Lineare Klassifikatoren

Volker Tresp

Einführung

- Lineare Klassifikatoren trennen Klassen durch eine lineare Hyperebene (genauer: affine Menge)
- In hochdimensionalen Problemen trennt schon eine lineare Trennebene die Klassen
- Lineare Klassifikatoren können nicht das *exclusive-or* Problem lösen
- Im Zusammenhang mit Basisfunktionen oder Kernelfunktionen können jedoch beliebige Trennflächen modelliert werden

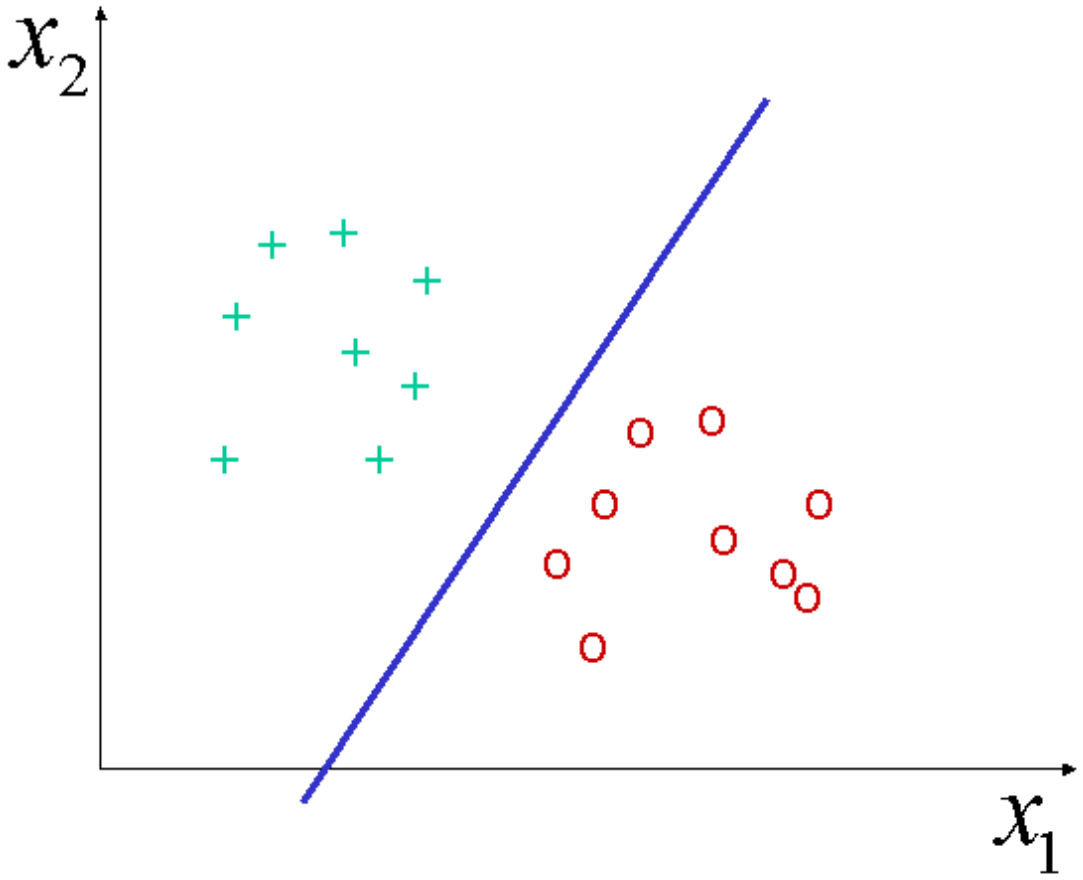
Einführung (2)

- Wir werden uns zunächst auf Klassifikatoren konzentrieren, die zwei Klassen $c = 1$ und $c = 0$ voneinander trennen und dann die Verfahren auf C Klassen verallgemeinern

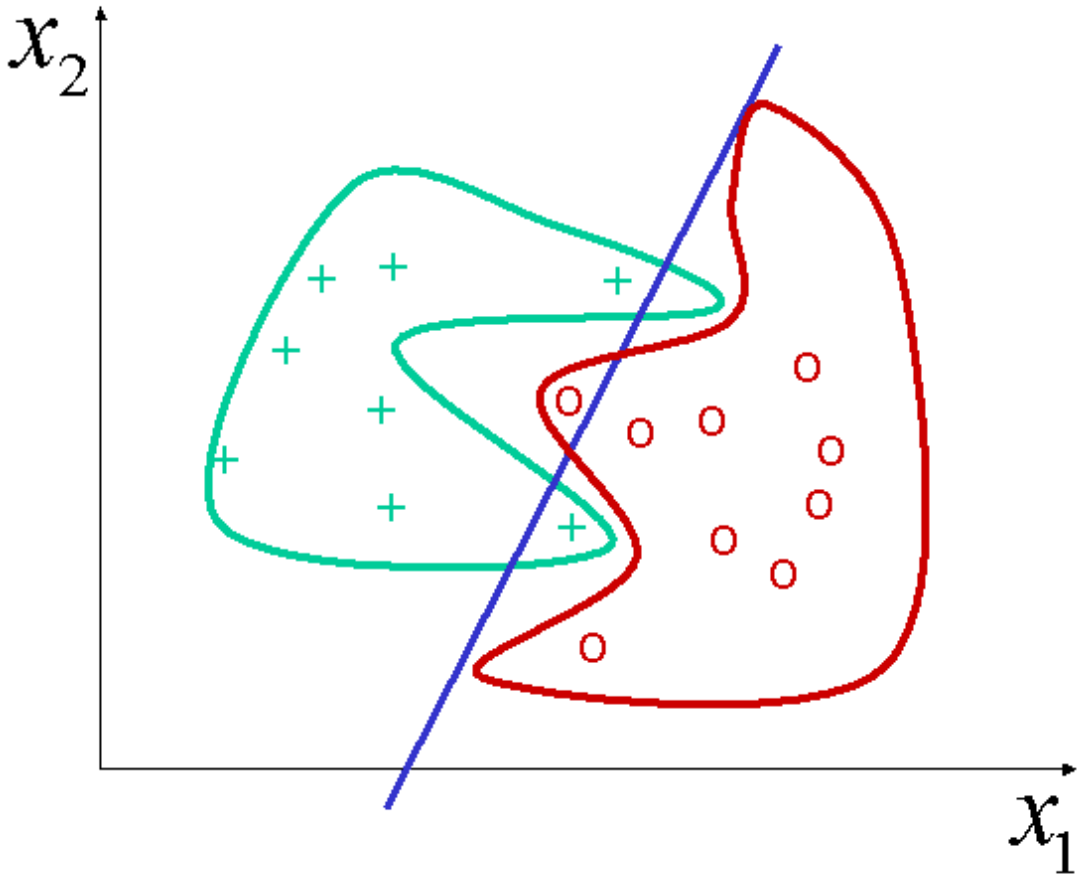
  Das Perceptron hatten wir bereits kennengelernt. Wir behandeln hier folgende weitere Ansätze

  - I. Klassifikation durch Regression
  - II. Logistische Regression
  - III. (Vapnik's) Optimal-trennende Hyperebenen
  - (IV. Appendix: (Fisher's) Lineare Diskriminanten-Analyse)

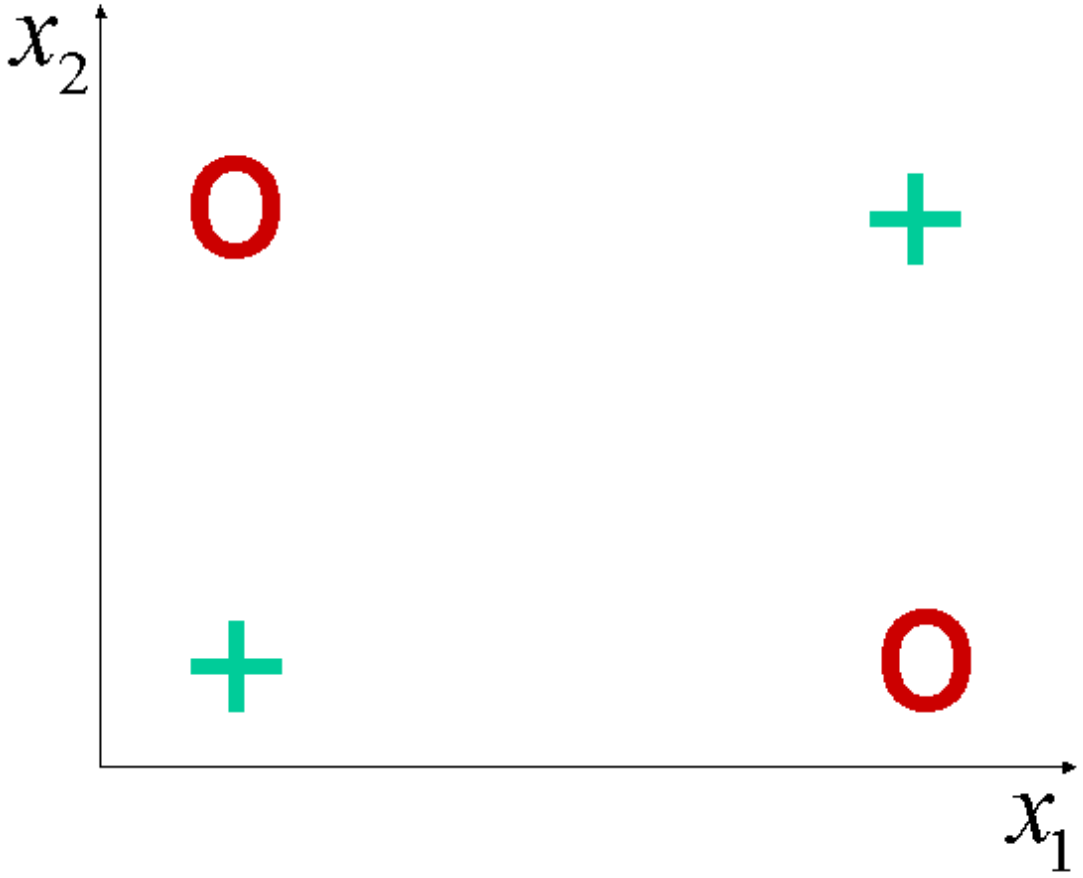
Zwei linear-separierbare Klassen



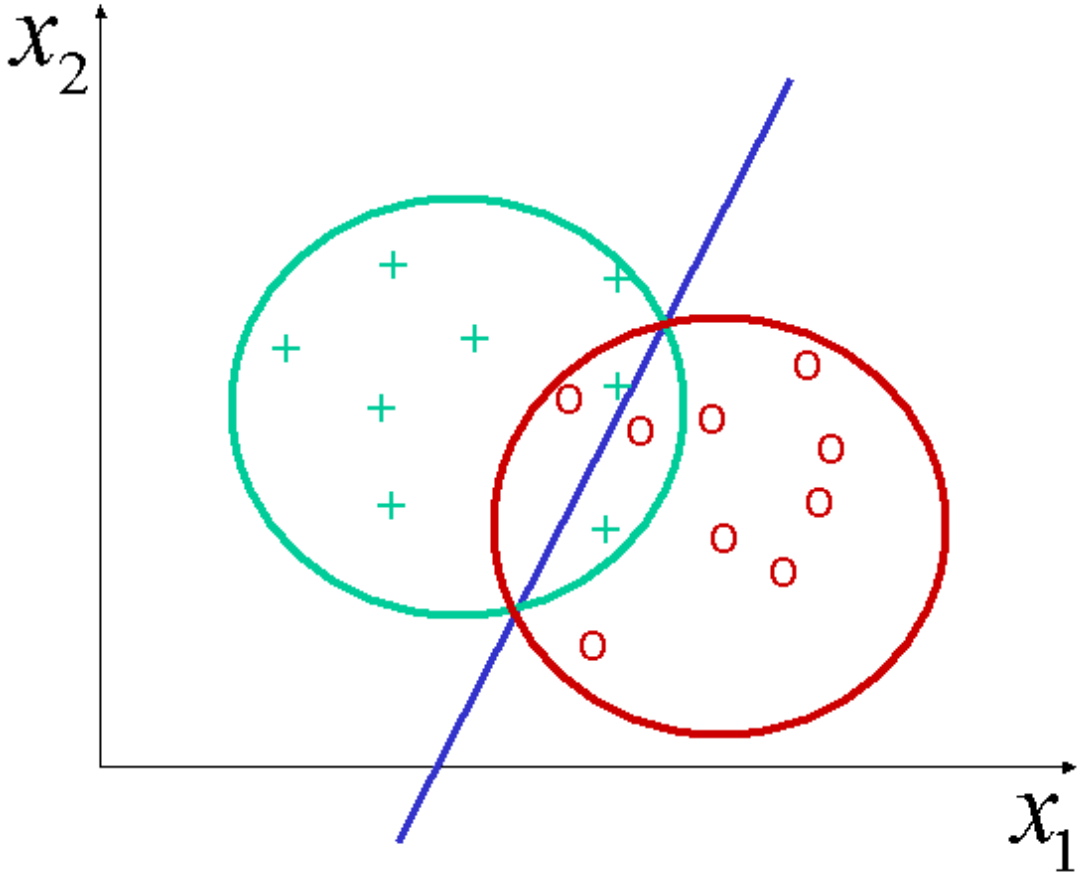
Zwei Klassen, die nicht linear separierbar sind



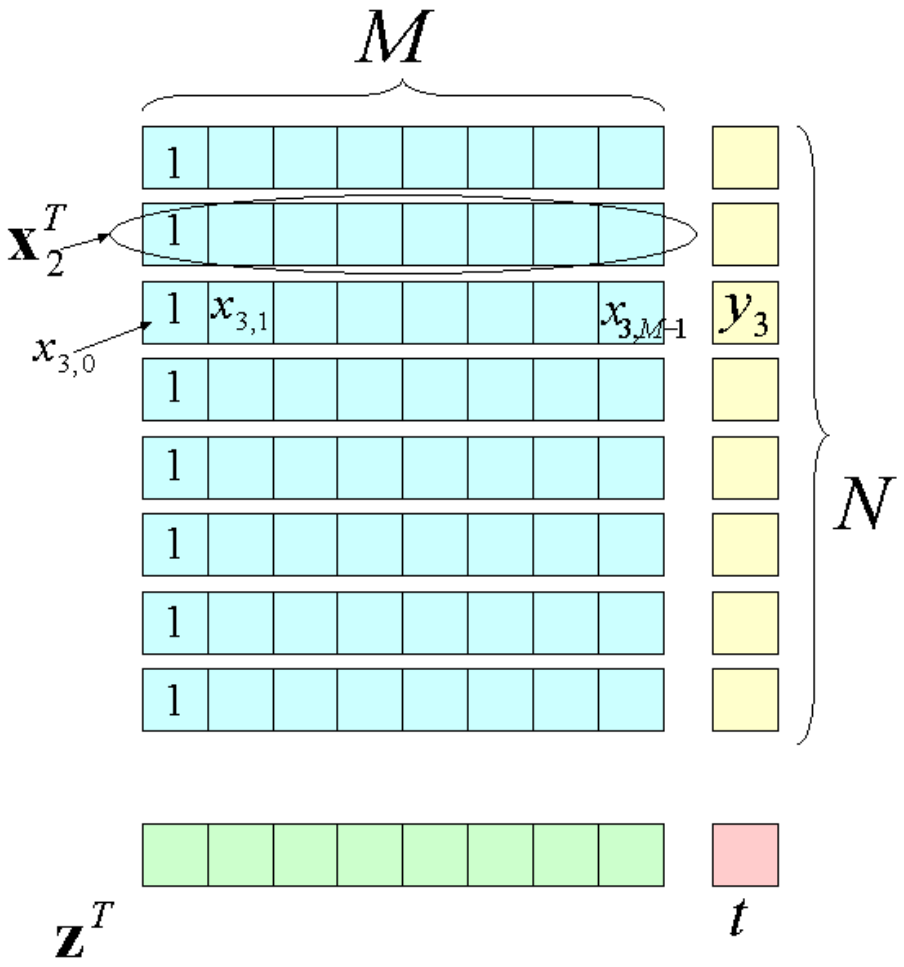
Das klassische Beispiel nicht-separierbarer Klassen: XOR



Separierbarkeit ist kein Ziel an sich: überlappende Klassen

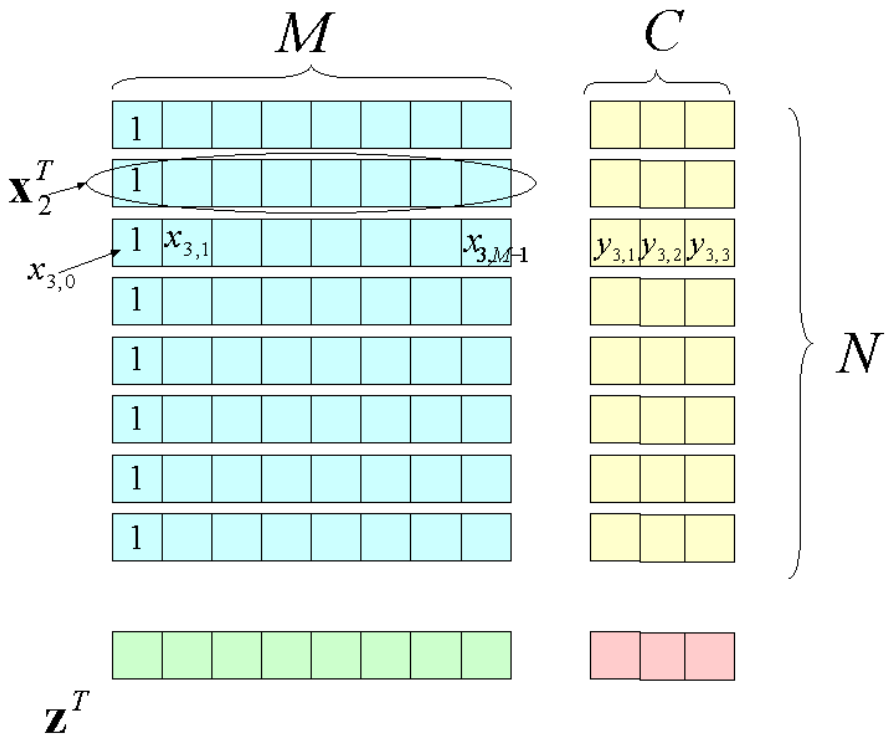


Die Datenmatrix für Klassifikation



X_j	j-te Eingangsvariable
X	$= (X_0, \ldots, X_{M-1})^T$ Vektor von Eingangsvariablen
M	Anzahl der Eingangsvariablen
N	Anzahl der Datenpunkte
Y	Ausgangsvariable
$\mathbf{x}_i$	$= (x_{i,0}, \ldots, x_{i,M-1})^T$ i-ter Eingangsvektor
$x_{i,j}$	j-te Komponente von $\mathbf{x}_i$
y_i	2 Klassen $\in \{0, 1\}$ oder $\in \{-1, 1\}$ C Klassen: $\in \{1, \ldots, C\}$
$\mathbf{d}_i$	$= (x_{i,0}, \ldots, x_{i,M-1}, y_i)^T$ i-tes Muster
D	$= \{\mathbf{d}_1, \ldots, \mathbf{d}_N\}$ (Trainings-) Datensatz
$\mathbf{z}$	Testeingangsvektor
t	Unbekannte Testzielgröße zu $\mathbf{z}$
$\mathbf{X}$	$= (\mathbf{x}_1, \ldots \mathbf{x}_N)^T$ Design matrix

Die Datenmatrix für Klassifikation mit Dummy-Variablen



M	Anzahl der Eingangsvariablen
N	Anzahl der Datenpunkte
$\mathbf{x}_i$	$= (x_{i,0}, \ldots, x_{i,M-1})^T$ i-ter Eingangsvektor
$x_{i,j}$	j-te Komponente von $\mathbf{x}_i$
$\mathbf{y}_i$	$= (y_{i,1}, \ldots, y_{i,C})^T$ i-ter Ausgangsvektor
$y_{i,j}$	j-te Komponente von $\mathbf{y}_i$ $= 1$, falls Muster i zu Klasse j gehört, ansonsten $= 0$
$\mathbf{d}_i$	$= (x_{i,0}, \ldots, x_{i,M-1}, y_{i,1} \ldots y_{i,C})^T$ i-tes Muster
D	$= \{\mathbf{d}_1, \ldots, \mathbf{d}_N\}$ (Trainings-) Datensatz
$\mathbf{X}$	$= (\mathbf{x}_1, \ldots \mathbf{x}_N)^T$ Design matrix
$\mathbf{Y}$	$= (\mathbf{y}_1, \ldots \mathbf{y}_C)^T$

I. Klassifikation durch Regression

- Lineare Regression:

$$f(\mathbf{x}_i, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j x_{i,j}$$

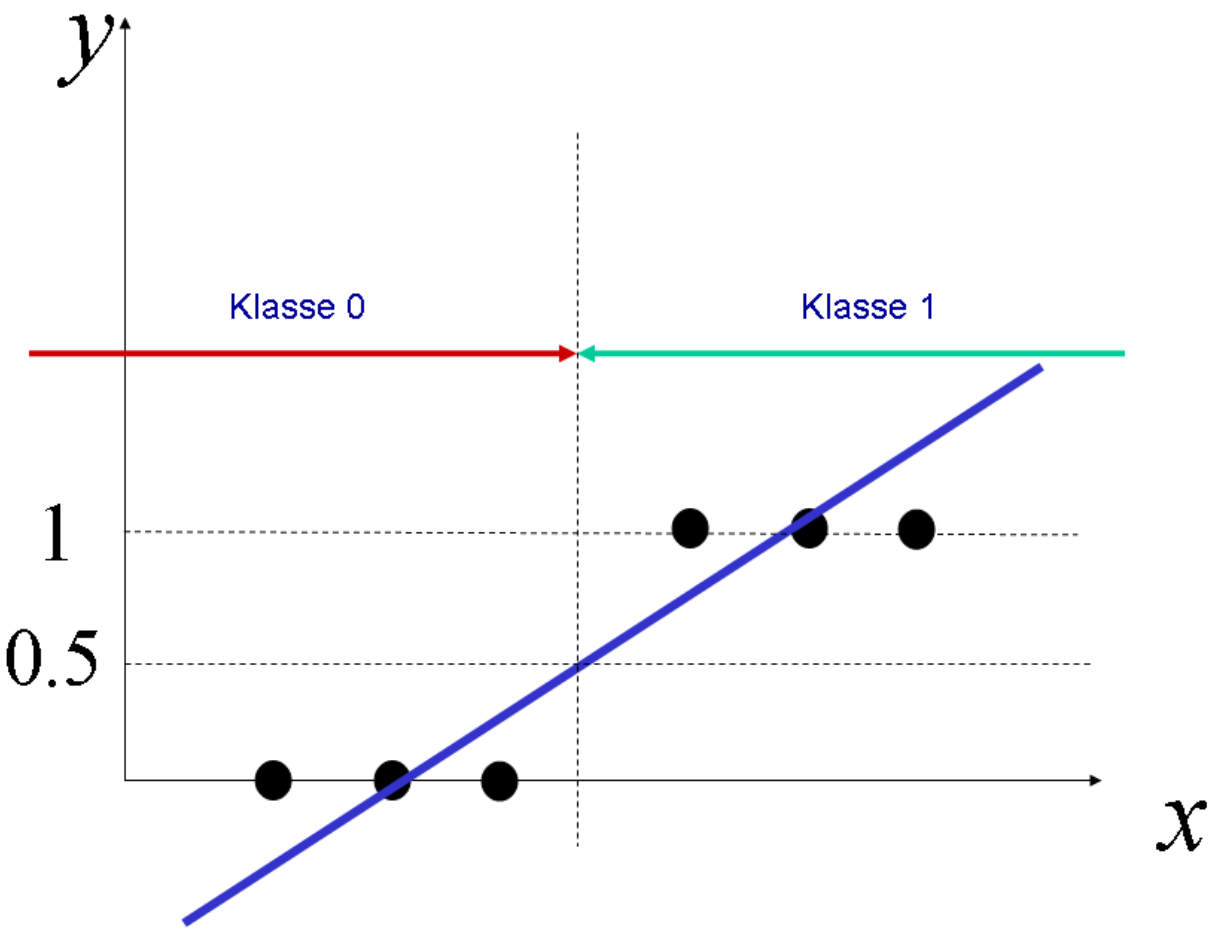
$$= \mathbf{x}_i^T \mathbf{w}$$

- Wir definieren als Zielgröße $y_i = 1$ falls Muster $\mathbf{x}_i$ zu Klasse 1 gehört und $y_i = 0$ falls Muster $\mathbf{x}_i$ zu Klasse 0 gehört
- Wir berechnen Gewichte $\mathbf{w}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ als LS-Lösung, genau wie in der linearen Regression
- Für einen neues Muster $\mathbf{z}$ berechnen wir $f(\mathbf{z}) = \mathbf{z}^T\mathbf{w}_{LS}$ und ordnen das Muster Klasse 1 zu falls $f(\mathbf{z}) > 1/2$; ansonsten ordnen wir das Muster Klasse 0 zu

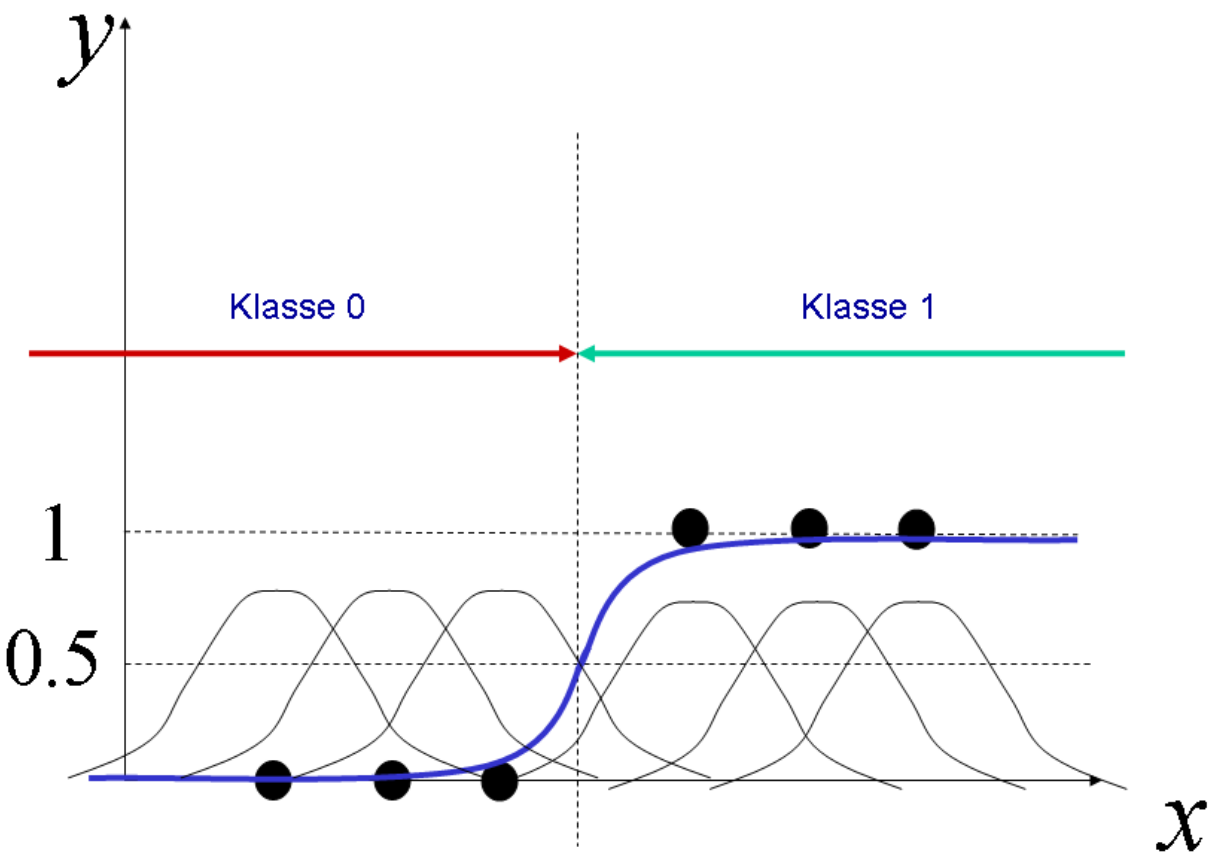
Klassifikation durch Regression (2)

- Asymptotisch konvergiert eine LS-Lösung zur Klassenwahrscheinlichkeit $P(c = 1|\mathbf{x})$; allerdings ist eine lineare Funktion in der Regel nicht fähig, die Klassenwahrscheinlichkeit zu repräsentieren (Bias!); dennoch kann die resultierende Klassifikationsentscheidung durchaus sinnvoll sein
- Man kann gute Klassifikationsentscheidungen in hohen Dimensionen erwarten und im Zusammenhang mit Basisfunktionen und Kernelfunktionen; in manchen Fällen erreicht man dann Konsistenz

Klassifikation durch Regression mit linearen Funktionen



Klassifikation durch Regression mit radialen Basisfunktionen



Klassifikation durch Regression: mehrere Klassen

- Man definiert einen Vektor von Zielgrößen $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,C})^T$ (Dummy-Variablen))
- $y_{i,j} = 1$, falls Muster x_i in Klasse $c = j$ gehört und $y_{i,j} = 0$ sonst
- Man führt für jede Komponente der Zielgröße eine getrennte lineare LS-Regression durch
- Für einen Mustervektor $\mathbf{z}$ berechnen wir mit dem linearen Modell den Wert jeder Zielvariablen $\widehat{t}_j$ und klassifizieren ihn als Klasse c, wobei

$$c = \arg\max_j \widehat{t}_j$$

- Klassische Ansatz bei Neuronalen Netzen
- Obwohl der Ansatz für lineare Klassifikation etwas simplistisch erscheint, ist die Performanz in Kombination mit Basisfunktionen oder Kernelfunktionen durchaus sehr gut! Dort wegen der großen Einfachheit und guten Performanz sehr populär

II. Logistische Regression: Zwei Klassen; Motivation

- Das Modell der linearen Regression war $f(x) = \mathbf{x}^T\mathbf{w}$
- Ein Modell für $P(c = 1|\mathbf{x})$ sollte zwischen 0 und 1 beschränkt sein
- Dies wird erfüllt, durch ein sogenanntes log-lineares Modell der Form

$$P(c = 1|\mathbf{x}) = \frac{1}{Z(x)}\exp(\mathbf{x}^T\mathbf{w}) \quad P(c = 0|\mathbf{x}) = \frac{1}{Z(x)}\exp(-\mathbf{x}^T\mathbf{w})$$

$$Z(x) = \exp(\mathbf{x}^T\mathbf{w}) + \exp(-\mathbf{x}^T\mathbf{w})$$

- Man teil diesen Zähler und Nenner durch $\exp(\mathbf{x}^T\mathbf{w})$ und erhält

$$P(c = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{x}^T\mathbf{w})}$$

II. Logistische Regression: Zwei Klassen

- Die Idee ist, dass man die lineare Funktion durch eine *squashing* Funktion auf in den Bereich $(0, 1)$ *quetscht*
- Modellannahme:

$$P(c = 1|\mathbf{x}) = f(\mathbf{x}) = \text{sig}(\mathbf{x}^T\mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{x}^T\mathbf{w})}$$

$$P(c = 0|\mathbf{x}) = 1 - \text{sig}(\mathbf{x}^T\mathbf{w}) = \frac{1}{1 + \exp(\mathbf{x}^T\mathbf{w})}$$

- Vorteil: der Ausgang ist begrenzt zwischen 0 und 1 und kann daher als Wahrscheinlichkeit interpretiert werden

Maximum-Likelihood Lösung

- Likelihood:

$$L = \prod_{i:y_i=1} \left(\frac{1}{1+\exp(-\mathbf{x}_i^T \mathbf{w})} \right) \times \prod_{i:y_i=0} \left(\frac{1}{1+\exp(\mathbf{x}_i^T \mathbf{w})} \right)$$

- Log-Likelihood-Kostenfunktion

$$l = \sum_{i:y_i=1} \log \left(\frac{1}{1+\exp(-\mathbf{x}_i^T \mathbf{w})} \right) + \sum_{i:y_i=0} \log \left(\frac{1}{1+\exp(\mathbf{x}_i^T \mathbf{w})} \right)$$

$$= - \sum_{i:y_i=1} \log(1+\exp(-\mathbf{x}_i^T \mathbf{w})) - \sum_{i:y_i=0} \log(1+\exp(\mathbf{x}_i^T \mathbf{w}))$$

- Bei Übereinstimmung von Zielgröße und Vorhersage entstehen Kosten $\longrightarrow 0$, bei Nichtübereinstimmung entstehen Kosten $\longrightarrow \infty$

Maximum-Likelihood Lösung: Lernen

- Die Ableitung der Log-Likelihood nach den Gewichten

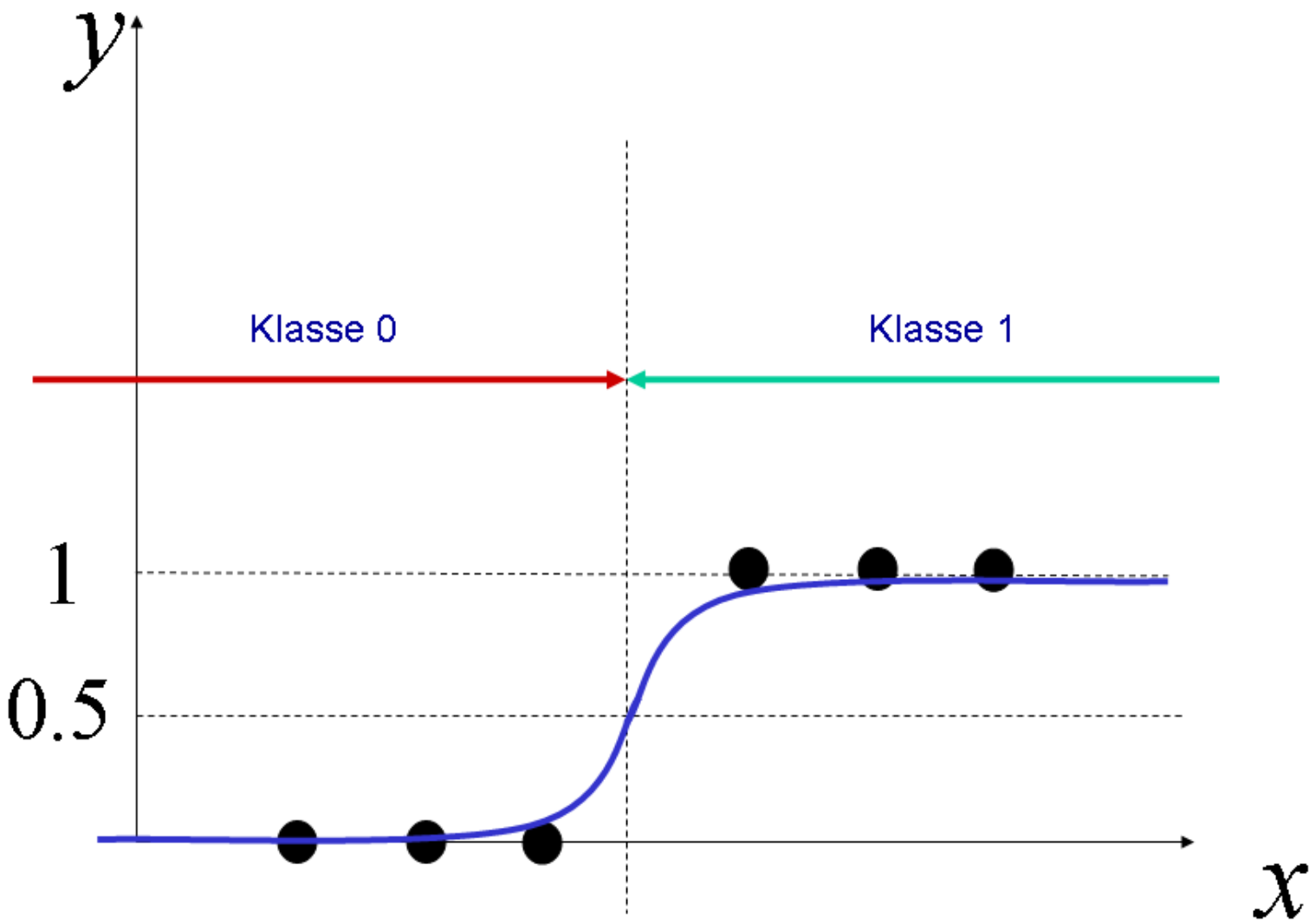
$$\frac{\partial l}{\partial \mathbf{w}} = \sum_{i:y_i=1} \frac{\mathbf{x}_i \exp(-\mathbf{x}_i^T \mathbf{w})}{1+\exp(-\mathbf{x}_i^T \mathbf{w})} - \sum_{i:y_i=0} \frac{\mathbf{x}_i \exp(\mathbf{x}_i^T \mathbf{w})}{1+\exp(\mathbf{x}_i^T \mathbf{w})}$$

$$= \sum_{i:y_i=1} \mathbf{x}_i(1-f(\mathbf{x}_i)) - \sum_{i:y_i=0} \mathbf{x}_i f(\mathbf{x}_i) = \sum_i \mathbf{x}_i(y_i - f(\mathbf{x}_i))$$

- Beachte: Ähnlichkeit zur Adaline Lernregel
- Gradientenbasierte Optimierung der Parameter (zur *Maximierung* der Log-Likelihood)

$$\mathbf{w} \longleftarrow \mathbf{w} + \eta \frac{\partial l}{\partial \mathbf{w}}$$

- Üblicherweise wird jedoch ein Newton-Raphson Verfahren zur Optimierung benutzt


y
Klasse 0
Klasse 1
1
0.5
x

Logistische Regression: Mehr als zwei Klassen

- Verschiedene Verallgemeinerungen sind möglich
- Beispiel:

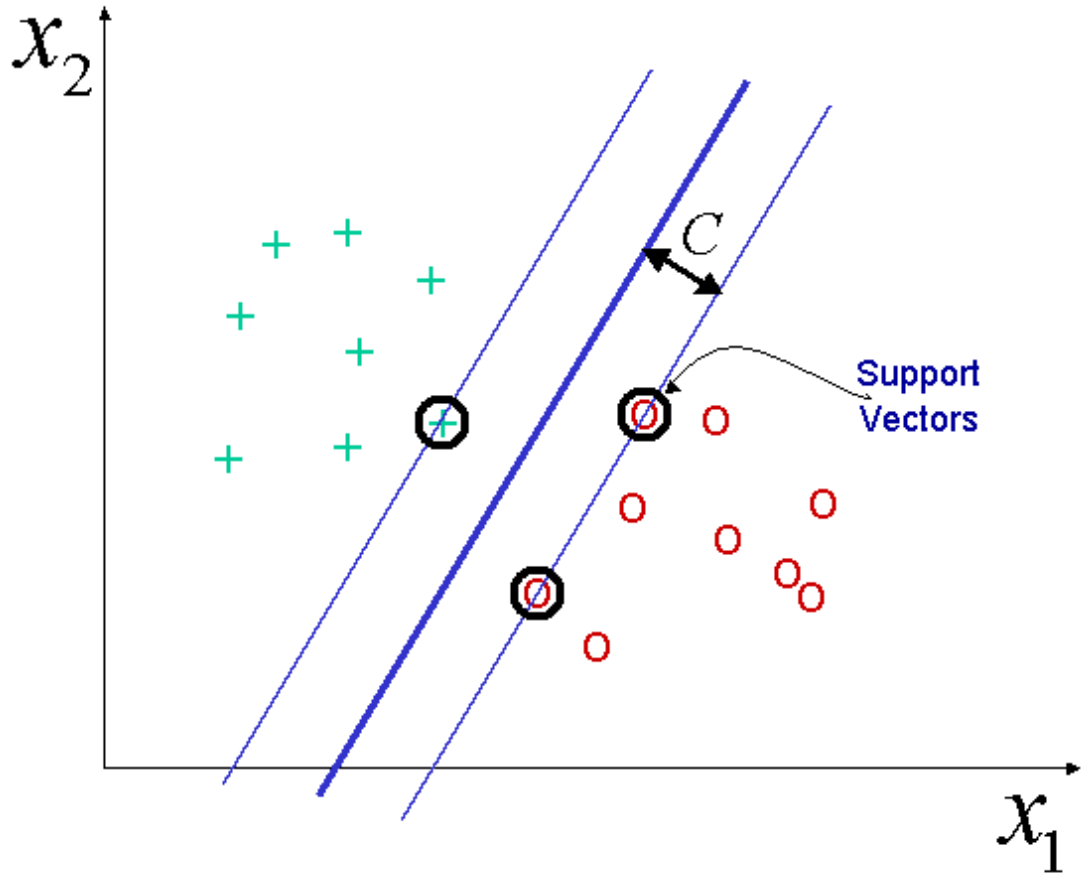
$$P(c = j|\mathbf{x}) = \frac{\exp(\mathbf{x}^T \mathbf{w}_j)}{\sum_j \exp(\mathbf{x}^T \mathbf{w}_j)}$$

und man entscheidet sich für die Klasse mit der höchsten a posteriori Wahrscheinlichkeit

III. (Vapnik's) Optimal Trennende Hyperebenen (Optimal Separating Hyperplanes)

- Die soweit vorgestellten Algorithmen trennen nicht notwendigerweise zwei Klassen, auch wenn diese separabel sind
- Dies ist auch unter Umständen richtig, da Klassen sich überlappen können
- Dennoch ist es wünschenswert, auch Algorithmen im Repertoire zu haben, die trennbare Klassen auch trennen (ähnlich wie beim Perzeptron)
- Vapnik stellte einen Algorithmus vor, der trennbare Klassen trennt; falls Klassen sich nicht trennen lassen, wird die Anzahl der Missklassifikationen klein gehalten
- Ziel ist es, die Klassen zu trennen und dabei den Margin $\mathcal{C}$ zu maximieren (falls Klassen separabel sind)

Optimal Trennende Hyperebenen (2D)



Optimal Trennende Hyperebenen (2)

- Die optimale Trennebene kann gefunden werden als

$$\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} \tilde{\mathbf{w}}^T \tilde{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{j=1}^{M-1} w_j^2$$

(beachte: man minimiert in Bezug auf (ganz) $\mathbf{w}$) unter der Nebenbedingung, dass

$$y_i(\mathbf{x}_i^T \mathbf{w}) = y_i \sum_{j=0}^{M-1} w_j x_{i,j} \geq 1 \quad i = 1, \ldots, N$$

wobei $\tilde{\mathbf{w}} = (w_1, \ldots, w_{M-1})$. (D.h. bei $\tilde{\mathbf{w}}$ fehlt der Offset w_0); $y_i \in \{-1, 1\}$

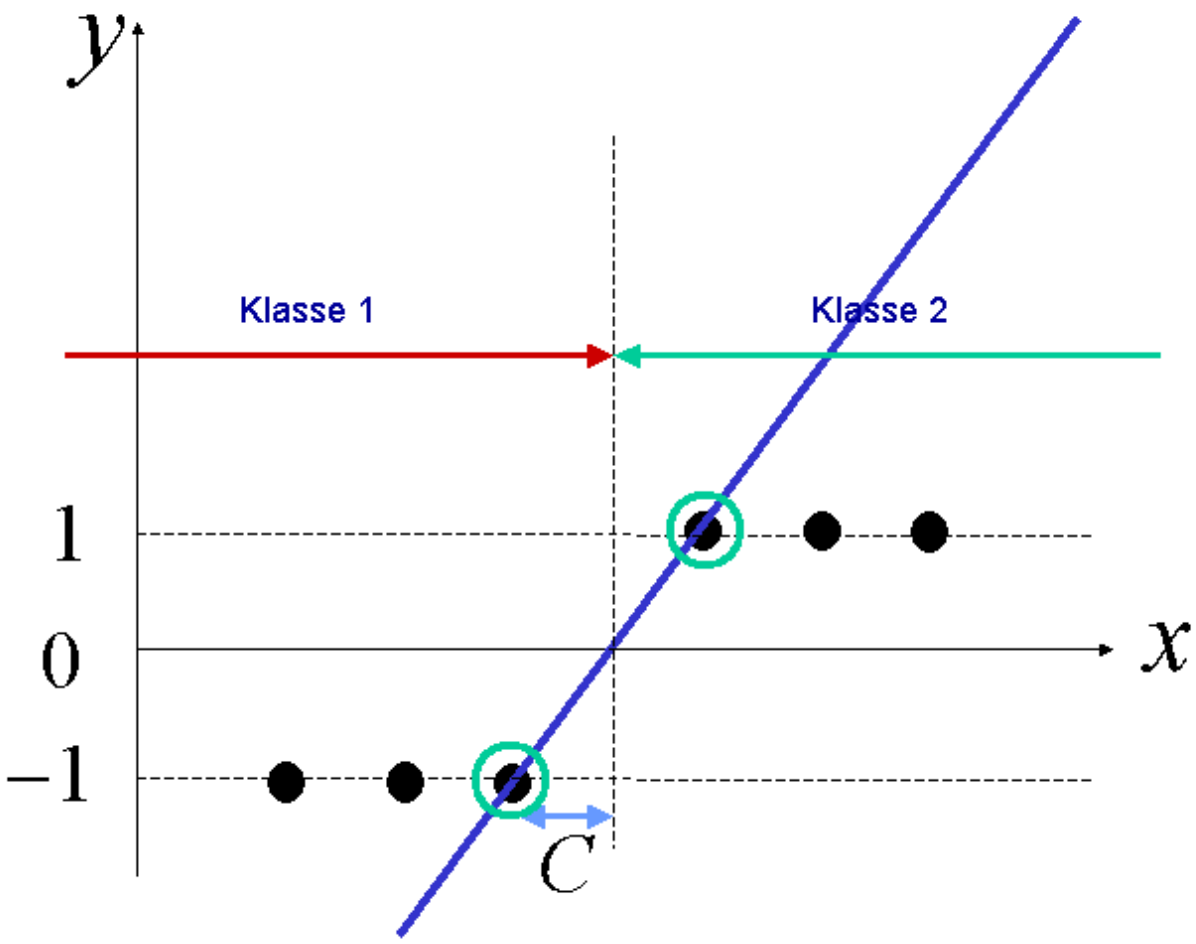
- Der Margin wird dann

$$\mathcal{C} = \frac{1}{||\tilde{\mathbf{w}}||}$$

- Für die *Support Vektoren* gilt,

$$y_i(\mathbf{x}_i^T \mathbf{w}) = 1$$

Optimal Trennende Hyperebenen (1D)



Optimierung: Optimal Separating Hyperplane

Zur Optimierung mit Randbedingungen (Ungleichheiten) definiert man die Lagrange Funktion

$$L_P = \frac{1}{2}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} - \sum_{i=1}^{N} \alpha_i[y_i(\mathbf{x}_i^T\mathbf{w}) - 1]$$

Die Lagrange Funktion wird in Bezug auf (ganz) $\mathbf{w}$ minimiert und in Bezug auf die Lagrange Multipikatoren $\alpha_i \geq 0$ maximiert (Sattelpunktlösung).

Intuition:

- Wenn eine Nebenbedingung nicht erfüllt ist, so ist $[y_i(\mathbf{x}_i^T\mathbf{w}) - 1] < 0$ und α_i wird anwachsen (beachte negatives Vorzeichen des 2ten Terms), welches durch die entsprechende Anpassung der Gewichte ausgeglichen werden muss, bis die Nebenbedingungen erfüllt sind; in diesem Fall wird am Ende $\alpha_i > 0$ sein

- Andererseits, wenn $[y_i(\mathbf{x}_i^T\mathbf{w}) - 1] > 0$, wird α_i kleiner werden und Null werden (wenn die Nebenbedingung nicht wieder aktiv werden)

- Schreiben wir äquivalent

$$L_P = \frac{1}{2}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} + \sum_{i=1}^{N} \alpha_i[1 - y_i(\mathbf{x}_i^T\mathbf{w})]$$

  erkennt man in Bezug auf $\mathbf{w}$ ein Kostenminimierungsproblem; der erste Term regularisiert die Parameter (außer w_0).

Karush-Kuhn-Tucker (KKT)-Bedingungen

Dies bedeutet, es muss gelten (Karush-Kuhn-Tucker (KKT)-Bedingung (1))

$$\alpha_i[y_i(\mathbf{x}_i^T\mathbf{w} - 1)] = 0 \quad \forall i$$

Durch Null-Setzen der Ableitungen von L_P nach $\tilde{\mathbf{w}}$ erhält man KKT (2)

$$\tilde{\mathbf{w}} = \sum_{i=1}^{N} \alpha_i y_i \tilde{\mathbf{x}}_i$$

und nach w_0 KKT (3)

$$0 = \sum_{i=1}^{N} \alpha_i y_i$$

Beachte, dass KKT (2) bedeutet, dass man die optimalen Parameter als lineare gewichtete Summe der Eingangsvektoren schreiben kann (Kernel-Trick)!

Wolfe-Dual

Durch einsetzen erhält man (Wolfe-Dual) das duale Optimierungsproblem (beachte: $\mathbf{w} = (w_0, \tilde{\mathbf{w}}^T)^T$, $\mathbf{x}_i = (1, \tilde{\mathbf{x}}_i^T)^T$,

$$L_D = \frac{1}{2} \sum_{i=1}^{N} \alpha_i y_i \tilde{\mathbf{x}}_i^T \sum_{i=1}^{N} \alpha_i y_i \tilde{\mathbf{x}}_i - \sum_{i=1}^{N} \alpha_i [y_i (\mathbf{x}_i^T \mathbf{w}) - 1]$$

$$= \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \tilde{\mathbf{x}}_i^T \tilde{\mathbf{x}}_k - \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \tilde{\mathbf{x}}_i^T \tilde{\mathbf{x}}_k$$

$$-w_0 \sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i$$

Die ersten beiden Terme sind gleich bis auf die Konstante. Der dritte Term ist im Optimum gleich Null.

und KKT (4)

$$\alpha_i \geq 0 \quad \forall i$$

Dies ist ein einfacheres konvexes Optimierungsproblem. (1), (2), (3),(4) und bilden die Karush-Kuhn-Tucker Bedingungen.

Optimierung: Zusammenfassung

- Man löst schließlich: Maximiere in Bezug auf die α_i

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \tilde{\mathbf{x}}_i^T \tilde{\mathbf{x}}_k$$

mit den Nebenbedingungen

$$\alpha_i \geq 0$$

und

$$0 = \sum_{i=1}^{N} \alpha_i y_i$$

Zwei Schreibweisen der Lösung

- Die Standardformulierung der Lösung ist eine gewichtete Summe über M Terme,

$$f(\mathbf{z}) = \text{sign}\left(\sum_{i=0}^{M-1} w_i z_i\right) = \text{sign}\left(\tilde{\mathbf{w}}^T \tilde{\mathbf{z}} + w_0\right)$$

- Alternativ lässt sich die Lösung schreiben als gewichtete Summe über N Terme,

$$f(\mathbf{z}) = \text{sign}\left(\sum_{i=1}^{N} y_i \alpha_i \tilde{\mathbf{x}}_i^T \tilde{\mathbf{z}} + w_0\right) = \text{sign}\left(\sum_{i=1}^{N} y_i \alpha_i k(\tilde{\mathbf{z}}, \tilde{\mathbf{x}}_i) + w_0\right)$$

mit Kernel

$$k(\tilde{\mathbf{z}}, \tilde{\mathbf{x}}_i) = \tilde{\mathbf{z}}^T \tilde{\mathbf{x}}_i$$

Die Terme mit $\alpha_i > 0$ definieren die Supportvektoren, die anderen Terme verschwinden in der Summe. Dies bedeutet, dass man die Lösung ebenfalls als gewichtete Summe der inneren Produkte des Eingangsvektors mit den Supportvektoren schreiben kann!

Optimal Trennende Hyperebenen: Nicht-trennbare Klassen

- Bei sich überlappenden Klassen führt man *slack* Variablen ξ_i ein:
- Die optimale Trennebene kann gefunden werden als

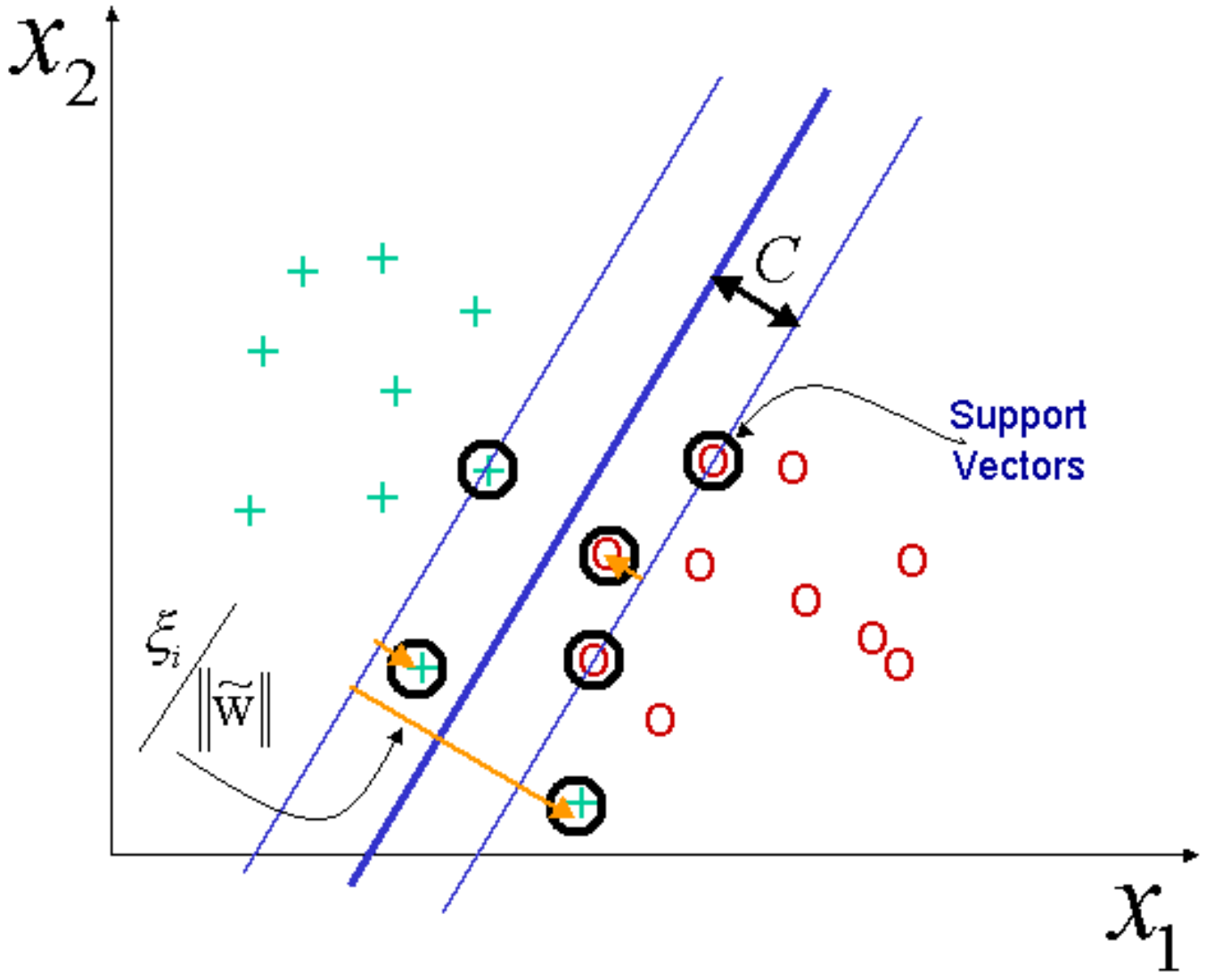
$$\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} \tilde{\mathbf{w}}^T \tilde{\mathbf{w}}$$

unter den Nebenbedingungen, dass

$$y_i(\mathbf{x}_i^T \mathbf{w}) \geq 1 - \xi_i \quad i = 1, \ldots, N$$

$$\xi_i \geq 0 \quad \sum_{i=1}^{N} \xi_i \leq 1/\gamma$$

- $\gamma > 0$ bestimmt den Kompromiss zwischen Trennbarbeit von Klassen und Überlappungsgrad. Für $\gamma \longrightarrow \infty$ erhält man den separierbaren Fall.


x_2
C
Support
Vectors
$\frac{\xi_i}{\|\tilde{w}\|}$
x_1

Optimal Trennende Hyperebenen: Kommentare

- Die optimale Trennebene wird gefunden über eine trickreiche Optimierung des resultierenden quadratischen Optimierungsproblems mit linearen Nebenbedingungen
- γ ist ein zu optimierender Hyperparameter (Kreuzvalidierung)

Optimierung über Penalty/Barrier Method

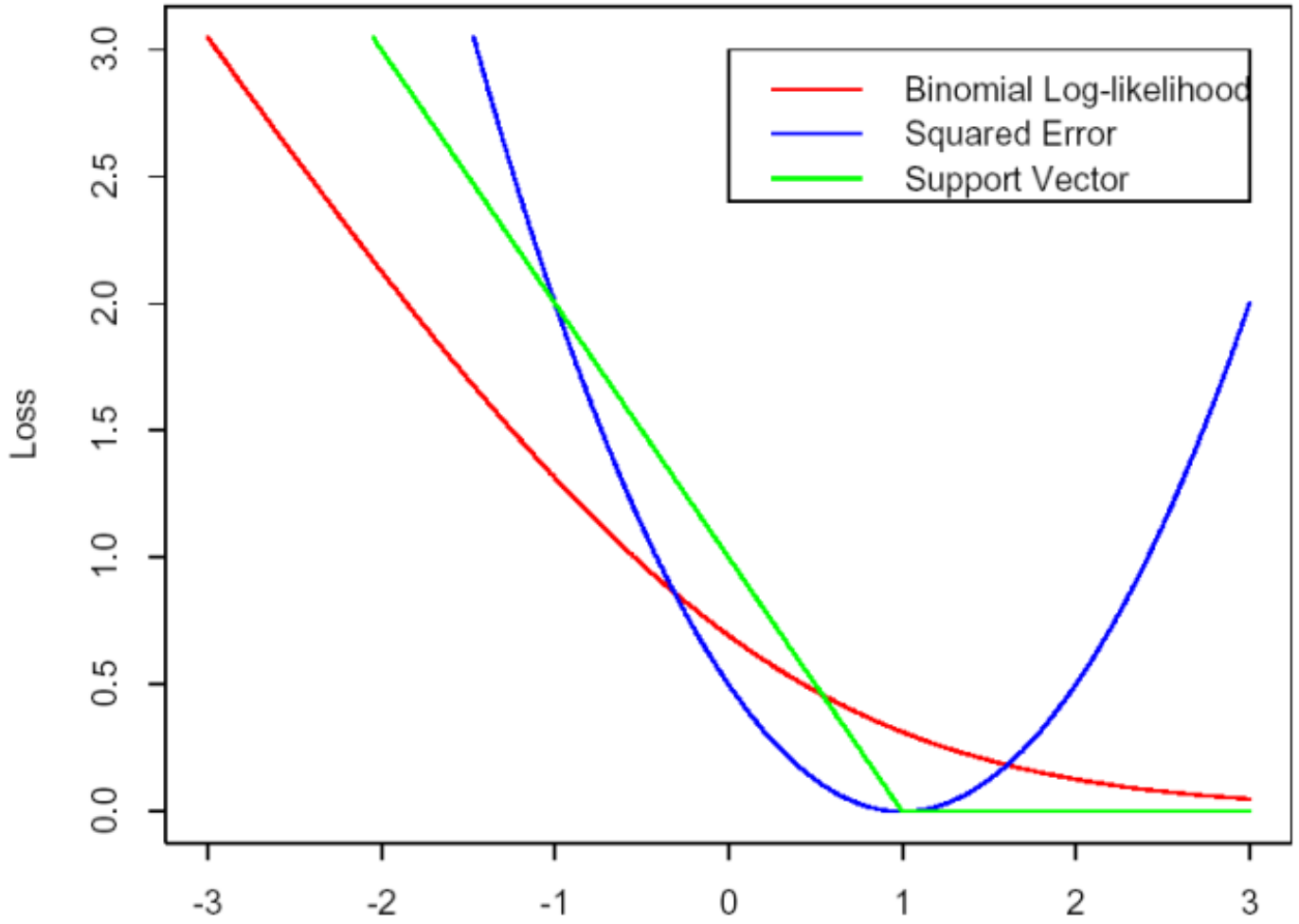
- Ziel: Transformation eines Optimierungsproblems mit Nebenbedingungen in ein Problem ohne Nebenbedingungen
- In der Penalty Methode wird ein Strafterm (penalty) zur Zielfunktion addiert für Punkte, die die Nebenbedingungen verletzen
- Die Barrier Methode (interior point method) ist ähnlich, nur dass der Strafterm unendlich ist für Punkte, die die Nebenbedingungen verletzen und endlich für Punkte "fast" die Nebenbedingungen verletzen
- Für unser Optimierungsproblem wählt man:

$$\arg\min_{\mathbf{w}} 2\gamma \sum |1 - y_i(\mathbf{x}_i^T \mathbf{w})|_+ + \tilde{\mathbf{w}}^T \tilde{\mathbf{w}}$$

wobei $|arg|_+ = \max(arg, 0)$. Man beginnt in der Optimierung mit kleinem γ und lässt es langsam anwachsen. Für $\gamma \to \infty$ verlangt man, dass alle Nebenbedingungen in der Lösung erfüllt sind. Lässt man γ endlich, dann erlaubt man "Slack".

Vergleich Der Kostenfunktionen

- Betrachten wir der Einfachheit einen Datenpunkt der Klasse 1
- Der Beitrag zur Kostenfunktion ist:
  - Optimal Trennenden Hyperebenen: $|1 - (\mathbf{x}_i^T \mathbf{w})|_+$
  - logistische Regression: $\log(1 + \exp(-\mathbf{x}_i^T \mathbf{w}))$
  - Klassifikation durch Regression: $(1 - (\mathbf{x}_i^T \mathbf{w}))^2$


Loss
0.0
0.5
1.0
1.5
2.0
2.5
3.0
-3
-2
-1
0
1
2
3
Binomial Log-likelihood
Squared Error
Support Vector