Thema: Support Vector Machines

Johannes Lächele

Zusammenfassung: *In dieser Seminararbeit wird das Basiswissen für das Verständnis von Support Vector Machines oder SVM vermittelt. Nach einer kurzen Motivation für Lernverfahren im Allgemeinen, wird zuerst die Grundidee der Klassifikation und des Lernens erläutert. Danach folgt eine Einführung in kernelbasiertes Lernen, sowie einige Anwendungsbeispiele.*

1 Motivation

In der Natur existieren zahlreiche Fälle, in denen ein Problem, das es zu lösen gilt, genau beschrieben und anschliessend berechnet werden kann. So lässt sich beispielsweise genau berechnen, wie sich Wachstumsprozesse in der Natur verhalten, oder wie sich Planeten auf ihrer Umlaufbahn bewegen. Allerdings gibt es Situationen, in denen eine solche genaue Mathematische Regelmäßigkeit mit hohem Rechenaufwand verbunden, nicht bekannt ist, oder im schlimmsten Fall nicht existiert. Dazu gehören z. B. die Handschrifterkennung, Simulation chemischer Prozesse und Bilderkennung.

Prinzipiell scheint es aber wohl doch möglich zu sein, mit Hilfe lernender Systeme zumindest einge dieser Probleme zu lösen, schliesslich kann der Mensch mit Beispielen und Übung Schreiben und Lesen lernen.

Seit Anbruch des Industriezeitalters und erst recht seit den Anfängen des Computers, sucht der Mensch nach einer Maschine, die „sebstständig Denkt” , d. h. auf Umwelteinflüsse intelligent reagiert und lernt, Fehler zu vermeiden. Die Vorstellung, nicht nur körperliche, sondern auch geistige Arbeit von Maschinen übernehmen zu lassen, treibt Forscher so stark an, dass zuweilen die Objektivität verloren geht. Nichts desto weniger existieren heute bereits Ansätze, die für bestimmte Probleme annehmbare Lösungen liefern, aber noch weit davon entfernt sind, tatsächlich als *Künstliche Intelligenz* bezeichnet zu werden. Computer, mit der Intelligenz eines Menschen, wird es in absehbarer Zeit nicht geben.

Diese Ausarbeitung soll nun einen Ansatz, den es im Bereich *Machine Learning* gibt, genauer Untersuchen: **Support Vector Maschines** oder **SVM** beschreiben ein kernelbasiertes Lernverfahren, das gute Erfolge erzielte in der Erkennung von Handschriften oder bei Gensequenzanalysen in der Gentechnik.

2 Learning Machines

Anhand eines Beispiels lässt sich am besten erklären, wie Lernkonzepte aus der Natur Vorbild sind für die Techniken/Verfahren des Maschinellen Lernens.

Hat eine Katze mehrere negative Erfahrungen mit einem Hund gemacht, dann lernt diese, Konfrontationen mit Hunden zu vermeiden. Sieht die Katze dann einen anderen Hund, d. h. sie klassifiziert ein anderes Tier als einen Hund, dann entscheidet sie anhand der Erfahrungen, die sie gemacht hat, wie sie reagiert – in diesem Fall Flucht.

Auf diese Methodik soll in diesem Abschnitt näher eingegangen werden.

2.1 Lineare Klassifikation

Die gebräuchlichste und zugleich einfachste Art der Klassifikation ist die binäre Klassifikation. Mit Hilfe der Funktion $f : X \subseteq \mathbb{R} \to \mathbb{R}$ wird ein Eingabevektor der Form $(x_1, \ldots, x_n)^t \in X$ der positiven Klasse zugeordnet, falls $f(x) \geq 0$ gilt, sonst der negativen Klasse.

Die lineare Funktion f lässt sich also schreiben als:

$$\begin{aligned} f(x) &= \langle w \cdot x \rangle + b \\ &= \sum_{i=1}^{n} w_i x_i + b \end{aligned}$$

wobei $w \in \mathbb{R}^n$ und $b \in \mathbb{R}$ Parameter der Funktion sind und deren Form bestimmen. Die Entscheidungsfunktion oder Hypothese $h(x)$ ist, wie oben beschrieben, die Signumsfunktion $\operatorname{sgn}(f(x))$, definiert als:

$$h(x) = \operatorname{sgn}(f(x)) = \begin{cases} 1 & \text{falls } f(x) \geq 0 \\ -1 & \text{sonst} \end{cases}$$

Informell erklärt, werden die Eingabevektoren in zwei Teile geteilt, wobei ein Bereich die positiven und der andere Bereich die negative Eingaben darstellt. Eingabevektoren, die sich durch eine solche *Hyperebene* trennen lassen, werden *linear separabel* genannt.

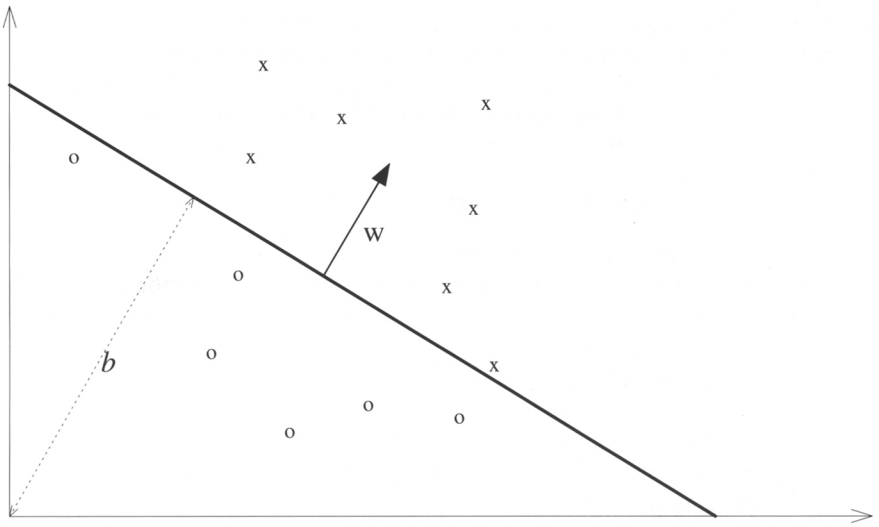

Abbildung 1: Daten in $\mathbb{R}^2$, getrennt durch eine Hyperebene gegeben durch (w, b)

In Abbildung 1 wird ein einfaches Beispiel für linear separierbare Daten gezeigt. Die dunkle Hyperebene (hier eine Linie) teilt den Eingaberaum in eine positive Region oberhalb und eine negative Region unterhalb der Linie ein. Durch den Orthogonalenvektor w wird die „Neigung” der Geraden bestimmt, während b diese parallel verschiebt.

Folgende Fragestellung drängt sich unweigerlich auf: Wie lässt sich eine solche Ebene finden, erlernen? Zur Beantwortung dieser Frage müssen zuerst einige Definitionen eingeführt werden:

Definition 2.1. *Sei X der Eingabe- und Y der Ausgaberaum. Oft ist $X \subseteq \mathbb{R}^n$ und $Y = \{-1, 1\}$ bei einer binären Klassifikation, bei einer Multi-Klassen Diskriminierung ist $Y = \{1, \ldots, m\}$ und $Y \subseteq \mathbb{R}$ im Fall der Regression. Desweiteren ist ein Trainingsset S definiert als*

$$S = ((x_1, y_1), \ldots, (x_l, y_l)) \subseteq (X \times Y)^l,$$

wobei l die Anzahl der Trainingsdaten angibt. Die x_i werden als Beispiele oder Instanzen und die y_i als deren Ergebnis bezeichnet.

Definition 2.2. *Der (funktionale) Abstand γ_i zwischen einem Beispiel (x_i, y_i) und einer Hyperebene, die durch (w, b) definiert ist, wird berechnet mit*

$$\gamma_i = y_i(\langle w \cdot x_i \rangle + b).$$

Des Weiteren soll $\gamma = \min_{1 \leq l} \gamma_i$ als der Abstand (oder Margin*) eines Trainingssets zu einer Hyperebene definiert werden.*

Als Abstand eines Trainingssets S wird der maximale geometrische Abstand aller Hyperebenen bezeichnet. Eine Hyperebene, die diesen Abstand realisiert, heißt Maximum Margin Hyperebene.

$\gamma_i > 0$ *impliziert eine richtige Klassifikation der Eingabe.*

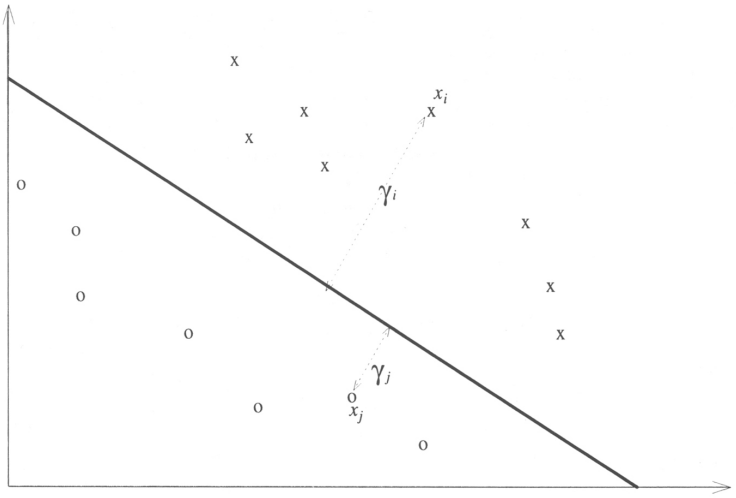

Abbildung 2: Der funktionale Abstand zweier Punkte

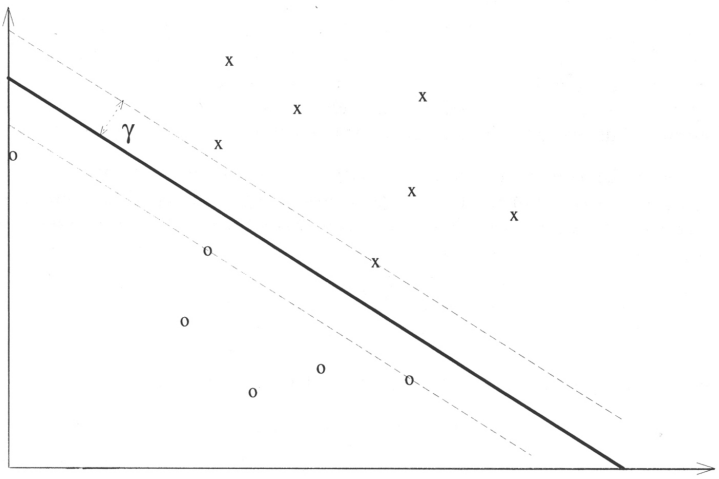

Abbildung 3: Der Abstand eines Trainingssets γ

Mit diesen Definitionen kann der Algorithmus für das Perzeptron (Tabelle 1) (entwickelt von Frank Rosenblatt 1956) beschrieben werden, Dieser hat garantiert eine Lösung, d. h. es wird immer eine Hyperebene gefunden, vorausgesetzt die Daten sind linear separabel.

$\eta \in \mathbb{R}^+$ beschreibt die Lernrate des Algorithmus.

$w_0 \leftarrow \overrightarrow{0}; b_0 \leftarrow 0; k \leftarrow 0$ `// wobei` k `die Fehleranzahl ist`
$R \leftarrow \max_{1 \leq i \leq l} \|x_i\|$
`repeat`
  `for` $i \leftarrow 1$ `to` l
    `if` $y_i(\langle w \cdot x_i \rangle + b) \leq 0$ `then`
      $w_{k+1} \leftarrow w_k + \eta y_i x_i$
      $b_{k+1} \leftarrow b_k + \eta y_i R^2$
      $k \leftarrow k + 1$
    `end if`
  `end for`
`until „keine Fehler in der for-Schleife“`
`return` (w_k, b_k)

Tabelle 1: Der Perzeptron Algorithmus (ursprüngliche Form)

wobei k die Fehleranzahl ist. Die Anzahl der Fehler dieses Alorithmus ist höchstens $\left(\frac{2R}{\gamma}\right)^2$. Ein Nachteil dieses Algorithmus jedoch ist, dass die gefundene Lösung nicht eindeutig ist! Schon die Reihenfolge, mit der die Beispiele betrachtet werden, kann eine andere Hyperebene als Lösung ergeben. In Abbildung 1 lassen sich viele Geraden finden, die diese Daten linear trennen würden. Es existieren Algorithmen, die eindeutige Hyperebenen erstellen. In Abschnitt 3 wird näher darauf eingegangen.

2.2 Weitere Klassifikationstypen

Außer der hier schon behandelten binären Klassifikation existieren noch weitere Typen. Zu den Wichtigsten gehören die Multi-Klassen-Diskriminierung und die Regression.

Ersteres handelt von SVM, die statt einer binären Klassifikation mit $Y = \{-1, 1\}$ mehrere Klassen $Y = \{1, \ldots, n\}$ besitzen. Diese lassen sich durch n verschiedene binäre SVM simulieren, indem für jede Klasse und deren Hyperebene eine SVM trainiert wird. Die Entscheidungsfunktion ist dann gegeben durch

$$c(x) = \arg \max_{1 \leq i \leq m} \left(\langle w_i \cdot x \rangle + b_i\right).$$

Geometrisch interpretiert, wird die Eingabe x derjenigen Klasse zugeordnet, deren Hyperebene am weitesten entfernt ist.

Die Problemstellung der linearen Regression besteht darin, eine lineare Funktion

$$f(x) = \langle w \cdot x \rangle + b$$

zu finden, die am besten die Werte eines gegebenen Trainingssets S interpoliert, wobei $Y \subseteq \mathbb{R}$.

Dieses Problem ist schon seit dem 18ten Jahrhundert bekannt. Die geläufigste Lösung ist die von Gauß und Legendre unabhängig vorgeschlagene Kleinste-Quadrate-Methode, bei der es die Summe der „Fehler im Quadrat" zu minimieren gilt. Ein Fehler bezeichnet hier die Entfernung des Datenpunktes x zur interpolierten Funktion $f(x)$. Am Anfang des Abschnitts 5 wird eine genauere Definition eingeführt.

$$L(w, b) = \sum_{i=1}^{l} (y_i - \langle w \cdot x_i \rangle - b)^2$$

Algorithmen wie der von Widrow-Hoff (auch bekannt als Adaline) [CST02, Seite 22] geben für dieses Problem eine Lösung an.

2.3 Duale Repräsentation

Bei genauerer Betrachtung des Perzeptron-Algorithmus zeigt sich, dass bei einer Fehlklassifikation positive Trainingsvektoren zu w addiert bzw. negative subtrahiert werden. Da bei der Initialisierung $w = \overrightarrow{0}$ gilt, folgt daraus, dass sich w definieren lässt als

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i.$$

Da das Vorzeichen der Klassifikation durch y_i gegeben wird, sind die $\alpha_i \in \mathbb{N}_0$ proportional zu der Anzahl der Fehlklassifikationen der x_i. Trainingspunkte, die leicht klassifiziert werden können, haben kleinere α_i, während schwierigere größere α_i besitzen. So lassen sich die Trainingsdaten nach deren Komplexität sortieren.

Die Entscheidungsfunktion $h(x)$ lässt sich nun folgendermaßen schreiben:

$$\begin{aligned} h(x) &= \operatorname{sgn}\left(\langle w \cdot x \rangle + b\right) \\ &= \operatorname{sgn}\left(\left\langle \sum_{j=1}^{l} \alpha_j y_j x_j \cdot x \right\rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{j=1}^{l} \alpha_j y_j \langle x_j \cdot x \rangle + b\right) \end{aligned}$$

Unter Verwendung von $h(x)$ lässt sich nun auch der Algorithmus 1 umformen.

$\alpha_0 \leftarrow \overrightarrow{0}; b_0 \leftarrow 0$
$R \leftarrow \max_{1 \leq i \leq l} \|x_i\|$
`repeat`
  `for` $i \leftarrow 1$ `to` l
    `if` $y_i \left(\sum_{j=1}^{l} \alpha_j y_j \langle x_j \cdot x_i \rangle + b \right) \leq 0$ `then`
      $\alpha_i \leftarrow \alpha_i + 1$
      $b \leftarrow b + y_i R^2$
    `end if`
  `end for`
`until „keine Fehler in der for-Schleife”`
`return` (α, b) `für Funktion` $h(x)$

Tabelle 2: Perzeptron Algorithmus in dualer Form

Bemerkung 2.3. Da bei jeder Fehlklassifikation genau ein Wert im Vektor α geändert wird, gilt:

$$\sum_{i=1}^{l} \alpha_i = \|\alpha\|_1 \leq \left(\frac{2R}{\gamma} \right)^2 .$$

Wobei die p-Norm definiert ist als $\|z\|_p = \left(\sum_{i=1}^{\dim(z)} |z_i|^p \right)^{\frac{1}{p}}$

Bemerkung 2.4. Die Trainingsdaten treten nun nicht mehr als Vektoren, sondern nur noch durch die Einträge der Matrix

$$G = \left(\langle x_i \cdot x_j \rangle \right)_{i,j=1}^{l}$$

auf, die bekannt ist als die *Gram Matrix*.

3 Maximum Margin

In der Bememerkung zu Algorithmus 1 wurde schon angesprochen, dass (unter gewissen Voraussetzungen) zwar immer eine Hyperebene gefunden wird, diese aber nicht Eindeutig ist. Es lassen sich mehrere Hyperebenen finden, die diese Daten linear trennen, doch welche von diesen ist die Beste ?

Mit „beste” ist diejenige Hyperebene gemeint, die nach der Trainingsphase möglichst gut generalisiert und dabei „sehr effizient” berechnet werden kann. „Sehr effizient” bedeutet,

dass der spätere Algorithmus Trainingsdaten mit über $100\,000$ Instanzen in annehmbarer Zeit verarbeiten kann. Es existieren verschiedene Algorithmen, die durch verschiedene Generalisierungsbeschränkungen motiviert werden. Der allgemeinste und bekannteste Berechnungsansatz ist der „Maximal Margin Classifier". Dieser funktioniert nur für linear separierbare Daten im Merkmalsraum und kann deswegen nicht in allen Situationen benutzt werden. Im Wesentlichen sucht dieser Algorithmus nach derjenigen Hyperebene mit dem größten Abstand γ. Dafür werden zwei weitere zur Hyperebene parallele Ebenen definiert:

$$\langle x \cdot x^+ \rangle = +1$$
$$\langle x \cdot x^- \rangle = -1$$

Wie in Bild 4 zu erkennen grenzen diese Ebenen den Bereich zwischen der Hyperebene und den Vektoren der positiven bzw. negativen Klasse ab.

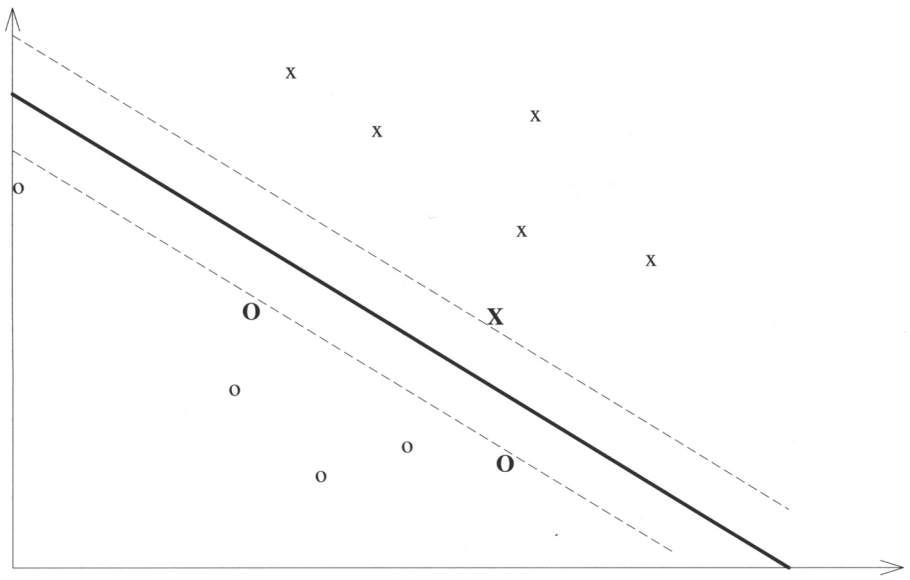

Abbildung 4: Beispiel einer Maximum Margin Hyperebene mit hervorgehobenen Support Vektoren und den Ebenen $\langle x \cdot x^+ \rangle = +1, \langle x \cdot x^- \rangle = -1$

Daraus lässt sich nun eine neue Darstellung für den Abstand γ definieren.

$$\begin{aligned}\gamma &= \frac{1}{2}\left(\left\langle \frac{w}{\|w\|_2}\cdot x^+\right\rangle - \left\langle \frac{w}{\|w\|_2}\cdot x^-\right\rangle\right)\\ &= \frac{1}{2\|w\|_2}\left(\langle w\cdot x^+\rangle - \langle w\cdot x^-\rangle\right)\\ &= \frac{1}{\|w\|_2}.\end{aligned}$$

Es stellt sich heraus, dass der Abstand γ nur noch von der Norm von w abhängt. Da das Maximum von γ gesucht ist, muss $\|w\|_2$ minimiert werden. In folgendem Satz wird dies als Optimierungsproblem festgehalten.

Satz 3.1. Für ein gegebenes, linear separierbares, Trainingsset

$$S = ((x_1, y_1), \ldots, (x_l, y_l))$$

repräsentiert diejenige Hyperebene, die folgendes Optimisierungsproblem

$$\begin{aligned} &minimiere_{w,b} && \langle w\cdot w\rangle,\\ &in\ Abhängigkeit\ von && y_i\left(\langle w\cdot x_i\rangle + b\right) \geq 1,\\ & && i = 1,\ldots,l\end{aligned}$$

löst, gerade die Maximum Margin Hyperebene mit geometrischen Abstand $\gamma = 1 \,/\, \|w\|_2$.

In [CST02, Kapitel 5.3] wird ein Ansatz beschrieben, mit dem sich dieses Optimierungsproblem in dualer Form schreiben lässt. Dafür wird folgende Formel von Lagrange in primärer Form benutzt:

$$L(w,b,\alpha) = \frac{1}{2}\langle w\cdot w\rangle - \sum_{i=1}^{l}\alpha_i\left[y_i\left(\langle w\cdot x_i\rangle + b\right) - 1\right],$$

wobei $\alpha_i \geq 0$ die Langrange-multiplikatoren sind. Durch Ableiten nach w und b folgt:

$$\begin{aligned}\frac{\partial L(w,b,\alpha)}{\partial w} &= w - \sum_{i=1}^{l} y_i\alpha_i x_i = \overrightarrow{0}\\ &\Rightarrow w = \sum_{i=1}^{l} y_i\alpha_i x_i\\ \frac{\partial L(w,b,\alpha)}{\partial b} &= \sum_{i=1}^{l} y_i\alpha_i = 0\\ &\Rightarrow 0 = \sum_{i=1}^{l} y_i\alpha_i\end{aligned}$$

Durch Resubstitution in die primäre Form folgt:

$$\begin{aligned} L(w,b,\alpha) &= \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle - \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle + \sum_{i=1}^{l} \alpha_i \\ &= \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle. \end{aligned}$$

Satz 3.2. Für ein gegebenes, linear separierbares, Trainingsset

$$S = ((x_1, y_1), \ldots, (x_l, y_l))$$

und unter der Voraussetzung, dass die Parameter α^* das folgende quadratische Optimierungsproblem lösen:

$$\begin{aligned} &maximiere && W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i \cdot x_j \rangle, \\ &in\ Abhängigkeit\ von && \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & && \alpha_i \geq 0, i = 1, \ldots, l \end{aligned}$$

realisiert der Vektor $w^* = \sum_{i=1}^{l} y_i \alpha_i^* x_i$ gerade die Maximum Margin Hyperebene mit dem geometrischen Abstand $\gamma = \frac{1}{\|w^*\|_2}$

Bemerkung 3.3. b wird im obigen Satz nicht mehr erwähnt, daher muss b^* durch die Begrenzungen der primären Form gefunden werden.

$$b^* = -\frac{\max_{y_i=-1}(\langle w^* \cdot x_i \rangle) + \min_{y_i=1}(\langle w^* \cdot x_i \rangle)}{2}$$

Mit dem Kuhn-Tucker Theorem [CST02, Seite 87] kann nun gefolgert werden, dass nur für diejenigen Eingaben x_i, deren funktioneller Abstand zur Hyperebene eins ist, die dazugehörigen $\alpha_i^* \neq 0$ sind. Für alle anderen Parameter gilt $\alpha_i^* = 0$.

4 Support Vektoren

Die Folgerungen aus dem vorigem Abschnitt bedeuten, dass nicht alle Eingabevektoren für die Darstellung von w benötigt werden. Nur diejenigen, die am nächsten zur Hyperebene liegen, sind tatsächlich notwendig, um diese zu definieren. In Abbildung 4 sind solche Vektoren markiert.

Definition 4.1. *Support Vektoren*
*Alle Eingabevektoren , deren $\alpha^* \neq 0$ ist, werden **Support Vektoren** genannt. Die Menge aller Support Vektoren wird mit **sv** bezeichnet. Des Weiteren kann die optimale Hyperebene, also die Maximum Margin Hyperebene, durch α^*, b^* beschrieben werden mit:*

$$\begin{aligned} f(x, \alpha^*, b^*) &= \sum_{i=1}^{l} y_i \alpha_i^* \langle x_i \cdot x \rangle + b^* \\ &= \sum_{i \in sv} y_i \alpha_i^* \langle x_i \cdot x \rangle + b^*. \end{aligned}$$

5 Kernel

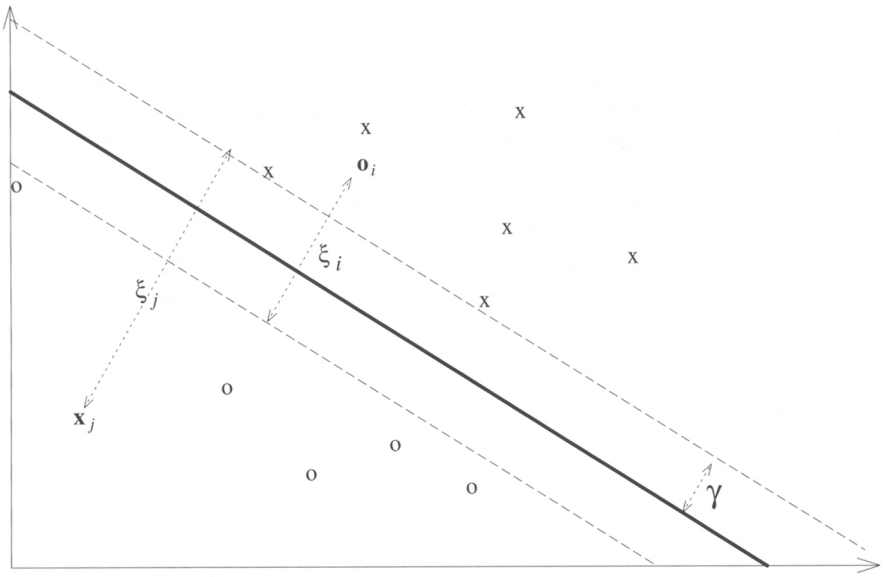

Abbildung 5: Zwei *Schlupfvariablen* für ein Klassifikationsproblem

Wie anfangs schon erwähnt haben lineare SVM nur die Möglichkeit, linear separierbare Daten zu trennen. Überschneiden sich die Datensätze wie in Abbildung 5, kann der angegebene Algorithmus keine Hyperebene finden – er endet in einer Endlosschleife. Eine Möglichkeit, dieses Problem zu umgehen, wäre, falsch klassifizierte Daten zuzulassen und nach einer Hyperebene zu suchen, die einerseits die Anzahl der falschen Klassifikationen und andererseits den Abstand dieser „falschen” Daten zur Ebene minimiert. Für diesen Ansatz wird eine sog. *margin slack variable* definiert als:

$$\xi\left(\left(x_i, y_i\right), (w, b), \gamma\right) = \xi_i = max\left(0, \gamma - y_i\left(\langle w, x_i \rangle + b\right)\right).$$

Informell gibt ξ an, wie weit ein falsch klassifizierter Datenpunkt x_i von der Hyperebene, welche den ganzen Datensatz teilt, entfernt ist. In Abbildung 5 sind zwei Beispiele für diese Variable eingezeichnet.

Aber: Es ist bewiesen, dass das Finden einer möglichst guten Trennung nicht separierbarer Daten mit der kleinsten Anzahl von Fehlern und Iterationsschritten $\mathcal{NP}$-Vollständig ist!

Es existieren eine Reihe von Heuristiken, auf die hier aber nicht näher eingegangen wird, siehe [CST02] bzw. [SCHSM01] für deren Definition.

Um das Problem der $\mathcal{NP}$-Vollständigkeit und Fehlklassifikation zu umgehen, wird ein Trick benutzt, der sog. *Kerneltrick*. Mit diesem wird nicht nur das Problem der Berechnungskosten gelöst, es lassen sich auch nicht separierbare Daten klassifizieren. Zunächst einmal wird folgende Definition einer Abbildung des Eingaberaums X in den *Merkmalsraum* F betrachtet.

Definition 5.1. *Ein* Repräsentationswechsel *beschreibt die Abbildung Φ, welche den* **Eingaberaum X** *in den* **Merkmalsraum F** *abbildet.*

$$\Phi : X \subseteq \mathbb{R}^n \to F \subseteq \mathbb{R}^N$$

$$X \ni x \mapsto \Phi(x) = (\Phi_1(x), \ldots, \Phi_N(x)) \in F$$

Dabei gilt für den Merkmalsraum $F = \{\, \Phi(x) |\, x \in X \}$.

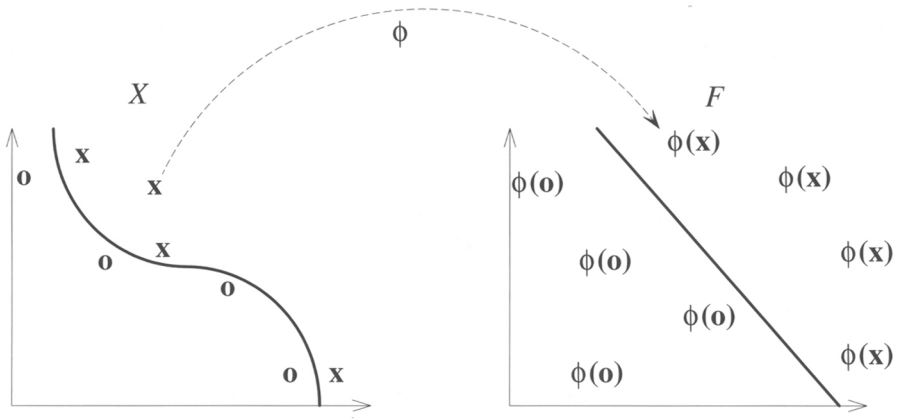

Abbildung 6: Beispiel für einen Repräsentationswechsel, um Daten linear klassifizieren zu können.

Beispiel 5.2. Es folgt ein Beispiel, das verdeutlichen soll, warum eine solche Abbildung sinnvoll ist. Das Gravitationsgesetz von Newton

$$f(m_1, m_2, r) = C\frac{m_1 m_2}{r^2},$$

welches die Gravitationskraft zweier Massen m_1, m_2 mit Abstand r beschreibt, lässt sich mit beobachtbaren Daten, also Massen und Entfernung berechnen. Allerdings kann eine lineare SVM diese Daten nicht direkt klassifizieren. Ein einfacher Repräsentationswechsel in einen passenden Merkmalsraum mit

$$(m_1, m_2, r) \mapsto (x, y, z) = (\ln m_1, \ln m_2, \ln r)$$

ergibt die Darstellung

$$g(x, y, z) = \ln f(m_1, m_2, r) = \ln C + \ln m_1 + \ln m_2 - 2 \ln r = c + x + y - 2z,$$

die von einer linearen Maschine gelernt werden kann, da es sich im letzten Ausdruck um eine Summe der Parameter handelt. Schon an diesem einfachen Beispiel lässt sich die die Mächtigkeit dieser Vereinfachung erkennen. Abbildung 6 zeigt die geometrische Vorstellung eines Repräsentationswechsels, und warum in X die Daten nicht von einer linearen SVM gelernt werden können.

Folgende Definitionen führen wichtige Begriffe für das Arbeiten im Merkmalsraum ein.

Definition 5.3. *Als* **Merkmale** *werden die Vektoren aus F bezeichnet. Sie beschreiben diejenigen Daten, die klassifiziert werden. Die ursprünglichen Daten, bzw. Vektoren aus X werden* **Attribute** *genannt.*

Die Suche einer möglichst guten Repräsentation der Attribute wird **Merkmalsauswahl** *genannt.*

Durch Repräsentationswechsel kann außerdem die Dimension, in der die SVM arbeitet, verändert werden. So führt das Streichen redundanter oder unnötiger Daten zu einer Dimensionsverkleinerung und dadurch zur Berechnungsoptimierung. Im Gegenzug wird die Dimension vergrößert, falls neue Attribute zur besseren Separierung hinzugefügt werden müssen. Diese können durch schon vorhandene Daten berechnet und so in einen neuen Zusammenhang gebracht werden.

Um also nichtlineare SVM zu erhalten, werden die Attribute zuerst mit einer nicht-linearen Abbildung $\Phi(x)$ in den Merkmalsraum abgebildet, in welchem SVM benutzt werden können. Infolgedessen werden Funktionen der Form

$$f(x) = \sum_{i=1}^{N} w_i \Phi_i(x) + b$$

für die Hypothese der SVM betrachtet.

wie schon in Abschnitt 2.3 gezeigt, existiert für jede Entscheidungsfunktion eine duale Repräsentation. Das bedeutet, dass sich die Entscheidungsfunktion als Linearkombination der Trainingsdaten darstellen lässt. Diese Vektoren treten dann nur noch als inneres Produkt auf:

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i \langle \Phi(x_i) \cdot \Phi(x) \rangle + b$$

Falls es nun möglich ist, das innere Produkt $\langle \Phi(x_i) \cdot \Phi(x) \rangle$ direkt im Merkmalsraum zu berechnen, sind die zwei Schritte (Abbildung nach und Klassifikation in F) auf einen Schritt reduziert. Solch eine *implizierte Abbildung* heißt *Kernel*.

Definition 5.4. *Ein **Kernel** ist eine Funktion K, sodass für alle $x, z \in X$ gilt:*

$$K(x, z) = \langle \Phi(x) \cdot \Phi(z) \rangle ,$$

wobei Φ der Definition der Abbildung von X in den Merkmalsraum F entspricht.

Mit dieser Definition lässt sich die Entscheidungsformel nochmals umformulieren zu

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b.$$

Es ist aus der Formel klar, dass jetzt nur noch die einzelnen inneren Produkte zur Berechnung notwendig sind. Dadurch ist der Aufwand für die Berechnung nur noch von l und nicht mehr von der Anzahl der Merkmale abhängig.

Definition 5.5. *Die für die Berechnung des Kernels notwendige Gram-Matrix*

$$G = (\langle x_i \cdot x_j \rangle)_{i,j=1}^{l}$$

*wird auch **Kernel-Matrix** genannt.*

6 Anwendungen

6.1 Handschrifterkennung

Die Handschrifterkennung stellt eines der ersten Klassifikationsprobleme mit praktischem Bezug dar, für die SVM eingesetzt wurden. Ursprünglich wurde eine Lösung dieses Problems von der US-Post motiviert, um das Sortieren handgeschriebener Postleitzahlen auf den Briefen zu automatisieren. Dafür wurden zwei Datenbanken mit Zeichen zur Verfügung gestellt: eine von der USPS („United States Postal Service"), die andere von der NIST („National Institute for Standard and Technology"), beide öffentlich zugänglich. Der Datensatz der USPS umfasste 7291 Trainings- und 2007 Testmuster, der der NIST 60 000 Trainings- und 10 000 Testmuster. Für die Klassifikation wurden verschiedene SVM implementiert, dazu gehörten Multi-Klassen SVM mit polynomiellen, sowie Gauß'schen Kernel. Es stellte sich heraus, dass alle ungefähr dieselbe Leistung hatten, und viele die gleichen Support Vektoren. Die genauen Ergebnisse können zusammengefasst in [VV98] nachgeschlagen werden.

Für die USPS-Daten wurden die folgenden polynomiellen und Gauß'schen Kernel benutzt:

$$K(x,y) = \left(\frac{\langle x \cdot y \rangle}{256}\right)^d$$

und

$$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{256\sigma^2}\right)$$

Mit den Werten $1 \leq d \leq 6$ und $0,1 \leq \sigma \leq 4,0$

Die Daten der USPS konnten mit dem polynomiellen Kernel und der Maximum-Margin-Klassifikation erst ab Grad 3 linear separiert werden. Mit Soft-Margin-Optimierung und Grad 1 entstanden $340/7291$ und mit Grad 2 $4/7291$ Fehlklassifikationen. Diese Experimente sind besonders interessant, da schon hochspezialisierte Algorithmen existieren, die durch großes a priori Wissen entstanden sind. Die Tatsache, dass SVM ohne große Spezialisierung genauso gute Werte erzielen, ist sicherlich bemerkenswert.

6.2 Bilderkennung

Eine Studie von Pontil und Verri [PV98] untersuchte die Performanz von SVM anhand von Bildern aus den COIL-(Columbia Object Image Library) Daten, die über das Internet zugänglich sind. Diese Datenbank stellt 7200 Bilder von 100 verschiedenen Objekten Verfügung. Diese wurden mit jeweils 5 Grad Versatz aufgenommen, sodass 72 Ansichten pro Objekt entstanden.

Um eine Klassifikation zu ermöglichen, mussten die Bilder von 128×128 Pixel großen Farbfotos zu 32×32 Pixel großen Bitmaps in Graustufen transformiert werden. Nach dieser Vorbereitung entstanden Vektoren mit 1024 Elementen, jeweils dargestellt durch 8 bit breite Zahlen. Ein einfaches Vektorprodukt dieser Daten wurde als Kernel gewählt sowie eine Maximum-Margin-Klassifikation implementiert.

Die 7200 Bilder wurden in zwei Gruppen, Trainings- und Testdaten, eingeteilt, wobei jede Gruppe das jeweils zweite Bild der 72 Ansichten aller 100 Objekte beinhaltete. Dadurch wurde für jedes der 100 Objekte eine eigene Klasse mit Trainings- und Testdaten erzeugt. Für das Experiment wurden jedoch nur 32 zufällig gewählte Objekte benutzt.

Es zeigte sich, dass durch die hohe Dimension der Eingaben scharf begrenzte Hyperebenen und einfache Kernel ausreichten, um diese Daten exakt zu trennen. Außerdem arbeitete die trainierte SVM sehr effektiv, nur künstliches Hinzufügen von Rauschwerten zu den Eingaben führte zu Fehlklassifikationen. Perzeptrone, die in derselben Umgebung trainiert wurden, schnitten im Vergleich viel schlechter ab.

Literatur

[CST02] Nello Cristianini und John Shawe-Taylor,

Support Vector Machines and other kernel-based learning methods,

Cambridge University Press 2002.

[SCHSM01] Bernhard Schölkopf und Alexander J. Smola,

Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond,

The MIT Press 2001.

[VV98] V. Vapnik,

Statistical Learning Theory,

Wiley 1998.

[PV98] M. Pontil und A.Verri,

Object recognition with support vector machines,

IEEE Trans. on PAMI, 20:637-646, 1998.