

# **Todo list**



## **Support Vector Machines (SVM)**

### **Computational Intelligence II**

Informatik - Software and Information Engineering  
Fachhochschule Vorarlberg

Erstellt von  
André Hopfgartner & Matthias Rupp

Dornbirn, am 5. Juni 2021

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>5</b>
<b>1 Intuition</b>	<b>6</b>
<b>2 Hard-Margin Support Vector Machine</b>	<b>7</b>
2.1 Problemdefinition . . . . .	7
2.2 Optimierungsproblem . . . . .	10
2.3 Lagrange Optimierung . . . . .	11
2.4 Lösung mittels Quadratic Programming Solver . . . . .	14
<b>3 Soft-Margin Support Vector Machine</b>	<b>16</b>
<b>4 Vergleich Soft-Margin und Hard-Margin Support Vector Machine</b>	<b>18</b>
<b>5 Nichtlineare Trennung</b>	<b>20</b>
5.1 Transformation der Problemstellung . . . . .	20
5.2 Kernel Trick . . . . .	22
5.2.1 Beispiel Kernel-Funktion . . . . .	22
5.2.2 Polynomieller Kernel . . . . .	23
5.2.3 Radial Basis Function Kernel . . . . .	24
5.2.4 SVM Definition mit Kernel . . . . .	24
5.2.5 Anforderungen an eine Kernel-Funktion . . . . .	25
<b>6 Pseudocode und Beispiele</b>	<b>27</b>
6.1 Hard-Margin SVM Pseudocode . . . . .	27
6.1.1 Berechnung von $\mathbf{K}$ . . . . .	28
6.2 Hard-Margin SVM - Beispiel . . . . .	28
6.3 Soft-Margin SVM Pseudocode . . . . .	30
6.4 Soft-Margin SVM Beispiel . . . . .	31
6.5 Kernel-Trick - Pseudocode . . . . .	32
6.5.1 Berechnung von $\mathbf{K}$ mit Kernelfunktion . . . . .	33
6.6 Kernel-Trick, Polynomieller Kernel - Beispiel . . . . .	34
6.7 Kernel-Trick, RBF Kernel - Beispiel . . . . .	36
6.8 Kernel-Trick, Vergleich verschiedener Methoden . . . . .	38
<b>Weiterführende Ressourcen</b>	<b>41</b>

# Abkürzungsverzeichnis

**SVM** Support Vector Machine

**QP** Quadratic Programming

**RBF** Radial Basis Function

# 1 Intuition

Das Ziel einer Support Vector Machine (SVM) ist die lineare Separation von zwei verschiedenen Klassen. Die Separation wird durch eine Ebene durchgeführt. Die Lage der Ebene wird so gewählt, dass um die Ebene ein möglichst breites Band entsteht. Abbildung 1.1 zeigt Beispiele für solche Trennungen. Weiters gibt es zwei Arten von SVMs:

- Eine Hard-Margin SVM trennt die Klassen so, dass keine Fehlklassifikationen entstehen.
- Eine Soft-Margin SVM erlaubt einzelne Fehlklassifikationen damit eine mitunter bessere Trennebene gefunden werden kann.

Diese beiden Arten von SVMs werden in den folgenden Kapiteln genauer betrachtet. Die Qualität der Trennung wird bei einer SVM durch die Breite des Trennbands beurteilt.

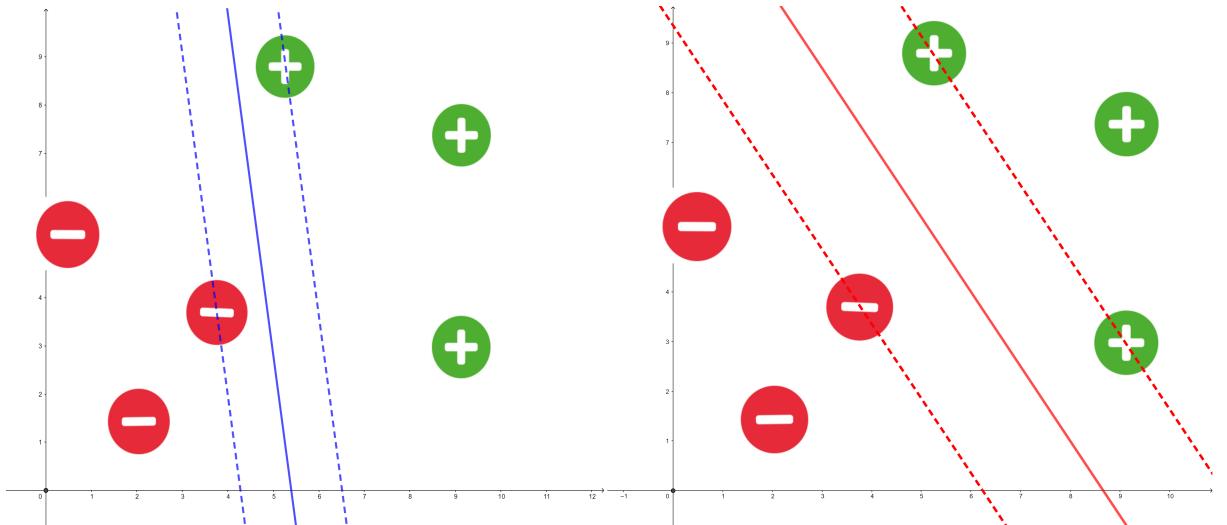


Abbildung 1.1: Abhängig von der Lage der Trennebene entstehen schmale (blau) oder breite (rot) Trennbänder. Das Ziel der SVM ist die Maximierung der Breite des Trennbands durch die Ermittlung der optimalen Lage der Trennebene.

## 2 Hard-Margin Support Vector Machine

Wie bereits in der Einführung erwähnt ist das Ziel einer Hard-Margin SVM ist die Trennung zweier Klassen ohne Fehler mit einem möglichst breiten Trennband. Diese Aussage wird in diesem Kapitel mathematisch formuliert und als Optimierungsproblem dargestellt. Weiters wird beschrieben wie dieses Problem in die Standardform von Quadratic Programming (QP) Problemen transformiert werden kann was für die Lösung nötig ist.

### 2.1 Problemdefinition

Gegeben sei ein Gewichtsvektor  $w \in \mathbb{R}^K$ , ein Bias  $b \in \mathbb{R}$ , ein beliebiger Punkt  $x_n \in \mathbb{R}^K$  und ein zugehöriges Label  $y_n \in \{-1, +1\}$ . Eine Ebene im Raum kann allgemein definiert werden durch:

$$w^T x_n + b = 0 \quad (2.1)$$

Mit einer bereits trainierten SVM können neue, noch unklassifizierte Eingabevektoren  $x_n$  wie folgt klassifiziert werden:

$$y = \text{sign}(w^T x_n + b) \quad \text{gleichbedeutend mit} \quad (2.2a)$$

$$w^T x_n + b > 0 \quad \text{für } y_n = +1 \quad (2.2b)$$

$$w^T x_n + b < 0 \quad \text{für } y_n = -1 \quad (2.2c)$$

Für die Herleitung führen wir eine noch striktere Regel ein. So soll für eine richtige Klassifikation eines gegebenen Eingabevektors  $x_n$  mit dem Label  $y_n$  gelten:

$$w^T x_n + b \geq +1 \quad \text{für } y_n = +1 \quad (2.3a)$$

$$w^T x_n + b \leq -1 \quad \text{für } y_n = -1 \quad (2.3b)$$

Durch Gleichung 2.3 wird sichergestellt, dass Werte aus dem Intervall  $]-1, +1[$  als Ergebnis nicht vorkommen können. Wie in Abbildung 2.1 dargestellt entsteht dadurch ein symmetrisches Trennband um die Trennebene in dem keine Punkte liegen. Ziel der SVM ist die Maximierung der Breite dieses Bandes.

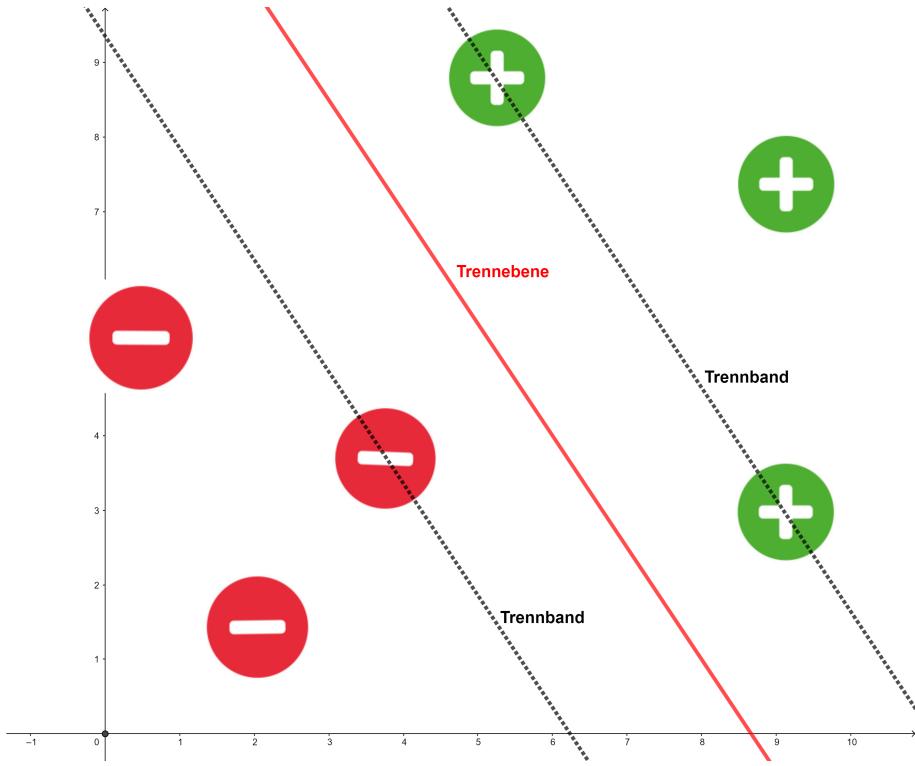


Abbildung 2.1: Um die Trennebene entsteht ein Trennband. Die am nächsten zu der Ebene liegenden Eingabevektoren liegen genau auf den Grenzen des Trennbands.

Gleichung 2.3 kann weiter verallgemeinert werden durch beidseitige Multiplikation mit  $y_n$ :

$$y_n(w^T x_n + b) \geq 1 \quad \text{für } y_n = +1 \quad (2.4a)$$

$$y_n(w^T x_n + b) \geq 1 \quad \text{für } y_n = -1 \quad (2.4b)$$

Für den Fall, dass  $x_n = \hat{x}$  genau an der Grenze des Trennbands liegt, gilt somit:

$$y_n(w^T \hat{x} + b) = 1 \quad (2.5)$$

Als nächsten Schritt bestimmen wir den euklidischen Normalabstand  $D$  eines beliebigen Punkts  $x_n \in \mathbb{R}^K$  zu der Ebene. Hierfür ist zuerst zu bemerken, dass  $w$  normal zur definierten Ebene steht.

**Lemma 2.1.1.** Eine Ebene sei definiert durch  $w^T x + b = 0$ . Der Vektor  $w$  steht normal zu der definierten Ebene.

*Beweis.* Man wähle zwei Punkte  $x_1, x_2 \in \mathbb{R}^K$  die auf der Ebene liegen. Somit muss gelten:

$$\begin{aligned} w^T x_1 + b &= 0 \\ w^T x_2 + b &= 0 \\ w^T(x_1 - x_2) = 0 &\Leftrightarrow \|w^T\| \|x_1 - x_2\| \cos(\alpha) = 0 \Leftrightarrow \alpha = 90^\circ \end{aligned} \tag{2.6}$$

□

Um den Normalabstand  $d$  eines beliebigen Punkts  $x_n$  zu ermitteln wählt man einen Punkt  $x$ , der auf der Ebene liegt, und projiziert den Vektor  $(x_n - x)$  auf den Einheitsvektor von  $w$ . Weil nur der tatsächliche Abstand zur Ebene relevant ist und nicht die Richtung nimmt man den Betrag.

$$\begin{aligned} d &= \left| \frac{w^T}{\|w\|} (x_n - x) \right| = \\ &= \frac{1}{\|w\|} |(w^T x_n - w^T x)| = \\ &= \frac{1}{\|w\|} |(w^T x_n + b - (w^T x + b))| \end{aligned} \tag{2.7}$$

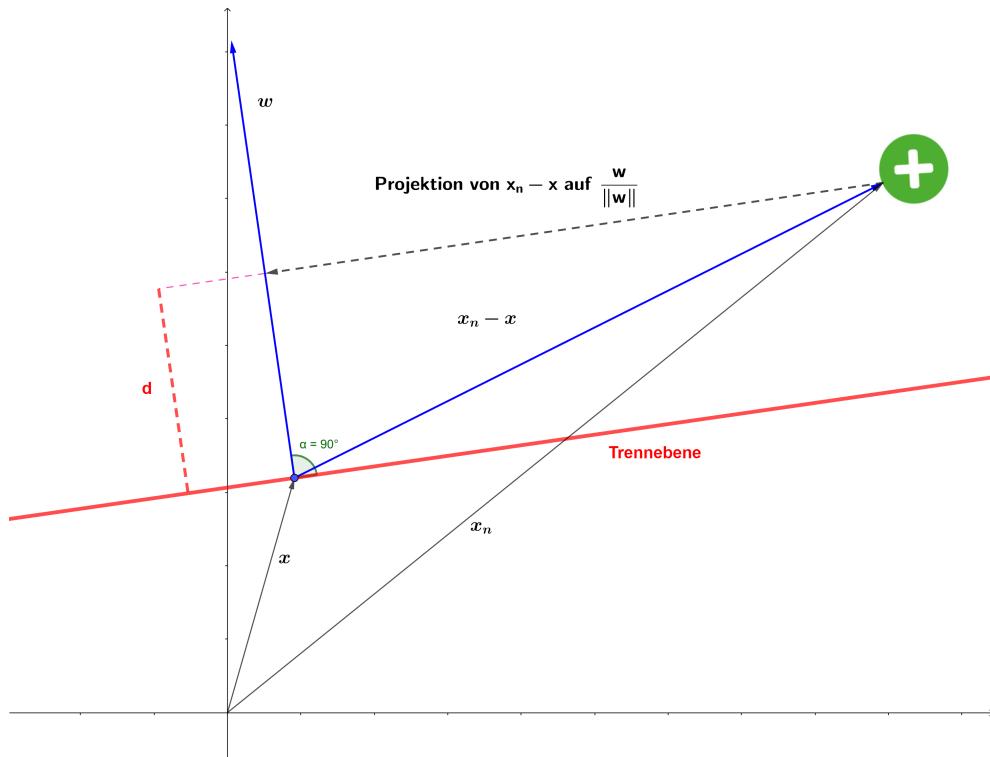


Abbildung 2.2: Durch die Projektion von  $(x_n - x)$  auf den Einheitsvektor von  $w$  kann der Normalabstand  $d$  von  $x_n$  zu der Ebene bestimmt werden.

Weil der Punkt  $x$  auf der Ebene liegt gilt  $w^T x + b = 0$  (Gleichung 2.1):

$$d = \frac{1}{\|w\|} |(w^T x_n + b)| \quad (2.8)$$

Trifft man nun die Annahme, dass  $x_n = \hat{x}$  der am nächsten zu der Trenngrenze liegende Punkt ist, so gilt aus Gleichung 2.5  $y_k(w^T \hat{x} + b) = 1 = |w^T \hat{x} + b|$  unter der Annahme, dass der Punkt richtig klassifiziert wurde. Somit ergibt sich der kleinste Abstand zur Trennebene  $D = d(\hat{x})$ , welcher zugleich der halben Breite des Trennbands entspricht, als:

$$D = \frac{1}{\|w\|} \quad (2.9)$$

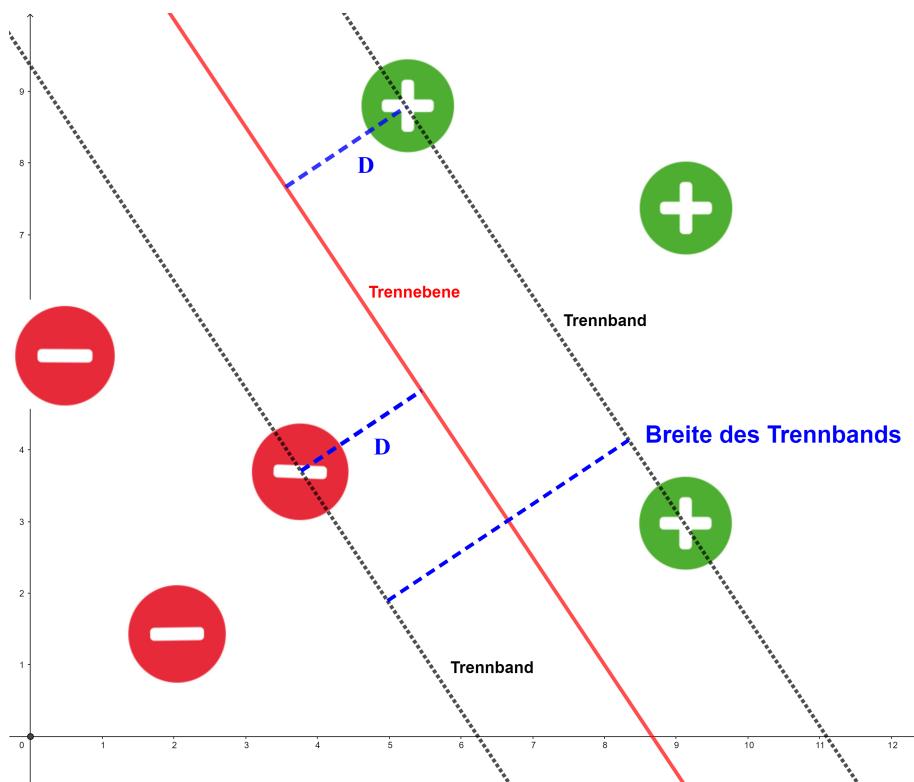


Abbildung 2.3: Der Normalabstand  $D$  von der Ebene zu den am nächsten liegenden Eingabevektoren entspricht genau der Hälfte der Breite des Trennbands.

## 2.2 Optimierungsproblem

Gleichung 2.9 beschreibt den Normalabstand zu dem am nächsten an der Ebene liegenden Punkt  $\hat{x}$ . Der Normalabstand ist gleichbedeutend mit der Hälfte der Breite

des Trennbands, wobei für die Optimierung der Faktor 2 keine Rolle spielt und vernachlässigt werden kann.

Das Ziel einer SVM ist die Maximierung der Breite des Trennbands für  $N$  Eingabevektoren  $\{x_1..x_N\}, x_n \in \mathbb{R}^K$ .

Bei dem beschriebenen Problem handelt es sich um ein Optimierungsproblem mit Nebenbedingungen:

$$\max_w \quad \frac{1}{\|w\|} \quad (2.10a)$$

$$\text{mit} \quad \min_{n=1..N} |w^T x_n + b| = 1 \quad (2.10b)$$

Gleichung 2.10b beschreibt hier den am nächsten zur Ebene gelegenen Punkt  $\hat{x}$  aus der gebenen Menge von Eingabevektoren in allgemeiner Form. Der Betrag lässt sich vermeiden durch die Anwendung von Gleichung 2.4:

$$|w^T x_n + b| = y_n(w^T x_n + b) \quad (2.11)$$

Durch Anwendung von Gleichung 2.11 in Gleichung 2.10b, Umformulierung der Maximierung in eine Minimierung und der Verallgemeinerung von  $\hat{x}$  auf beliebige Punkte  $x_n$  erhält man:

$$\min_w \quad \frac{1}{2} w^T w \quad (2.12a)$$

$$\text{mit} \quad y_n(w^T x_n + b) \geq 1 \text{ für } n = 1..N \quad (2.12b)$$

Die Verallgemeinerung von Gleichung 2.10b auf Gleichung 2.12b auf beliebige Punkte ist so möglich, weil durch Gleichung 2.5 sichergestellt ist, dass für beliebige Punkte  $y_n(w^T x_n + b) \geq 1$  gilt.

## 2.3 Lagrange Optimierung

Das beschriebene Optimierungsproblem beinhaltet eine Ungleichung in Gleichung 2.12b. Um die Lagrangegleichung aufzustellen zu können wird zuerst die Nebenbedingung umgeformt:

$$\min_w \quad \frac{1}{2} w^T w \quad (2.13a)$$

$$\text{mit} \quad y_n(w^T x_n + b) - 1 \geq 0 \text{ für } n = 1..N \quad (2.13b)$$

Das Problem kann nun als Lagrangegleichung dargestellt werden:

$$\min_{w,b} \quad \mathcal{L}(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{n=1}^N \alpha_n (y_n(w^T x_n + b) - 1) \quad (2.14a)$$

$$\max_{\alpha_n} \quad \alpha_n \geq 0 \text{ für } n = 1..N \quad (2.14b)$$

Weil Gleichung 2.13b eine Ungleichung der Form  $\geq 0$  beinhaltet werden diese Terme von der zu maximierenden Funktion subtrahiert. Nun kann die uneingeschränkte Optimierung von Gleichung 2.14a nach  $w$  und  $b$  durchgeführt werden indem die Ableitungen bestimmt und 0 gesetzt werden:

$$\begin{aligned}\nabla_w \mathcal{L} &= w - \sum_{n=1}^N \alpha_n y_n x_n \stackrel{!}{=} \vec{0} \\ w &= \sum_{n=1}^N \alpha_n y_n x_n\end{aligned}\tag{2.15}$$

$$\begin{aligned}\frac{\partial}{\partial b} \mathcal{L} &= - \sum_{n=1}^N \alpha_n y_n \stackrel{!}{=} 0 \\ \sum_{n=1}^N \alpha_n y_n &= 0\end{aligned}\tag{2.16}$$

Die Ergebnisse von Gleichung 2.15 und Gleichung 2.16 können in Gleichung 2.14a eingesetzt werden. Hierfür wird zuerst die Summe in Gleichung 2.14a in Teilsummen zerlegt:

$$\begin{aligned}\mathcal{L}(w, b, \alpha) &= \frac{1}{2} w^T w - \sum_{n=1}^N \alpha_n (y_n (w^T x_n + b) - 1) = \\ &= \frac{1}{2} w^T w - \left[ \sum_{n=1}^N \alpha_n y_n b - \sum_{n=1}^N \alpha_n + \sum_{n=1}^N \alpha_n y_n w^T x_n \right]\end{aligned}\tag{2.17}$$

Weil  $\sum_{n=1}^N \alpha_n y_n = 0$  (Gleichung 2.16) gilt fällt der Term  $\sum_{n=1}^N \alpha_n y_n b$  weg:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} w^T w - \left[ - \sum_{n=1}^N \alpha_n + \sum_{n=1}^N \alpha_n y_n w^T x_n \right]\tag{2.18}$$

Vergleicht man den Term  $\sum_{n=1}^N \alpha_n y_n w^T x_n$  mit dem Ergebnis von Gleichung 2.15 erkennt man, dass  $\sum_{n=1}^N \alpha_n y_n w^T x_n = w^T w$  gilt. Dies kann ausgeschrieben werden als:

$$\mathcal{L}(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^M y_n y_m \alpha_n \alpha_m x_n^T x_m\tag{2.19}$$

Gleichung 2.19 beschreibt das Optimierungsproblem ohne Abhängigkeit von  $w$  und

$b$ , wir haben jetzt also eine Maximierung für  $\alpha$  mit Nebenbedingungen:

$$\max_{\alpha} \quad \mathcal{L}(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^M y_n y_m \alpha_n \alpha_m x_n^T x_m \quad (2.20a)$$

$$\text{mit} \quad \alpha_n \geq 0 \text{ für } n = 1..N \quad (2.20b)$$

$$\sum_{n=1}^N \alpha_n y_n = 0 \text{ für } n = 1..N \quad (2.20c)$$

Bei dem in Gleichung 2.20 beschriebene Problem handelt es sich um ein Quadratic Programming Problem, ersichtlich an dem Term  $x_n^T x_m$ , welches beispielsweise mittels eines Quadratic Programming Solvers gelöst werden kann. Als Ergebnis erhält man einen Vektor  $\alpha$  mit allen  $\alpha_n$ . Durch Einsetzen in  $w = \sum_{n=1}^N \alpha_n y_n x_n$  kann  $w$  bestimmt werden.

Betrachtet man den Ergebnisvektor  $\alpha$  so wird man feststellen, dass sehr viele Werte 0 ergeben. In Gleichung 2.14a befindet sich der Term  $\alpha_n (y_n(w^T x_n + b) - 1)$ . Der Term  $(y_n(w^T x_n + b) - 1)$  kann als Schlupf bezeichnet werden. Das Produkt von Schlupf und  $\alpha_n$  kann nur 0 werden wenn entweder der Schlupf 0 ist oder  $\alpha_n$ . Umgekehrt bedeutet dies, dass alle Vektoren, die einen minimalen Abstand zu der Trennebene haben, ein  $\alpha_n \neq 0$  aufweisen. Vektoren, die diese Bedingung erfüllen, werden Stützvektoren genannt.

Mit dieser Erkenntnis kann Gleichung 2.15 erneut analysiert werden:

$$w = \sum_{n=1}^N \alpha_n y_n x_n \quad (2.21)$$

Weil nur Stützvektoren ein  $\alpha_n \neq 0$  aufweisen und somit auch nur Stützvektoren einen Beitrag zu  $w$  leisten, kann Gleichung 2.21 stark vereinfacht werden:

$$w = \sum_{n \text{ ist Stützvektor}} \alpha_n y_n x_n \quad (2.22)$$

Der Gewichtsvektor  $w$  hängt also lediglich von den Stützvektoren ab, deren Anzahl in der Regel gering ist.

Noch offen ist die Bestimmung des Bias  $b$ . Weil für Stützvektoren  $y_n(w^T x_n + b) = 1$  gilt (Gleichung 2.5) kann der Bias  $b$  aus jedem beliebigen Stützvektor bestimmt werden:

$$\begin{aligned} b &= \frac{1}{y_n} - w^T x_n = \\ &= y_n - w^T x_n \end{aligned} \quad (2.23)$$

## 2.4 Lösung mittels Quadratic Programming Solver

Weil es sich bei dem in Gleichung 2.20 beschriebene Optimierungsproblem um ein Quadratic Programming Problem handelt kann dieses auch mittels eines Quadratic Programming Solvers gelöst werden. Die Implementationsdetails für diese Lösungsverfahren werden an dieser Stelle nicht weiter ausgeführt. Für die Anwendung eines Quadratic Programming Solvers muss das Problem in die Standardform von QP Problemen umformuliert werden:

$$\min_x = \frac{1}{2}x^T Qx + cx + d \quad (2.24)$$

$Q \in \mathbb{R}^{n \times n}$  ist eine symmetrische reelle Matrix mit Koeffizienten, die einen quadratischen Beitrag leisten.  $c \in \mathbb{R}$  ist ein Faktor für den linearen Beitrag und  $d \in \mathbb{R}$  ist ein fixer Anteil.

Unter der Voraussetzung, dass  $\max_x f(x) = \min_x (-f(x))$  gilt können wir unser Maximierungsproblem aus Gleichung 2.20 in ein Minimierungsproblem umformen:

$$\min_{\alpha} \mathcal{L}(\alpha) = \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^M y_n y_m \alpha_n \alpha_m x_n^T x_m - \sum_{n=1}^N \alpha_n \quad (2.25)$$

Als nächsten Schritt müssen wir die Koeffizienten aus den Summen extrahieren,  $\alpha$  und  $y$  als Vektor darstellen, das Problem in die Standard-QP-Form bringen und die Nebenbedingungen als Matrizenmultiplikation darstellen:

$$\min_{\alpha} \mathcal{L}(\alpha) = \frac{1}{2} \alpha^T Q \alpha + (-1^T) \alpha \quad (2.26a)$$

$$\text{mit } Q = \begin{bmatrix} y_1 y_1 x_1^T & y_1 y_2 x_1^T & \dots & y_1 y_N x_1^T \\ y_2 y_1 x_2^T & y_2 y_2 x_2^T & \dots & y_2 y_N x_2^T \\ \vdots & \vdots & \vdots & \vdots \\ y_N y_1 x_N^T & y_N y_2 x_N^T & \dots & y_N y_N x_N^T \end{bmatrix} \quad (2.26b)$$

$$\text{für } y^T \alpha = 0 \quad (2.26c)$$

$$0 \leq \alpha \leq \infty \quad (2.26d)$$

An dieser Stelle ist zu bemerken dass es sich bei der Matrix  $Q$  um eine  $N \times N$  Matrix handelt. Für eine große Anzahl an Trainingsdaten ist dies sehr problematisch beziehungsweise nicht mehr lösbar. Hier können andere Optimierungsverfahren wie beispielsweise in Platt (1998) beschrieben verwendet werden.

In dieser Darstellung kann das Problem direkt an ein Quadratic-Programming Solver Framework übergeben werden. Als Ergebnis erhalten wir einen Vektor  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ .

Daraus kann, wie bereits zuvor erwähnt,  $w$  und  $b$  bestimmt werden:

$$w = \sum_{n=1}^N \alpha_n y_n x_n \quad (2.27)$$

Für die Biasberechnung wird ein beliebiger Stützvektor  $x_k$  benötigt. Dieser kann bestimmt werden indem der Ergebnisvektor  $\alpha$  betrachtet wird. Ein Stützvektor  $x_k$  muss  $\alpha_k \neq 0$  aufweisen.

$$b = \frac{1}{y_k} - w^T x_k \quad (2.28)$$

Die SVM ist nun vollständig definiert, neue Eingabevektoren  $x$  können wie folgt klassifiziert werden:

$$y = sign(w^T x + b) \quad (2.29)$$

### 3 Soft-Margin Support Vector Machine

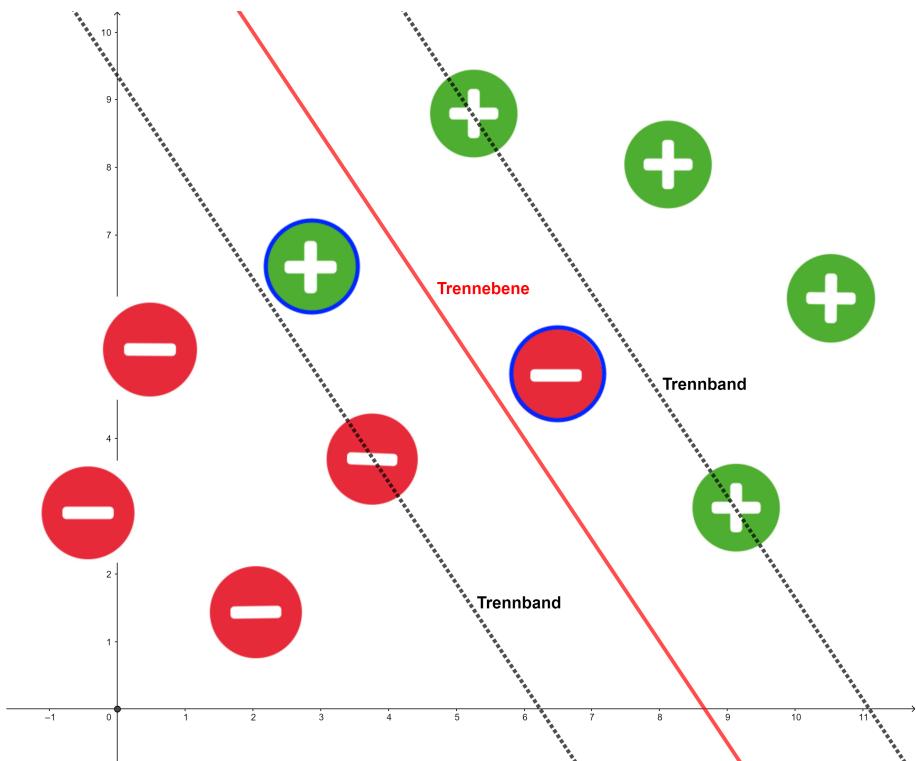


Abbildung 3.1: Die Eingabevektoren können nicht ohne einzelne Fehlklassifizierungen (blau markiert) linear getrennt werden.

In den Überlegungen in Kapitel 2 wurde angenommen, dass alle Eingabevektoren linear separierbar sind ohne dass Fehlklassifikationen auftreten. Dieses Problem kann durch eine Hard-Margin SVM, wie in den Kapiteln zuvor gezeigt, gelöst werden. Trifft diese Annahme für eine Menge von Eingabevektoren nicht mehr zu, wie in Abbildung 3.1 dargestellt, kann der bisher beschriebene Algorithmus keine lineare Trennebene finden. Durch die Einführung von positiven Fehlervariablen  $\xi_n \in \mathbb{R}^K, \xi_n \geq 0$  in Gleichung 2.3 kann dieses Problem umgangen werden und trotzdem eine fehlerbehaftete, lineare Trennebene gefunden werden:

$$w^T x_n + b \geq +1 - \xi_n \quad \text{für } y_n = +1 \quad (3.1a)$$

$$w^T x_n + b \leq -1 + \xi_n \quad \text{für } y_n = -1 \quad (3.1b)$$

Damit ein Eingabevektor  $x_n$  nun, gemäß Gleichung 3.1, falsch klassifiziert werden kann muss der zu dem Vektor gehörende Fehler  $\xi_n$  über 1 steigen. Eine obere Grenze für die Anzahl der aufgetretenen Fehler kann somit beschrieben werden durch  $\sum_{n=1}^N \xi_n$ .

Basierend auf dieser Grundlage kann eine Kostenfunktion  $E$  basierend auf der Anzahl der Fehler formuliert werden:

$$E = C \left( \sum_{n=1}^N \xi_n \right) \quad (3.2)$$

Der Parameter  $C \in \mathbb{R}, C \geq 0$  bestimmt die Höhe der Bestrafung von Fehlern und kann frei gewählt werden.

Erweitert man die zu minimierende Funktion  $\frac{1}{2} w^T w$  aus Gleichung 2.13 um die Kostenfunktion  $C(\sum_{n=1}^N \xi_n)$  so erhält man ein neues Optimierungsproblem:

$$\min_w \quad \frac{1}{2} w^T w + C \left( \sum_{n=1}^N \xi_n \right) \quad (3.3a)$$

$$\text{mit} \quad y_n (w^T x_n + b) - 1 \geq 0 \quad \text{für } n = 1..N \quad (3.3b)$$

Wendet man die in Abschnitt 2.3 beschriebenen Schritte auf das in Gleichung 3.3 beschriebene Problem an erhält man folgendes Optimierungsproblem:

$$\max_{\alpha} \quad \mathcal{L}(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^M y_n y_m \alpha_n \alpha_m x_n^T x_m \quad (3.4a)$$

$$\text{mit} \quad 0 \leq \alpha_n \leq C \quad \text{für } n = 1..N \quad (3.4b)$$

$$\sum_{n=1}^N \alpha_n y_n = 0 \quad \text{für } n = 1..N \quad (3.4c)$$

Sehr bemerkenswert hier ist, dass der einzige Unterschied zu Gleichung 2.20 die Beschränkung der Lagrange Multiplikatoren durch  $C$  ist. Umgekehrt bedeutet dies, dass eine Hard-Margin SVM durch Gleichung 3.4 beschrieben werden kann wenn der Parameter  $C$  sehr groß gewählt wird.

## 4 Vergleich Soft-Margin und Hard-Margin Support Vector Machine

Ein großer Vorteil der Soft-Margin SVM ist die Verminderung des Einflusses von Ausreißern auf die Trenngrenze. Bei der Hard-Margin SVM kann mitunter ein Ausreißer, der näher als alle anderen Punkte bei der anderen Klasse liegt, die Lage der Trenngrenze bestimmen. Die so bestimmte Trenngrenze liegt mitunter viel näher bei der abzugrenzenden Klasse als eine optimale Trenngrenze liegen würde. Ein Beispiel für eine solche Situation ist in Abbildung 4.1 dargestellt. Die Soft-Margin SVM hat die Möglichkeit Fehlklassifikationen zuzulassen und kann dadurch bessere liegende Trennebenen finden die zu stabileren Klassifikationsergebnissen für neue Daten führt.

Aufgrund des zuvor genannten Verhaltens tendiert die Hard-Margin SVM im Allgemeinen zu Overfitting. Aus diesem Grund sollte die Soft-Margin SVM bevorzugt werden und der Parameter  $C$  abhängig von den gegebenen Trainingsdaten gewählt werden.

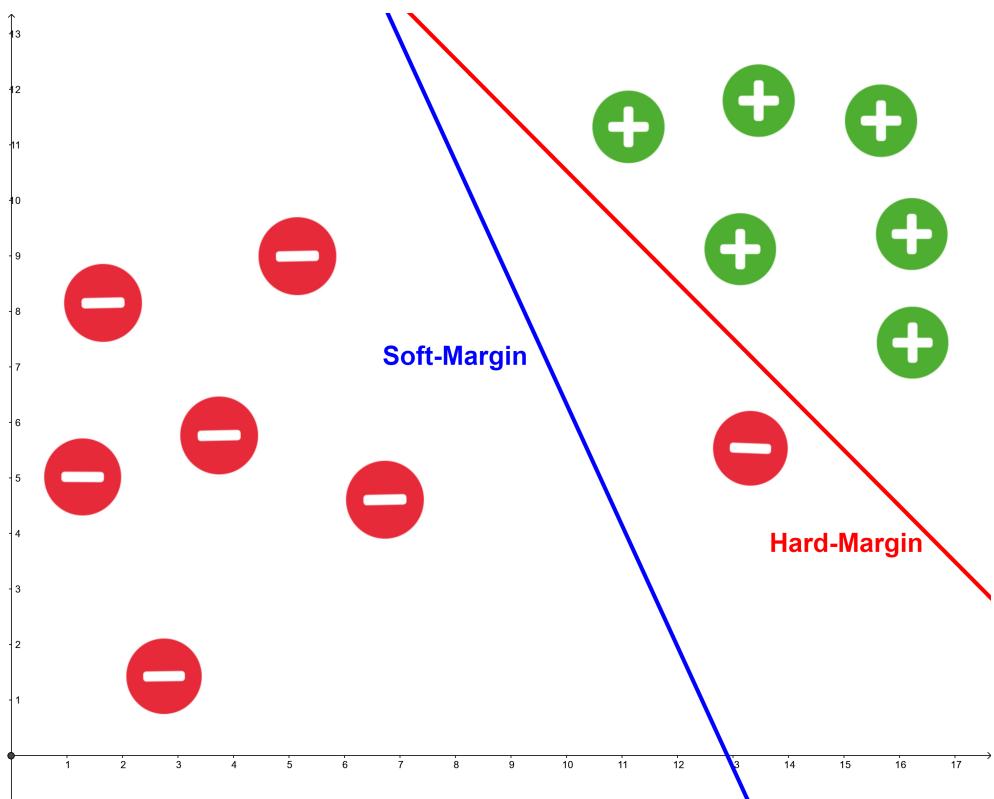


Abbildung 4.1: Die Hard-Margin SVM trennt die Klassen zwar ohne Fehler, die Lage der Trennebene ist allerdings sehr nahe bei den grünen Punkten. Dadurch werden neue Eingabevektoren, die nahe bei der Trengrenze liegen, mit großer Wahrscheinlichkeit falsch klassifiziert. Die Soft-Margin SVM „opfert“ eine Fehlklassifikation für eine viel bessere, stabilere Lage der Trengrenze.

# 5 Nichtlineare Trennung

Wird eine nichtlineare Trennung von Eingabevektoren gewünscht lässt sich dies nicht direkt durch eine SVM realisieren, da eine SVM in ihrer Grundform ausschließlich linear trennen kann. Werden die Eingabevektoren allerdings mittels einer Funktion so transformiert, dass diese linear trennbar sind, kann eine SVM auch nichtlinear trennbare Eingabevektoren trennen. In Abschnitt 5.1 wird zuerst eine Intuition für die Transformation aufgebaut. Diese Intuition wird anschließend in Abschnitt 5.2 weiter formalisiert und verallgemeinert.

## 5.1 Transformation der Problemstellung

Sind die Eingabevektoren nicht linear trennbar können diese mittels einer Transformation  $\Phi(x) : \mathbb{R}^K \rightarrow \mathbb{R}^L$  in einen Raum, in dem diese trennbar sind, transformiert werden. Beispiele für solche Transformationen sind in Abbildung 5.1 und Abbildung 5.2 dargestellt.

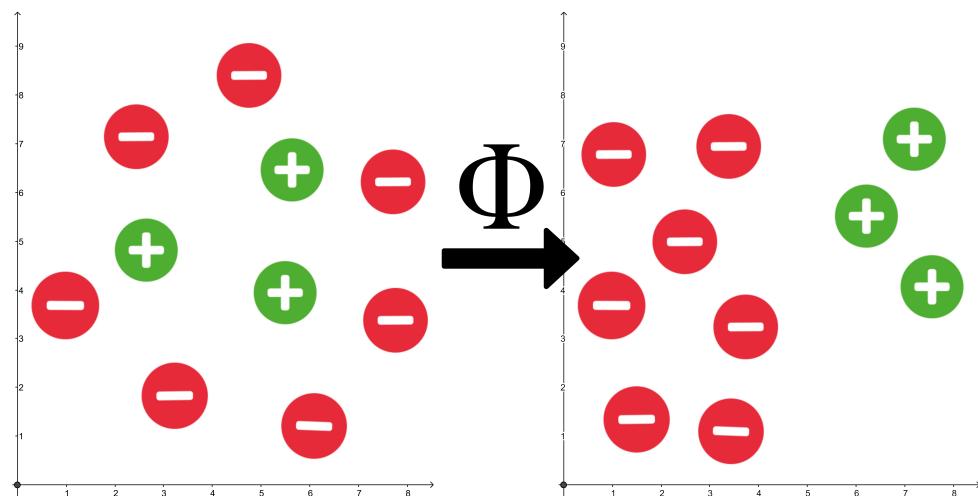


Abbildung 5.1: Die Eingabevektoren werden mittels einer Funktion  $\Phi(x)$  transformiert. Die transformierten Vektoren sind linear trennbar.

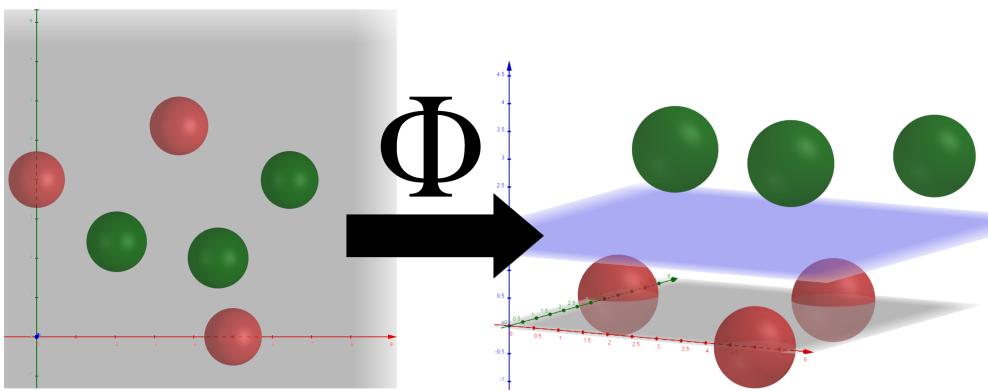


Abbildung 5.2: Die Eingabevektoren werden mittels einer Funktion  $\Phi(x)$  in einen höherdimensionalen Raum transformiert. Die transformierten Vektoren sind durch eine Ebene linear trennbar.

Wird statt allen Eingabevektoren  $x$  deren transformierte Vektoren  $\Phi(x)$  in Gleichung 3.4 eingesetzt erhält man:

$$\max_{\alpha} \quad \mathcal{L}(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^M y_n y_m \alpha_n \alpha_m \Phi(x_n)^T \Phi(x_m) \quad (5.1a)$$

$$\text{mit} \quad 0 \leq \alpha_n \leq C \text{ für } n = 1..N \quad (5.1b)$$

$$\sum_{n=1}^N \alpha_n y_n = 0 \text{ für } n = 1..N \quad (5.1c)$$

Wie in Gleichung 5.1 ersichtlich sind die einzigen Zusatzkosten gegenüber einer Soft-Margin SVM die Berechnungen der mitunter höherdimensionalen Skalarprodukte  $\Phi(x)^T \Phi(x)$ . Die Anzahl der Lagrange-Faktoren  $\alpha$  und die Dimension der Q-Matrix aus Gleichung 5.16 bleibt gleich weil diese nur von der Anzahl der Eingabevektoren und nicht von der Dimension derer abhängig sind.

Die Stützvektoren dieser Methode befinden sich in dem neuen Raum  $\mathbb{R}^L$ , weil auch nur die transformierten Eingabevektoren betrachtet werden. Durch die, mitunter nichtlineare, Transformation  $\Phi$  ist es so möglich mittels einer SVM nichtlineare Trennungen durchführen zu können.

Das Problem dieses Ansatzes ist die Wahl der Transformationsfunktion  $\Phi(x)$ . Im Normalfall ist eine solche Funktion nicht bekannt. Trotzdem ist die Erkenntnis, dass die Dimension der Vektoren das Optimierungsproblem nicht groß beeinflusst und dass durch eine Transformation nichtlineare Trenngrenzen bestimmt werden können, äußerst relevant für die weitere Betrachtung. In Abschnitt 5.2 wird beschrieben wie die Zusatzkosten der Skalarprodukte auch umgangen werden können.

## 5.2 Kernel Trick

Wie in Abschnitt 5.1 beschrieben können die Eingabevektoren mittels einer Funktion  $\Phi$  in einen anderen, beliebigen Raum transformiert werden. In diesem Kapitel wird diese Idee der Transformation verallgemeinert betrachtet.

Für die Betrachtung wird das Optimierungsproblem und die Formeln für  $w$  und  $b$  für die Soft-Margin SVM so abgeändert, dass statt allen Eingabevektoren  $x$  deren transformierte Form  $z = \Phi(x)$  eingesetzt wird:

$$\max_{\alpha} \quad \mathcal{L}(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^M y_n y_m \alpha_n \alpha_m z_n^T z_m \quad (5.2a)$$

$$\text{mit} \quad 0 \leq \alpha_n \leq C \text{ für } n = 1..N \quad (5.2b)$$

$$\sum_{n=1}^N \alpha_n y_n = 0 \text{ für } n = 1..N \quad (5.2c)$$

Für die Berechnung von  $w$  und  $b$  ergibt sich:

$$w = \sum_{n=1}^N \alpha_n y_n z_n \quad (5.3)$$

$$b = \frac{1}{y_k} - w^T z_k \quad (5.4)$$

Wie bereits zuvor erwähnt hängt das Problem ausschließlich von den Skalarprodukten der Eingabevektoren, die jetzt in ihrer transformierten Form verwendet werden, ab. Werden die Eingabevektoren mittels  $\Phi$  in einen sehr hochdimensionalen, mitunter unendlich dimensionalen, Raum transformiert kann die Berechnung des Skalarprodukts sehr aufwändig bis gar nicht berechnbar werden.

Durch die Einführung einer Kernel-Funktion  $K(x, x') = z_1^T z_2 = \Phi(x)^T \Phi(x')$ , die das Skalarprodukt der transformierten Eingabevektoren berechnet ohne die Eingabevektoren tatsächlich in den neuen Raum zu transformieren, kann das Problem der Berechnung von hochdimensionalen Skalarprodukten umgangen werden.

### 5.2.1 Beispiel Kernel-Funktion

Die Idee einer Kernel-Funktion wird nun anhand eines einfachen Beispiels gezeigt. Folgende Kernel-Funktion für  $x, x' \in \mathbb{R}^2$  sei gegeben:

$$\begin{aligned} K(x, x') &= (1 + x^T x')^2 = \\ &= (1 + x_1 x'_1 + x_2 x'_2)^2 = \\ &= 1 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1 x'_1 + 2x_2 x'_2 + 2x_1 x'_1 x_2 x'_2 \end{aligned} \quad (5.5)$$

Diese Kernel-Funktion scheint auf den ersten Blick nicht, wie zuvor definiert, einem Skalarprodukt der transformierten Vektoren  $\Phi(x)$  und  $\Phi(x')$  zu entsprechen.

Angenommen die verwendete Transformationsfunktion  $\Phi$  entspräche:

$$\Phi(x) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2) \quad (5.6)$$

Angewandt auf die Vektoren  $x$  und  $x'$ :

$$\begin{aligned} \Phi(x) &= (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2) \\ \Phi(x') &= (1, x_1'^2, x_2'^2, \sqrt{2}x_1', \sqrt{2}x_2', \sqrt{2}x_1'x_2') \\ \Phi(x)^T \Phi(x') &= 1 + x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_1' x_2 x_2' \end{aligned} \quad (5.7)$$

Werden die Ergebnisse von Gleichung 5.5 und Gleichung 5.7 verglichen erkennt man, dass die Kernel-Funktion  $(1 + x^T x')^2$  tatsächlich dem Skalarprodukt der mit  $\Phi$  transformierten Vektoren  $x$  und  $x'$  entspricht.

$\Phi$  transformiert in diesem Beispiel 2-dimensionale Vektoren nach  $\mathbb{R}^6$ . Mittels der Kernel-Funktion kann das Skalarprodukt der transformierten Vektoren durch  $K(x, x') = (1 + x^T x')^2$  berechnet werden ohne dass die Vektoren  $x$  und  $x'$  tatsächlich in den neuen Raum transformiert werden müssen.

In dem Beispiel wurde die Berechnung von 6-dimensionalen Skalarprodukten durchgeführt durch die Berechnung mittels der Kernel-Funktion, die lediglich 2-dimensionale Skalarprodukte berechnen muss. Führt die Funktion  $\Phi$  eine Transformation in eine viel höhere oder unendlich hohe Dimension aus ist der Vorteil einer solchen Kernel-Funktion sofort ersichtlich.

### 5.2.2 Polynomieller Kernel

Die in Unterabschnitt 5.2.1 verwendete Kernel-Funktion kann verallgemeinert werden. Seien die Eingabevektoren  $x \in \mathbb{R}^d$  und die Transformationsfunktion  $\Phi : \mathbb{R}^d \rightarrow \mathbb{Z}$  ein Polynom der Ordnung  $Q$ , dann kann eine Kernel-Funktion wie folgt definiert werden:

$$\begin{aligned} K(x, x') &= (1 + x^T x')^Q = \\ &= (1 + x_1 x_1' + x_2 x_2' + \dots + x_d x_d')^Q \end{aligned} \quad (5.8)$$

Die in Gleichung 5.8 beschriebene Kernel-Funktion wird polynomieller Kernel genannt. Weil durch das Polynom eine Vielzahl an Faktoren entsteht kann zur Kompenstation dieser Faktoren die Kernel-Funktion mit den Skalierungsfaktoren  $a$  und  $b$  erweitert werden:

$$K(x, x') = (ax^T x' + b)^Q \quad (5.9)$$

Mittels dieser Kernel-Funktion kann die Berechnung eines Skalarprodukts eines beliebig großen Polynoms des Grades  $Q$  durch die Berechnung eines Skalarprodukts

in  $\mathbb{R}^d$  und der Exponentiation mit  $Q$  durchgeführt werden ohne dass die Vektoren tatsächlich in den Polynomraum transformiert werden müssen.

### 5.2.3 Radial Basis Function Kernel

Eine weitere mögliche Kernel-Funktion ist der Radial Basis Function (RBF) Kernel:

$$K(x, x') = \exp \gamma \|x - x'\|^2 \quad (5.10)$$

Hierbei ist  $\gamma$  ein frei wählbarer Parameter. Die zu diesem Kernel zugehörige Transformationsfunktion  $\Phi$  bildet in einen unendlich dimensionalen Raum ab. Dies kann wie folgt gezeigt werden (für den einfachsten Fall):

$$\begin{aligned} K(x, x') &= \exp(-(x - x')^2) = \\ &= \exp(-x^2 + 2xx' - x'^2) = \\ &= \exp(-x^2) \exp(2xx') \exp(-x'^2) = \\ &= \exp(-x^2) \sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!} \exp(-x'^2) \end{aligned} \quad (5.11)$$

Durch die Taylorexpansion von  $\exp(2xx')$  wird die Unendlichkeit dieses Raums sichtbar. Das Ergebnis von Gleichung 5.11 hat bereits die für ein Skalarprodukt benötigte Symmetrie von  $\exp(-x^2) \cdot \exp(-x'^2)$  und  $(x)^k \cdot (x')^k$ . Die Anteile  $\frac{2^k}{k!}$  können gleichmäßig auf  $x$  und  $x'$  aufgeteilt werden indem je die Wurzel der Anteile zu  $x$  und  $x'$  multipliziert werden. Damit wurde gezeigt dass der RBF Kernel das Skalarprodukt in einem unendlich dimensionalen Raum berechnet.

### 5.2.4 SVM Definition mit Kernel

Ein Input  $x$ , transformiert mit einer Funktion  $z = \Phi(x)$ , kann mit einer SVM wie folgt klassifiziert werden:

$$y(x) = \text{sign}(w^T z + b) \quad (5.12)$$

Diese Definition setzt voraus, dass die Funktion  $\Phi$  bekannt ist. Das Ziel der neuen Definition ist die Vermeidung von einzeln auftretenden transformierten Eingabevektoren  $z$  in den Formeln um das gesamte Problem mittels einer Kernel-Funktion  $K(x, x')$  ausdrücken zu können ohne die Funktion  $\Phi$  tatsächlich kennen zu müssen.

$$w = \sum_{z_n \text{ ist SV}} \alpha_n y_n z_n \quad (5.13)$$

Setzt man Gleichung 5.13 in Gleichung 5.12 ein, erhält man:

$$\begin{aligned} y(x) &= \text{sign}\left(\sum_{\alpha_n > 0} \alpha_n y_n z_n^T z + b\right) = \\ &= \text{sign}\left(\sum_{\alpha_n > 0} \alpha_n y_n K(x_n, x) + b\right) \end{aligned} \quad (5.14)$$

Für  $b$  ergibt sich durch Einsetzen von Gleichung 5.13 für einen beliebigen Stützvektor  $x_k$ :

$$\begin{aligned} b &= \frac{1}{y_k} - w^T z_k = \\ &= \frac{1}{y_k} - \sum_{\alpha_n > 0} \alpha_n y_n K(x_n, x_k) = \\ &= y_k - \sum_{\alpha_n > 0} \alpha_n y_n K(x_n, x_k) = \end{aligned} \quad (5.15)$$

Die SVM ist nun vollständig definiert durch die Verwendung der Kernel-Funktion ohne dass die Transformationsfunktion  $\Phi$  bekannt sein muss. Es wird keine einzige tatsächliche Transformation eines Eingabevektors mittels  $\Phi$  benötigt und es können beliebig dimensionale Räume durch die Verwendung von entsprechenden Kernel-Funktionen verwendet werden.

Da die Funktion  $\Phi$  überhaupt nicht bekannt sein muss impliziert das, dass beliebige Kernel-Funktionen verwendet werden können. Das ist in der Tat möglich, allerdings müssen die Kernel-Funktionen bestimmte Eigenschaften erfüllen. Diese werden in Unterabschnitt 5.2.5 erläutert.

Das in Abschnitt 2.4 beschriebene Lösungsverfahren ist für das neue Problem nach wie vor gültig, lediglich die  $Q$ -Matrix muss entsprechend angepasst werden. Es muss nur statt  $x_n^T x_m$   $K(x_n, x_m)$  eingesetzt werden:

$$\min_{\alpha} \quad \mathcal{L}(\alpha) = \frac{1}{2} \alpha^T Q \alpha + (-1^T) \alpha \quad (5.16a)$$

$$\text{mit } Q = \begin{bmatrix} y_1 y_1 K(x_1, x_1) & y_1 y_2 K(x_1, x_2) & \dots & y_1 y_N K(x_1, x_N) \\ y_2 y_1 K(x_2, x_1) & y_2 y_2 K(x_2, x_2) & \dots & y_2 y_N K(x_2, x_N) \\ \vdots & \vdots & \vdots & \vdots \\ y_N y_1 K(x_N, x_1) & y_N y_2 K(x_N, x_2) & \dots & y_N y_N K(x_N, x_N) \end{bmatrix} \quad (5.16b)$$

$$\text{für } y^T \alpha = 0 \quad (5.16c)$$

$$0 \leq \alpha \leq \infty \quad (5.16d)$$

Mit dieser Änderung kann das Problem wieder an ein Quadratic Programming Framework übergeben werden um die  $\alpha$  Werte zu bestimmen. Die SVM lässt sich somit in Kombination mit Kernel-Funktionen auf beliebige binäre Klassifikationsprobleme anwenden.

### 5.2.5 Anforderungen an eine Kernel-Funktion

Wie in den Kapiteln zuvor gezeigt wurde gibt es eine Vielzahl von verschiedenen Kernel-Funktionen. Wenn eine neue Funktion als Kernel-Funktion verwendet werden

soll muss diese einem Skalarprodukt in einem Raum entsprechen. Um dies überprüfen zu können gibt es zwei verschiedene Ansätze:

- Für eine vermutlich richtige Kernel-Funktion wird konstruktiv versucht die zugehörige Transformationsfunktion  $\Phi$  zu bestimmen. Diese Vorgehensweise wurde in Unterabschnitt 5.2.1 durchgeführt.
- Der Kernel  $K(x, x')$  ist ein gültiger Kernel wenn die Funktion  $K(x, x')$  symmetrisch ist und die Matrix

$$K = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots & K(x_1, x_N) \\ K(x_2, x_1) & K(x_2, x_2) & \dots & K(x_2, x_N) \\ \vdots & \vdots & \vdots & \vdots \\ K(x_N, x_1) & K(x_N, x_2) & \dots & K(x_N, x_N) \end{bmatrix}$$

positiv semi-definit ist für jedes beliebige  $x_1..x_N$ . Die Forderung der positiv semi-definiten Matrix wird auch Satz von Mercer genannt und garantiert bei Erfüllung dass eine Funktion  $\Phi$  existiert die in einen Raum abbildet dessen Skalarprodukte durch die Kernel-Funktion beschrieben werden können.

# 6 Pseudocode und Beispiele

In diesem Kapitel werden Pseudocodes von verschiedensten Varianten von SVMs präsentiert, von einer Hard-Margin SVM bis zu einer Soft-Margin SVM mit RBF-Kernel. Weiters werden anhand von Beispielen die Funktionsweisen und Eigenschaften der verschiedenen SVMs anschaulich demonstriert.

## 6.1 Hard-Margin SVM Pseudocode

Es folgt der Pseudocode mit Anmerkungen für eine Hard-Margin Support Vector Machine.

Hard-Margin SVM	Zeile
Initialisiere $\mathbf{x}, \mathbf{y}$	1
$\mathbf{Q} = (\mathbf{y}\mathbf{y}^T)\mathbf{K}$	2
$\mathbf{c} = (-1, -1, \dots, -1)^T$	3
$\mathbf{A} = \text{diag}(-1, -1, \dots, -1)$	4
$\mathbf{b} = (0, 0, \dots, 0)^T$	5
$\mathbf{A}_{\text{eq}} = \mathbf{y}^T$	6
$b_{\text{eq}} = 0$	7
$\mathbf{l}\mathbf{b} = (0, 0, \dots, 0)^T$	8
$\mathbf{u}\mathbf{b} = C * (1, 1, \dots, 1)^T$	9
$\alpha = QPSolver(\mathbf{Q}, \mathbf{c}, \mathbf{A}, b, \mathbf{A}_{\text{eq}}, \mathbf{b}_{\text{eq}})$	10
$\mathbf{w} = \sum_{n=SV} \alpha_n y_n x_n$	11
$bias = \frac{1}{y_n} - \mathbf{w}^T x_n$	12

### Anmerkungen

- Zeile 1: Initialisieren von Werte- und Klassenvektoren
- Zeile 2: Berechnen der Matrix Q
- Zeile 3: Berechnen von c
- Zeile 4, 5: Berechnen der Ungleichheitsbedingungen
- Zeile 6, 7: Berechnen der Gleichheitsbedingungen
- Zeile 8: Lösen mittels Quadratic Programming
- Zeile 9: Berechnung Gewichte mit Stützvektoren
- Zeile 10: Berechnung bias mit beliebigem Stützvektor

### 6.1.1 Berechnung von $\mathbf{K}$

Im Folgenden der Pseudocode für die Berechnung von  $\mathbf{K}$ .

K Berechnung	Zeile
Initialisiere $\mathbf{x}$	1
For $i = 1$ To $N$	2
For $j = 1$ To $N$	3
$\mathbf{K}(i, j) = x_i \cdot x_j$	4
Ende For	5
Ende For	6

$N$  ist hierbei die Anzahl der Eingabevektoren.

## 6.2 Hard-Margin SVM - Beispiel

Mit einer Hard-Margin SVM können linear separierbare Punkte getrennt werden. Abbildung 6.1 zeigt ein Beispiel.

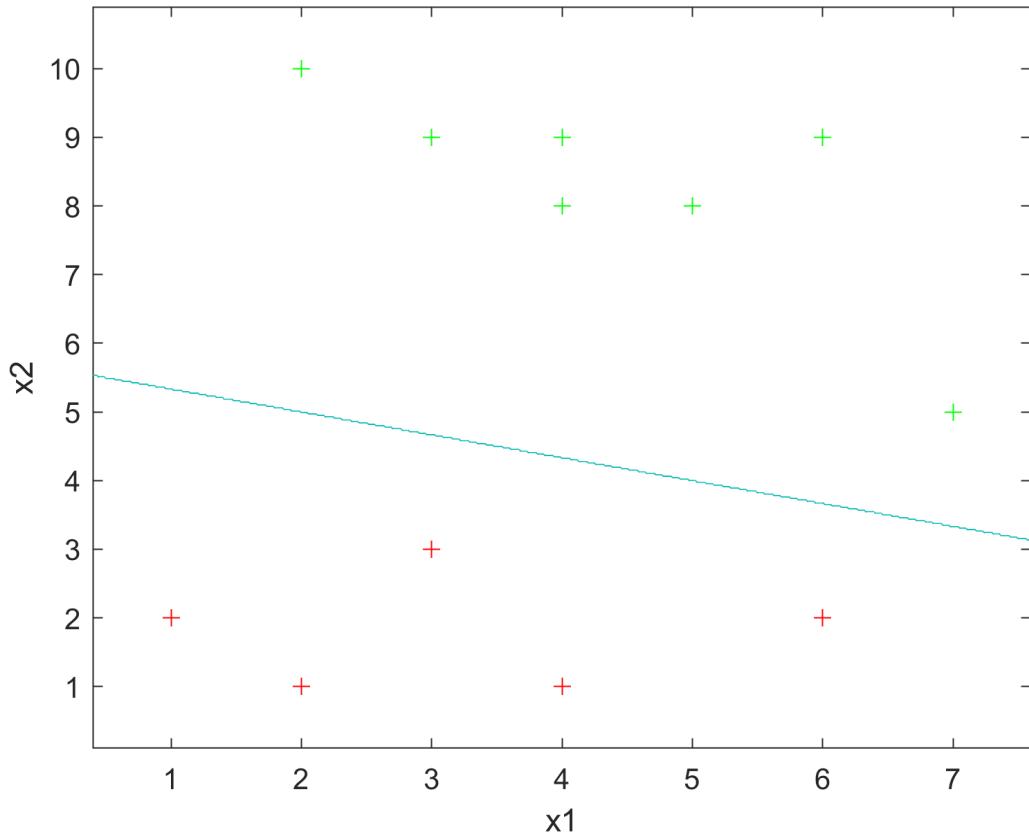


Abbildung 6.1: Linear separierbarer Datensatz mit Grenze von Hard-Margin SVM

Wie ersichtlich hat die Grenze, die „hard margin“ einen Mindestabstand zu allen Punkten. Abbildung 6.2 zeigt den Vergleich zwischen einer Hard-Margin SVM, einem Algorithmus mit Pseudoinversen und  $\alpha$ -LMS.

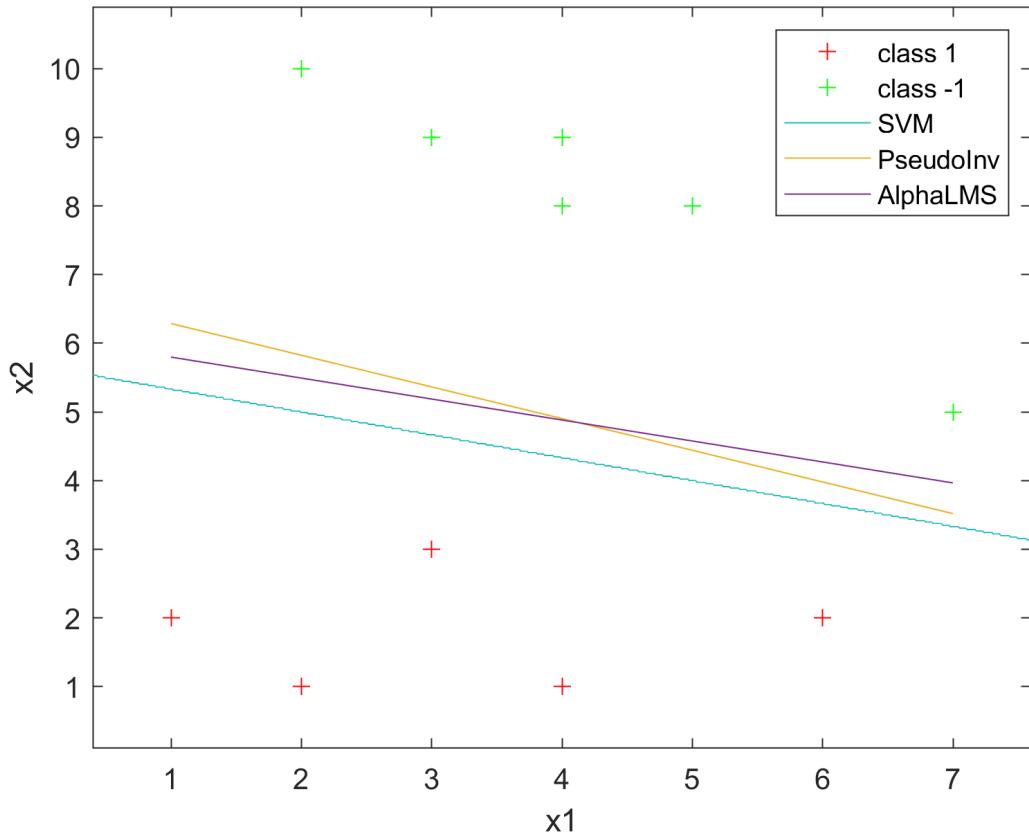


Abbildung 6.2: Linear separierbarer Datensatz mit Grenze von Hard-Margin SVM, Pseudoinverser und  $\alpha$ -LMS

Alle Entscheidungsgrenzen trennen den Datensatz sauber. Im Gegensatz zur Hard-Margin SVM ist bei den anderen Algorithmen der Abstand der Punkte zur Grenze unerheblich.

### 6.3 Soft-Margin SVM Pseudocode

Der Pseudocode für die Soft-Margin SVM unterscheidet sich nur geringfügig von der für die Hard-Margin SVM. Man erweitert den Algorithmus um einen Bestrafungsparameter  $C$  und die untere und obere Grenze.

Soft-Margin SVM	Zeile
Initialisiere $\mathbf{x}, \mathbf{y}, C$	1
$\mathbf{Q} = (\mathbf{y}\mathbf{y}^T)\mathbf{K}$	2
$\mathbf{c} = (-1, -1, \dots, -1)^T$	3
$\mathbf{A} = diag(-1, -1, \dots, -1)$	4
$\mathbf{b} = (0, 0, \dots, 0)^T$	5
$\mathbf{A}_{eq} = \mathbf{y}^T$	6
$b_{eq} = 0$	7
$\mathbf{lb} = (0, 0, \dots, 0)^T$	8
$\mathbf{ub} = C * (1, 1, \dots, 1)^T$	9
$\alpha = QPSolver(\mathbf{Q}, \mathbf{c}, \mathbf{A}, \mathbf{b}, \mathbf{A}_{eq}, \mathbf{b}_{eq}, \mathbf{lb}, \mathbf{ub})$	10
$\mathbf{w} = \sum_{n=SV} \alpha_n y_n x_n$	11
$bias = \frac{1}{y_n} - \mathbf{w}^T x_n$	12

### Anmerkungen

- Zeile 1: Initialisieren von Werte- und Klassenvektoren und Bestrafungsparameter C
- Zeile 2: Berechnen der Matrix Q
- Zeile 3: Berechnen von c
- Zeile 4, 5: Berechnen der Ungleichheitsbedingungen
- Zeile 6, 7: Berechnen der Gleichheitsbedingungen
- Zeile 8: untere Grenze  
(kann auch  $(-\infty, -\infty, \dots, -\infty)$  gewählt werden)
- Zeile 9: Berechnung obere Grenze mit C
- Zeile 10: Lösen mittels Quadratic Programming
- Zeile 11: Berechnung Gewichte mit Stützvektoren
- Zeile 12: Berechnung bias mit beliebigem Stützvektor

## 6.4 Soft-Margin SVM Beispiel

Im Gegensatz zur Hard-Margin SVM kann bei der Soft-Margin SVM der Mindestabstand, oder genauer der Bestrafungsfaktor für Abweichungen von diesem, eingestellt werden. So kann mit einer geringen Bestrafung auch bei nichtlinear separierbaren Datensätzen ein gutes Ergebnis erzielt werden, vorausgesetzt die Klassen sind nur leicht durchmischt. Ein Beispiel, mit einer Grenze von einer Hard-Margin SVM zum Vergleich, ist in Abbildung 6.3 dargestellt.

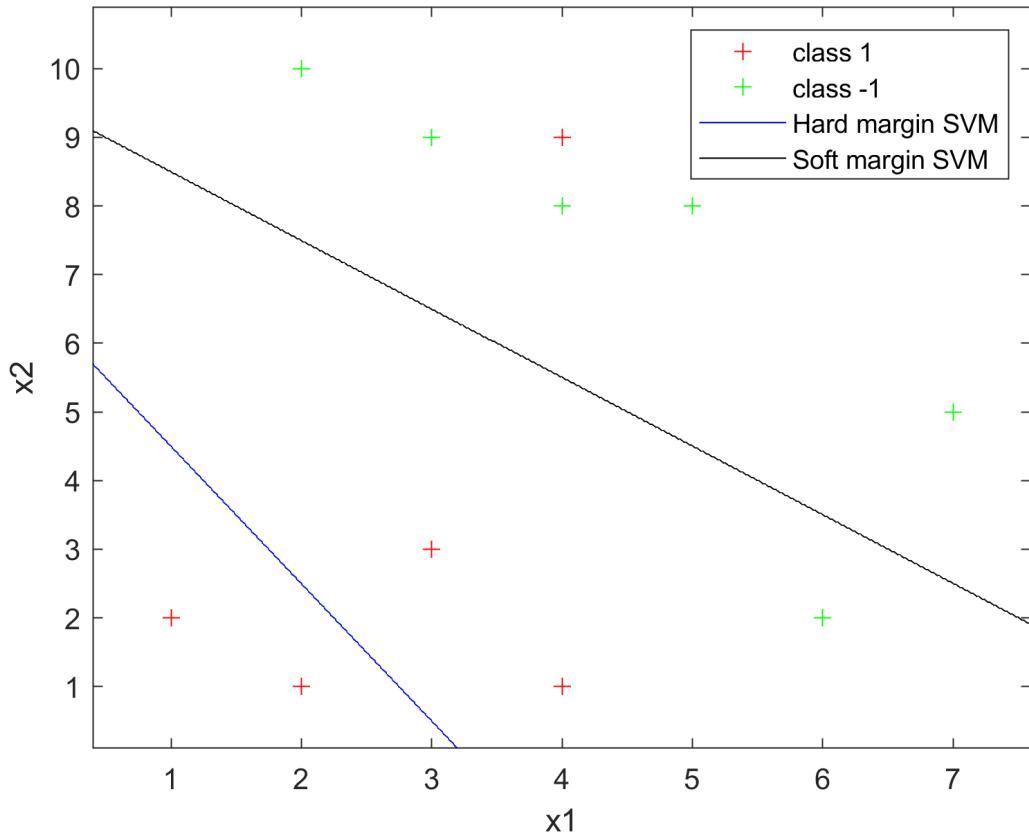


Abbildung 6.3: Nichtlinear separierbarer Datensatz mit Grenze von Hard-Margin SVM und Soft-Margin SVM

Es ist ersichtlich, dass die Soft-Margin SVM trotz der zwei Punkte, die eine lineare Trennung verhindern, eine gute Grenze zieht. Die beiden Ausreißer werden hierbei quasi ignoriert. Anders bei der Hard-Margin SVM, die durch die Ausreißer stark beeinträchtigt wird und keine Trenngrenze findet. Wird der Bestrafungsparameter  $C$  sehr hoch eingestellt, so verhält sich eine Soft-Margin SVM identisch zu einer Hard-Margin SVM.

## 6.5 Kernel-Trick - Pseudocode

Bei der Erweiterung mit dem Kerneltrick kann die Soft-Margin SVM als Basis gewählt werden. Im Algorithmus ändert sich hierbei nur die Berechnung von  $\mathbf{K}$  (siehe Anmerkung) und die des bias. Weiters wird anstatt des Gewichtsvektors  $\mathbf{w}$  der Vektor  $\alpha$  zurückgegeben und dessen positiven Teile  $\alpha > 0$  für die Klassifikation verwendet.

Soft-Margin SVM mit Kernel-Trick	Zeile
Initialisiere $\mathbf{x}, \mathbf{y}, C$	1
$\mathbf{Q} = (\mathbf{y}\mathbf{y}^T)\mathbf{K}$	2
$\mathbf{c} = (-1, -1, \dots, -1)^T$	3
$\mathbf{A} = \text{diag}(-1, -1, \dots, -1)$	4
$\mathbf{b} = (0, 0, \dots, 0)^T$	5
$\mathbf{A}_{\text{eq}} = \mathbf{y}^T$	6
$b_{eq} = 0$	7
$\mathbf{lb} = (0, 0, \dots, 0)^T$	8
$\mathbf{ub} = C * (1, 1, \dots, 1)^T$	9
$\alpha = QPSolver(\mathbf{Q}, \mathbf{c}, \mathbf{A}, b, \mathbf{A}_{\text{eq}}, \mathbf{b}_{eq}, \mathbf{lb}, \mathbf{ub})$	10
$bias = y_k - \sum_{\alpha_n > 0} \alpha_n y_n KF(x_n, x_k)$	11

Anmerkungen
Zeile 1: Initialisieren von Werte- und Klassenvektoren und Bestrafungsparameter C
Zeile 2: Berechnen der Matrix Q
Zeile 3: Berechnen von c
Zeile 4, 5: Berechnen der Ungleichheitsbedingungen
Zeile 6, 7: Berechnen der Gleichheitsbedingungen
Zeile 8: untere Grenze (kann auch $(-\infty, -\infty, \dots, -\infty)$ gewählt werden)
Zeile 9: Berechnung obere Grenze mit C
Zeile 10: Lösen mittels Quadratic Programming
Zeile 11: Berechnung bias $x_k$ ist ein beliebiger Stützvektor, $y_k$ die dazugehörige Klasse. Aufgrund von Rechenenauigkeiten empfiehlt sich ein Vergleich von $\alpha_n$ mit nicht exakt 0 sondern einem sehr kleinen Wert, e.g. $10^{-10}$

$KF$  ist hierbei die Kernel-Funktion. Der Algorithmus ist für alle Kernel (e.g. RBF oder polynomiell) gleich. Natürlich muss die für die Soft-Margin SVM verwendete Klassifizierung auch angepasst werden.

### 6.5.1 Berechnung von $\mathbf{K}$ mit Kernelfunktion

Im Folgenden der Pseudocode für die Berechnung von  $\mathbf{K}$ , wenn der Kernel Trick verwendet wird. Die einzige Änderung ist, dass anstatt des Skalarproduktes die Kernelfunktion verwendet wird.

K Berechnung	Zeile
Initialisiere $\mathbf{x}$	1
For $i = 1$ To $N$	2
For $j = 1$ To $N$	3
$\mathbf{K}(i, j) = KF(x_i, x_j)$	4
Ende For	5
Ende For	6

$N$  ist hierbei die Anzahl der Eingabevektoren,  $KF$  ist die Kernelfunktion. Durch diese Berechnung spart man sich die Transformation der Eingabe in höhere Dimensionen, die Essenz des Kernel Tricks.

## 6.6 Kernel-Trick, Polynomieller Kernel - Beispiel

Ein polynomieller Kernel der Form  $(ax^T x' + b)^Q$  wird für dieses Beispiel verwendet. Beim polynomiellen Kernel gilt es herauszufinden, durch welches Polynom sich eine Trennung der eigentlich nichtlinearen Problemstellung erreichen lässt. Hierfür benötigt man entweder Erfahrung oder Experimente. Für das folgende Experiment gilt  $b = 1, a = 1$ , der Exponent  $Q$  wird variiert. Abbildung 6.4 zeigt die Entscheidungsgrenzen für verschiedene  $Q$  für den Datensatz KM40M2-NN aus Übung 5b) des Aufgabenblattes.

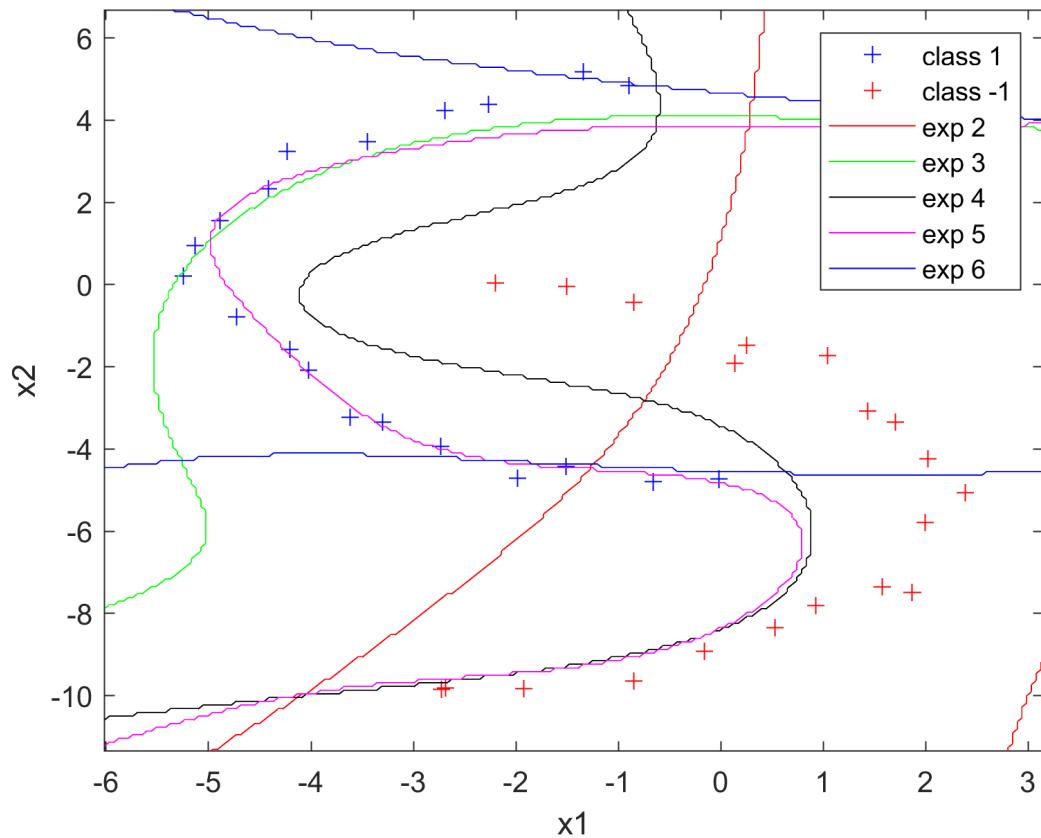


Abbildung 6.4: Nichtlinear separierbarer Datensatz mit Grenze von SVM mit polynomiellem Kernel für  $Q = 2, \dots, 6$

Man kann die verschiedenen Grenzen beobachten und graphisch analysieren. Da sich durch den Kerneltrick der Rechenaufwand in Grenzen hält, kann über die Variation von  $Q$  und gegebenenfalls auch  $a, b$  gut experimentiert werden, um ein geeignetes Polynom zu finden. Bei genauerer Betrachtung bietet sich bei diesem Beispiel ein Exponent von  $Q = 4$  an. In Abbildung 6.5 ist die Entscheidungsgrenze für dieses Polynom abgebildet.

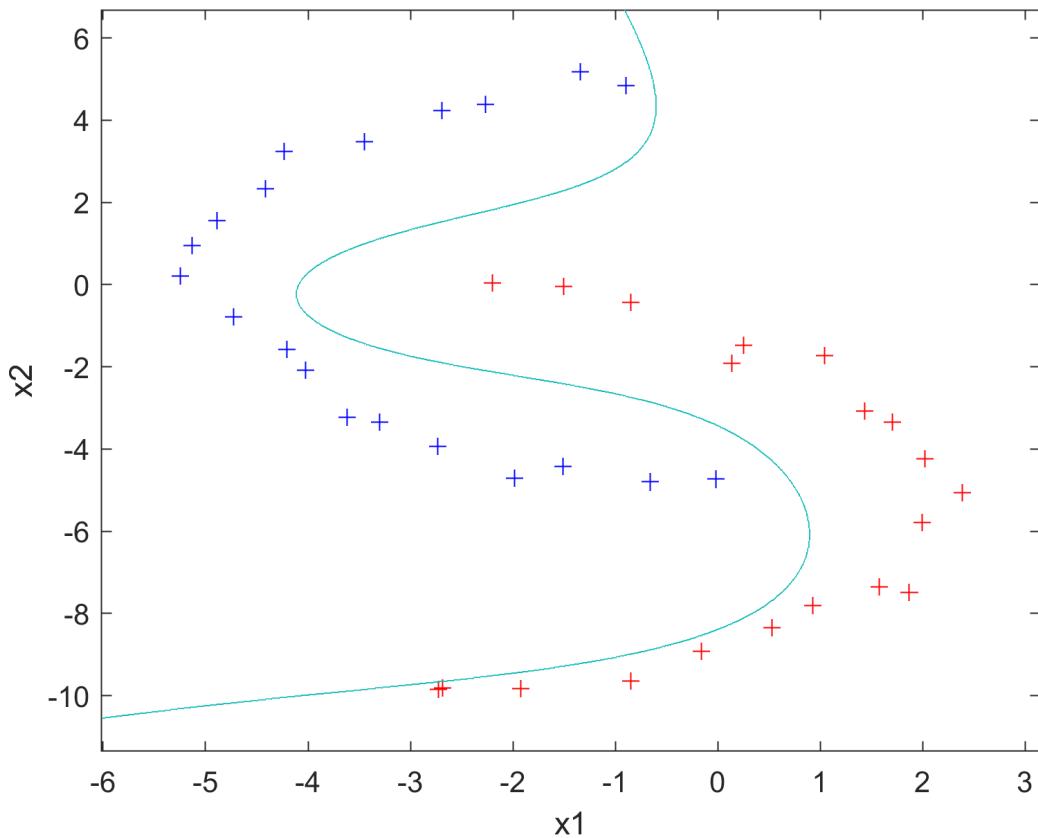


Abbildung 6.5: Nichtlinear separierbarer Datensatz mit Grenze von SVM mit polynomiellem Kernel für  $Q = 4$

Die Punkte werden, auch bei diesem nichtlinearen Problem, sauber getrennt.

## 6.7 Kernel-Trick, RBF Kernel - Beispiel

Anders als beim polynomiellem Kernel muss beim RBF Kernel nicht nach einem Polynom gesucht werden. Dafür muss der Parameter  $\gamma$  korrekt eingestellt werden. Abbildung 6.6 zeigt die Ergebnisse mit verschiedenen Werten für  $\gamma$ , angewandt auf den Datensatz KM40M2-NN.

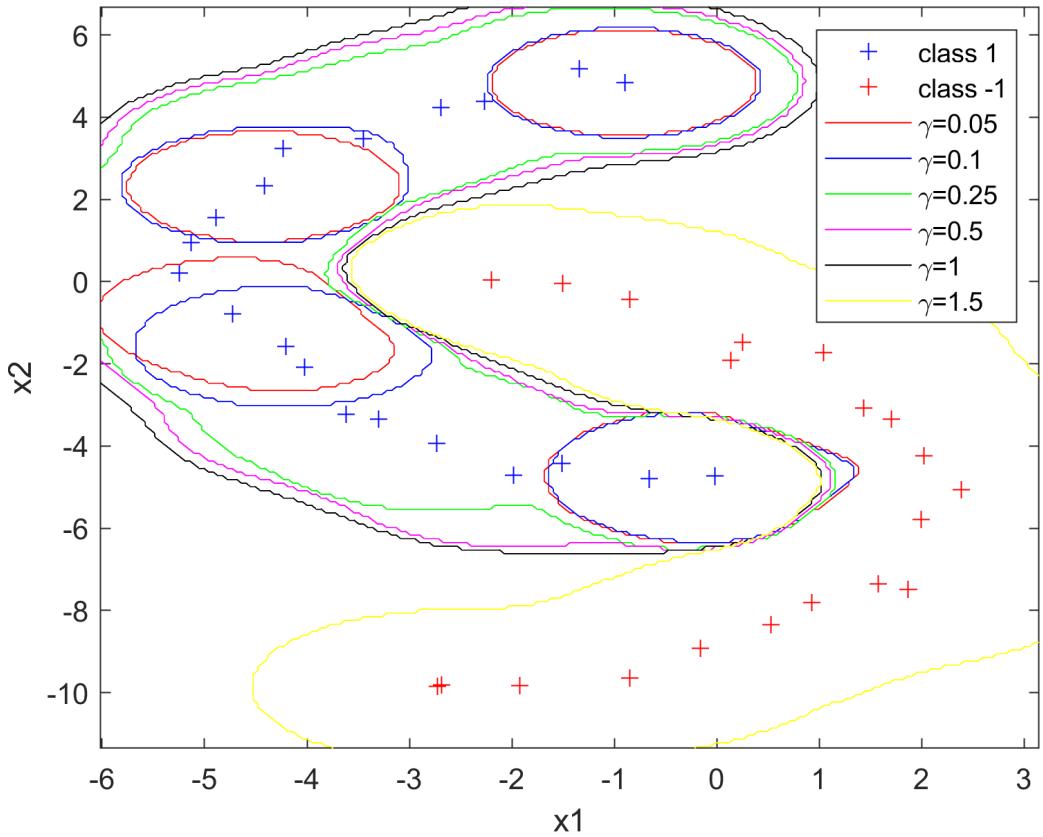


Abbildung 6.6: Nichtlinear separierbarer Datensatz mit Grenze von SVM mit RBF-Kernel für verschiedene  $\gamma$

Man sieht, dass sich bei manchen Werten für  $\gamma$  mehrere kleinere abgegrenzte Bereiche bilden, aber nicht eine saubere Trenngrenze gezogen wird. Mit  $\gamma = 1$  und  $\gamma = 1.5$  wird sauber getrennt. Bei  $\gamma = 1$  werden die Punkte der Klasse 1 von einer Grenze umschlossen, bei  $\gamma = 1.5$  die Punkte der Klasse -1.  
Die Grenze für  $\gamma = 1$  wird noch einmal separat in Abbildung 6.7 dargestellt.

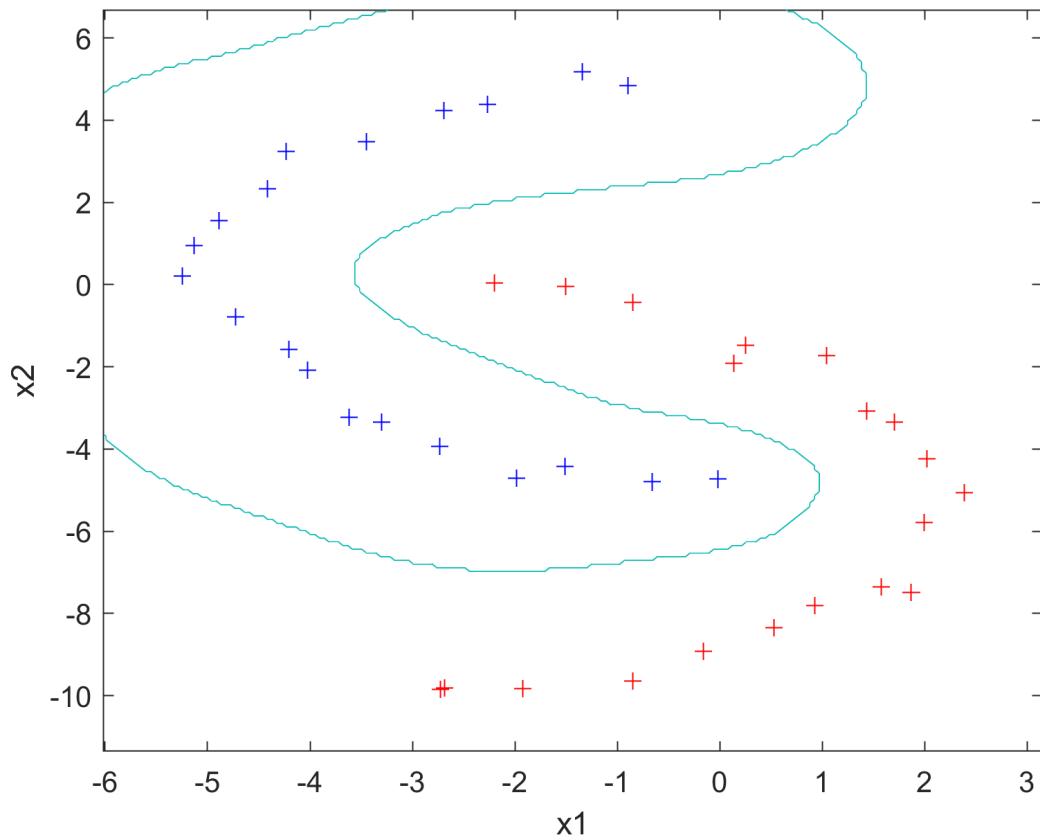


Abbildung 6.7: Nichtlinear separierbarer Datensatz mit Grenze von SVM mit RBF-Kernel für  $\gamma = 1$

Man sieht, dass die Punkte schön getrennt werden. Auch das Umschließen einer Klasse, das typisch für SVM mit RBF-Kernel ist, sieht man gut.

## 6.8 Kernel-Trick, Vergleich verschiedener Methoden

Vergleicht man die SVM mit Kernel Trick mit bereits aus der Vorlesung bekannten Methoden, erhält man Abbildung 6.8.

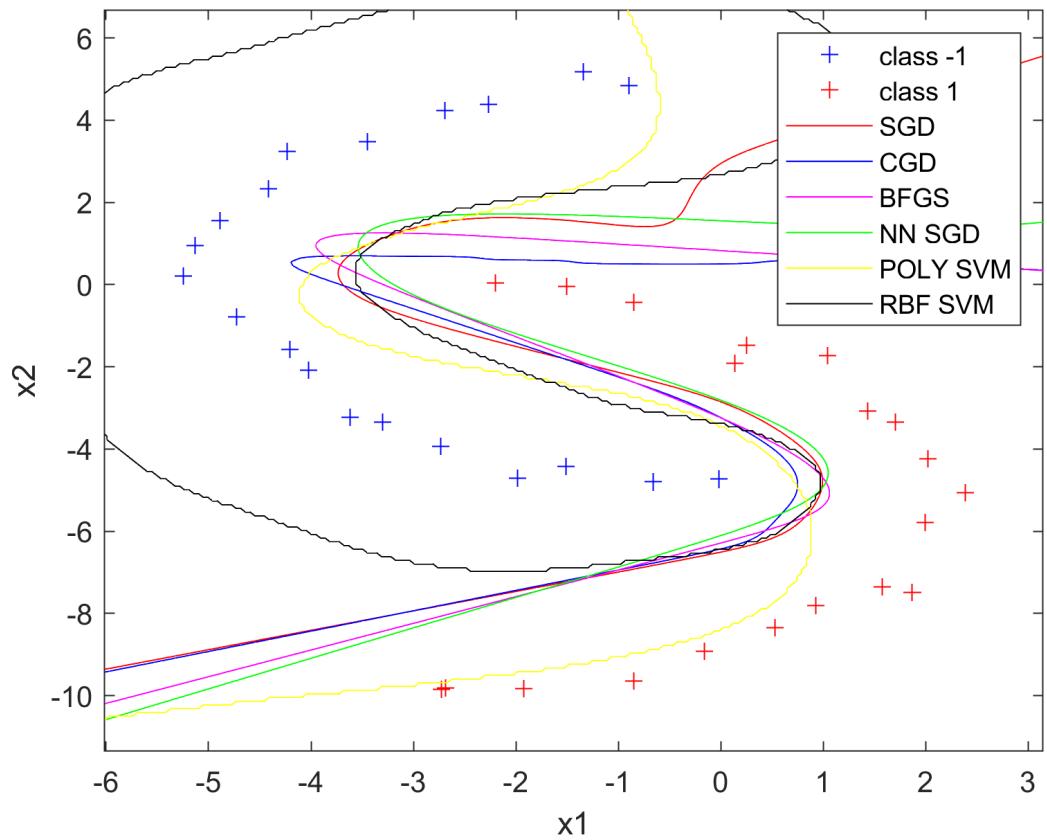


Abbildung 6.8: Nichtlinear separierbarer Datensatz mit Grenzen verschiedener Methoden

Man sieht, dass alle Methoden trennen. Wichtig ist hier, dass die SVM-Methoden stabiler sind, durch die Abwesenheit der zufällig initialisierten Gewichtsmatrix der anderen Methoden. Vergleicht man die Zeit, die die Algorithmen brauchen, erhält man Abbildung 6.9.

```
SGD, time taken:  
13.7960  
  
CGD, time taken:  
1.5080  
  
BFGS, time taken:  
15.7030  
  
NN SGD, time taken:  
1.3670  
  
POLY SVM, time taken:  
0.0420  
  
RBF SVM, time taken:  
0.0300
```

Abbildung 6.9: Dauer verschiedener Methoden für nichtlinearen Datensatz

Man sieht, dass die SVM-Methoden wesentlich schneller sind. Das verwendete QP-Solver Framework trägt hier sicher einiges dazu bei. Die Kombination aus kurzer Dauer und Stabilität macht das Experimentieren zum Finden der richtigen Hyperparameter ( $\gamma$  oder  $Q, a, b$ ) mit den SVM-Methoden angenehmer.

# Weiterführende Ressourcen

- Abu-Mostafa, Yaser S., Malik Magdon-Ismail und Hsuan-Tien Lin (2012). *Learning From Data*. AMLBook. 213 S. ISBN: 978-1-60049-006-4.
- Bishop, Christopher M. (23. Aug. 2016). *Pattern Recognition and Machine Learning*. Softcover reprint of the original 1st ed. 2006 Edition. New York, NY: Springer. 758 S. ISBN: 978-1-4939-3843-8.
- Boyd, Stephen und Lieven Vandenberghe (2004). *Convex Optimization*. Cambridge, UK ; New York: Cambridge University Press. 727 S. ISBN: 978-0-521-83378-3. URL: <https://web.stanford.edu/~boyd/cvxbook/>.
- Burges, Christopher J.C. (1. Juni 1998). „A Tutorial on Support Vector Machines for Pattern Recognition“. In: *Data Mining and Knowledge Discovery* 2.2, S. 121–167. ISSN: 1573-756X. DOI: 10.1023/A:1009715923555. URL: <https://doi.org/10.1023/A:1009715923555> (besucht am 06.03.2021).
- Kecman, Vojislav (8. Juni 2001). *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. Reprint Edition. MIT Press. 576 S. ISBN: 978-0-262-52790-3.
- Platt, John (Apr. 1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. MSR-TR-98-14, S. 21. URL: <https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines/>.
- Winston, Patrick (2010). „6.034 Artificial Intelligence“. Massachusetts Institute of Technology: MIT OpenCourseWare. URL: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010>.