	0,975	0,830	0,016
0,08	18.591	0,964	0,747	0,013
0,09	16.804	0,951	0,653	0,010
0,10	15.000	0,934	0,523	0,008
0,11	13.369	0,915	0,380	0,008
0,12	12.177	0,900	0,254	0,006
0,13	11.157	0,883	0,148	0,005
0,14	10.245	0,875	0,084	0,005

Tabelle 3.2: Ergebnisse der Klassifizierung mit der W-SVM

```
                30          % Integralbild erzeugen
 0.23   23469  31          I(2:end,2:end) = cumsum(cumsum(X(:,:,i),1),2);
                32
                33          % Zugriff auf Integralbild
40.17   23469  34          l11 = I(l1);l12 = I(l2);l13 = I(l3);l14 = I(l4);
                35
                36          % Berechnung von exp(xu)
 7.56   23469  37          IG = exp(sum(l5.*(l11 - l12 - l13 + l14)));
                38
                39          % Klassifikation
 0.44   23469  40          f = sign(exp(reshape(X(:,:,i),1,400)*reshape(X(:,:,i),400,1)*-p)*...
                41              IG*wsv + b);
                42
                43          % Klassifikation in einem Schritt
49.32   23469  44          f = sign(exp(reshape(X(:,:,i),1,400)*reshape(X(:,:,i),400,1)*-p)*...
                45              exp(sum(l5.*(I(l1) - I(l2) - I(l3) + I(l4))))*wsv + b);
```

Abbildung 3.6: Auszug aus dem MATLAB - Profiler (linke Spalte: benötigte Zeit für die eizelnen Schritte)

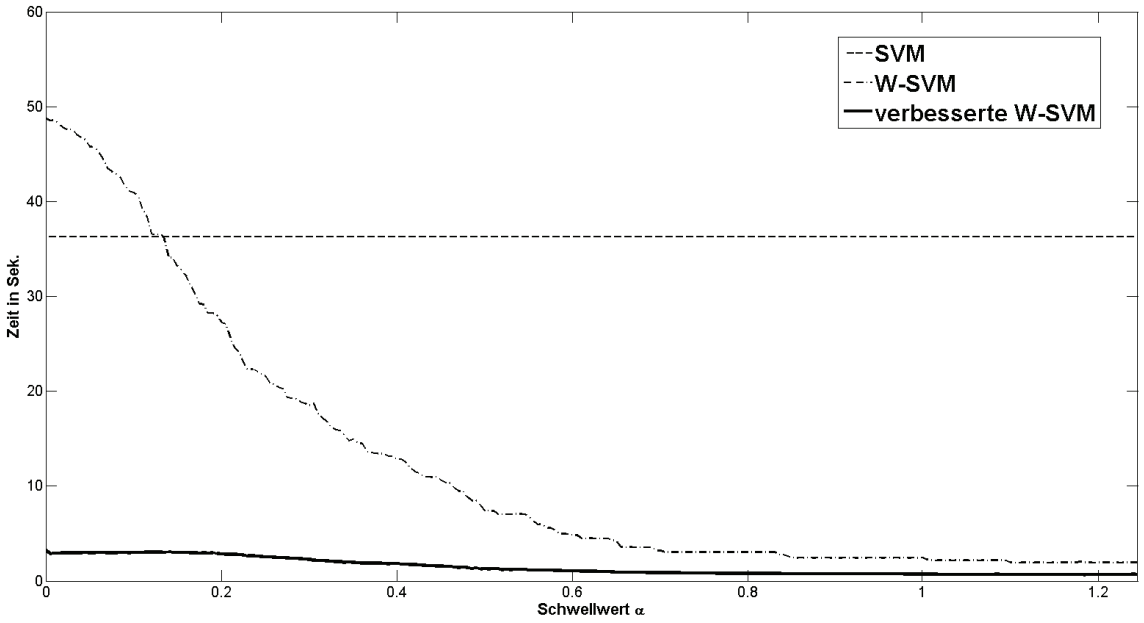

Abbildung 3.7: Klassifizierung mit der verbesserten W-SVM

- 45). Bei der Klassifikation macht der Zugriff auf das Integralbild ca. 80% der Gesamtzeit aus. Somit gilt es, die Anzahl der Zugriffe auf das Integralbild zu verringern, um so die Klassifizierung weiter zu beschleunigen. Viele Rechtecke kommen in der gleichen Größe und Position in den W-SVn vor, aber mit unterschiedlichem Grauwert. Beim Zugriff auf das Integralbild werden damit häufig die selben Rechtecke abgefragt. Um dies zu vermeiden werden alle Einträge, die mehrfach in der Liste der Rechtecke auftreten, entfernt. Jedes Rechteck ist somit einmalig in der Liste vorhanden. Die Beschleunigung der verbesserten W-SVM ist in der Abbildung 3.8 dargestellt. Die Klassifizierung verläuft jetzt deutlich schneller als bei der SVM, auch bei geringeren Schwellwerten. Der optimale Schwellwert für die W-SV liegt bei $\alpha = 0,01$. Bei diesem Wert wird die schnellste Zeit gemessen. Für die Klassifizierung benötigt die verbesserte W-SVM 16-mal weniger Zeit als die eigentliche W-SVM bei gleicher Klassifikationsrate und 12-mal weniger Zeit als die SVM bei fast gleicher Klassifikationsrate (siehe Tabelle 3.2).

Beim optimalen Schwellwert sinkt die Anzahl der Zugriffe auf das Integralbild von 183.996 $(45.999 * 4)$ auf 4.108 $(1.027 * 4)$. Die Auswirkungen auf die Zeiten zeigt die Abbildung 3.8. Der Zugriff auf das Integralbild beträgt jetzt nur noch ca. 35% der für die Klassifizierung benötigten Zeit. Das vorgestellte Verfahren ist somit mit kleinen Korrekturen deutlich schneller als das derzeitig verwendete Verfahren. Die Klassifizierungsgenauigkeit konnte gegenüber der SVM sogar

```
                78              % Integralbild erzeugen
0.25    23469   79              I(2:end,2:end) = cumsum(cumsum(X(:,:,i),1),2);
                80
                81              % Zugriff auf Integralbild
1.06    23469   82              l11 = I(l1);l12 = I(l2);l13 = I(l3);l14 = I(l4);
                83
                84              % Berechnung von exp(xu)
1.76    23469   85              IG = exp(g*(l11 - l12 - l13 + l14));
                86
                87              % Klassifikation
0.38    23469   88              f = sign(exp(reshape(X(:,:,i),1,400)*reshape(X(:,:,i),400,1)*-p)*...
                89                  wsv*IG + b);
                90
                91              % Klassifikation in einem Schritt
2.97    23469   92              f = sign(exp(reshape(X(:,:,i),1,400)*reshape(X(:,:,i),400,1)*-p)*...
                93                  wsv*exp(g*(I(l1) - I(l2) - I(l3) + I(l4))) + b);
```

Abbildung 3.8: Auszug aus dem MATLAB-Profiler für verbesserte W-SVM (linke Spalte: benötigte Zeit für die eizelnen Schritte)

noch etwas gesteigert werden. Eine weitere Beschleunigung des oben beschriebenen Verfahrens war mit MATLAB nicht möglich.

Die Idee, dass sich einige Werte bereits vor der Klassifizierung berechnen und speichern lassen, wurde zum Vergleich auch auf die Original-SVM übertragen. Dabei konnte eine deutliche Zeitsteigerung erzielt werden. Der Testdatensatz ließ sich nun in weniger als einer Sekunde klassifizieren bei gleichbleibender Genauigkeit.

Kapitel 4

Reduzierte Support-Vektor-Maschine

In diesem Kapitel wird ein weiteres Verfahren zur Beschleunigung der Klassifikation von einer Support-Vektor-Maschine beschrieben. Die Beschleunigung wird durch die Verringerung der Anzahl der Vektoren erreicht. Die Idee wurde bereits mehrfach erfolgreich umgesetzt [10] [11]. Im ersten Abschnitt werden die theoretischen Grundlagen des Verfahrens erläutert. Der nächste Abschnitt beschreibt die Annäherung an den Normalenvektor einer SVM. Der veränderte Algorithmus für die Klassifizierung wird im Abschnitt 3 beschrieben. Die Ergebnisse der Anwendung des Verfahrens werden im letzten Abschnitt präsentiert.

4.1 Theoretische Grundlagen

Eine weitere Möglichkeit, die Berechnung der Entscheidungsfunktion zu beschleunigen, ist die Reduzierung der Komplexität einer Support-Vektor-Maschine. Die Laufzeit der Klassifikation mit einer SVM ist linear abhängig von der Anzahl der Support-Vektoren. Je mehr Support-Vektoren vorhanden sind, desto länger dauert die Klassisfikation eines Patches. Um die Komplexität einer gegebenen SVM zu verringern, wird eine Näherung an die SVM bestimmt, die viel weniger Support-Vektoren besitzt. Eine solche Näherung bezeichnet man als reduzierte Support-Vektor-Maschine (RVM).

Sei $\Psi_{SVM} = \sum_{i=1}^{N_x} \alpha_i \Phi(x_i) \in F$ der Normalenvektor einer SVM mit N_x Vektoren und $\Psi_{RVM} = \sum_{i=1}^{N_z} \beta_i \Phi(z_i) \in F$ der Normalenvektor einer RVM mit $N_z \ll N_x$. Die reduzierten Vektoren z_i lassen sich dann durch die Minimierung von:

$$\|\Psi_{SVM} - \Psi_{RVM}\|^2 \tag{4.1}$$

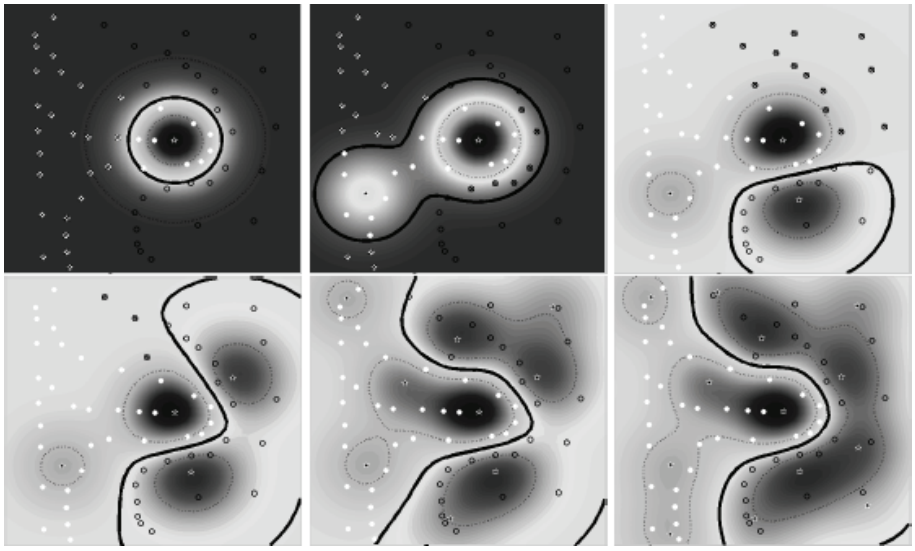

Abbildung 4.1: Eregbnis der sequentiellen Anwendung von reduzierten Vektoren (Sterne) auf ein Testdatensatz. Es ist der Einfluß von 1, 2, 3, 4, 9 und 13 Vektoren dargestellt. Dunkle Bereiche weisen auf einen starken Einfluß der Vektoren für die Klassifikation hin. [Quelle: [11]]

bezüglich z_i und β_i bestimmen. Aus den ermittelten z_i lassen sich die optimalen β_i berechnen. Die reduzierten Vektoren (RV) haben den Vorteil, dass sie jeden Wert annehmen können. Sie sind nicht auf die Trainingsvektoren beschränkt wie die Support-Vektoren. Für eine Annäherung an die SVM sind somit nur wenige RV erforderlich. Zum Beispiel lässt sich eine SVM mit mehr als 8000 Support-Vektoren durch eine RVM mit nur 100 RV annähern. Ein weiterer Vorteil einer RVM ist die hierarchische Anordnung der Vektoren. Der erste reduzierte Vektor unterscheidet die Daten am besten. Die meisten Daten, die vom ersten Vektor falsch klassifiziert wurden, werden vom zweiten Vektor richtig klassifiziert usw. (siehe Abb. 4.1). Durch Ersetzen des Normalenvektors Ψ_{SVM} durch Ψ_{RVM} in (2.14) erhält man folgende Entscheidungsfunktion:

$$f(x_{neu}) = sgn\left(\sum_{i=1}^{N_z} \beta_i K(z_i, x_{neu}) + b\right) \tag{4.2}$$

Eine zusätzliche Beschleunigung wird durch die Verwendung einer Kaskade erreicht. Hierbei wird ein Patch zunächst nur mit $(\beta_1\ ,\ z_1)$ klassifiziert. Anschließend wird geprüft, ob $\beta_1 \langle \Phi(z_1), \Phi(x) \rangle + b\,(*)$ so groß ist, dass der Rest $\sum_{i=2}^{N_z} \beta_i \langle \Phi(z_i), \Phi(x) \rangle + b$ das Vorzeichen von $(*)$ nicht mehr ändert. Ist dies der Fall, kann die Klassifizierung an dieser Stelle beendet werden. Ansonsten müssen weitere Support-Vektoren zur Klassifizierung herangezogen werden, bis sich das Vorzeichen nicht mehr ändert. In den meisten Fällen benötigt man so nicht die volle RVM für die Klassifizierung und erreicht somit eine weitere Beschleunigung.

4.2 Reduzierte Vektoren

Angenommen $\beta = (\beta_1, \ldots, \beta_{N_z})$ und $z = (z_1, \ldots, z_{N_z})$ sind gegeben und gesucht ist der Koeffizient β_{N_z+1} und Vektor z_{N_z+1}. Durch eine Erweiterung der zu minimierenden Zielfunktion (4.1) lässt sich β_{N_z+1} und z_{N_z+1} bestimmen:

$$\min \left\| \sum_{i=1}^{N_x} \alpha_i \Phi(x_i) - \sum_{j=1}^{N_z} \beta_j \Phi(z_j) - \beta_{N_z+1} \Phi(z_{N_z+1}) \right\|_F^2 \tag{4.3}$$

Für $N_z = 0$ entfällt der mittlere Term.

Setzt man: $\tilde{\alpha} = (\alpha_1, \ldots, \alpha_{N_x}, -\beta_1, \ldots, -\beta_{N_z})^t$ und $\tilde{x} = (x_1, \ldots, x_{N_x}, z_1, \ldots, z_{N_z})$, folgt für die zu minimierende Zielfunktion (4.3):

$$\begin{aligned}
f(\beta_{N_z+1}, z_{N_z+1}) &= \left\| \sum_{i=1}^{M} \tilde{\alpha}_i \Phi(\tilde{x}_i) - \beta_{N_z+1} \Phi(z_{N_z+1}) \right\|_F^2 \\
&= \left\| \sum_{i=1}^{M} \tilde{\alpha}_i \Phi(\tilde{x}_i) \right\|_F^2 - 2 \left\langle \sum_{i=1}^{M} \tilde{\alpha}_i \Phi(\tilde{x}_i) \ , \ \beta_{N_z+1} \Phi(z_{N_z+1}) \right\rangle + \left\| \beta_{N_z+1} \Phi(z_{N_z+1}) \right\|_F^2 \\
&= \sum_{i,j=1}^{M} \alpha_i \alpha_j \left\langle \Phi(x_i), \Phi(x_j) \right\rangle - 2\beta_{N_z+1} \sum_{i=1}^{M} \tilde{\alpha}_i \left\langle \Phi(\tilde{x}_i), \Phi(z_{N_z+1}) \right\rangle + \beta_{N_z+1}^2 \underbrace{\left\| \Phi(z_{N_z+1}) \right\|^2}_{=1} \\
&= \tilde{\alpha}^t B_1 \tilde{\alpha} - 2\beta_{N_z+1} \tilde{\alpha}^t B_2 + \beta_{N_z+1}^2
\end{aligned} \tag{4.4}$$

mit $M = N_x + N_z$, $B_1 = \left(\left\langle \Phi(\tilde{x}_i) \ , \ \Phi(\tilde{x}_j) \right\rangle \right)_{i,j=1,\ldots,M} = \left(exp\left(-\gamma \left\| \tilde{x}_i - \tilde{x}_j \right\|^2 \right) \right)_{i,j=1,\ldots,M}$ und $B_2 = \left(\left\langle \Phi(\tilde{x}_i) \ , \ \Phi(z_{N_z+1}) \right\rangle \right)_{i=1,\ldots,M}^t = \left(exp\left(-\gamma \left\| \tilde{x}_i - z_{N_z+1} \right\|^2 \right) \right)_{i=1,\ldots,M}^t$.

Die notwendige Bedingung für (4.4) erhält man, in dem (4.4) nach β_{N_z+1} abgeleitet und gleich Null gesetzt wird:

$$\begin{aligned}
\frac{\partial f(\beta_{N_z+1}, z_{N_z+1})}{\partial \beta_{N_z+1}} &= -2\widetilde{\alpha}^t B_2 + 2\beta_{N_z+1} = 0 \\
\Rightarrow \quad \widehat{\beta}_{N_z+1} &= \widetilde{\alpha}^t B_2
\end{aligned} \tag{4.5}$$

Durch Einsetzen von (4.5) in die Gleichung (4.4) muss die Zielfunktion nur noch bezüglich z_{N_z+1} minimiert werden:

$$\begin{aligned} f(\beta_{N_z+1}, z_{N_z+1}) &= \widetilde{\alpha}^t B_1 \widetilde{\alpha} - 2\widetilde{\alpha}^t B_2 B_2^t \widetilde{\alpha} + \left(\widetilde{\alpha}^t B_2\right)^2 \\ &= \widetilde{\alpha}^t B_1 \widetilde{\alpha} - \left(\widetilde{\alpha}^t B_2\right)^2 \\ &= \widetilde{\alpha}^t B_1 \widetilde{\alpha} - \left(\sum_{i=1}^{M} \widetilde{\alpha}_i exp\left(-\gamma \left\| \widetilde{x}_i - z_{N_z+1} \right\|^2\right) \right)^2 \end{aligned} \tag{4.6}$$

Für die Minimierung von (4.6) wird die MATLAB Funktion `fminsearch` genutzt. Die Funktion verwendet für die Minimierung den einfachen Nelder-Mead Simplex-Algorithmus. Als Ergebnis liefert die Funktion den Vektor $\widehat{z}_{N_z+1}$, für den (4.6) minimal wird. Durch Einsetzen von $\widehat{z}_{N_z+1}$ in (4.5) lässt sich $\widehat{\beta}_{N_z+1}$ berechnen.

Eine weitere Möglichkeit $\widehat{\beta}$ bei gegebenen $\widehat{z}$ zu berechnen, wird in [4] beschrieben. Diese Variante liefert ein optimales β^*, das besser oder mindestens genauso gut ist wie das Original $\widehat{\beta}$:

$$\begin{aligned} f\left(\widehat{\beta}, \widehat{z}\right) &= \alpha^t B_5 \alpha - 2\widehat{\beta}^t B_3 \alpha + \widehat{\beta}^t B_4 \widehat{\beta} \\ \Rightarrow \frac{\partial f\left(\widehat{\beta}, \widehat{z}\right)}{\partial \widehat{\beta}} &= -B_3 \alpha + B_4 \widehat{\beta} = 0 \\ \Rightarrow \beta^* &= B_4^{-1} B_3 \alpha \end{aligned} \tag{4.7}$$

mit $B_5 = \left(exp\left(-\gamma \left\| x_i - x_j \right\|^2\right)\right)_{i,j=1,\ldots,N_x}$, $B_3 = \left(exp\left(-\gamma \left\| x_i - \widehat{z}_j \right\|^2\right)\right)_{i=1,\ldots,N_x, j=1,\ldots,N_{\widehat{z}}}$ und $B_4 = \left(exp\left(-\gamma \left\| \widehat{z}_i - \widehat{z}_j \right\|^2\right)\right)_{i,j=1,\ldots,N_{\widehat{z}}}$.

Die Berechnung von β^* und $\widehat{z}$ ist im Algorithmus 2 dargestellt. In der Abbildung 4.2 sind einige reduzierte Vektoren, die dabei entstehen, dargestellt.

Algorithmus 2 Funktion: RV(x, α, γ)

Eingabe: x, α, γ

Ausgabe: $\widehat{z}, \beta^*$

1: $\widetilde{x} = x \quad , \quad \widetilde{\alpha} = \alpha \quad , \quad N_x =$ Anzahl der Support-Vektoren $\quad , \quad M = N_x$

2: **while** $\|\Psi_{SVM} - \Psi_{RVM}\|^2$ sich wesentlich ändert **do**

3: $\quad B_1 = \left(exp\left(-\gamma \left\|\tilde{x}_i - \tilde{x}_j\right\|^2\right)\right)_{i,j=1,\ldots,M}$

4: $\quad \widetilde{x}_{M+1} \leftarrow$ mittels `fminsearch`-Funktion bestimmen

5: $\quad B_2 = \left(exp\left(-\gamma \|\tilde{x}_i - \widetilde{x}_{M+1}\|^2\right)\right)^t_{i=1,\ldots,M}$

6: $\quad B_3 = \left(exp\left(-\gamma \left\|\widetilde{x}_i - \widetilde{x}_j\right\|^2\right)\right)_{i=1,\ldots,N_x \ , \ j=N_x+1,\ldots,M+1}$

7: $\quad B_4 = \left(exp\left(-\gamma \left\|\widetilde{x}_i - \widetilde{x}_j\right\|^2\right)\right)_{i,j=N_x+1,\ldots,M+1}$

8: $\quad \widetilde{\alpha}_{N_x+1,\ldots,M+1} = -\left(B_4^{-1} B_3 \alpha\right)$

9: $\quad M = M + 1$

10: **end while**

11: $\widehat{z} = \widetilde{x}_{N_x+1,\ldots,M}$

12: $\beta^* = -\widetilde{\alpha}_{N_x+1,\ldots,M}$

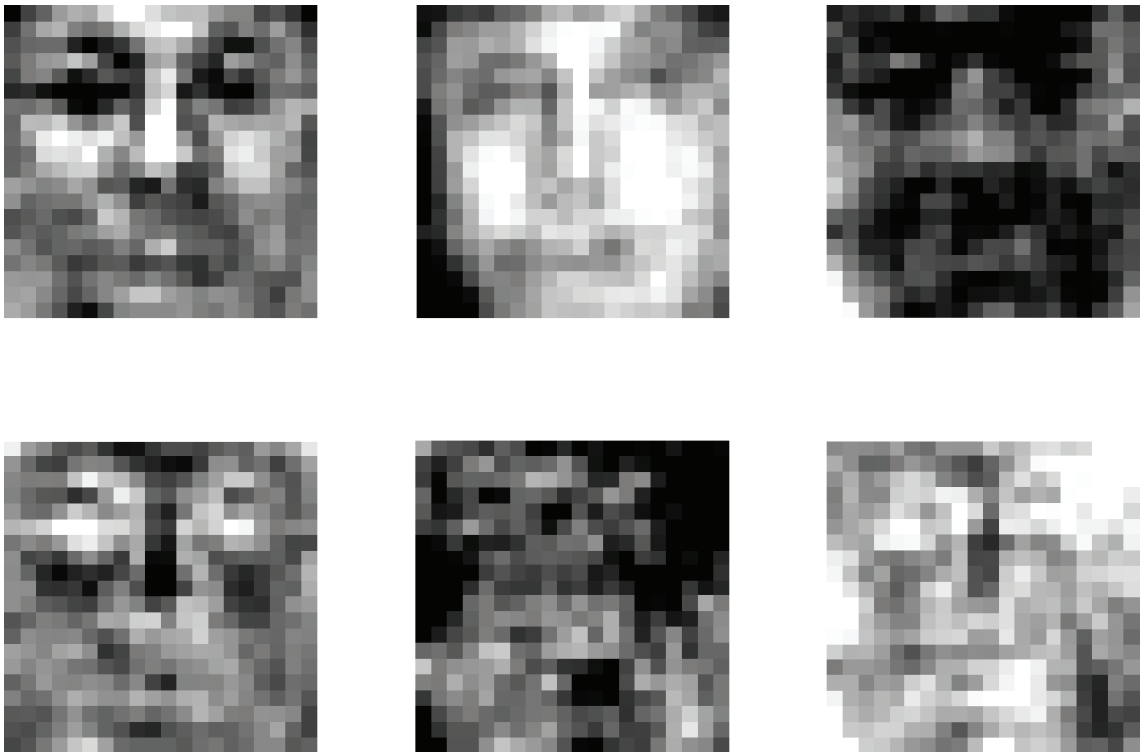

Abbildung 4.2: Beispiele für reduzierte Vektoren, oben: Gesichter, unten: nicht-Gesichter

4.3 Klassifizierung

Wie bereits im Abschnitt 4.1 beschrieben, wird für die Klassifizierung eine Kaskade verwendet. Die Berechnung des Restes in jeder Kaskadenstufe wäre aber zu zeitaufwändig. Daher wird stattdessen $\sum_{i=m+1}^{N_z} \min(\beta_i^*, 0)$ bzw. $\sum_{i=m+1}^{N_z} \max(\beta_i^*, 0)$ mit $m =$ Kaskadenstufe berechnet. Denn es gilt:

$$\langle \Phi(z_i), \Phi(x) \rangle = K(z_i, x) = \exp\left(- \underbrace{\gamma \|z_i - x\|^2}_{\geq 0} \right) \leq 1$$

und somit:

$$\begin{aligned} \sum_{i=1}^{m} \beta_i^* \langle \Phi(z_i), \Phi(x) \rangle \quad + \underbrace{\sum_{i=m+1}^{N_z} \beta_i^* \langle \Phi(z_i), \Phi(x) \rangle}_{} \quad + \quad b \quad \lessgtr \quad 0 \\ \leq \sum_{i=m+1}^{N_z} \max(\beta_i^*, 0) \quad \leq \sum_{i=m+1}^{N_z} |\beta_i^*| \\ \geq \sum_{i=m+1}^{N_z} \min(\beta_i^*, 0) \quad \geq - \sum_{i=m+1}^{N_z} |\beta_i^*| \end{aligned}$$

Der Vorteil von $\sum_{i=m+1}^{N_z} \min(\beta_i^*, 0)$ bzw. $\sum_{i=m+1}^{N_z} \max(\beta_i^*, 0)$ gegenüber $\sum_{i=m+1}^{N_z} |\beta_i^*|$ bzw. $-\sum_{i=m+1}^{N_z} |\beta_i^*|$ ist, dass nur Gewichte von Vektoren berücksichtigt werden, die das Vorzeichen der Signum-Funktion ändern könnten (siehe Abbildung 4.5). Zu untersuchende Patches, die weit von der Hyperebene entfernt sind, können somit mit wenigen reduzierten Vektoren klassifiziert werden. Bei gegebenen β^* lässt sich $\sum_{i=m+1}^{N_z} \min(\beta_i^*, 0) + b$ bzw. $\sum_{i=m+1}^{N_z} \max(\beta_i^*, 0) + b$ für jede Stufe der Kaskade bereits vor der Klassifizierung berechnen und speichern. Die Berechnung von (4.2) für ein Patch ist im Algorithmus 3 dargestellt.

Vor der Klassifizierung wird eine Datei geladen, die folgende Werte enthält:

- RV - Matrix der reduzierten Vektoren $\widehat{z}$
- w - Vektor β^*
- b - Bias
- γ - Skalierungsfaktor für den Gauß-Kern
- b-min - $\left(\sum_{i=2}^{N_z} \min(\beta_i^*, 0) + b \, , \, \ldots \, , \, \sum_{i=N_z}^{N_z} \min(\beta_i^*, 0) + b \right)$
- b-max - $\left(\sum_{i=2}^{N_z} \max(\beta_i^*, 0) + b \, , \, \ldots \, , \, \sum_{i=N_z}^{N_z} \max(\beta_i^*, 0) + b \right)$

Algorithmus 3 Klassifizierung mit RVM

$f = 0$
for $m = 1, \ldots, N_z - 1$ **do**
  $f = f + \beta_m^* \exp\left(-\gamma \|z_m - x\|^2\right)$
  if $f > 0$ AND $f - \sum_{i=m+1}^{N_z} \min(\beta_i^*, 0) + b > 0$ **then**
    break
  else if $f < 0$ AND $f + \sum_{i=m+1}^{N_z} \max(\beta_i^*, 0) + b < 0$ **then**
    break
  end if
  if $m == N_z - 1$ **then**
    $f = f + \beta_m^* \exp\left(-\gamma \|z_m - x\|^2\right) + b$
  end if
end for
$f = sgn(f)$

4.4 Tests und Ergebnisse

Die Tests fanden unter den gleichen Bedingungen wie in Abschnitt 3.4 statt. Zunächst wurden die reduzierten Vektoren ermittelt. Die Abbildung 4.3 zeigt den Verlauf der Zielfunktion 4.1 mit individuell berechnetem β (RVM_{ind}) und mit dem optimalen β^* (RVM_{opt}). Als Startvektor für die `fminsearch`-Funktion kam der Support-Vektor Nummer 78 zum Einsatz, da er vor allem bei den ersten Vektoren die beste Näherung liefert. Die Berechnung eines RV kann unter Umständen Tage dauern, daher wurde die Anzahl der Iteration der Funktion auf 10^7 begrenzt. Es ist deutlich zu erkennen, dass die ersten Vektoren den größten Einfluß auf die zu minimierende Zielfunktion haben. Ab ca. 20 Vektoren geht die Minimierung nur noch in kleinen Schritten voran, sowohl für die RVM mit individuellem β als auch für die RVM mit dem optimalen β^*. Obwohl die Näherung von RVM_{opt} ab dem 18. Vektor schlechter ist, hat dies keine negative Auswirkung auf die Genauigkeit der Klassifikation (siehe Tabelle 4.1). Aufgrund der guten Näherung der ersten Vektoren pendelt sich die Genauigkeit schnell bei 99% ein. Bei der RVM mit individuellem β hingegen treten bei den ersten Vektoren größere Schwankungen bei der Genauigkeit auf. Bei beiden Verfahren erzielt der erste Vektor nur eine geringe Genauigkeit. Die meisten Patches werden als Gesichter klassifiziert (siehe Flaschalarmrate), daher auch die hohe Erkennungsrate. Aber schon beim nächsten Vektor steigt die Genauigkeit bei beiden RVMn. Ein Großteil, der vom ersten Vektor falsch er-

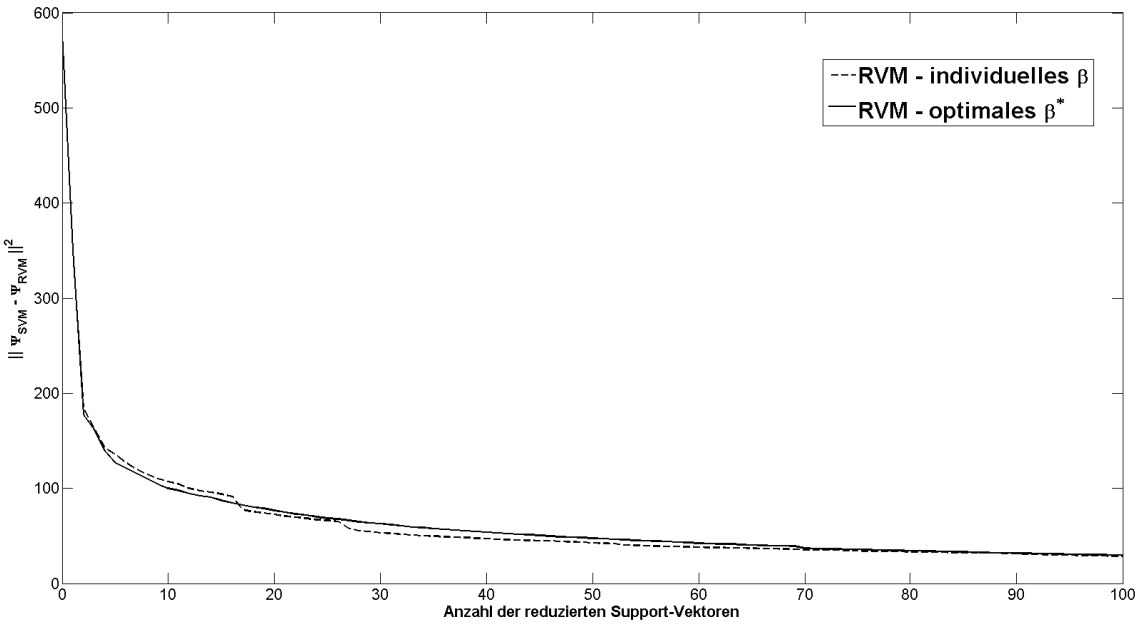

Abbildung 4.3: Güte der Approximation $||\Psi_{SVM} - \Psi_{RVM}||$ des SVM-Normalenvektors

kannt wurde, wird jetzt richtig klassifiziert. Bei beiden RVMn reichen fast 10 Vektoren aus, um die Genauigkeit und Erkennungsrate der SVM (Genauigkeit = 0.990 und Erkennungsrate = 0.992) zu erreichen. Bei 20 Vektoren und mehr erhöht sich sogar teilweise die Genauigkeit und die Erkennungsrate. Für die Klassifizeriung wurde deutlich weniger Zeit benötigt als mit der SVM (Zeit = ca. 36 s). Zum Vergleich wurde z nach dem Verfahren von Romdhani et. al. [11] berechnet. Bei diesem Verfahren wird z_i durch Lösen von

$$z_i = \frac{\sum_{j=1}^{M} \alpha_j \exp\left(-\gamma \left\|x_j - z_i\right\|^2\right) x_j}{\sum_{j=1}^{M} \alpha_j \exp\left(-\gamma \left\|x_j - z_i\right\|^2\right)}$$

mittels Fixpunkt-Iteration bestimmt. β wird auf die gleiche Weise wie in 4.7 bestimmt. Das Ergebnis der Näherung ist in der Abbildung 4.4 dargestellt. Mit dem Verfahren von Romdhani wurde der Normalenvektor der SVM deutlich besser angenähert. Dies zeigt sich auch in der Genauigkeit und Erkennungsrate, siehe Tabelle 4.2. Gegenüber den vorhergehenden RVMn reichen weniger Vektoren für ein gutes Ergebnis. Das einfache Simplex-Verfahren der `fminsearch`-Funktion liefert somit kein optimales Ergebnis trotz langer Berechnungszeit. Da die Ergebnisse der `fminsearch`-Funktion vom Startvektor abhängen, wurde versucht, diesen zu verbessern. Dazu wurde jeweils der Startvektor nach dem Prinzip von Romdhani berechnet und der Funktion übergeben. Das Ergebnis ist in der Abbildung 4.4 dargestellt. Gegenüber der RVM mit optimalem β^* bringt diese Variante keine größeren Vorteile trotz besserer Startvektoren. Bei den Anfangsvektoren ist die

N_z	Zeit [s]	Genauigkeit		Erkennungsrate		Falschalarmrate	
		RVM_{ind}	RVM_{opt}	RVM_{ind}	RVM_{opt}	RVM_{ind}	RVM_{opt}
1	0,155	0,307	0,307	1,000	1,000	0,802	0,802
2	0,286	0,888	0,972	0,180	0,801	0,000	0,001
3	0,424	0,985	0,985	0,935	0,904	0,008	0,002
4	0,563	0,702	0,987	0,998	0,983	0,344	0,012
5	0,722	0,990	0,985	0,980	0,995	0,008	0,017
10	1,545	0,986	0,990	0,991	0,991	0,015	0,011
20	2,801	0,992	0,990	0,991	0,993	0,008	0,011
30	4,221	0,992	0,990	0,987	0,993	0,007	0,011
50	7,013	0,992	0,990	0,994	0,993	0,013	0,011
80	11,186	0,992	0,990	0,991	0,992	0,008	0,011
100	14,050	0,990	0,990	0,994	0,992	0,012	0,010

Tabelle 4.1: Ergebnisse der Klassifizierung mittels RVM mit individuellem β und optimalem β^*

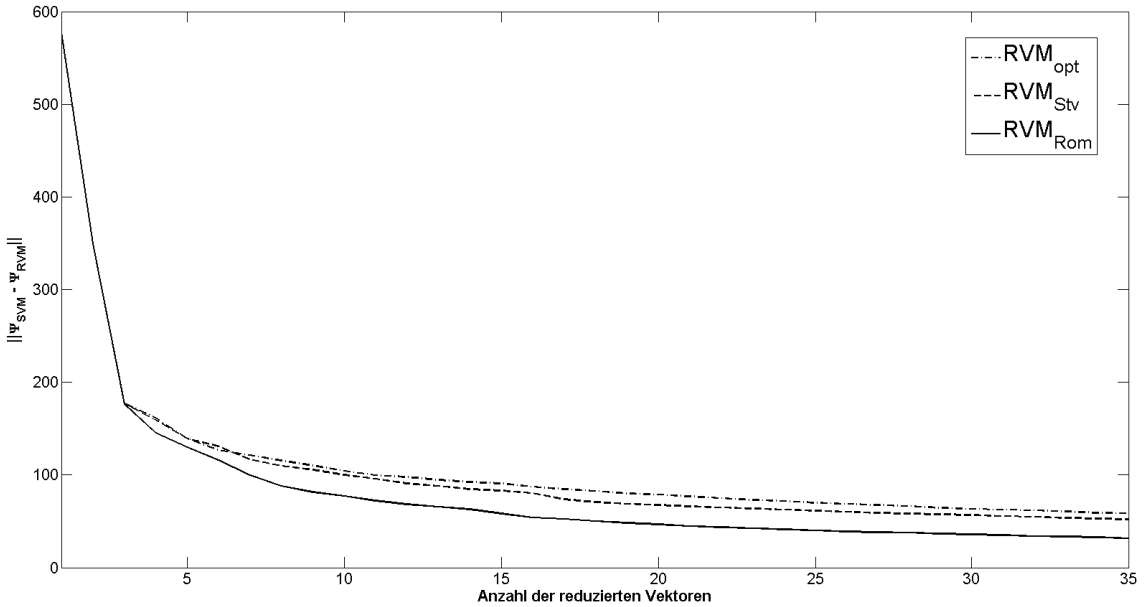

Abbildung 4.4: Vergleich weiterer RVMn

N_z	**Genauigkeit**		**Erkennungsrate**		**Falschalarmrate**	
	RVM_{Rom}	RVM_{Stv}	RVM_{Rom}	RVM_{Stv}	RVM_{Rom}	RVM_{Stv}
1	0,307	0,864	1,000	0,000	0,803	0,000
2	0,961	0,258	0,715	1,000	0,004	0,859
3	0,975	0,903	0,989	0,289	0,027	0,001
4	0,990	0,952	0,962	0,650	0,006	0,001
5	0,975	0,974	0,985	0,960	0,023	0,001
10	0,990	0,913	0,989	0,997	0,010	0,100
15	0,991	0,957	0,992	0,998	0,009	0,050
20	0,990	0,890	0,992	0,998	0,011	0,127
25	0,990	0,992	0,993	0,986	0,011	0,007
30	0,990	0,994	0,993	0,977	0,011	0,003

Tabelle 4.2: Ergebnisse der Klassifizierung mit der RVM nach Romdhani RVM_{Rom} und mit verbessertem Startvektor RVM_{Stv}

N_z	**Stufen**				**Zeit** [s]			
	RVM_{ind}	RVM_{opt}	RVM_{Rom}	RVM_{Stv}	RVM_{ind}	RVM_{opt}	RVM_{Rom}	RVM_{Stv}
5	4,8	4,9	5,0	4,9	1,346	1,359	1,410	1,344
10	9,8	9,8	9,9	9,9	2,410	2,354	2,399	2,367
15	14,7	14,6	14,5	14,9	3,201	3,347	3,293	3,440
20	19,6	19,4	19,8	19,6	4,261	4,352	4,456	4,470
25	24,6	24,2	24,5	24,3	5,234	5,375	5,421	5,375
30	29,6	29,2	28,9	28,9	6,315	6,392	6,2683	6,315
35	34,3	34,0	34,3	33,9	7,3102	7,227	7,379	7,239

Tabelle 4.3: durchschnittliche Anzahl der durchlaufenen Stufen der Kaskade und die dafür benötigte Zeit

N_z	Genauigkeit			
	RVM_{ind}	RVM_{opt}	RVM_{Rom}	RVM_{Stv}
5	0,627	0,986	0,608	0,969
10	0,993	0,986	0,990	0,992
15	0,983	0,988	0,934	0,994
20	0,977	0,991	0,992	0,983
25	0,993	0,988	0,993	0,992
30	0,988	0,990	0,993	0,992
35	0,990	0,986	0,979	0,990

Tabelle 4.4: Genauigkeit mit RVM + Kaskade

N_z	Erkennungsrate			
	RVM_{ind}	RVM_{opt}	RVM_{Rom}	RVM_{Stv}
5	0,999	0,988	0,680	0,771
10	0,979	0,988	0,977	0,991
15	0,996	0,995	0,995	0,977
20	0,996	0,991	0,989	0,995
25	0,990	0,993	0,981	0,990
30	0,993	0,993	0,990	0,992
35	0,993	0,994	0,997	0,993

Tabelle 4.5: Erkennungsrate mit RVM + Kaskade

N_z	Falschalarmrate			
	RVM_{ind}	RVM_{opt}	RVM_{Rom}	RVM_{Stv}
5	0,431	0,014	0,403	0,004
10	0,005	0,015	0,014	0,008
15	0,019	0,013	0,018	0,004
20	0,025	0,010	0,007	0,012
25	0,007	0,012	0,005	0,008
30	0,012	0,012	0,007	0,008
35	0,011	0,012	0,023	0,011

Tabelle 4.6: Falschalarmrate mit RVM + Kaskade

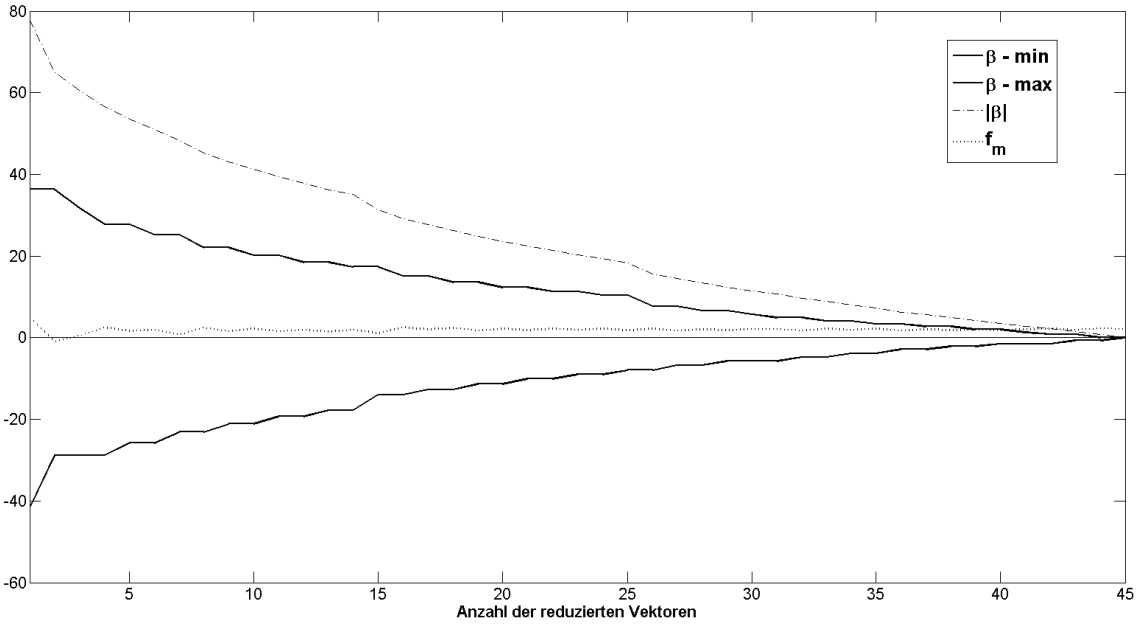

Abbildung 4.5: Verlauf von β - min und β - max im Vergleich zu $|\beta|$, am Beispiel einer Klassifizierung eines Gesichtes

Näherung sogar teilweise schlechter. Auch die Genauigkeit, Erkennungs- und Falschalarmrate konnten nicht weiter verbessert werden.

Als nächstes wurde die Kaskade mit den vier RVMn getestet. Die Ergebnisse sind in den Tabellen 4.3 bis 4.6 dargestellt. Zeitlich gesehen bringt die Kaskade keinen weiteren Vorteil eher im Gegenteil. Das liegt zum einen daran, dass fast alle reduzierten Vektoren durchlaufen werden, bevor die Kaskade abbricht. Der Grund hierfür ist, dass die β-min bzw. β-max gegenüber dem Funktionswert f fast bis zum Schluß um einiges größer sind (siehe Abbildung 4.5). Zum anderen kann durch das späte Abbrechen der Kaskade die benötigte Zeit für die *if*-Abfragen der Kaskade nicht kompensiert werden. Die Genauigkeit, Erkennungs- und Falschalarmrate haben sich gegenüber der Berechnung mit der RVM ohne Kaskade nicht verändert, da durchschnittlich fast alle Vektoren für die Klassifizierung verwendet wurden.

Der in Abbildung 4.5 dargestellte Verlauf von β - min und β - max ist in etwa für jede Anzahl von reduzierten Vektoren gleich. Auch bei der Nutzung von 100 Vektoren nähern sich die beiden Kurven erst bei ca. 95 Vektoren der Funktionskurve f an. Die Funktionskurve und die beiden anderen Kurven nähern sich nur langsam an, da β_i nur langsam gegen Null streben (siehe Abbildung 4.6). Für eine schnelle Kaskade müsste der Verlauf schneller sein.

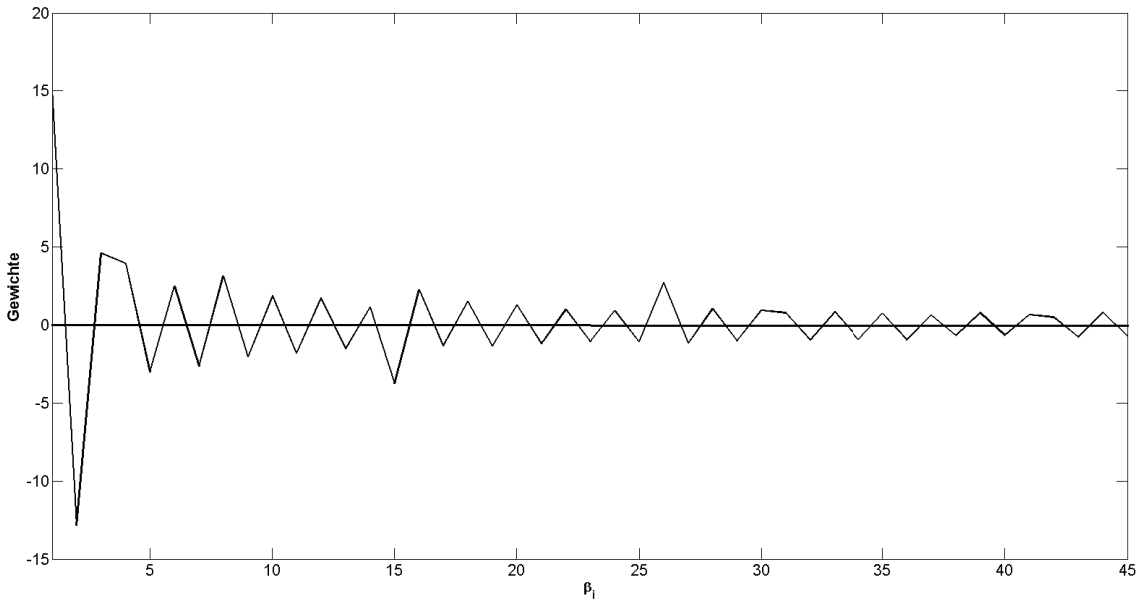

Abbildung 4.6: Verlauf von β_i am Beispiel von RVM$_{opt}$

Mit dem in diesem Kapitel beschriebenen Verfahren der RVM konnten gute Ergebnisse erzielt werden. Gegenüber der Original-SVM konnte die Berechnungszeit um ein 12-faches verringert werden bei gleichbleibender Genauigkeit. Tests haben aber gezeigt, dass sich die reduzierten Vektoren mit dem Verfahren von Romdhani schneller berechnen lassen und ein etwas besseres Ergebnis liefern. Die verwendete Kaskade hat keine zusätzliche Beschleunigung gebracht.

Kapitel 5

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden zwei Verfahren zur Beschleunigung der Klassifikationsphase einer Support-Vektor-Maschine (SVM) vorgestellt und auf einer vorgegebenen SVM angewandt. Bei dem Verfahren Wavelet-Support-Vektor-Maschine(W-SVM) wird eine Beschleunigung durch die Reduzierung der Anzahl der durchzuführenden Rechenoperationen erreicht. Zum einen lassen sich einige Terme bereits vor der Klassifizierung berechnen und speichern und zum anderen werden die Vorteile eines Integralbildes ausgenutzt. Mit Hilfe eines Integralbildes lässt sich die Summe aller Grauwerte des Eingabebildes innerhalb eines beliebigen Rechtecks in konstanter Zeit berechnen. Pro Rechteck sind jeweils vier Rechenoperationen erforderlich. Für die Anwendung dieser Methode müssen die Support-Vektoren eine blockartige Struktur aufweisen. Dazu werden die Support-Vektoren mit einer Wavelet-Shrinkage-Funktion bearbeitet. Die so entstandenen Rechtecke werden in einer Tabelle aufgelistet und gespeichert. Das Verfahren wurde in MATLAB implementiert und mit einer kleinen SVM getestet. Die Rechenzeit für die Klassifizierung konnte deutlich gesenkt werden bei gleichbleibender bzw. teilweise verbesserter Genauigkeit.

Eine andere Möglichkeit für die beschleunigte Berechnung der Entscheidungsfunktion ist die Anwendung der reduzierten Support-Vektor-Maschine(RVM). Die Dauer der Klassifikation ist von der Anzahl der Support-Vektoren abhängig. Bei dieser Methode wird eine Näherung an die Original-SVM bestimmt, die aus viel weniger Vektoren besteht. Die durchgeführten Tests in MATLAB belegen, dass durch die Reduzierung der Anzahl der Vektoren eine Reduzierung der Rechenzeit erreicht wurde bei gleichbleibender Genauigkeit. Im Vergleich zum Verfahren von Romdhani et al. [11] liefert die `fminsearch`-Funktion keine optimalen Vektoren für die Näherung. Eine weitere Beschleunigung sollte durch die Verwendung einer Kaskade erzielt werden. Dies wurde

jedoch nicht erreicht, da die berechneten Gewichte β_i für die Kaskadenstruktur ungeeignet sind. Die Kaskade wurde fast bei jedem Patch vollständig durchlaufen.

Der Schwerpunkt dieser Arbeit lag auf der Reduzierung der Rechenzeit bei der Gesichtserkennung bei mindestens gleichbleibender Genauigkeit. Mit beiden Verfahren wurde dieses Ziel erreicht. Die Rechenzeit konnte bei der W-SVM um das 16-fache verringert werden und bei der reduzierten SVM um das 12-fache. Die Berechnung der W-SVM und RVM erfolgte jeweils automatisiert und die Parameter, wie zum Beispiel Bias und Kern-Funktion, wurden aus der vorgegebenen SVM übernommen.

Um eine weitere Beschleunigung der Rechenzeiten zu erzielen, wäre zu untersuchen, ob eine Kombination von beiden vorgestellten Verfahren die Rechenzeit verringert. Hierzu wird zunächst eine Näherung an die Original-SVM bestimmt. Im nächsten Schritt erfolgt die Transformation der reduzierten Vektoren zu den Wavelet-Vektoren. Anschließend werden die Rechtecke in einer Tabelle aufgelistet und gespeichert. Die Berechnung der Klassifizierung verläuft dann wie bei der W-SVM. Da bei der W-SVM die Parameter der RVM übernommen werden, ist von der Verwendung einer Kaskade abzusehen bzw. ist die Kaskadenstruktur zu überdenken.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[2] F . Girosi E. Osuna, R. Freund. Support vector machines: Training and applications. *Massachusetts Institute of Technology*, 1996.

[3] Tatjana Eitrich. Support-vektor-maschinen und ihre anwendung auf datensätze aus der forschung.

[4] Bernhard Schölkopf et al. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5), 1999.

[5] M. Vingron F. Markowetz, L . Edler. Support vector machines for protein fold class prediction. *Biometrical Journal*, 2003.

[6] Roger Fletcher. *Practical Methods of Optimization*. 2. Aufl. John Wiley & Sons, Inc., 1987.

[7] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Proc. of thé European Conférence on Machine Learning (ECML)*, 1998.

[8] Paul Viola & Michael Jones. Rapid object detection using a boosted cascade of simple features. *Accepted Conference on Computer Vision and Pattern Recognition*, 2001.

[9] Colin Campbell Kristin P. Bennett˙ Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, 2:1–13, 2000.

[10] S. Romdhani & T. Vetter M. Rätsch, G. Teschke. Wavelet frame accelerated reduced support vector machines. *IEEE Transactions on Image Processing*, 17(12):2456–2464, 2008.

[11] B. Schölkopf & A. Blake S. Romdhani P. Torr˙ Computationally efficient face detection. *Proceedings of the 8th International Conference on Computer Vision*, 2001.

[12] Carl Taswell. The what, how, and why of wavelet shrinkage denoising. *Computing in Science & Engineering*, 2:12–19, 2000.

[13] Vapnik und Chervonenkis. Theory of pattern recognition. 1974.