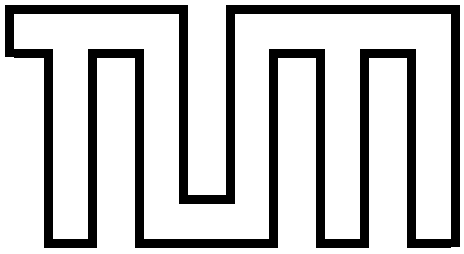

Technische Universität München

Fakultät für Informatik

Forschungs- und Lehreinheit Informatik IX

Support Vector Machines

Hauptseminar

Robert Rackl

Betreuer: Dipl.-Inform. Simone Hämmerle

Abgabetermin: 8. Juli .2004

Inhaltsverzeichnis

Kapitel 1

Einleitung - Was sind Support Vector Machines

Immer wieder liest man Aussagen wie "Comupter sind intelligent", "Computer können lernen" und fast jeder kennt den Begriff "Künstliche Intelligenz". Aber können Comuter wirklich denken? Nun, sicherlich nicht so gut wie wir Menschen. Doch Computer schlagen sich ziemlich gut darin, in eng begrenzten Wissensgebieten zu lernen. Automatische Beweiser können aus bekanntem Wissen Folgerungen erschließen und so neue wahre Aussagen formen. Im Roboter Fußball (besonders in der Small-League), werden die ersten Gehversuche unternommen, in Echtzeit Strategien zu planen und durchzuführen. Ein weiteres Beispiel für lernende Programme sind die Bayes-Filter um Spam E-Mails automatsich zu erkennen. In vielen Bereichen kann man große Fortschritte im Lernen von Computern beobachten.

Wie wir ja auch bereits in einigen der vorangegangenen Vorträge gehört haben, ist es sehr oft nötig in digitalen Bildern gewisse Merkmale zu erkennen, wie zum Beispiel Ecken, Kanten und Flächen. Oder es werden Übereinstimmungen in Paaren von Bildern gesucht um dann daraus z.B. die Verschiebung zwischen zwei Aufnahmen des Selben Motivs zu berechnen. Leider ist es nach wie vor meistens noch nötig, diese sogenannten point correspondancesper Maus Klick einzugeben. Einer sehr langwierige Arbeit. Diesen Prozess zu automatisieren ist ein großes Ziel in der Bildverarbeitung.

Ein sehr bekannter Vertreter im Bereich Computerlernen sind die Neuronalen Netze. Doch diese Modelle sind theoretisch sehr schwer zu erfassen und und mathematisch komplex. Hier kommen die Support Vector Machines (kurz SVMs) ins Spiel. Eine SVM kann auch 'lernen' und dann Aussagen treffen, doch ist das theoretische Modell, welches dahintersteht wesentlich simpler. Durch ein paar mathematische Künststücke, die ich im Folgenden Vorstellen werde, bleiben SVMs leicht zu implementieren.

Kapitel 2

Support Vector Machines

2.1 Lernen Anhand von Beispielen

An eine Support Vector Machine wird folgende Aufgabe gestellt. Gegeben sei eine Menge von zufällig verteilten Vektoren in einem n-Dimensionalen Raum. Jeder dieser Vektoren hat eine binäre Eigenschaft. Entweder er gehört zu einer Klasse oder eben nicht.

$$\{\vec{x}_i, y_i\} \quad \text{mit} \quad i = 1, \ldots, l \; \vec{x}_i \in \mathbb{R}^n, \; y_i \in \{+1, -1\} \tag{2.1}$$

Gesucht ist nun eine Funktion, die jedem dieser Vektoren seine Eigenschaft, im Folgenden als +1 und -1 dargestellt, zuordnet.

$$f : \mathbb{R}^n \to \{+1, -1\} \tag{2.2}$$
$$f(\vec{x}_i) = y_i \tag{2.3}$$

Zusätzlich soll die Funktion f nun auch noch die Eigenschaft haben, dass sie neue Punkte, die mit der gleichen Wahrscheinlichkeitsverteilung wie die Eingabedaten erstellt bzw. gemessen wurden, ebenfalls so gut wie möglich der korrekten Klasse zuordnet. Das heißt also, die Funktion soll die beiden Klassen nicht nur trennen, sondern etwas Bildlich gesprochen, diese auch möglichst gut "in der Mitte" zerteilen.

2.2 Die VC-Dimension

Um eine derartige Funktion zu finden, kann man sich vorher noch eine Eigenschaft dieser Funktionen ansehen. Bei l Punkten in der Eingabemenge gibt es insgesamt 2^l Möglichkeiten jedem dieser Punkte eine Eigenschaft zuzuordnen. Man kann sich vorstellen jeder

Punkte wäre mit +1 oder -1 beschriftet. Angenommen, wir hätten eine Klasse von Funktionen gefunden, die diese Punkte für jede Beschriftungsmöglichkeit korrekt zerteilt, dann sagt man diese Klasse von Funktionen 'zerschlägt' (engl. "shatter") diese Punkte.

Die VC-Dimension ist nun definiert als die maximale Anzahl an Trainingspunkten die von einer gegebenen Klasse von Funktionen zerschlagen werden kann.

2.3 Fehlerminimierung

Wie wird nun das "Zerteilen in der Mitte" mathematisch formuliert. Wie kann eine Zerteilung besser oder schlechter sein als eine andere. Schnell wird klar, dass es nötig ist eine Fehlerfunktion einzuführen. Wir benötigen also eine Klasse von Funktionen bei denen es Möglich ist zu berechnen wie gut oder schlecht sie zerteilen. Für SVMs sind das die so genannten Hyperebenen:

$$(\vec{w} \cdot \vec{x}) + b = 0 \quad \text{mit} \quad \vec{w} \in \mathbb{R}^n, b \in \mathbb{R} \tag{2.4}$$

Zu welcher Klasse ein Punkt dann gehört erhält man durch die Entscheidungsfunktion

$$f(x) = sign((\vec{w} \cdot \vec{x}) + b) \tag{2.5}$$

Nun liegt es auf der Hand, dass die "beste" Hypereben, also diejenige Ebene, die die Eingabepunkte am besten teilt, diejenige mit dem größten Abstand zu den Punkten der beiden Klassen ist. Erstaunlich ist, dass es hierfür eine analoge und einfache geometrische Formulierung gibt. Die optimale Hyperebene ist die Mittelsenkrechte über der kürzesten Verbindung der konvexen Hüllen der beiden Punktklassen. Siehe Abbildung 2.1 auf Seite 5.

Durch die Skalierung der Gewichtungsvektoren gibt es auf jeder Seite der Hyperebene mindestens einen Punkt, der am nähesten an der teilenden Ebene liegt. Diese beiden (oder mehr) Punkte haben jeweils den Abstand $2/\|w\|$ lotrecht zur Ebene. Dieser Abstand liefert nun genau die gesuchte Kostenfunktion $\|w\|$ die es gilt zu minimieren - unter der Nebenbedinung

$$y_i \cdot ((\vec{w} \cdot \vec{x}_i) + b) - 1 \geq 0 \tag{2.6}$$

Die genannten nahen Punkte zur Hyperebene, also alle Punkte für die die Gleicheit in 2.6 erfüllt ist, heißen "Support Vektoren". Sie sind deswegen so wichtig, da nur sie allein die Lage der Hyperebene festlegen. Selbst wenn man alle anderen Eingabevektoren verschiebt (natürlich nicht über die beiden äußeren Hyperebenen hinüber) ändert sich nichts an der trennenden Hyperebene.

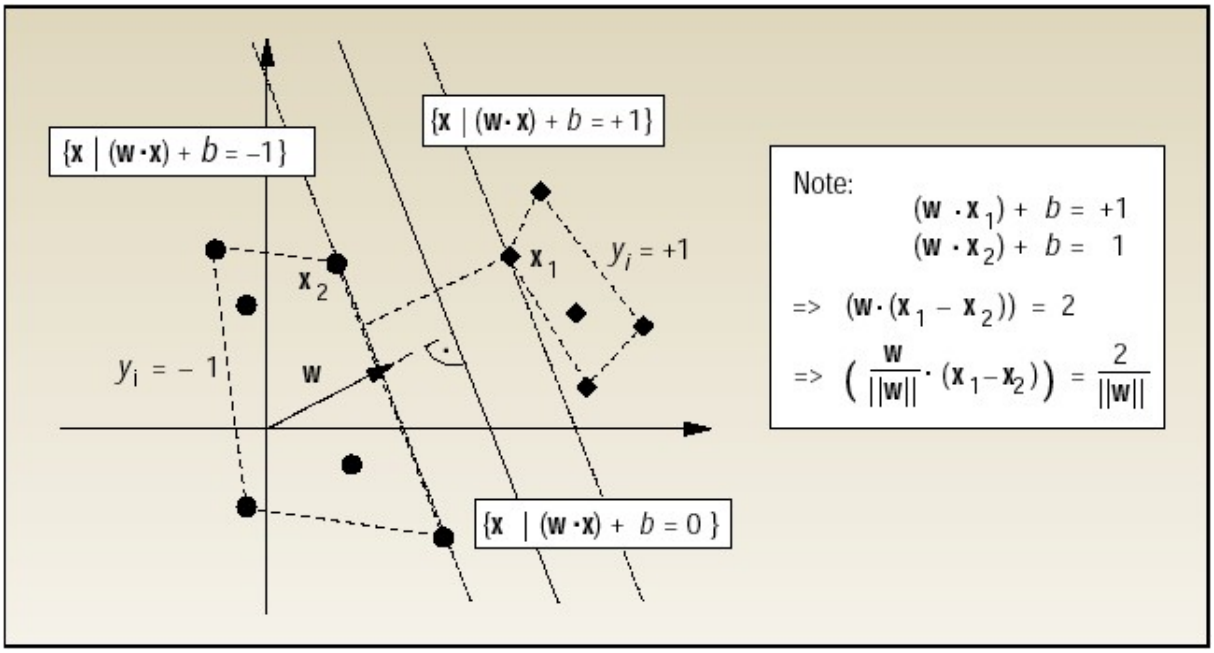

Abbildung 2.1: Die optimale Hyperebene. Es gibt einen Gewichtungsvektor $\vec{w}$ und eine Schwelle b, so dass für alle Punkte x_i gilt: $y_i \cdot ((\vec{w} \cdot \vec{x}) + b) > 0$ Wenn man nun $\vec{w}$ und b so skaliert, dass die Punkte am nähesten zur Hyperebene $|(\vec{w} \cdot \vec{x}) + b| = 1$ erfüllen, kann man die Hyperebene in der Form $y_i \cdot ((\vec{w} \cdot \vec{x}) + b) \geq 1$ schreiben. Quelle: [Hea98]

2.4 Lagrange Faktoren

Wie minimiert man eine Fehlerfunktion unter Nebenbedingungen? Hierzu kann man die Lagrange Faktoren zur Hilfe nehmen. Für jede der Ungleichheitsbedingungen in 2.6 führt man einen positiven Lagrange Faktor α_i ein, multipliziert die Bedingung mit diesem Faktor und subtrahiert das von der zu minimierenden Funktion: [1]

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{l} \alpha_i (y_i(\vec{w} \cdot \vec{x}_i + b) - 1) \tag{2.7}$$

Nun gilt es ein Minimum dieser Funktion bezogen auf $\vec{w}$ und b zu finden unter folgenden Bedingungen:

- Die Ableitungen von L bezüglich $\vec{w}$ und b an der gefundenen Stell müssen Null werden für alle α_i
- Die Lagrange Faktoren α_i bleiben alle größer gleich Null.

Aus der ersten Bedingung folgen die beiden Ableitungen

[1] Der Abstand der Support Vektoren soll möglichst groß sein, d.h.
maximiere $2/\|\vec{w}\| \Leftrightarrow$ minimiere $\|\vec{w}\| \Leftrightarrow$ minimiere $1/2\|\vec{w}\|^2$

$$\frac{\delta L}{\delta \vec{w}} = 0 \quad \Leftrightarrow \quad w = \sum_i \alpha_i y_i \vec{x}_i \tag{2.8}$$

$$\frac{\delta L}{\delta b} = 0 \quad \Leftrightarrow \quad 0 = \sum_i \alpha_i y_i \tag{2.9}$$

Diese in 2.7 eingesetzt ergibt

$$L = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\vec{x}_i \cdot \vec{x}_j) \tag{2.10}$$

Man stellt fest, dass die Eingabevektoren $\vec{x}$ in dieser Gleichung nur noch als Skalarprodukt vorkommen. Diese Eigenschaft lässt sich ausnützen.

2.5 Übergang von Weltkoordinaten in den Hilbertschen Raum

Ein hilbertscher Raum [Hil] ist ein (möglicher- aber nicht notwendigerweise unendlich dimensionaler) vollständiger und linearer Raum in dem ein inneres Produkt definiert ist (z.B. das Skalarprodukt). Wir betrachten nun eine Abbildung $\Phi : \mathbb{R}^n \rightarrow H$ die unsere Eingabevektoren in einen Hilbertraum überführen. Wie schon erwähnt ist es für die Berechnung der SVM lediglich nötig Skalarprodukte zwischen zwei Eingabevektoren auszüführen. Diese Berechnung kann nun aber auch in dem neuen Raum H durchgeführt werden. Da aber nun H sehr hochdimensional sein kann ist diese Berechnung sehr aufwendig. Gäbe es jedoch eine sogenannte Kernel Funktion k, so dass

$$k(\vec{x}, \vec{y}) = (\Phi(\vec{x}) \cdot \Phi(\vec{y})) \tag{2.11}$$

würde diese für die Berechnung des Lern-Algorithmuses genügen. Selbst wenn die Abbildung Φ gar nicht bekannt ist, kann die SVM allein durch Kenntnis dieses Kernels trainiert werden. Man muss also gar nicht wirklich eine Abbildung in den neuen Raum definieren, sondern es reicht schon sich ein Skalarprodukt im Hilbert-Raum festzulegen. Natürlich kann man in einfachen Fällen daraus dann auf die Abbildung zurück schließen, doch das ist wie schon gesagt gar nicht nötig. Durch den Übergang in einen geeigneten höherdimensionalen Feature Raum ist es dann möglich eine trennende Hyperebene zu finden wo es im Eingaberaum noch nicht möglich war. Siehe Abbildung 2.2 auf Seite 7.

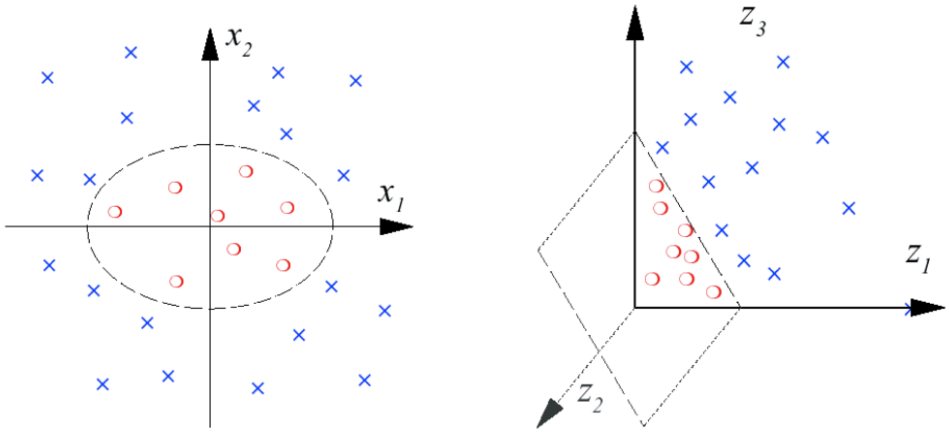

Abbildung 2.2: Ein Beispiel für einen zweidimensionalen Eingaberaum und zwei Klassen von Punkten. Im Eingaberaum kann keine trennende Hyperebene (keine 2D-Gerade!) gefunden werden. Durch den Übergang in einen dreidimensionalen Feature Raum mit Hilfe von Polynomen zweiter Ordnung kann dort dann eine trennende 3D-Hyperebene gefunden werden. Projeziert man diese Hyperebene zurück in den Eingaberaum erhält man eine nichtlineare Elipse als Entscheidungsgrenze. Quelle: MIT-Press Die Abbildungsfunktion Φ zum Übergang in den Feature Raum ist hier: $\Phi(\vec{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$

Radiale	$k(x,y) = exp(-\|x \cdot y\|^2/c)$
polynomielle Kernel	$k(x,y) = ((x \cdot y) + \sigma)^d$
Sigmoide Kernel	$k(x,y) = tanh(\kappa x \cdot y + \delta)$

Tabelle 2.1: In der Praxis verwendet man als Kernel oft diese Funktionen.

2.6 Testen von neuen Kandidaten

Nun ist unsere Support Vector Machine fertig und kann anfangen zu arbeiten. Um für einen neuen gegebenen Vektor $\vec{x}$ zu entscheiden zu welcher Klasse dieser gehört, muss das Skalarprodukt von $\vec{x}$ mit der Normalen auf unserer teilenden Hyperebene $\vec{w}$ ausgewertet werden. Dies alles jedoch projeziert im Hilbertraum H. Mit Hilfe von 2.8 erhält man

$$f(\vec{x}) = sign(\Phi(\vec{w}) \cdot \Phi(\vec{x}) + b) \tag{2.12}$$

$$\Leftrightarrow f(\vec{x}) = sign(\sum_{i=1}^{s} \alpha_i y_i \cdot k(\vec{s}_i, \vec{x}_i) + b) \tag{2.13}$$

Wobei s die Anzahl der Supportvektoren und die s_i eben diese Supportvektoren seien.

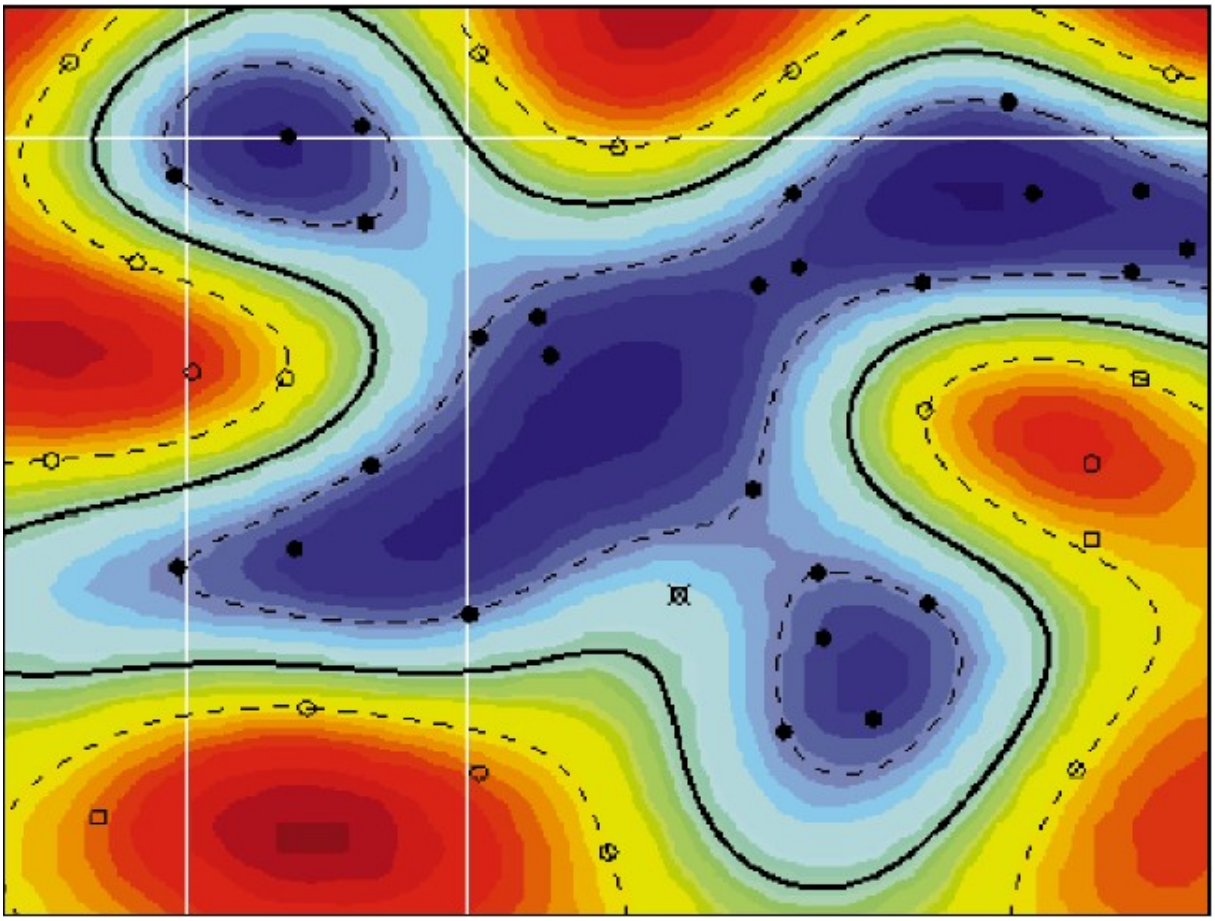

Abbildung 2.3: Hier sieht man sehr schön wie sich eine Radiale Basisfunktion als Abbildung in den Feature Raum auswirkt. Die Kreise und Punkte im Bild sind die zweidimensionalen Eingabevektoren für die SVM. Die durchgezogene schwarze Linie ist die errechnete Entscheidungsebene. Diese Ebene ist selbstverständlich nur im Feature Raum eine flache Ebene. Das Bild zeigt jedoch den Eingaberaum. So sieht man also das durch die Umkehrabbildung zurück projizierte Bild der Hyperebene. Die vom Algorithmus gefundenen Support Vektoren liegen auf den gestrichelten Linien. Quelle: [Hea98]

Kapitel 3

Anwendungen

Support Vector Machines finden in einem breiten Spektrum von Anwengungsfällen Verwendung. So zum Beispiel in der Schrifterkennung, in der Dedektierung von Gesichtern und in der automatischen Kategorisierung von Texten. In vielen Bereichen finden SVMs sogar schneller optimalere Lösungen als bisher angewandte Techniken.

3.1 Gesichtserkennung

Nur um Mißverständnissen vorzubeugen: Es geht hier nicht um die Zuordnung von Gesichtern zu Namen, sondern erst einmal nur um die Frage: Wo in einem Bild befindet sich ein Gesicht? Hierzu wird wie folgt vorgegangen. Gegeben ist ein schwarz-weiß Foto in dem Gesichter vorkommen. Gesucht ist die genaue Postion dieser Gesichter im Bild. Außerdem sollen auch noch Gesichtszüge in unterschiedlicher Größe erkannt werden. Je nachdem ob sich die betreffende Person zum Zeitpunkt der Aufnahme nahe oder fern der Kamera befand. Aus dem Eingabebild werden nun kleine quadratische Teilbilder ausgeschnitten, zum Beispiel 20x20 Pixel groß. Und zwar alle Teilbilder, oder zumindest in einer gewissen (möglichst kleinen) Schrittweite alle, d.h. also diese einzelnen Quadrate überschneiden sich untereinander. Dann wird das Eingabebild skaliert und der Vorgang wiederholt. Man erhält eine ganze Menge an Daten. Jedes dieser 20x20 Pixel großen Quadrate ist nun ein Eingabepunkt (in einem 400 dimensionalen Raum) für unsere Support Vector Machine.

Selbstverständlich muss die SVM nun erst trainiert werden. Dafür müssen von Hand einige bzw. besser gesagt möglichst viele dieser Eingabepunkte ausgewählt werden und als Gesicht bzw. kein Gesicht klassifiziert werden. Die SVM wird sehr gut funktionieren, wenn alle Gesichts-Quadrate im Eingaberaum nahe zusammen liegen. Diese Annahme ist durchaus sinnvoll, denn Gesichter sehen ja untereinander ähnlich aus. In jedem Gesicht kommen zwei Augen eine Nase und ein Mund vor und das auch noch in mindestens ähnlicher Anordnung. Ich betone noch einmal, dass die Eingabepunkte, welche als Gesichter klassifiziert werden noch lange nicht durch eine Hyperebene vom Rest trennbar sind. Mo-

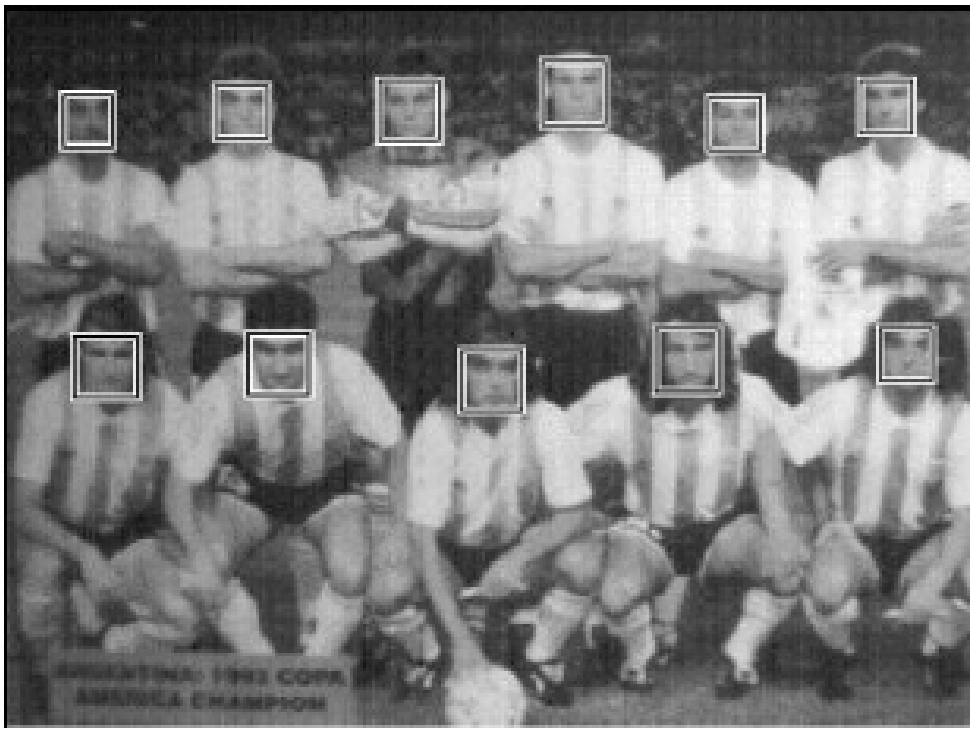

Abbildung 3.1: Eine SVM erkennt die Position der Gesichter einer Fußballmannschaft.

mentan sind wir noch im Eingaberaum. Erst durch einen geeigneten Übergang in den Feature Raum werden sich die beiden Klassen dann (hoffentlich) durch eine Hyperebene trennen lassen.

Nachdem die SVM dann einmal trainiert ist, kann sie nun auch in neuen Bildern Gesichter erkennen.

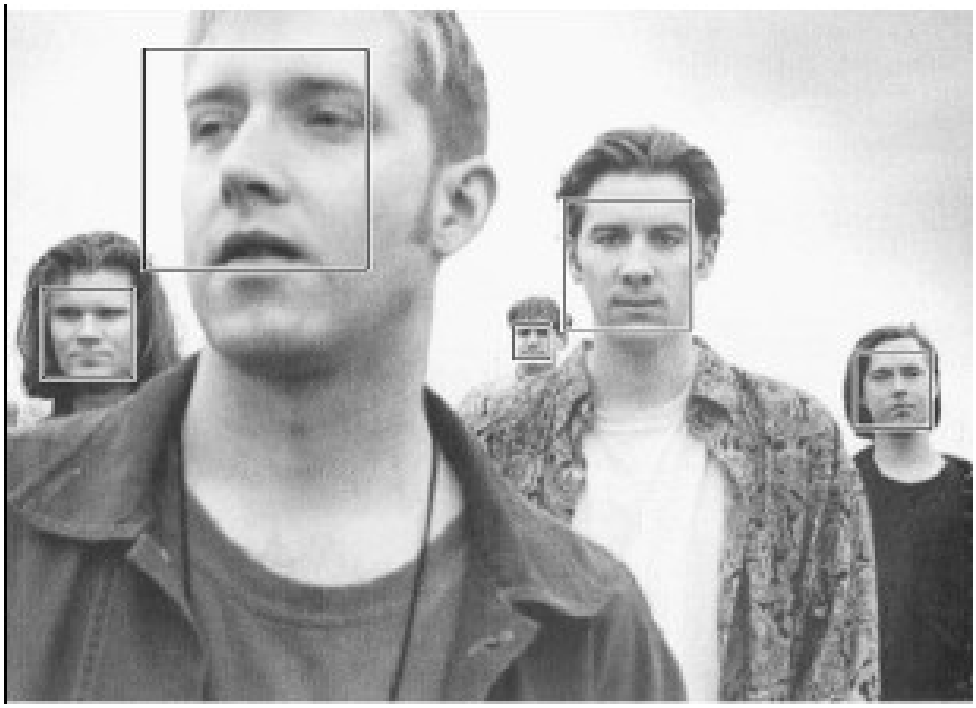

Abbildung 3.2: Ein Beispiel mit unterschiedlich großen Gesichtern.

Abbildung 3.3: Ein wunderschönes Beispiel anhand eines sehr komplexen Bildes. Es ist bemerkenswert wie viele Gesichter noch erkannt werden. Selbst das menschliche Auge könnte kaum besser schließen. Erwähnenswert ist jedoch noch die Person halbrechts oben, der eine Hand vor seinem Gesicht hält. Die SVM versagt, doch das menschliche Auge erkennt dort klar ein Gesicht.

Literaturverzeichnis

[Bur98] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Hea98] Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4), 1998.

[Hil] Hilbert-raum. http://wikipedia.de/wiki/Hilbertraum.

[MRTS01] K.-R. Müller, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Intelligent Systems*, 12(2), May 2001.

[SVM] Die seite über support vector machines in der englischen wikipedia. http://en.wikipedia.org/wiki/Support_vector_machine.

Index