

# Linear separierbare Probleme

Mustererkennung und Klassifikation, Vorlesung No. 9<sup>1</sup>

M. O. Franz

13.12.2007

---

<sup>1</sup> falls nicht anders vermerkt, sind die Abbildungen entnommen aus Duda et al., 2001 

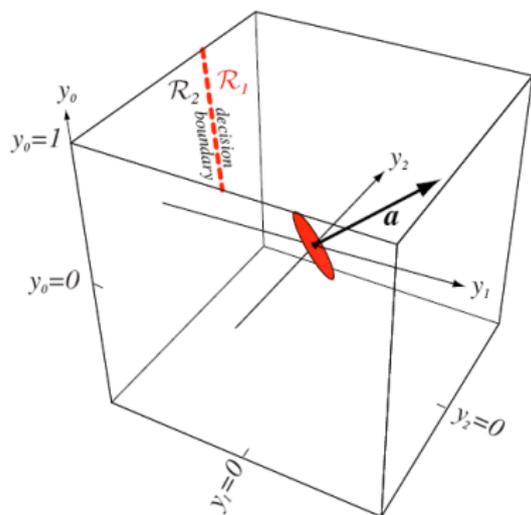
# Übersicht

- 1 Linear trennbare 2-Kategorienprobleme
- 2 Gradientenabstieg
- 3 Perzeptron

# Übersicht

- 1 Linear trennbare 2-Kategorienprobleme
- 2 Gradientenabstieg
- 3 Perzeptron

# Erweiterte Merkmalsvektoren



Einführung einer zusätzlichen konstanten Koordinate  $x_0 = 1$  vereinfacht Notation.

$$g(x) = w_0 + \sum_{i=1}^d w_i x_i = \sum_{i=0}^d w_i x_i$$

Erweiterter Merkmalsvektor:

$$y = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

Erweiterter Gewichtsvektor:

$$a = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ w \end{bmatrix}$$

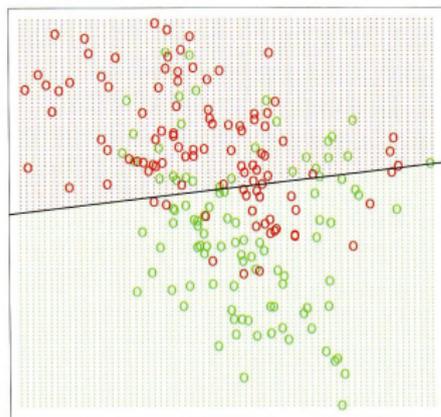
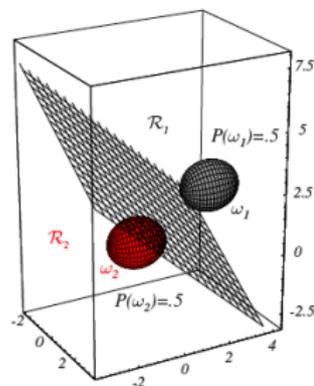
# Linear trennbare 2-Kategorienprobleme (1)

**Aufgabe:** Bestimme den Gewichtsvektor  $a$  einer linearen Diskriminantenfunktion

$$g(x) = a^\top y$$

so, daß die gegebenen Beispiele aus beiden Klassen  $\omega_1$  und  $\omega_2$  durch die entsprechende Hyperebene getrennt werden.

Probleme, für die ein solcher Gewichtsvektor existiert, heißen **linear trennbar**.



## Linear trennbare 2-Kategorienprobleme (2)

Korrekte Klassifikation genau dann, wenn

- $a^\top y_i > 0$  für  $y_i \in \omega_1$
- $a^\top y_i < 0$  für  $y_i \in \omega_2$

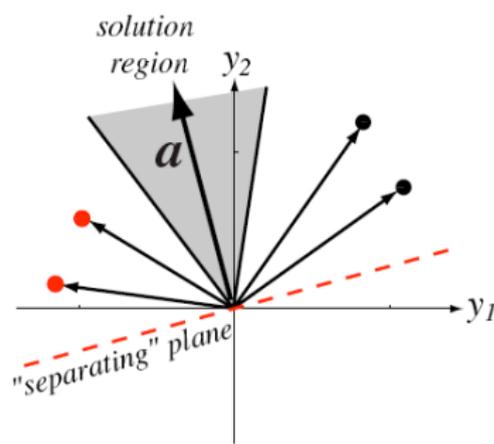
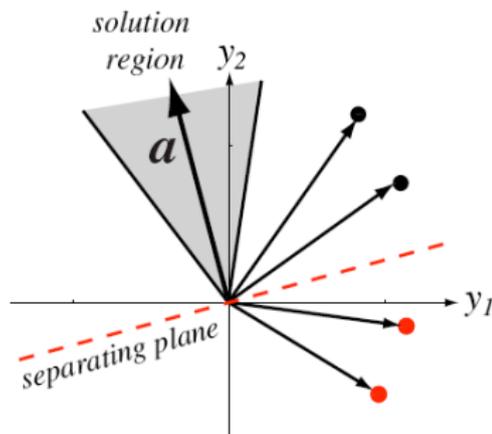
Eine vereinfachte Schreibweise ergibt sich, wenn alle Beispiele aus  $\omega_2$  durch ihre Negative ersetzt werden. Dadurch ändert sich die Aufgabe in:

Finde einen Gewichtsvektor, für den

$$a^\top y > 0$$

für **alle** Beispiele gilt. Solch ein Gewichtsvektor heißt auch **separierender Vektor** oder **Lösungsvektor**.

# Gewichtsraum



Jeder Gewichtsvektor  $a$  ist ein Punkt im **Gewichtsraum**. Jedes Beispiel  $y_i$  beschränkt die möglichen Lösungsvektoren auf einen Halbraum, denn es muß  $a^\top y_i > 0$  gelten.

Die Schnittmenge aus allen Halbräumen ist die **Lösungsregion** des Problems.

# Übersicht

- 1 Linear trennbare 2-Kategorienprobleme
- 2 Gradientenabstieg**
- 3 Perzeptron

# Gradientenabstieg (1)

**Problem:** Finde Lösungsvektor für eine Menge von  $n$  Ungleichungen  $a^\top y_i > 0$ .

**Lösungsansatz:** Definiere eine **Fehlerfunktion**  $J(a)$ , die genau dann minimal wird, wenn  $a$  ein Lösungsvektor ist  $\Rightarrow$  Statt  $n$  Ungleichungen zu lösen, muß nun eine skalare Funktion minimiert werden.

Häufig wird zur Minimierung ein **Gradientenabstieg** eingesetzt: Starte an einer beliebigen Stelle  $a_1$  im Gewichtsraum, berechne dort den Gradienten  $\nabla J(a_1)$  (Richtung des steilsten Anstiegs), bewege Dich einen Schritt in umgekehrter Richtung (steilster Abstieg), usw.

$$a_{k+1} = a_k - \eta_k \nabla J(a_k).$$

Schrittgröße wird durch die **Lernrate**  $\eta_k$  kontrolliert.

## Gradientenabstieg (2)

### Algorithmus 1: Gradientenabstieg

```
begin initialize  $a$ , Schwellwert  $\theta$ ,  $\eta_k$ ,  $k = 0$ 
  do  $k \leftarrow k + 1$ 
     $a \leftarrow a - \eta_k \nabla J(a)$ 
  until  $|\eta_k \nabla J(a)| < \theta$ 
  return  $a$ 
end
```

Probleme:

- Gradientenabstieg findet nur lokale Minima.
- Wenn  $\eta_k$  zu klein gewählt wird, ist die Konvergenzrate zu klein, oder die Prozedur konvergiert gegen einen Wert, der kein Minimum ist.
- Wenn  $\eta_k$  zu groß gewählt wird, kann das Ziel verfehlt werden, oder die Prozedur oszilliert zwischen 2 Werten, oder divergiert sogar.

# Newtonverfahren (1)

Das Newtonverfahren setzt **automatisch** die richtige Schrittweite und Abstiegsrichtung.

**Annahme:** Die Fehlerfunktion kann gut durch die Taylorentwicklung zweiter Ordnung angenähert werden:

$$J(a) \approx J(a_k) + \nabla J^\top (a - a_k) + \frac{1}{2}(a - a_k)^\top H(a - a_k)$$

mit der **Hesse-Matrix**  $H_{ij} = \partial_{ij}^2 J$  der zweiten Ableitungen. Einsetzen von  $a_{k+1} = a_k - \eta_k \nabla J$  ergibt

$$J(a_{k+1}) \approx J(a_k) - \eta_k \|\nabla J\|^2 + \frac{1}{2} \eta_k^2 \nabla J^\top H \nabla J$$

wird minimiert durch die Lernrate (Ableitung nach  $\eta = 0$  setzen)

$$\eta_k = \frac{\|\nabla J\|^2}{\nabla J^\top H \nabla J}$$

## Newtonverfahren (2)

Stattdessen kann die Fehlerfunktion direkt durch  $a$  minimiert werden:

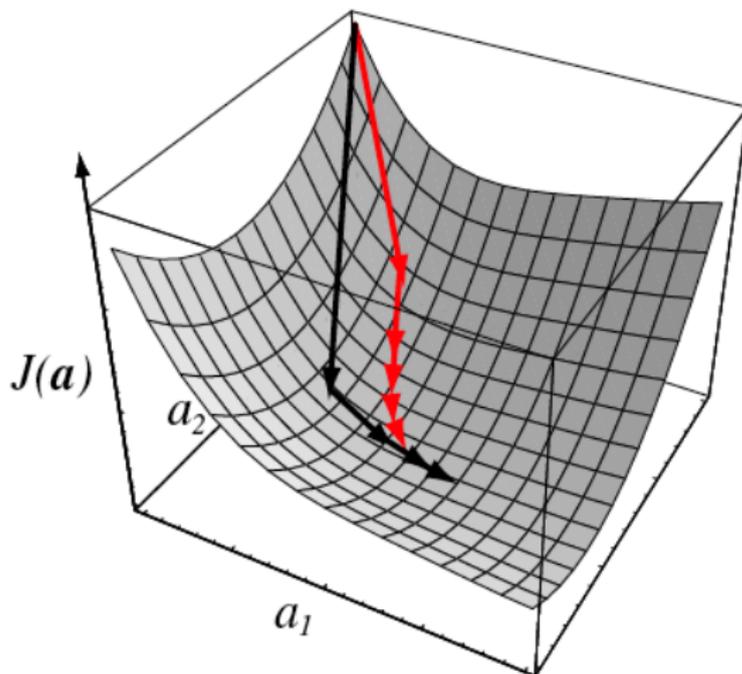
$$a_{k+1} = a_k - H^{-1} \nabla J$$

### Algorithmus 2: Newtonverfahren

```
begin initialize  $a$ , Schwellwert  $\theta$ 
do
   $a \leftarrow -H^{-1} \nabla J(a)$ 
until  $|H^{-1} \nabla J(a)| < \theta$ 
return  $a$ 
end
```

- Schnellere Konvergenz als einfacher Gradientenabstieg
- $H$  ist manchmal nicht invertierbar
- Berechnung von  $H^{-1}$  kann in hochdimensionalen Problemen aufwendig sein  $O(d^3)$ .

# Gradientenabstieg und Newtonverfahren



# Übersicht

- 1 Linear trennbare 2-Kategorienprobleme
- 2 Gradientenabstieg
- 3 Perzeptron**

# Perzeptron-Kriterium

Um Gradientenabstieg anwenden zu können, muß eine Fehlerfunktion angegeben werden.

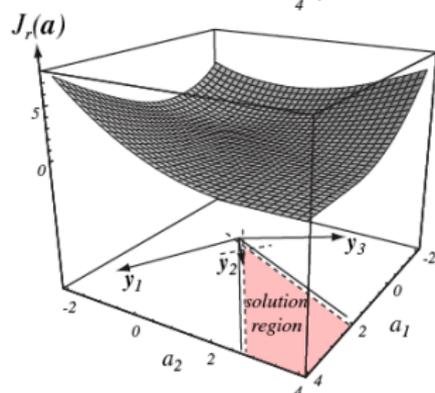
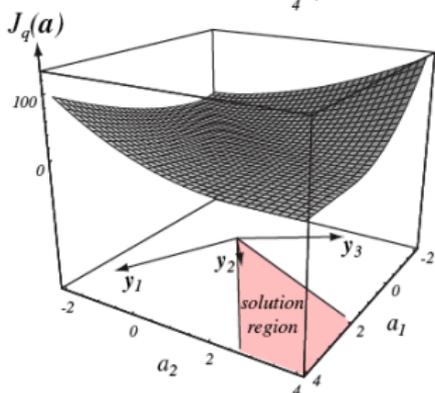
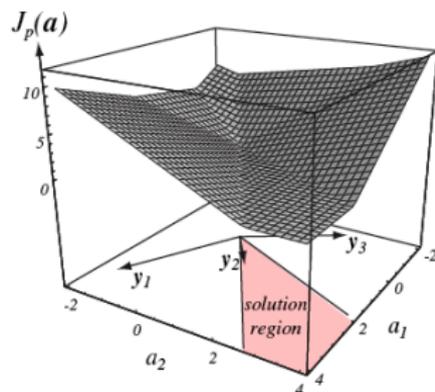
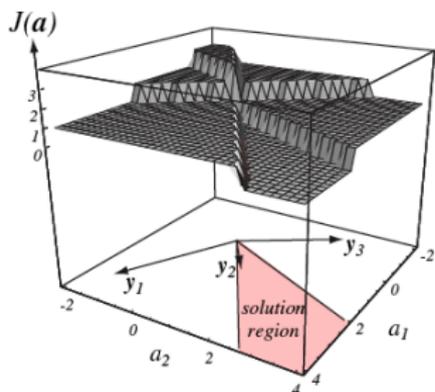
- Anzahl der falsch klassifizierten Beispiele eignet sich nicht für Gradientenabstieg, da diese Funktion stückweise konstant ist.
- Bessere Alternative: **Perzeptron-Kriterium**

$$J_p(a) = \sum_{y \in \mathcal{Y}} (a^\top y)$$

$\mathcal{Y}$  : Menge der durch  $a$  falsch klassifizierten Beispiele. Da für falsch klassifizierte  $y_i$  immer  $a^\top y_i < 0$  gilt, ist  $J_p$  nie negativ und nur gleich 0 bei einem Lösungsvektor.

- Das Perzeptron-Kriterium ist stückweise linear und proportional zur Summe der Distanzen aller falsch klassifizierten Beispiele zur Hyperebene.

# Vergleich verschiedener Fehlerfunktionen



# Perzeptron-Algorithmus

Gradient von  $J_p$ :

$$\partial_{a_j} J_p = \partial_{a_j} \sum_{y \in \mathcal{Y}} (-a_j y_j) = \sum_{y \in \mathcal{Y}} y_j \quad \Rightarrow \quad \nabla J_p = \sum_{y \in \mathcal{Y}} (-y)$$

$$\text{Lernregel: } a_{k+1} = a_k - \eta_k \nabla J_p = a_k + \eta_k \sum_{y \in \mathcal{Y}_k} y$$

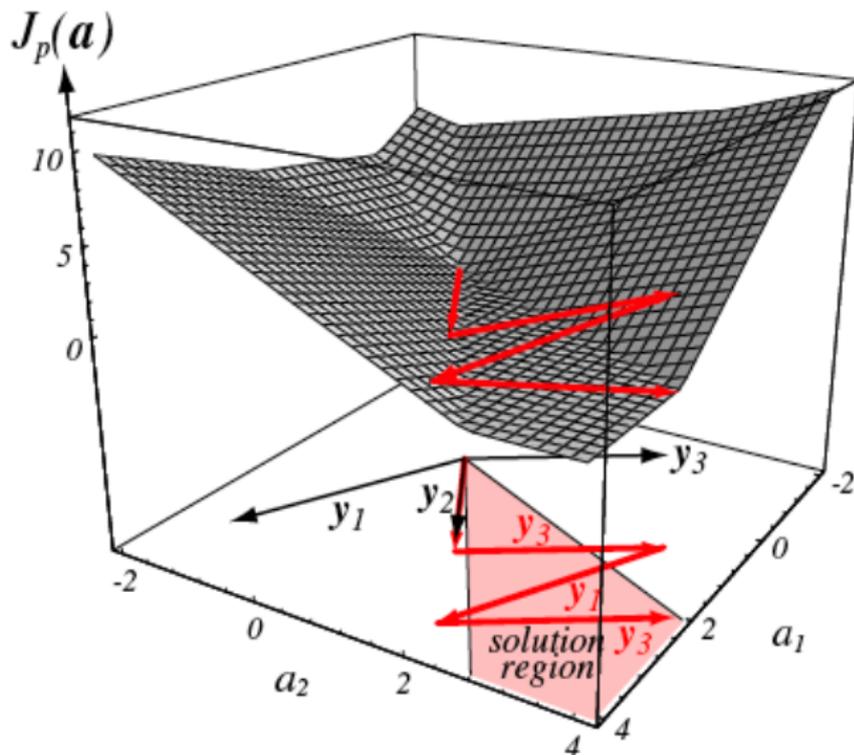
## Algorithmus 3: Batch-Perzeptron

```

begin initialize  $a, \eta_k$ , Schwellwert  $\theta, k = 0$ 
  do  $k \leftarrow k + 1$ 
     $a \leftarrow a + \eta_k \sum_{y \in \mathcal{Y}_k} y$ 
  until  $|\eta_k \sum_{y \in \mathcal{Y}_k} y| < \theta$ 
  return  $a$ 
end
  
```

Den nächsten Gewichtsvektor erhält man durch Addition eines Teils der Summe aller falsch klassifizierten Beispiele.

# Beispiel: Perzeptron bei 3 Datenpunkten



# Vereinfachung: Perzeptron mit Einzelbeispielkorrektur

Algorithmus 3: Batch-Perzeptron mit Einzelbeispielkorrektur und

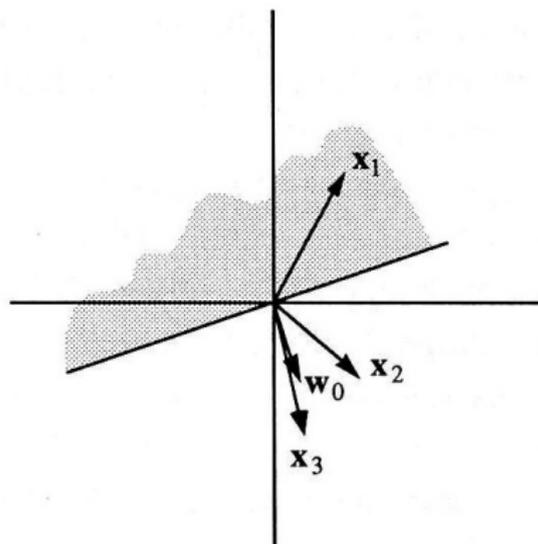
$$\eta_k = 1$$

```
begin initialize  $a$ ,  $k = 0$ 
  do  $k \leftarrow (k + 1) \bmod n$ 
    if  $y_k$  falsch klassifiziert then  $a \leftarrow a + y_k$ 
  until alle  $y_i$  richtig klassifiziert
  return  $a$ 
end
```

- Korrektur wird nur vorgenommen, wenn ein Beispiel falsch klassifiziert wird.
- Wenn das Problem linear trennbar ist, läßt sich beweisen, daß nur eine endliche Anzahl von Korrekturen notwendig sind.
- Falls ein Beispiel  $y_k$  einen größeren Winkel als  $90^\circ$  zu  $a$  bildet, wird  $a$  in Richtung  $y_k$  gezogen.

# Aufgabe 9.1

1) Anfangssituation



Konstruieren Sie für die links gezeigte Ausgangssituation mit 3 Beispielen die Zwischenschritte und das Ergebnis des Perzeptron-Algorithmus mit Einzelbeispielkorrektur.

## Algorithmus 3

```

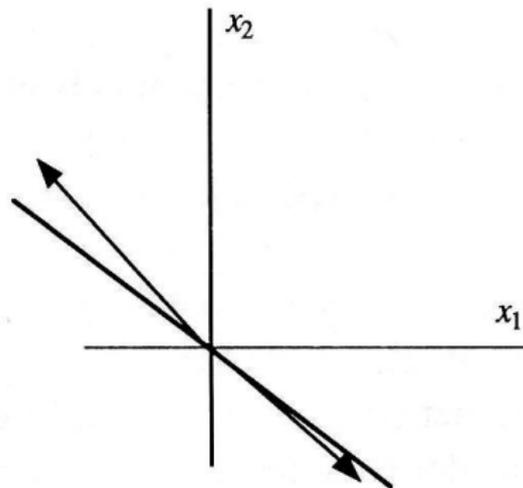
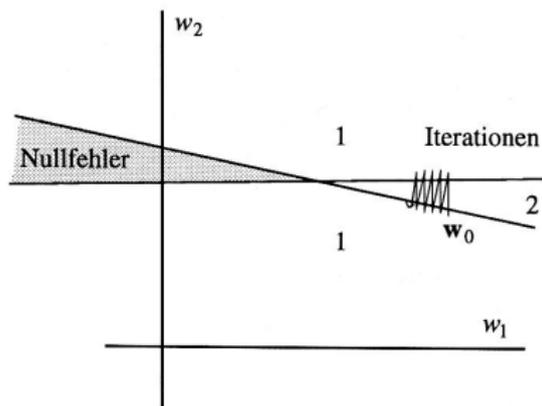
begin initialize  $a, k = 0$ 
  do  $k \leftarrow (k + 1) \bmod n$ 
    if  $y_k$  falsch klassifiziert then
 $a \leftarrow a + y_k$ 
  until alle  $y_i$  richtig klassifiziert
  return  $a$ 
end
```

[Rojas, 1993]

# Eigenschaften des Perzeptron-Algorithmus

- Längere Vektoren ziehen den Gewichtsvektor stärker an als kürzere  $\Rightarrow$  Wenn alle Daten den gleichen Einfluß haben sollen, ist Normierung sinnvoll.
- Falls ein Gewichtsvektor normierte Beispiele trennt, trennt er auch die unnormierten Beispiele.
- Die Länge des Gewichtsvektors steigt langsam mit der Anzahl der Korrekturen. Jede weitere Korrektur dreht den Vektor daher um einen kleiner werdenden Winkel (entspricht kleiner werdenden Lernrate).

# Ungünstiger Fall für den Perzeptron-Lernalgorithmus



[Rojas, 1993]

In ungünstigen Fällen kann die Rechenzeit exponentiell in der Anzahl der Beispiele sein.

# Perzeptron-Lernen als lineares Programm

**Ursprüngliche Formulierung des Problems:** Finde einen Gewichtsvektor  $a$ , für den  $a^\top y_i > 0$  für alle Beispiele  $y_i$  gilt.

Schreibe  $y_i$  als Zeilen einer Matrix  $Y$

$$Y = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,d} \\ \vdots & \ddots & \vdots & \\ y_{n,1} & y_{n,2} & \cdots & y_{n,d} \end{pmatrix} \Rightarrow Ya > 0$$

Einführung von **künstlichen Variablen**  $\tau = (\tau_1, \tau_2, \dots, \tau_n)^\top$  mit  $\tau_i > 0$ :

$$Ya + \tau > 0$$

**Aufgabe:** Minimiere  $\tau$  und  $a$ , wobei  $Ya > 0$  und  $\tau > 0$  gelten soll  $\Rightarrow$  **Lineares Programm** läuft in **polynomieller** Zeit. Wenn  $\tau_{\text{opt}} = 0$ , dann ist das Problem linear trennbar.

# Relaxationsmethoden

Das Perzeptron-Kriterium ist zwar stetig, nicht aber seine Ableitung. Dies kann zu numerischen Instabilitäten beim Gradientenabstieg an den Unstetigkeiten der Ableitung führen.

Die alternative Fehlerfunktion

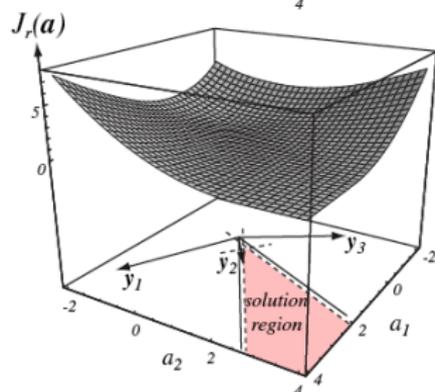
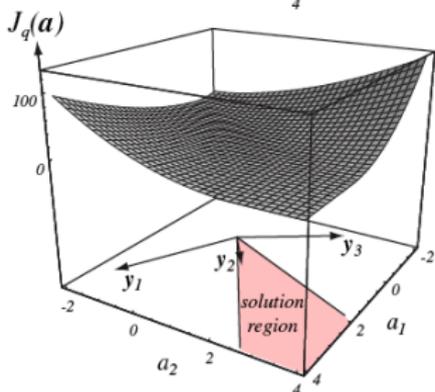
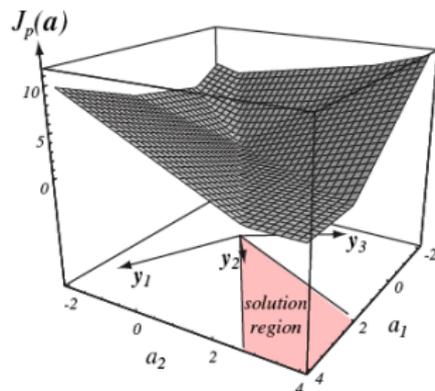
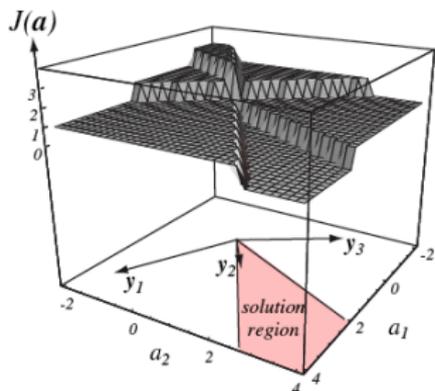
$$J_q(a) = \sum_{y \in \mathcal{Y}} (a^\top y)^2$$

hat keine Unstetigkeiten im Gradienten, strebt aber oft gegen die Grenzen der Lösungsregion (insbesondere  $a = 0$ ). Außerdem kann die Lösung von wenigen großen Vektoren dominiert werden. Bessere Fehlerfunktion:

$$J_r(a) = \frac{1}{2} \sum_{y \in \mathcal{Y}} \frac{(a^\top y - b)^2}{\|y\|^2}$$

Der resultierende Gradientenabstieg ist ein Beispiel für die sog. **Relaxationsmethoden**.

# Vergleich verschiedener Fehlerfunktionen



# Supportvektormaschinen

Die bisherigen Verfahren haben i.A. keine eindeutige Lösung, sondern eine ganze Lösungsregion. Im Gegensatz dazu hat die SVM eine eindeutige Lösung: aus der Lösungsregion wird diejenige Hyperebene gewählt, die **maximalen Abstand** zu den nächsten Trainingspunkten hat.

