Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Lineare Klassifikatoren, Kernel-Maschinen

Christoph Sawade/Niels Landwehr
Jules Rasetaharison
Tobias Scheffer

Überblick

- Perzeptron-Algorithmus (Rosenblatt, 1960): Modell neuronaler Informationsverarbeitung
- Kann „nur“ sehr eingeschränkte Klasse linear separierbarer Funktionen lernen (Minsky & Papert, 1969).
- Multilayer-Perzeptron, Backpropagation-Algorithmus: Kann beliebig komplizierte Separatoren lernen, aber extrem ineffizient.
- Vapnik, 1979: Statistische Lerntheorie führt zur Support-Vektor-Maschine (Kernel-Maschine).
- Kernel-Maschinen: Gute Klassifikatoren und effizient, aber keine biologische Plausibilität.

Was man über Hyperebenen wissen sollte…

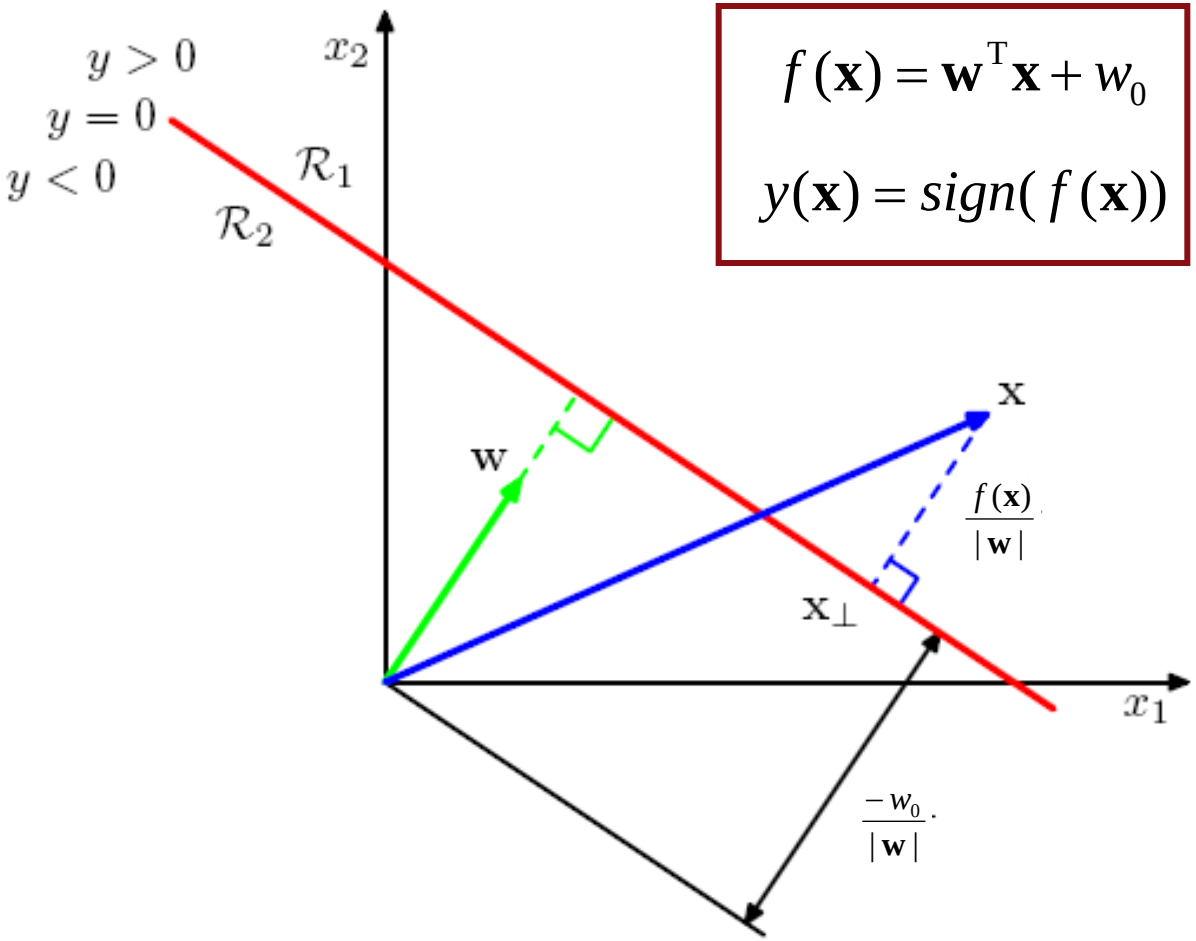
- Wir beschreiben eine Hyperebene durch $\langle \mathbf{w}, \mathbf{x} \rangle = 0$, wobei $\mathbf{x}, \mathbf{w} \in \mathbb{R}^d$

1. Was ist $\mathbf{w}$?
2. Wann gilt $\langle \mathbf{w}, \mathbf{x} \rangle > 0$ bzw. $\langle \mathbf{w}, \mathbf{x} \rangle < 0$

- Die Ebene ist ein Model, mit dem klassifiziert werden kann

1. Wie bestimmt man eine derartige Ebene für eine Menge $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$?
2. Wie viele gibt es?
3. Welche ist die beste (für ungesehene Daten)?

Lineare Klassifikatoren



$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

$$y(\mathbf{x}) = sign(f(\mathbf{x}))$$

Lineare Klassifikatoren

- Umformulierung mit zusätzlichem, konstanten Eingabeattribut x_0=1:
  - $f(\mathbf{x}) = \mathbf{w}_{(1..n)}^{\mathrm{T}} \mathbf{x}_{(1..n)} + w_0$

$$= (w_1 \ \ldots \ w_n) \begin{pmatrix} x_1 \\ \ldots \\ x_n \end{pmatrix} + w_0 = (w_0 \ \ w_1 \ \ldots \ w_n) \begin{pmatrix} x_0 \\ x_1 \\ \ldots \\ x_n \end{pmatrix}$$

$$= \mathbf{w}_{(0..n)}^{\mathrm{T}} \mathbf{x}_{(0..n)}$$

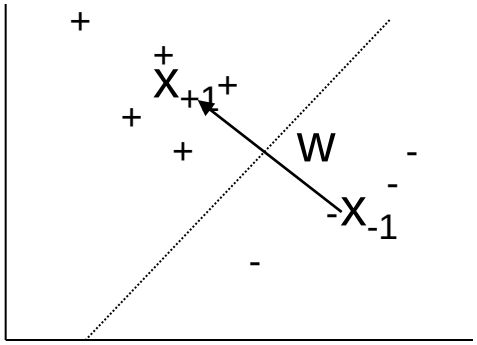
$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \mathbf{x} + w_0$$

$$y(\mathbf{x}) = sign(f(\mathbf{x}))$$

Bestimmung von linearen Klassifikatoren

- „Ad hoc“-Ansätze
  - Rocchio, Fisher-Diskriminante
  - K-nearest neighbour
- Gradientenabstieg
  - Perzeptron, Margin-Perzeptron
- Maximal-Margin-Ansätze
  - SVM (hard- and softmargin)
  - Logistic Regression
- Kernelmaschinen
  - Nicht-lineare Klassifikatoren
  - Strukturierte Ein- und Ausgabe

Rocchio



- $\overline{\mathbf{x}}_{-1}$: Mittelpunkt der neg. Beispiele
- $\overline{\mathbf{x}}_{+1}$: Mittelpunkt der pos. Beispiele
- Trennebene: Normalenvektor = $(\mathbf{x}_{+1}\text{-}\mathbf{x}_{\text{-}1})$

$$f(\mathbf{x}) = (\overline{\mathbf{x}}_{+1} - \overline{\mathbf{x}}_{-1})\mathbf{x} + w_0$$

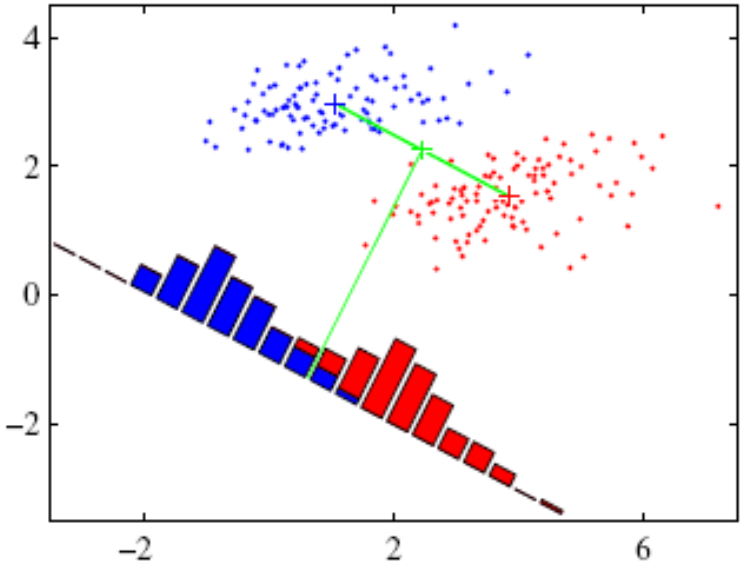
- Bestimmung von w_0: Mittelpunkt $(\mathbf{x}_{+1}+\mathbf{x}_{\text{-}1})/2$ muss auf der Ebene liegen.

$$f((\overline{\mathbf{x}}_{-1} + \overline{\mathbf{x}}_{+1})/2) = (\overline{\mathbf{x}}_{+1} - \overline{\mathbf{x}}_{-1})(\overline{\mathbf{x}}_{-1} + \overline{\mathbf{x}}_{+1})/2 + w_0 = 0$$

$$\Leftrightarrow w_0 =$$

Rocchio

- Trennebenen hat maximalen Abstand von den Mittelpunkten der Klassen.
- Trainingsbeispiele können falsch klassifiziert werden.
- **Problem:** Differenz der Mittelwerte kann schlechter Normalenvektor für Diskrimination sein.



Fisher-Diskriminante

- Idee:
  - Differenz der Mittelwerte soll große Projektion auf Normalenvektor haben.

$$\max\left[m_{+1} - m_{-1} = \mathbf{w}^{\mathrm{T}}\overline{\mathbf{x}}_{+1} - \mathbf{w}^{\mathrm{T}}\overline{\mathbf{x}}_{-1}\right]$$

  - Innerhalb der Klassen soll $f(\mathbf{x})$ geringe Varianz haben.

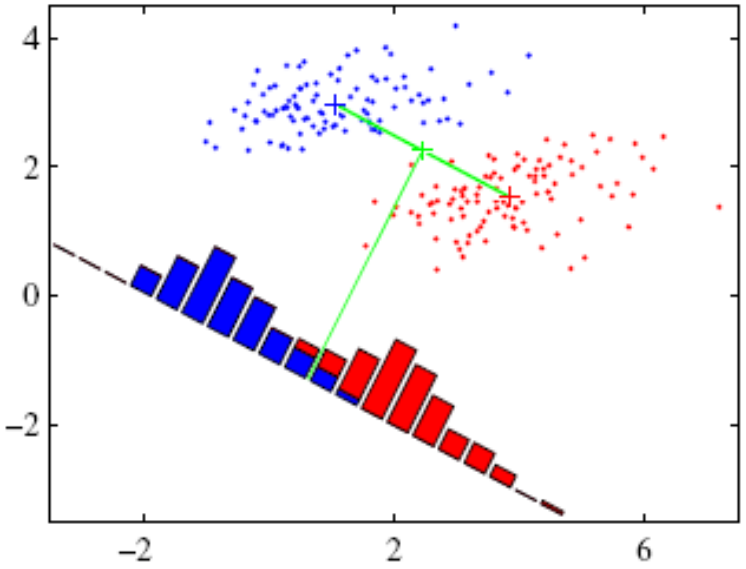
$$\min\left[s_{+1}^2 = \sum_{(\mathbf{x},+1)\in L}(\mathbf{w}^{\mathrm{T}}\mathbf{x} - \mathbf{w}^{\mathrm{T}}\overline{\mathbf{x}}_{+1})^2\right],$$

$$\min\left[s_{-1}^2 = \sum_{(\mathbf{x},-1)\in L}(\mathbf{w}^{\mathrm{T}}\mathbf{x} - \mathbf{w}^{\mathrm{T}}\overline{\mathbf{x}}_{-1})^2\right]$$

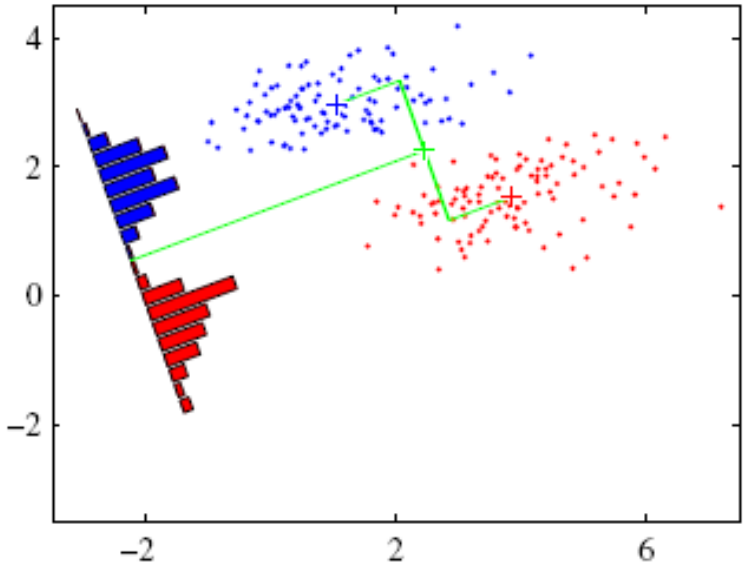
Fisher-Diskriminante

- Fisher-Optimierungskriterium:

$$J(\mathbf{w}) = \frac{m_{+1} - m_{-1}}{s_{+1}^2 + s_{-1}^2}$$



Rocchio



Fisher

Fisher-Diskriminante

- Fisher-Optimierungskriterium, umgeformt:

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_B\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_W\mathbf{w}}$$

S_B... *K*ovarianz zwischen den Klassen

S_W... Kovarianz innerhalb der Klassen

- Maximieren: Ableiten, Null setzen, nach $\mathbf{w}$ auflösen:

$$\mathbf{w} \propto \mathbf{S}_W^{-1}(\bar{\mathbf{x}}_{+1} - \bar{\mathbf{x}}_{-1})$$

$$\mathbf{S}_W = \sum_{(\mathbf{x},+1)\in L}(\mathbf{x} - \bar{\mathbf{x}}_{+1})(\mathbf{x} - \bar{\mathbf{x}}_{+1})^{\mathrm{T}} + \sum_{(\mathbf{x},-1)\in L}(\mathbf{x} - \bar{\mathbf{x}}_{-1})(\mathbf{x} - \bar{\mathbf{x}}_{-1})^{\mathrm{T}}$$

$$\mathbf{S}_B = (\bar{\mathbf{x}}_{-1} - \bar{\mathbf{x}}_{+1})(\bar{\mathbf{x}}_{-1} - \bar{\mathbf{x}}_{+1})^{\mathrm{T}}$$

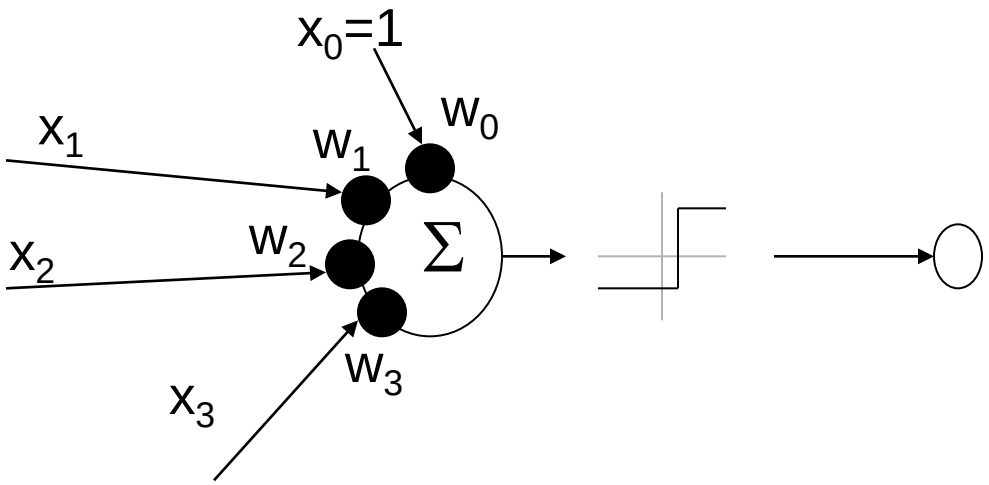
K-nearest neighbour (KNN)

- Algorithmus

1. Bestimme die k nächsten Nachbarn von $\bar{\mathbf{x}}$
   - $\mathbf{x}_1 = \arg\min_{\mathbf{x} \in D} d(\bar{\mathbf{x}}, \mathbf{x}), \ldots, \mathbf{x}_k = \arg\min_{\mathbf{x} \in D \setminus \{\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}\}} d(\bar{\mathbf{x}}, \mathbf{x})$
2. Gebe akkumulierte Klasse für $\bar{\mathbf{x}}$ zurück
   - z.B.: Durchschnitt oder Mehrheitsklasse

- **Problem:** Bisherige Ansätze reduzieren den Klassifikator nur auf Durchschnitt von Beispielen oder einzelne Repräsentanten

Perzeptron

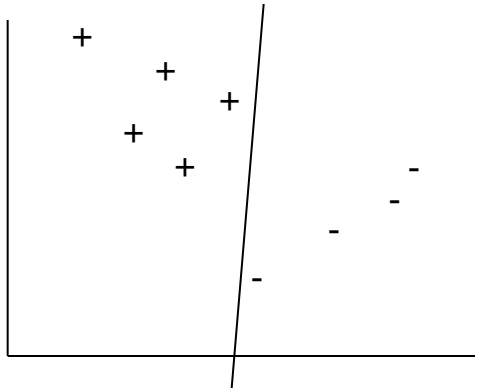


$$sign\left(\sum_{i=1}^{n} w_i x_i + w_0\right) = sign\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0\right)$$

- Originalreferenz:
  F.Rosenblatt: „*The perceptron: A probabilistic model for information storage and organization in the brain*”. 1958

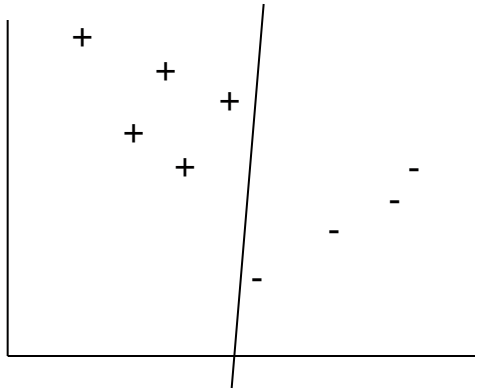
Perzeptron

- Lineares Modell:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Ziel:
  - Für alle Beispiele $(\mathbf{x}_i, +1)$:
    $$f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0$$
  - Für alle Beispiele $(\mathbf{x}_i, -1)$:
    $$f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i < 0$$



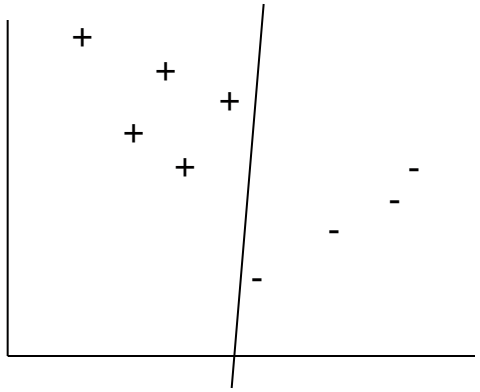
Perzeptron

- Lineares Modell:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Ziel:
  - Für alle Beispiele $(\mathbf{x}_i, +1)$:
    $f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0$
  - Für alle Beispiele $(\mathbf{x}_i, -1)$:
    $f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i < 0$
- Ziel: Für alle Beispiele:
  - $y_i f(\mathbf{x}_i) = y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0$
  - = Beispiel liegt auf der richtigen Seite der Ebene.



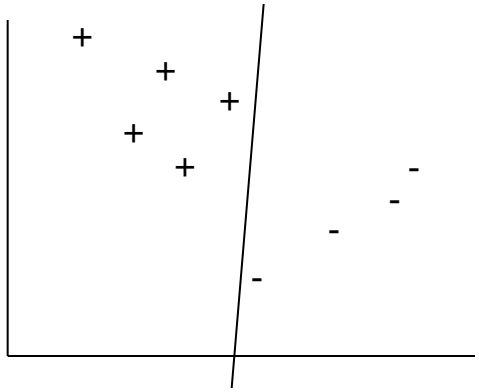
Perzeptron

- Lineares Modell:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Ziel: Für alle Beispiele:
  - $y_i f(\mathbf{x}_i) = y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0$
- Perzeptron-Optimierungskriterium:
  - $J_P(\mathbf{w}) = \sum_{(\mathbf{x}_i, y_i) \in L} \max\left\{-y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i, 0\right\}$



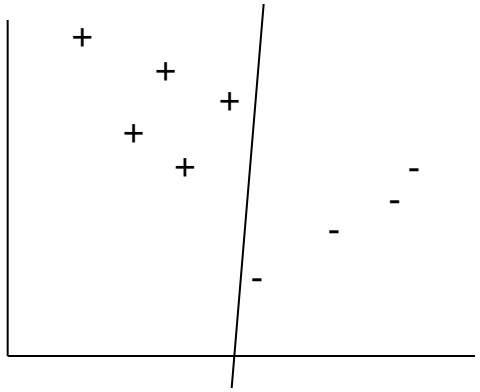
Perzeptron

- Lineares Modell:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Ziel: Für alle Beispiele:
  - $y_i f(\mathbf{x}_i) = y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0$
- Perzeptron-Optimierungskriterium:
  - $J_P(\mathbf{w}) = \sum_{(\mathbf{x}_i, y_i) \in L} \max\left\{-y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i, 0\right\}$
- Subgradient für Beispiel $(\mathbf{x}_i, y_i)$:
  - $\nabla_i J_P(\mathbf{w}) = \begin{cases} 0, & \text{wenn } y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0 \\ -y_i \mathbf{x}_i & \text{sonst} \end{cases}$



Perzeptron

- Lineares Modell:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Perzeptron-Optimierungskriterium:
  - $J_P(\mathbf{w}) = \sum_{(\mathbf{x}_i, y_i) \in L} \max\left\{-y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i, 0\right\}$
- Subgradient für Beispiel $(\mathbf{x}_i, y_i)$:
  - $\nabla_i J_P(\mathbf{w}) = \begin{cases} 0, & \text{wenn } y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0 \\ -y_i \mathbf{x}_i & \text{sonst} \end{cases}$
- Gradientenabstieg: Wiederhole, für alle Beispiele mit $y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i \leq 0$
  - $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$



Perzeptron-Algorithmus

- Lineares Modell:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$

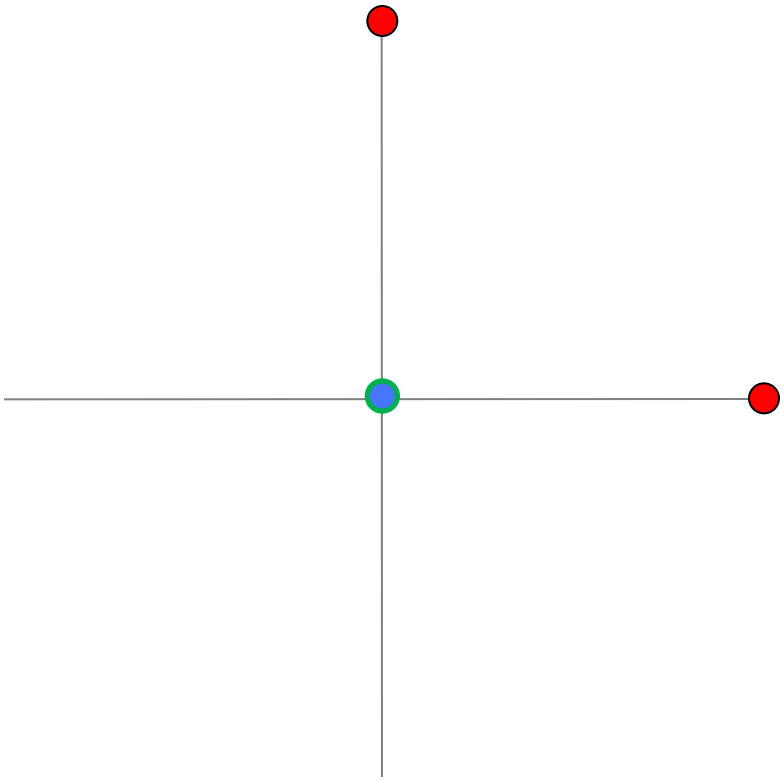
- Perzeptron-Trainingsalgorithmus:
- Solange noch Beispiele $(\mathbf{x}_i, y_i)$ mit der Hypothese inkonsistent sind, iteriere über alle Beispiele
  - Wenn $y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i \leq 0$ dann $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$

Perzeptron-Algorithmus: Beispiel

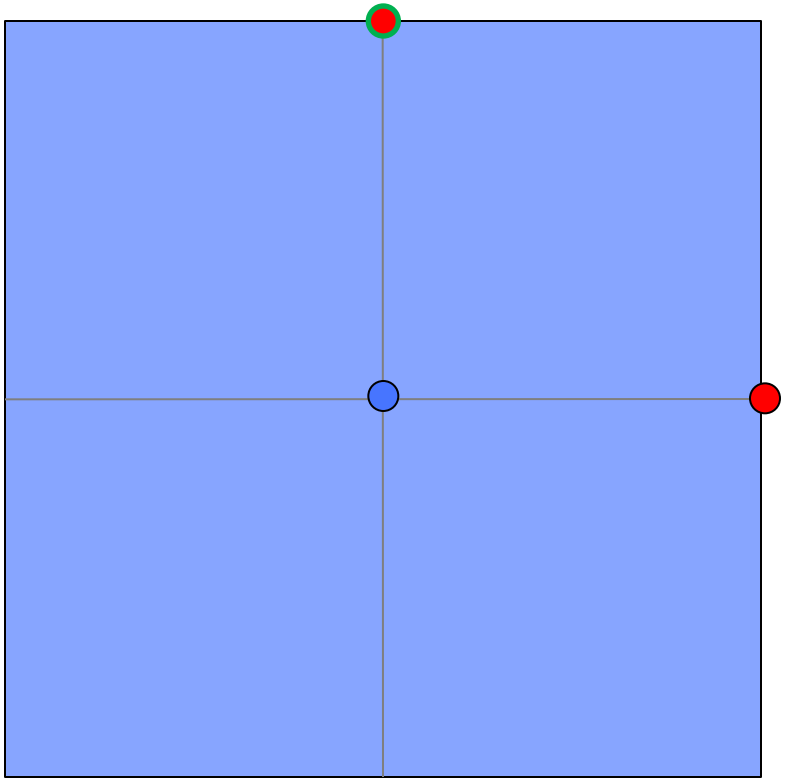
- Wenn $y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i \leq 0$ dann $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$

x_1	x_2	y	w_0	w_1	w_2	$f(\mathbf{x})$
0	0	-1	0	0	0	
0	1	+1				
1	0	+1				
0	0	-1				
0	1	+1				
1	0	+1				
0	0	-1				
0	1	+1				
1	0	+1				
0	0	-1				
0	1	+1				
1	0	+1				

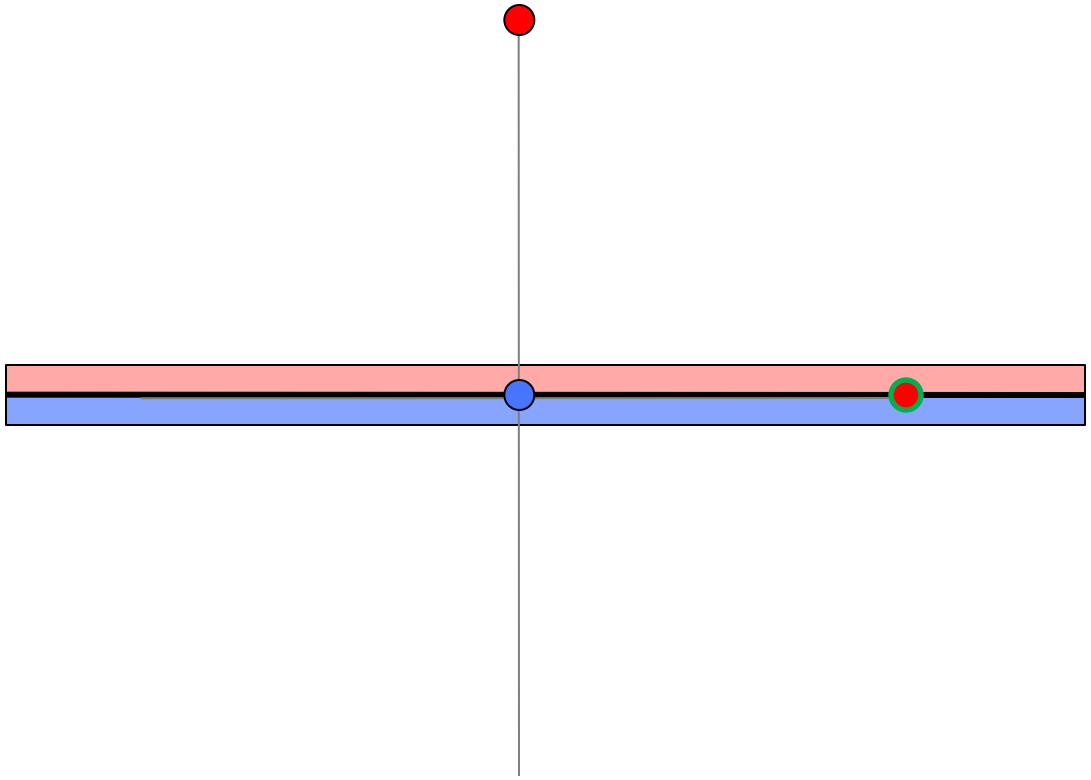
1. Iteration (1/3)



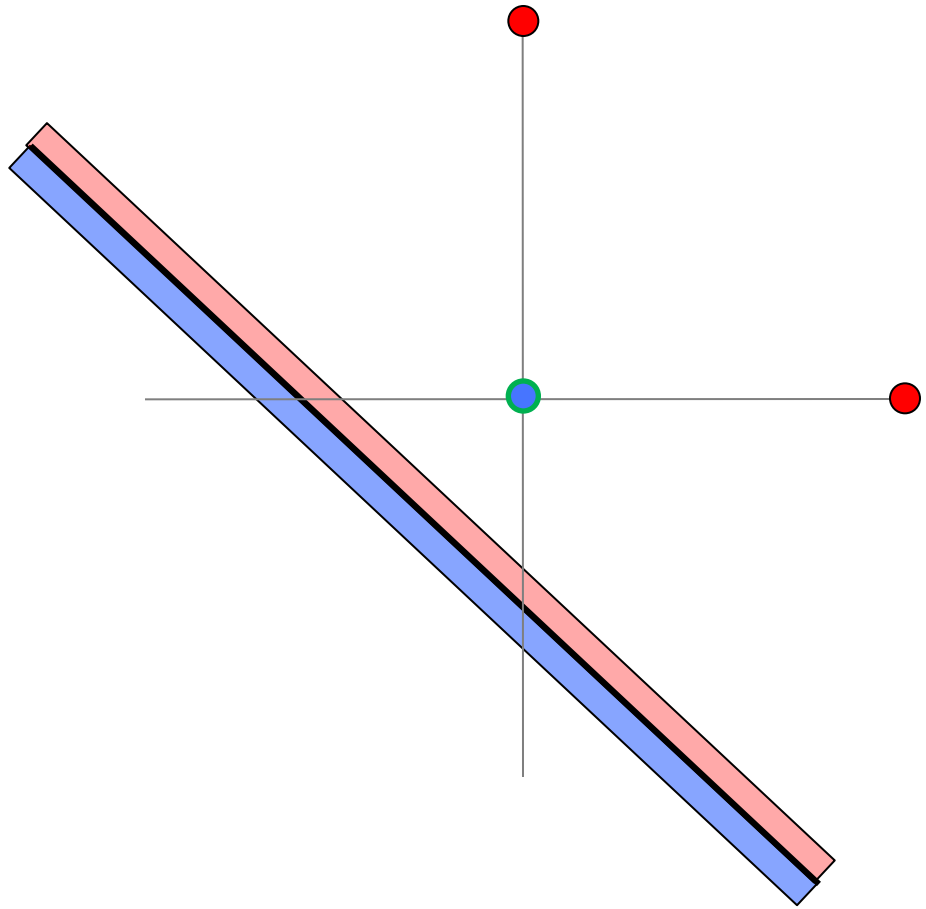
1. Iteration (2/3)



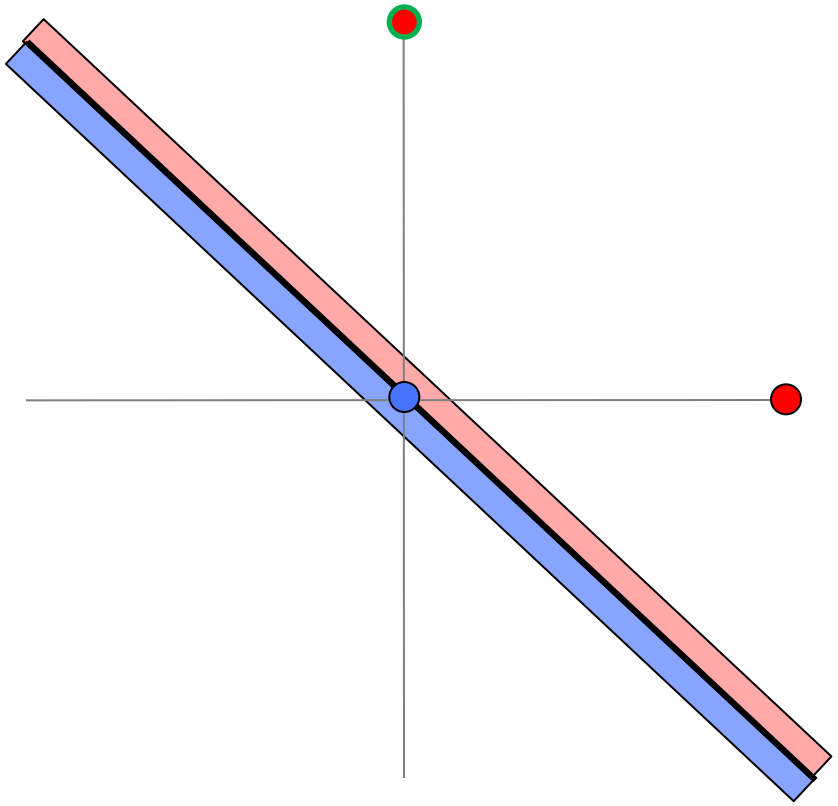
1. Iteration (3/3)



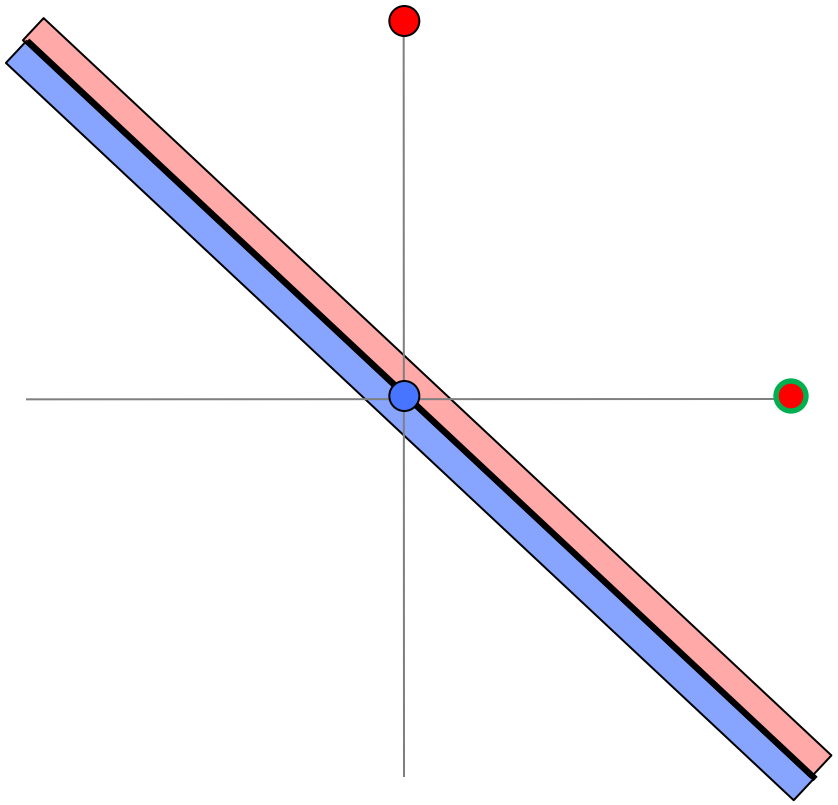
2. Iteration (1/3)



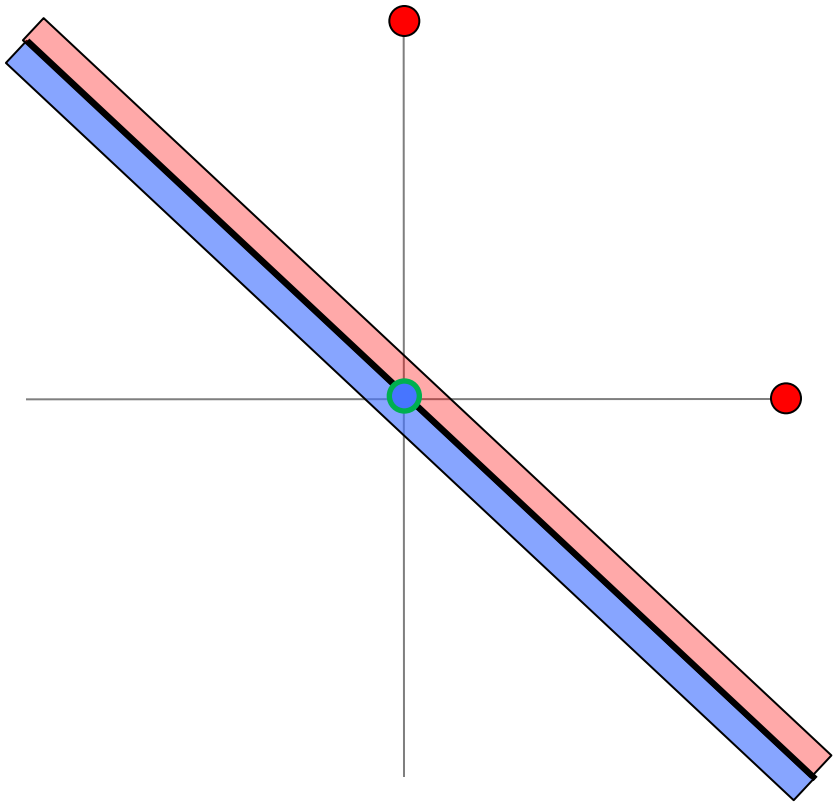
2. Iteration (2/3)



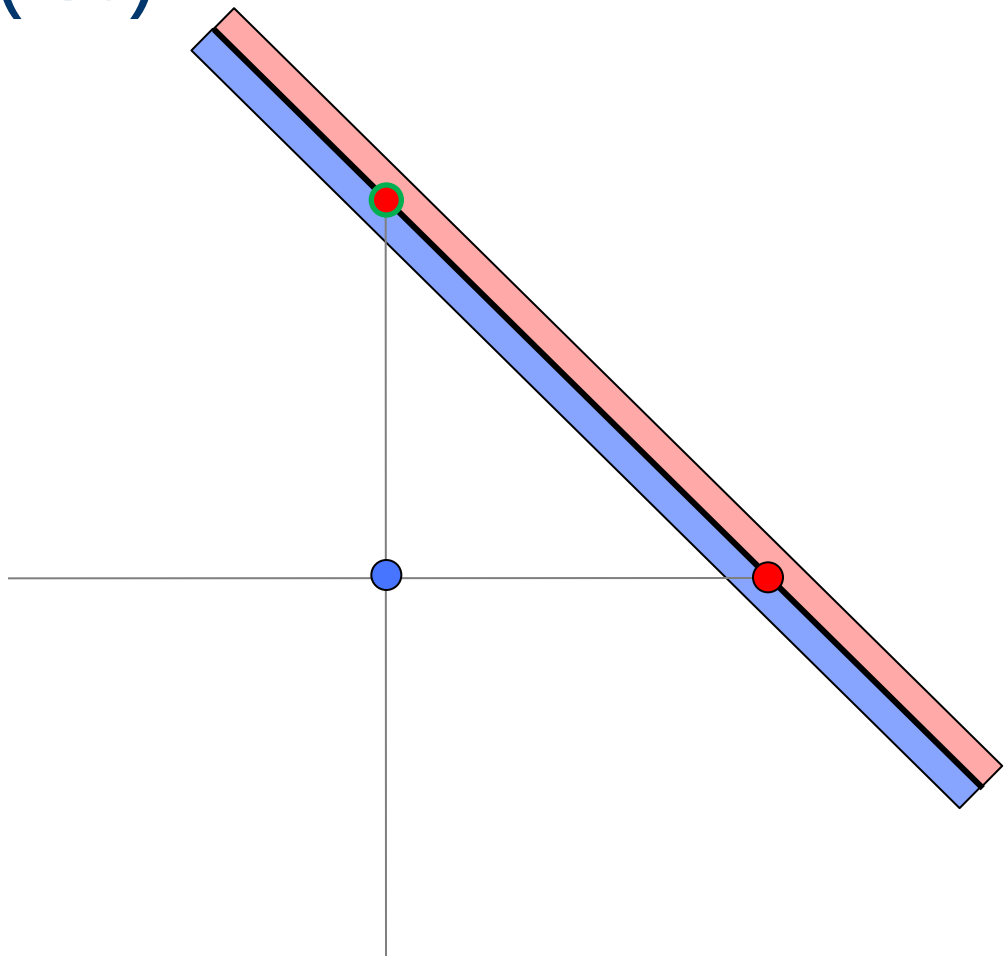
2. Iteration (3/3)



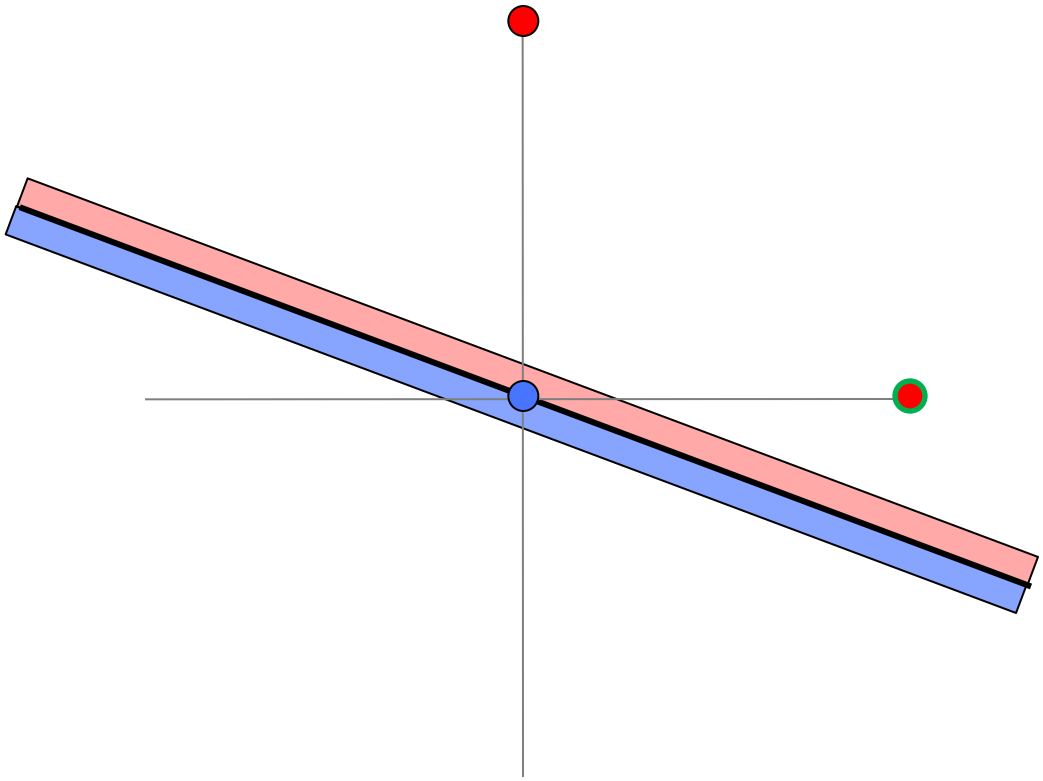
3. Iteration (1/3)



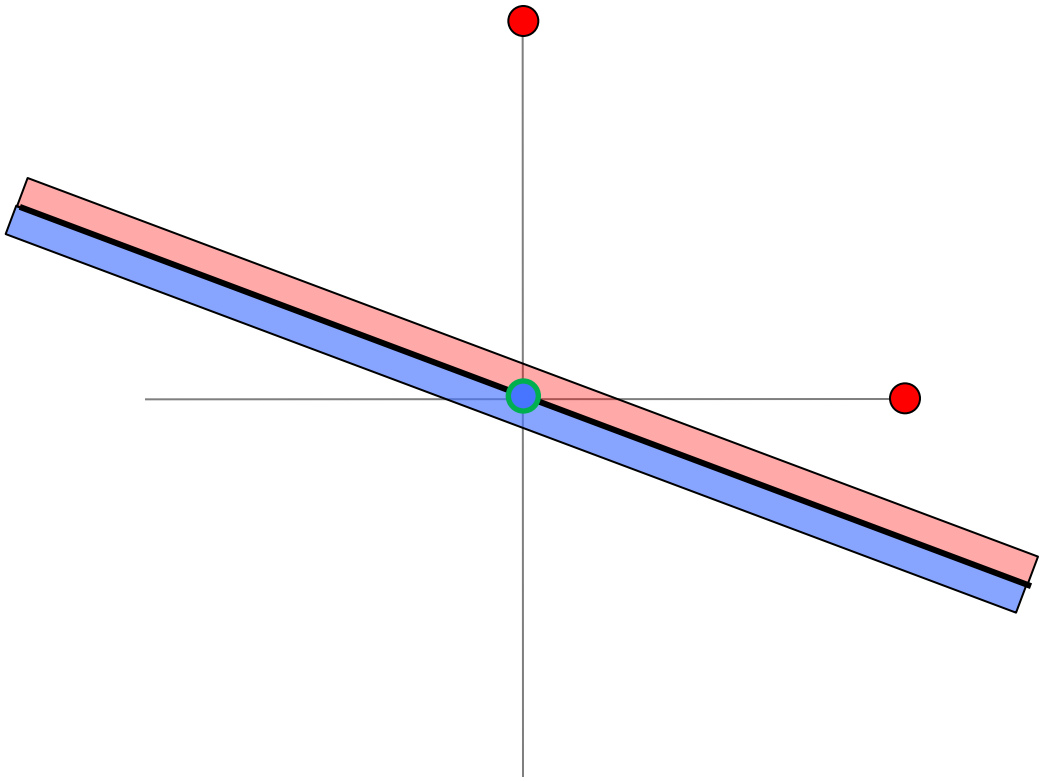
3. Iteration (2/3)



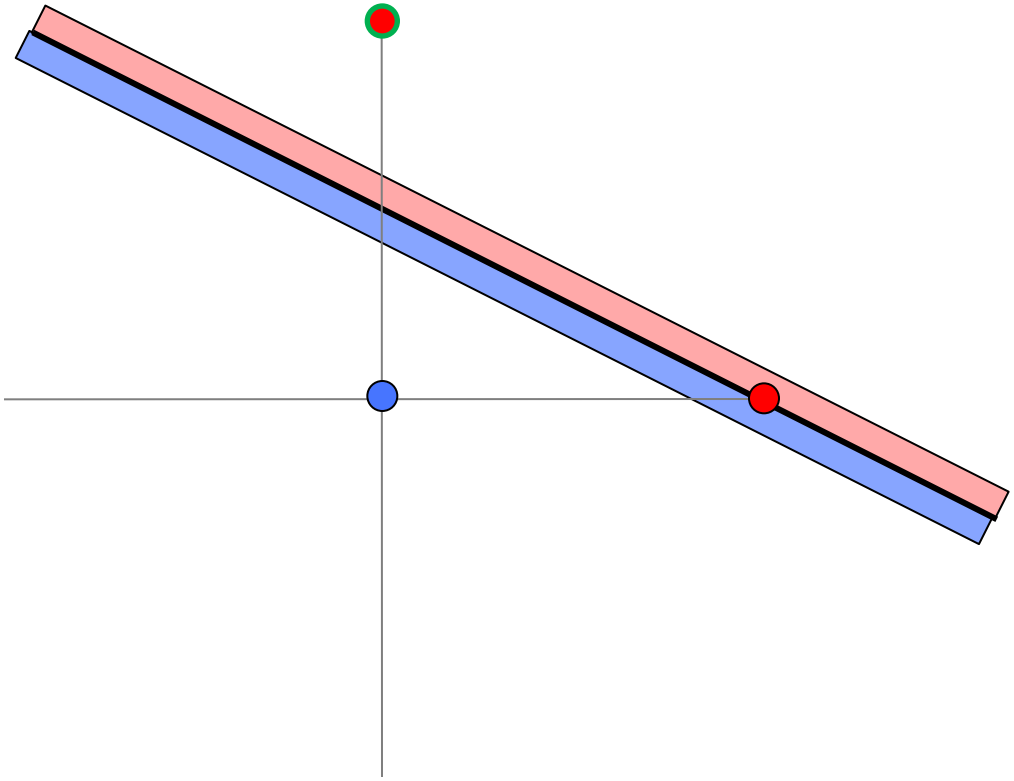
3. Iteration (3/3)



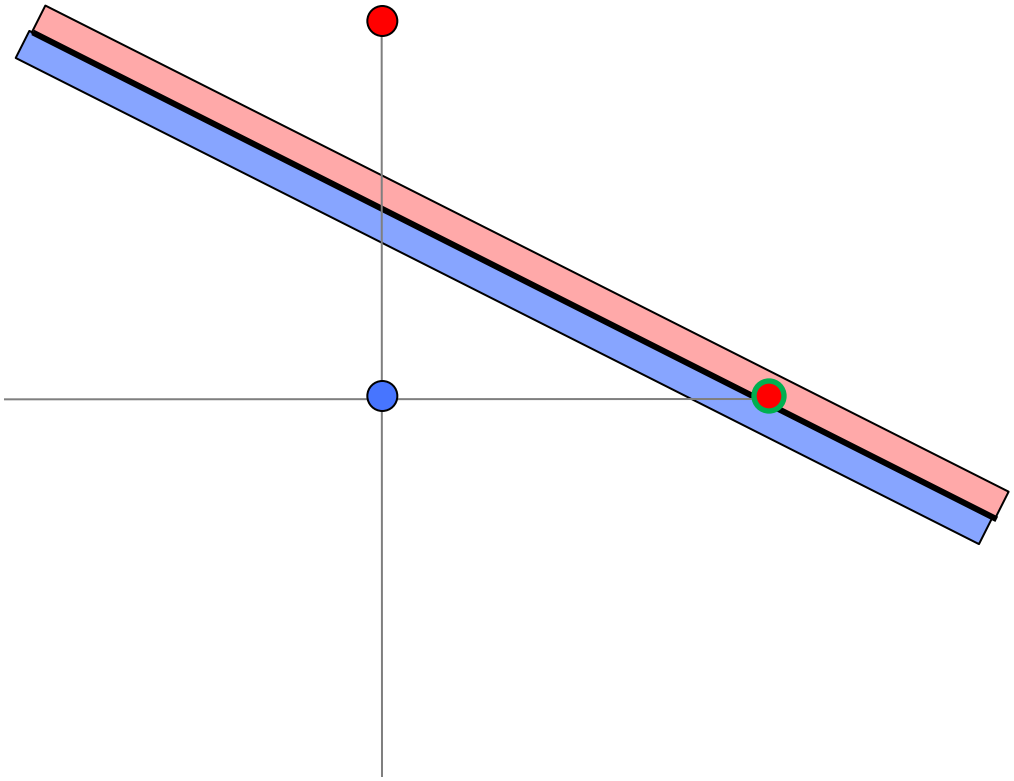
4. Iteration (1/3)



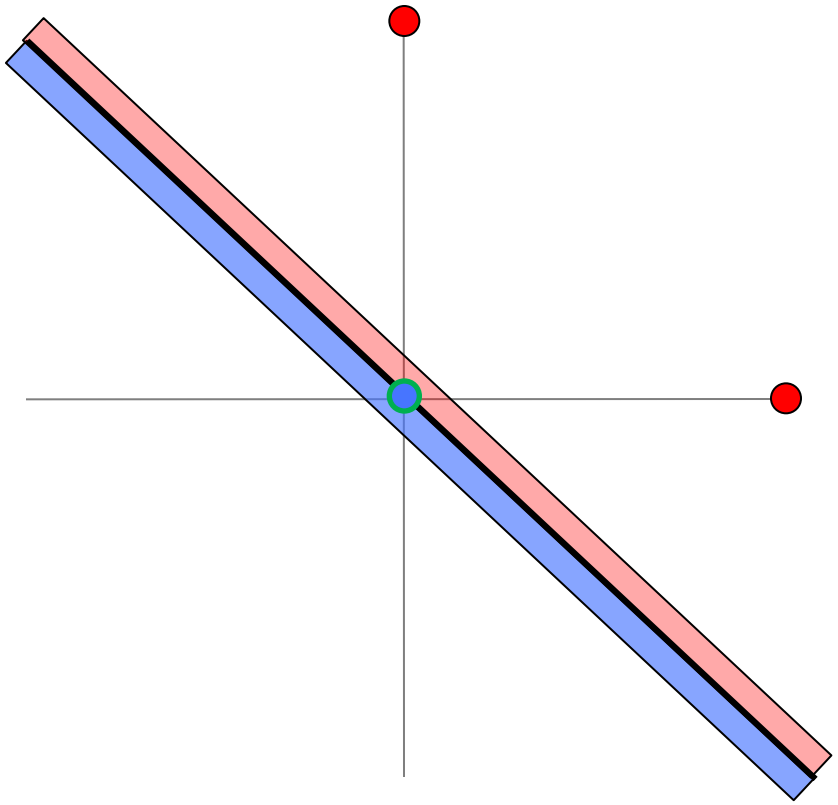
4. Iteration (2/3)



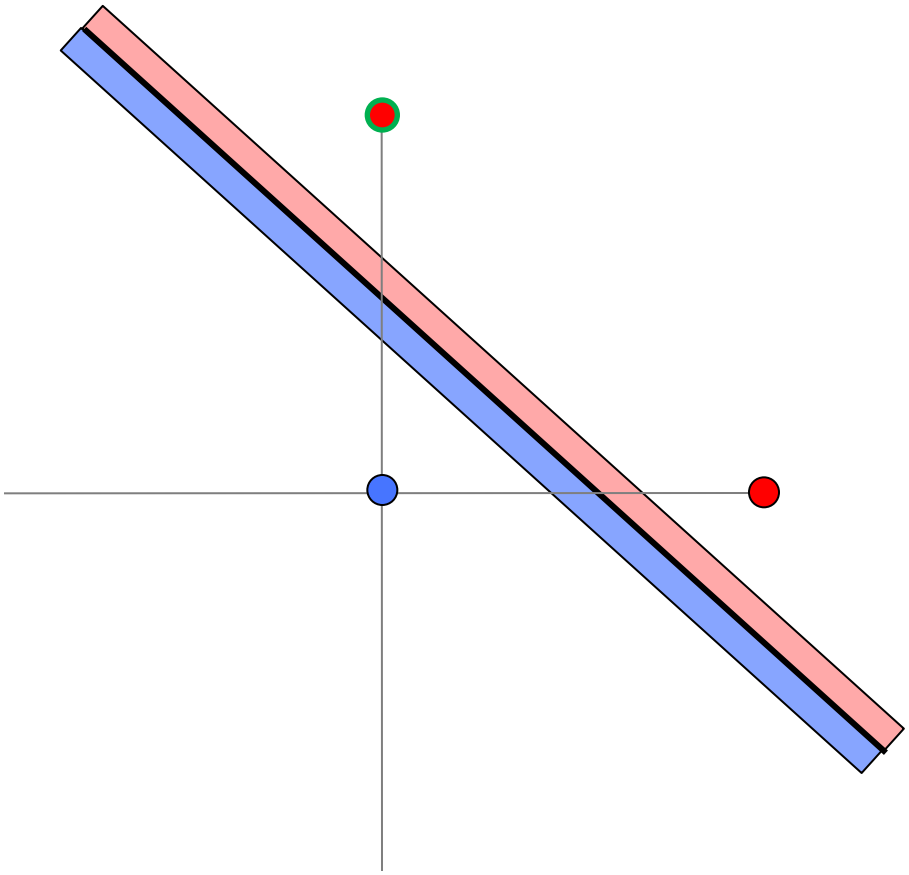
4. Iteration (2/3)



5. Iteration (1/3)



5. Iteration (2/3)



Perzeptron-Algorithmus: Beispiel

- Wenn $y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i \leq 0$ dann $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$

x_1	x_2	y	w_0	w_1	w_2	$f(\mathbf{x})$
0	0	-1	0	0	0	0
0	1	+1	-1	0	0	-1
1	0	+1	0	0	1	0
0	0	-1	1	1	1	1
0	1	+1	0	1	1	1
1	0	+1	0	1	1	1
0	0	-1	0	1	1	0
0	1	+1	-1	1	1	0
1	0	+1	0	1	2	1
0	0	-1	0	1	2	0
0	1	+1	-1	1	2	1
1	0	+1	-1	1	2	0
…	…	…	…	…	…	…
			-1	2	2	

Perzeptron-Algorithmus

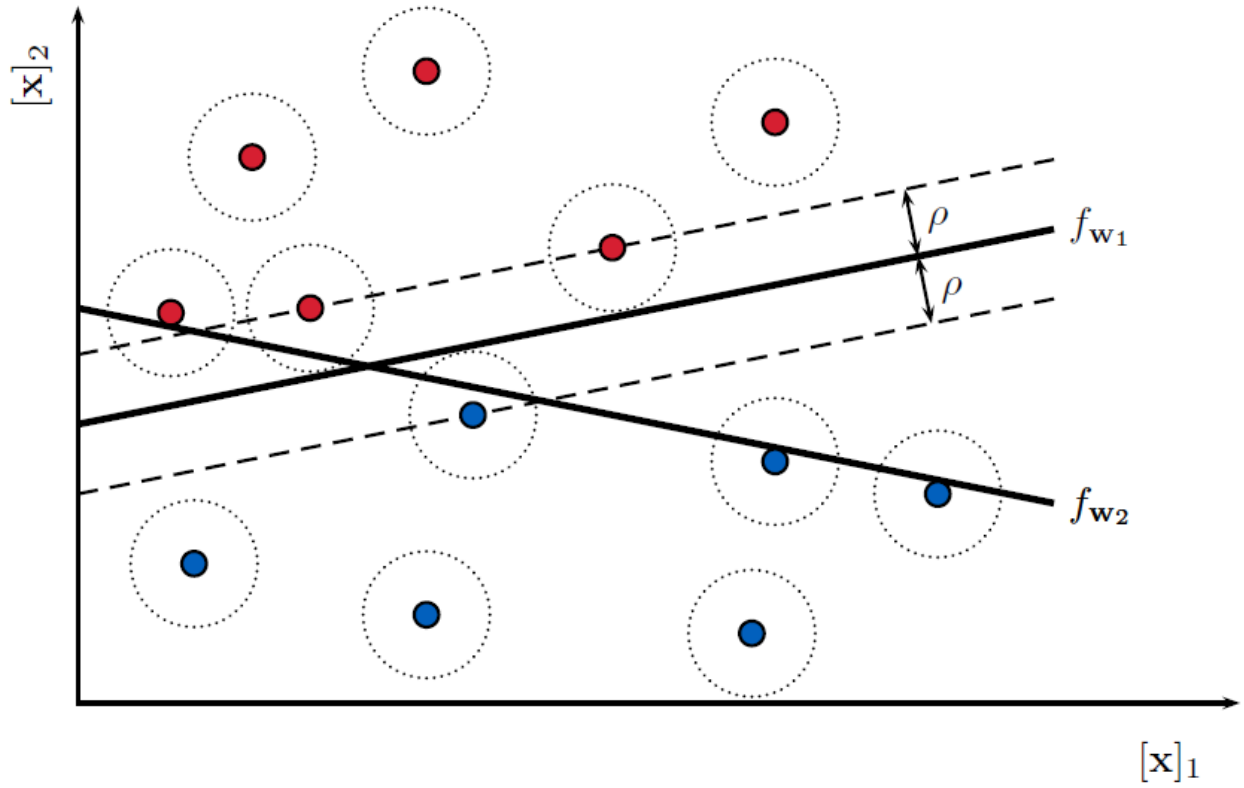
- Optimierungskriterium ist konvex.
- Perzeptron findet immer eine Trennebene, wenn eine existiert.
  - Beweis?
- Existiert immer eine Trennebene?

Probleme

1. Wenn linear-separierbar, dann mehre Modelle: Welches ist das beste?
   - Lösung: Maximal-Margin-Ansatz
2. Daten verrauscht / falsch gelabelte Daten
   - Lösung: Slack zulassen
3. Keine Fehlerwahrscheinlichkeit der Klassifikation
   - Lösung: Logistic Regression
4. Daten nicht linear-separierbar
   - Lösung: Kernel

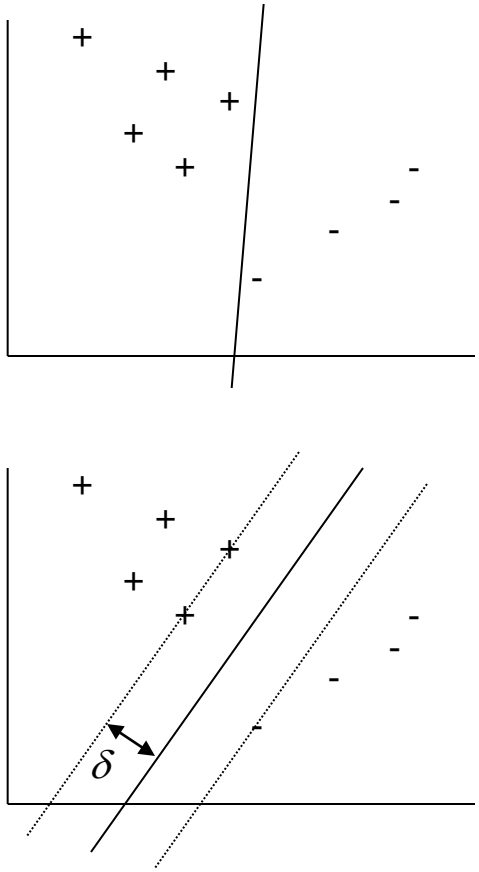
Probleme

1. Wenn linear-separierbar, dann mehre Modelle: Welches ist das beste?
   - Lösung: Maximal-Margin-Ansatz



Margin-Perzeptron

- Perzeptron-Klassifikation:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Perzeptron: für alle Beispiele muss gelten
  - $y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 0$
  - = Beispiel liegt auf der richtigen Seite der Ebene.
- Margin-Perzeptron:
  - $y_i \left(\frac{\mathbf{w}^{\mathrm{T}}}{|\mathbf{w}|} \mathbf{x}_i \right) > \delta$
  - = Beispiel mindestens δ von Trennebene entfernt.

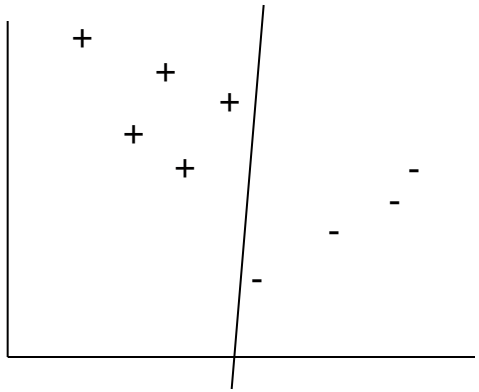


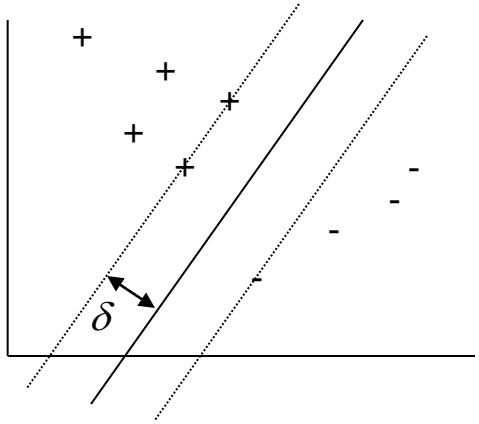
Margin-Perzeptron-Algorithmus

- Lineares Modell:
  - $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Perzeptron-Trainingsalgorithmus:
  - Solange noch Beispiele $(\mathbf{x}_i, y_i)$ mit der Hypothese inkonsistent sind, iteriere über alle Beispiele
  - Wenn $y_i\left(\frac{\mathbf{w}^{\mathrm{T}}}{|\mathbf{w}|}\mathbf{x}_i\right) \leq \delta$ dann $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$
- Margin δ muss angegeben werden

Margin-Maximierung

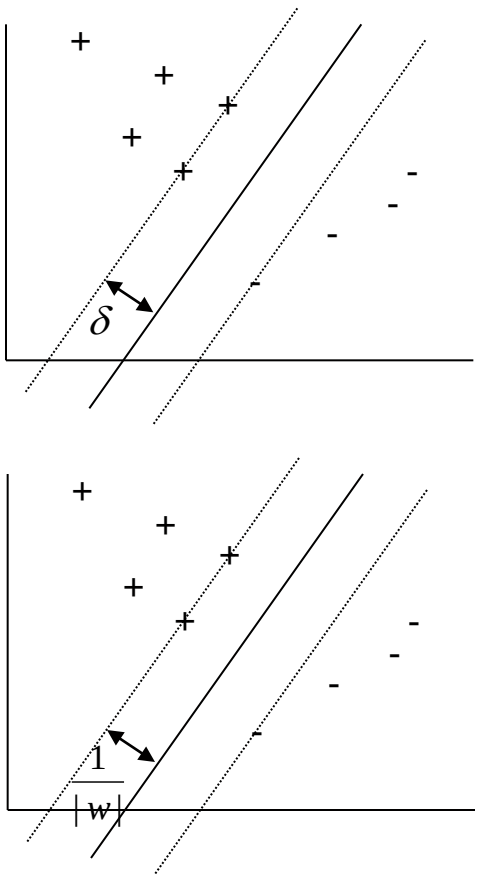
- Perzeptron: für alle Beispiele muss gelten $y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i > 0$
- Margin-Perzeptron: $y_i \left(\frac{\mathbf{w}^{\mathrm{T}}}{|\mathbf{w}|} \mathbf{x}_i \right) > \delta$
  - Finde Ebene, die alle Beispiele mindestens δ von Ebene entfernt.
  - Fester, voreingestellter Wert δ.
- Margin-Maximierung:
  - Finde Ebene, die alle Beispiele mindestens δ von Ebene entfernt.
  - Für den größtmöglichen Wert δ.





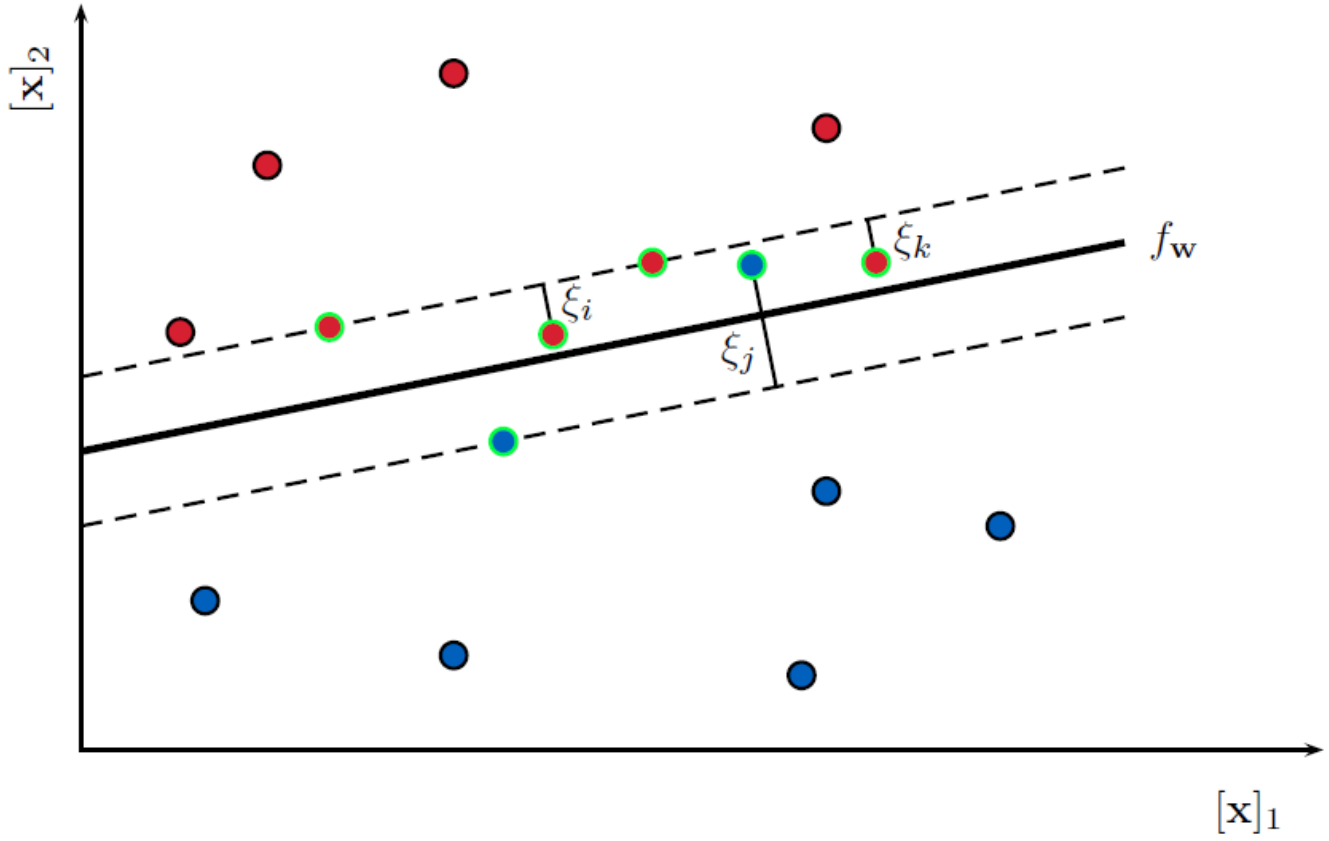
Margin-Maximierung

- Margin-Maximierung: $y_i\left(\frac{\mathbf{w}^{\mathrm{T}}}{|\mathbf{w}|}\mathbf{x}_i\right) > \delta$
  - Finde Ebene, die alle Beispiele mindestens δ von Ebene entfernt.
  - Für den größtmöglichen Wert δ.
- Maximiere δ unter der Nebenbedingungen: für alle Beispiele $(\mathbf{x}_i, y_i)$ gelte $y_i\left(\frac{\mathbf{w}^{\mathrm{T}}}{|\mathbf{w}|}\mathbf{x}_i\right) > \delta$
- = Minimiere $|\mathbf{w}|$ unter der Nebenbedingung:
  - für alle Beispiele $(\mathbf{x}_i, y_i)$: $y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i > 1$



Probleme

2. Daten verrauscht / falsch gelabelte Daten
   - Lösung: Slack zulassen

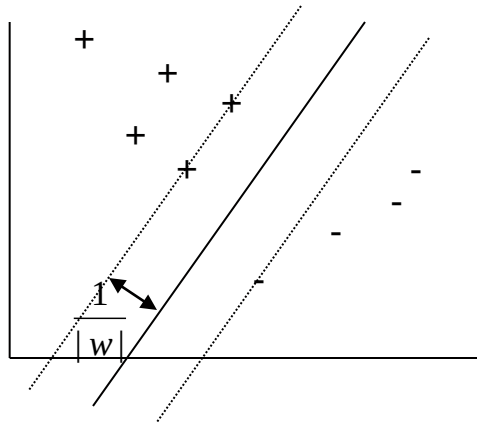


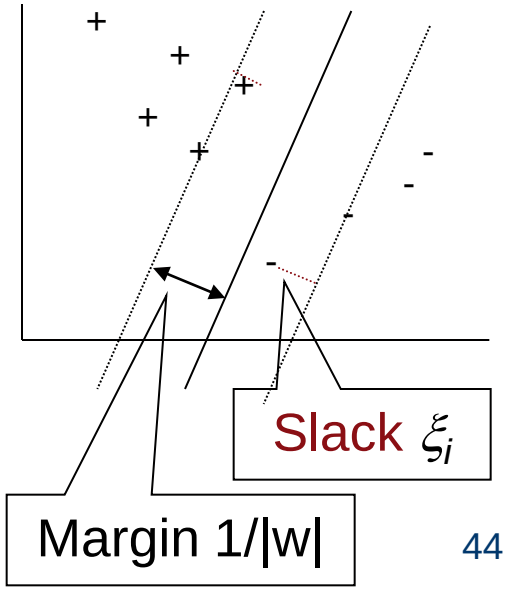
Soft-Margin-Maximierung

- Hard-Margin-Maximierung:
  - Minimiere $|\mathbf{w}|$ unter der Nebenbedingungen:
  - für alle Beispiele $(\mathbf{x}_i, y_i)$: $y_i \mathbf{w}^T \mathbf{x}_i > 1$
- Soft-Margin-Maximierung:
  - Minimiere $|\mathbf{w}| + C \sum_i \xi_i$ unter den Nebenbedingungen:
  - für alle Beispiele $(\mathbf{x}_i, y_i)$:

    $$y_i \mathbf{w}^T \mathbf{x}_i > 1 - \xi_i$$

  - Alle $\xi_i \geq 0$.
- Soft-Margin-Ebene existiert immer, Hard-Margin-Ebene nicht!



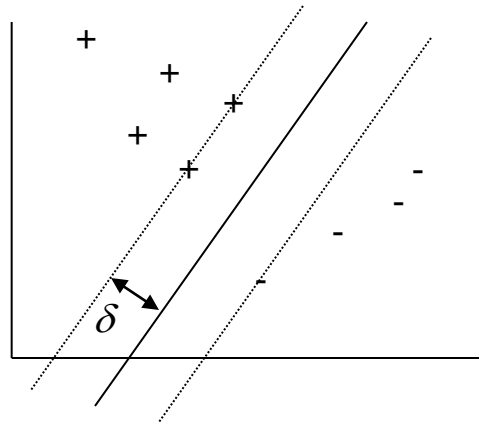


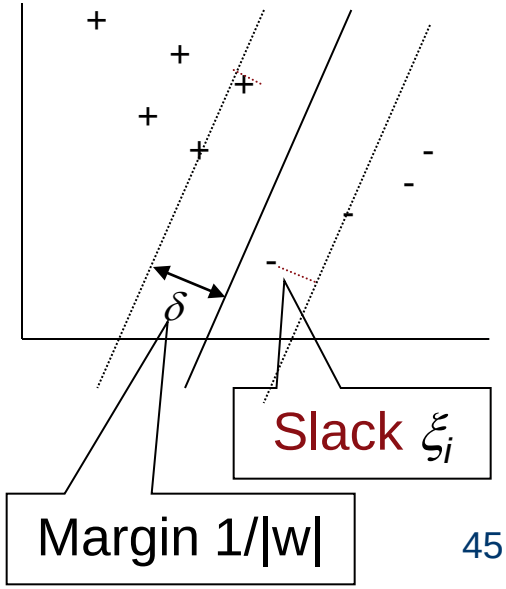
Soft-Margin-Maximierung

- Soft-Margin-Maximierung:
  - Minimiere $|\mathbf{w}|+C\sum_i \xi_i$ unter den Nebenbedingungen:
  - für alle Beispiele $(\mathbf{x}_i, y_i)$:

    $$y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i > 1-\xi_i$$

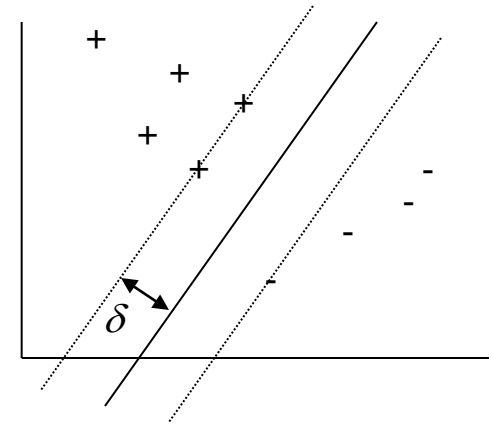
  - Alle $\xi_i \geq 0$.
- Einsetzen von ξ_i in Optimierungskriterium ergibt
  - Minimiere: $|\mathbf{w}|+C\sum_i \max\{0,1-y_i \mathbf{w}^{\mathrm{T}} \mathbf{x}_i\}$

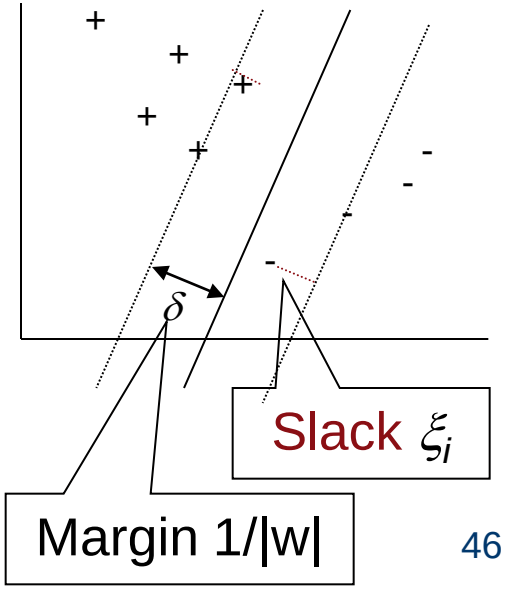




Soft-Margin-Maximierung

- Soft-Margin-Maximierung:
  - Minimiere: $|\mathbf{w}| + C\sum_i \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\}$
- Minimierung mit Gradientenverfahren.
- Annäherung durch differenzierbare Variante (Huber- statt Hinge-Loss), dann Newton-Verfahren.
- Kriterium ist konvex, es gibt genau ein Minimum.
- Primale Support Vector Machine.
- $O(n^2)$, unter 100.000 Beispiele, unter 100.000 Attribute.



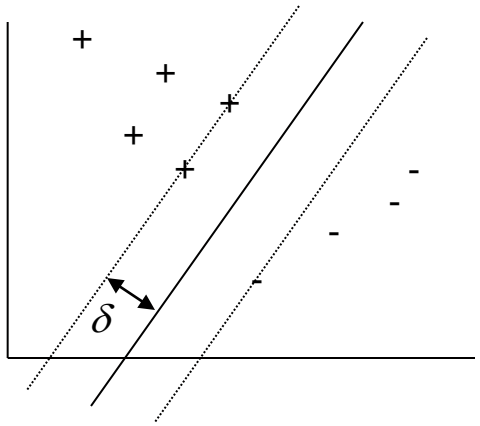


Soft-Margin-Maximierung

- Soft-Margin-Maximierung:
  - Minimiere: $E_H(\mathbf{w}) = |\mathbf{w}| + C\sum_i \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\}$
- Minimierung z.B. mit Gradientenverfahren.
- Wiederhole:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial E_H(\mathbf{w})}{\partial \mathbf{w}}$$

Enthält Summe über alle Beispiele

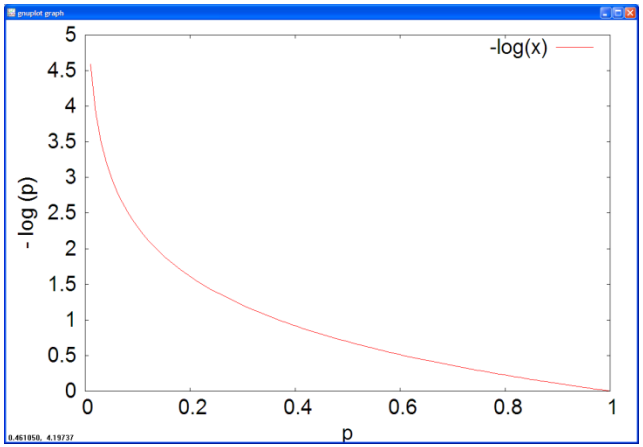


- Originalreferenz:

V.N.Vapnik: „*The nature of statistical learning theory*”. 1996
C.Cortes & V.N.Vapnik: “*Support-vector networks*”.1995

Interpretation als MAP-Hypothese

- Primales SVM-Optimierungskriterium:
  - Minimiere $|\mathbf{w}| + C\sum_i \max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\}$
  - Maximiere $\underbrace{-|\mathbf{w}|}_{\text{Log-Prior } \log P(\theta)} \underbrace{- C\sum_i \max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\}}_{\text{Log-Likelihood } \log P(L|\theta)}$
- Max $P(\theta \mid L) = P(L \mid \theta)P(\theta)$ $\equiv$ Max $\log P(\theta \mid L) = \log P(L \mid \theta) + \log P(\theta)$
- Margin = Prior
  - $P(\theta) = e^{-|w|}$
- Slack = - Log-Likelihood
  - $P(L \mid \theta) = \prod_{i=1}^{m} e^{-\xi_i}$



Interpretation als MAP-Hypothese

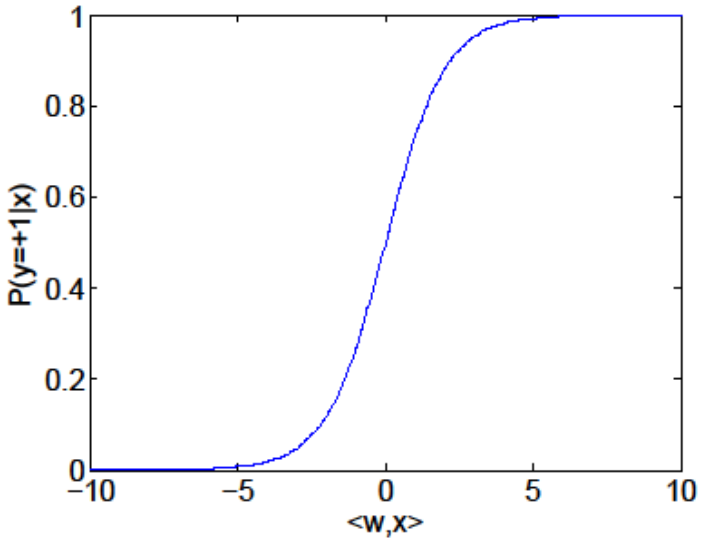
- SVM-Klassifikator ist MAP-Hypothese, wenn
  - Margin = Prior und
  - Slack = -Log-Likelihood.
- Gibt es ein SVM-Analogon zur Bayes-Hypothese?
  - Bayes Point Machine.

Bayes Point Machine

- Hard-Margin,
  - Likelihood = 0 bei Verletzung der Ebene, 1 sonst.
- Gleichverteilter Prior,
  - gleiches $P(\mathbf{w})$ für alle $\mathbf{w}$.
- Bayesian Model Averaging, Mitteln über alle $\mathbf{w}$.
  - Posterior = 0, wenn ein Beispiel Ebene verletzt,
  - konstanter Posterior für alle anderen $\mathbf{w}$.
- Bayes Point, Mittelpunkt des Version Space.
  - Liefert Bayes-optimale Entscheidung.
  - Approximative Bestimmung, Mittelpunkt des größten Kreises, der in Version Space passt (SVM).

Probleme

3. Keine Fehlerwahrscheinlichkeit der Klassifikation
   - Lösung: Logistic Regression



Logistische Regression

- SVM: großer Entscheidungsfunktionswert ~ hohe Sicherheit der Vorhersage.
- Aber: beim Lernen nicht auf korrekte Kalibrierung der Klassenwahrscheinlichkeiten optimiert.
- $f(\mathbf{x})=18.3 \rightarrow$ Risiko eines Fehlers?

- Logistische Regression: Vorhersage der Klassenwahrscheinlichkeit.

Logistische Regression

- Bayes‘ Regel:
  - $$P(y=+1\,|\,\mathbf{x}) = \frac{p(\mathbf{x}\,|\,y=+1)P(y=+1)}{p(\mathbf{x}\,|\,y=+1)P(y=+1)+p(\mathbf{x}\,|\,y=-1)P(y=-1)}$$
    $$= \frac{1}{1+\frac{p(\mathbf{x}\,|\,y=-1)P(y=-1)}{p(\mathbf{x}\,|\,y=+1)P(y=+1)}} = \frac{1}{1+\exp(-a)} = \sigma(a)$$
- Log-odd ratio:
  - $$a = \ln \frac{p(\mathbf{x}\,|\,y=+1)P(y=+1)}{p(\mathbf{x}\,|\,y=-1)P(y=-1)}$$

Logistische Regression

- Likelihood jeder Klasse normalverteilt, gemeinsame Kovarianzmatrix für beide Klassen.
  - $$p(\mathbf{x} \mid y) = \frac{1}{(2\pi)^{d/2} \mid \Sigma \mid^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \mu)^{\mathrm{T}} \Sigma^{-1} (\mathbf{x} - \mu) \right]$$

- Log-odds ratio:

$$a = \ln \frac{\frac{1}{(2\pi)^{d/2} \mid \Sigma \mid^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \mu_{+1})^{\mathrm{T}} \Sigma^{-1} (\mathbf{x} - \mu_{+1}) \right] P(y = +1)}{\frac{1}{(2\pi)^{d/2} \mid \Sigma \mid^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \mu_{-1})^{\mathrm{T}} \Sigma^{-1} (\mathbf{x} - \mu_{-1}) \right] P(y = -1)}$$

$$= \left[-\frac{1}{2} (\mathbf{x} - \mu_{+1})^{\mathrm{T}} \Sigma^{-1} (\mathbf{x} - \mu_{+1}) \right] - \left[-\frac{1}{2} (\mathbf{x} - \mu_{-1})^{\mathrm{T}} \Sigma^{-1} (\mathbf{x} - \mu_{-1}) \right] + \ln \frac{P(y = +1)}{P(y = -1)}$$

$$= \underbrace{\left[\Sigma^{-1} (\mu_{+1} - \mu_{-1}) \right]}_{\mathbf{w}} \mathbf{x} + \underbrace{\left[-\frac{1}{2} (\mu_{+1}^{\mathrm{T}} \Sigma^{-1} \mu_{+1}) + \frac{1}{2} (\mu_{-1}^{\mathrm{T}} \Sigma^{-1} \mu_{-1}) + \ln \frac{P(y = +1)}{P(y = -1)} \right]}_{w_0}$$

Logistische Regression

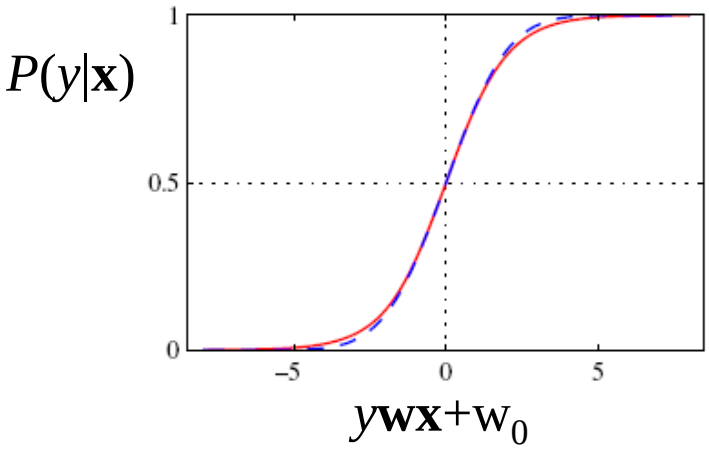
- Likelihood jeder Klasse normalverteilt, gemeinsame Kovarianzmatrix für beide Klassen.
  - $$p(\mathbf{x} \mid y) = \frac{1}{(2\pi)^{d/2} \mid \Sigma \mid^{1/2}} \exp\left[-\frac{1}{2} (\mathbf{x} - \mu)^{\mathrm{T}} \Sigma^{-1} (\mathbf{x} - \mu) \right]$$
- Log-odds ratio:

$$a = \underbrace{\left[\Sigma^{-1} (\mu_{+1} - \mu_{-1}) \right]}_{\mathbf{w}} \mathbf{x} + \underbrace{\left[-\frac{1}{2} (\mu_{+1}^{\mathrm{T}} \Sigma^{-1} \mu_{+1}) + \frac{1}{2} (\mu_{-1}^{\mathrm{T}} \Sigma^{-1} \mu_{-1}) + \ln \frac{P(y=+1)}{P(y=-1)} \right]}_{w_0}$$

Logistische Regression

- Wenn zwei Klassen jeweils normalverteilte Likelihood mit derselben Kovarianzmatrix haben, dann nimmt $P(y|\mathbf{x})$ diese Form an:

$$P(y \mid \mathbf{x}) = \frac{1}{1+\exp(-y\mathbf{w}\mathbf{x}+w_0)} = \underbrace{\sigma(\underbrace{y\mathbf{w}\mathbf{x}+w_0}_{\text{linearer Klassifikator}})}_{\text{logistische Funktion}}$$



Logistische Regression

- Bisher: Motivation der Form des logistischen Klassifikationsmodells.
- Falls Klassenverteilungen bekannt wären, könnten wir $\mathbf{w}$ und w_0 aus μ_{+1}, μ_{-1} und Σ herleiten.
- Sind aber nicht bekannt. Verteilungsannahme muss auch nicht stimmen.
- Jetzt: Wir finden wir tatsächlich Parameter $\mathbf{w}$ und w_0?

Logistische Regression

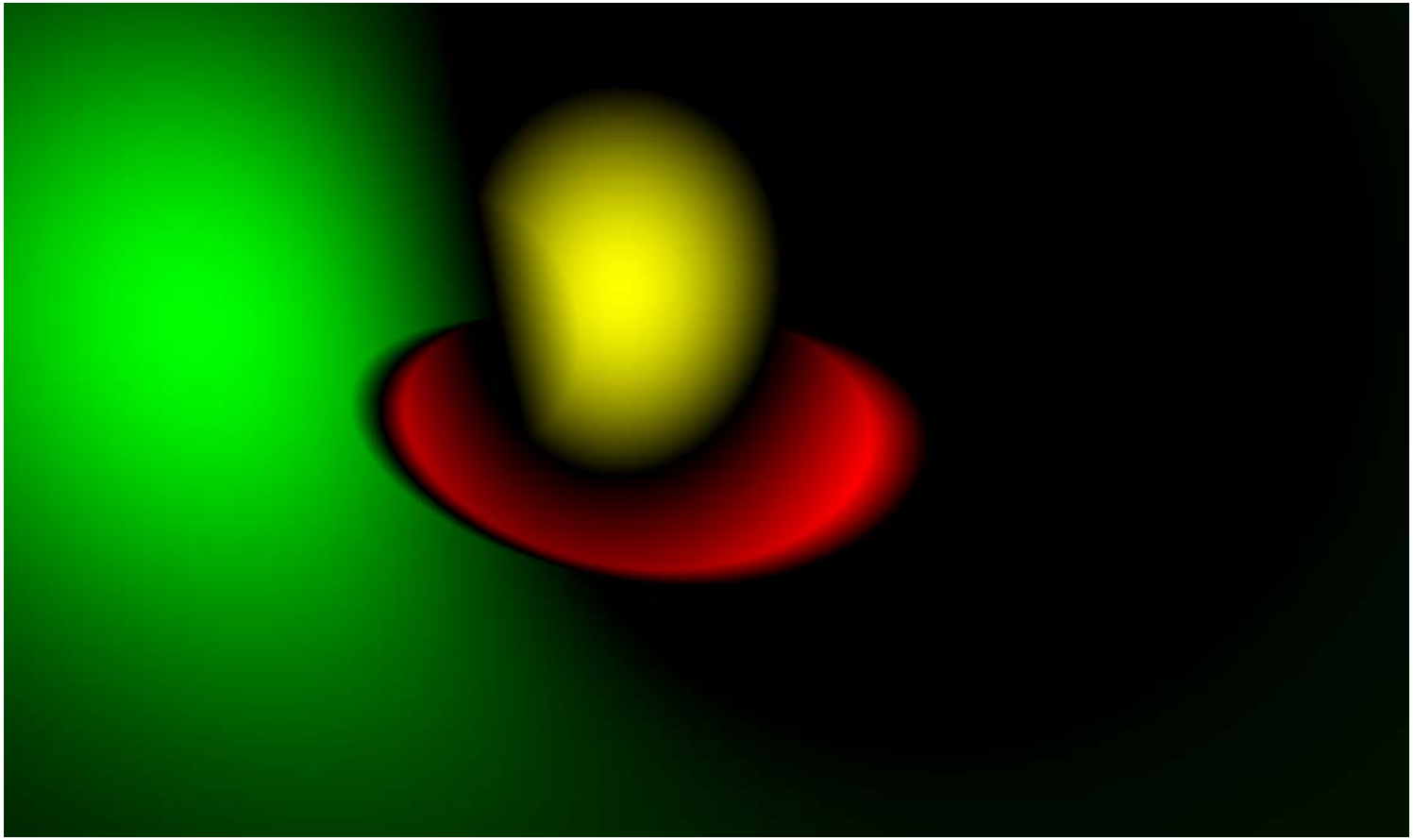
- Prior über Parameter:
  - Normalverteilung, $\mathbf{w} \sim \mathrm{N}[0,\sigma]$.
- Posterior:
  - $$P(\mathbf{w} \mid L) \propto \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \mathbf{w}) p(\mathbf{w})$$
    $$= \prod_{i=1}^{N} \sigma(\mathbf{w}^{\mathrm{T}} \mathbf{x}_i)^{[[y_i=+1]]} (1-\sigma(\mathbf{w}^{\mathrm{T}} \mathbf{x}_i))^{[[y_i=-1]]} p(\mathbf{w})$$
- Verlustfunktion+Regularisierer:
  - $$E(\mathbf{w}, L) = -\log p(\mathbf{w} \mid L)$$
    $$= -\sum_{i=1}^{N} [[y_i = +1]] \log \sigma(\mathbf{w}^{\mathrm{T}} \mathbf{x}_i) + [[y_i = -1]] \log(1-\sigma(\mathbf{w}^{\mathrm{T}} \mathbf{x}_i)) - \frac{\mathbf{w}^{\mathrm{T}} \mathbf{w}}{\sigma'^2}$$

Logistische Regression

- Verlustfunktion ist konvex und differenzierbar.
- Gradientenabstieg führt zum Minimum.

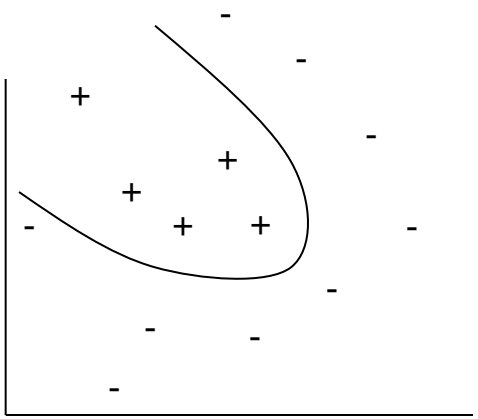
Probleme

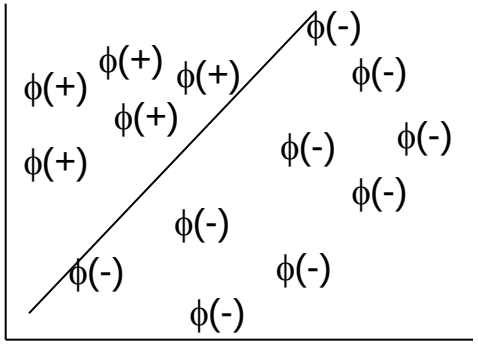
4. Daten nicht linear-separierbar
   - Lösung: Kernel



Kernel

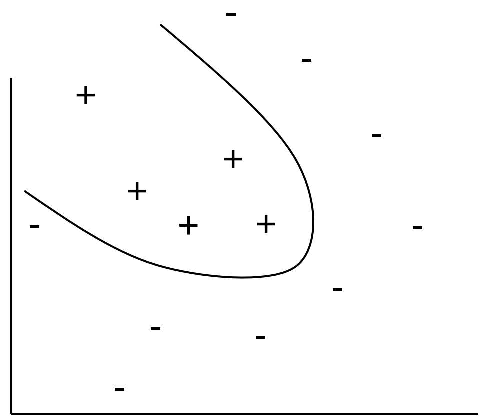
- Lineare Klassifikatoren:
  - Oft adäquat, aber nicht immer.
- Idee: Beispiele in anderen Raum abbilden, in dem sie linear klassifizierbar sind.
- Abbildung
  - $\mathbf{x} \mapsto \varphi(\mathbf{x})$
- Zugehöriger Kernel
  - $k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)$
- Kernel = Inneres Produkt = Ähnlichkeit der Beispiele.

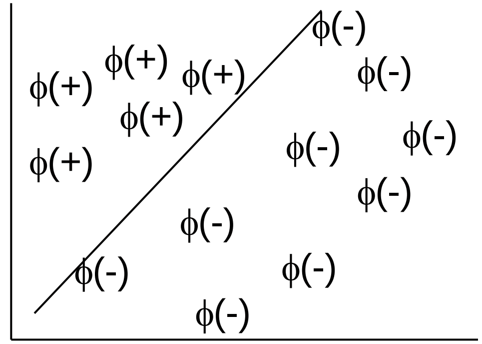




Kernel

- Kernel
  - $k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)$
- Gram-Matrix oder Kernel-Matrix
  - $K = (k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$
- Matrix der inneren Produkte = Ähnlichkeiten zwischen Beispielen, N x N-Matrix.





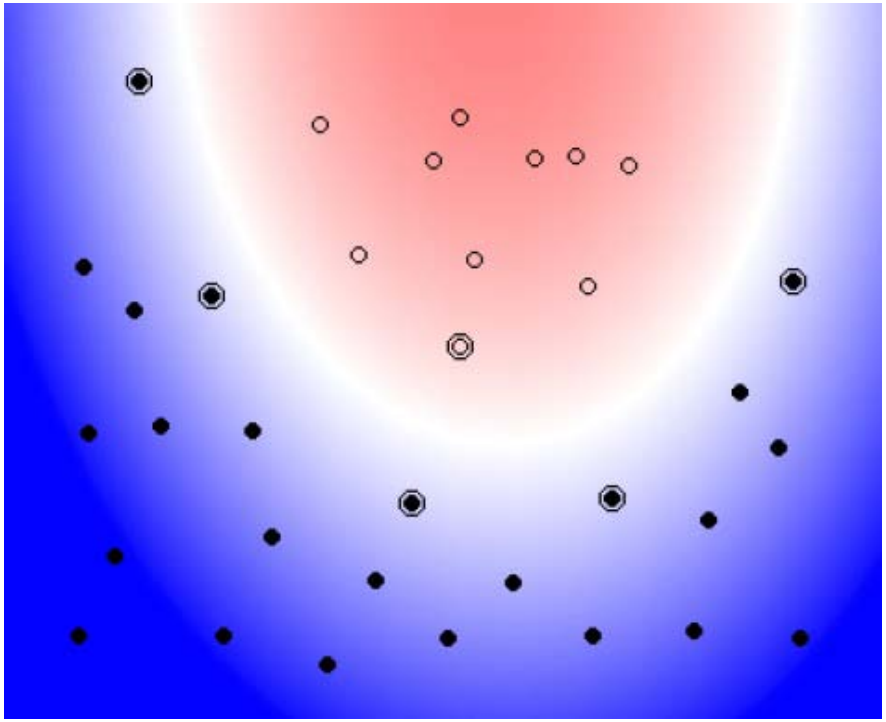
Kernel-Funktionen

- Polynomielle Kernels $k_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + 1)^d$
- Radiale Basisfunktion $k_{\text{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma(\mathbf{x}_i - \mathbf{x}_j)^2}$
- Sigmoide Kernels, String-Kernels (z.B. zum Klassifizieren von Gensequenzen).
- Graph-Kernels zum Lernen mit strukturierten Instanzen.

- Weitere Literatur:
  B.Schölkopf, A.J.Smola: „*Learning with Kernels*”. 2002

Polynomielle Kernel

- Kernel: $k_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + 1)^d$
- Welcher Transformation φ entspricht das?
- Beispiel: 2-D Originalraum, d=2.



Polynomielle Kernel

- Kernel: $k_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_j + 1)^d$, 2D-Eingabe, d=2.
  - $k_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_j + 1)^d$

$$= \left(\begin{pmatrix} \mathbf{x}_{i1} & \mathbf{x}_{i2} \end{pmatrix} \begin{pmatrix} \mathbf{x}_{j1} \\ \mathbf{x}_{j2} \end{pmatrix} + 1 \right)^2$$

Polynomielle Kernel

- Kernel: $k_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + 1)^d$, 2D-Eingabe, d=2.
  - $k_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + 1)^d$

$$= \left(\begin{pmatrix} \mathbf{x}_{i1} & \mathbf{x}_{i2} \end{pmatrix} \begin{pmatrix} \mathbf{x}_{j1} \\ \mathbf{x}_{j2} \end{pmatrix} + 1 \right)^2 = \left(\mathbf{x}_{i1}\mathbf{x}_{j1} + \mathbf{x}_{i2}\mathbf{x}_{j2} + 1 \right)^2$$

$$= \left(\mathbf{x}_{i1}^2\mathbf{x}_{j1}^2 + \mathbf{x}_{i2}^2\mathbf{x}_{j2}^2 + 2\mathbf{x}_{i1}\mathbf{x}_{j1}\mathbf{x}_{i2}\mathbf{x}_{j2} + 2\mathbf{x}_{i1}\mathbf{x}_{j1} + 2\mathbf{x}_{i2}\mathbf{x}_{j2} + 1 \right)$$

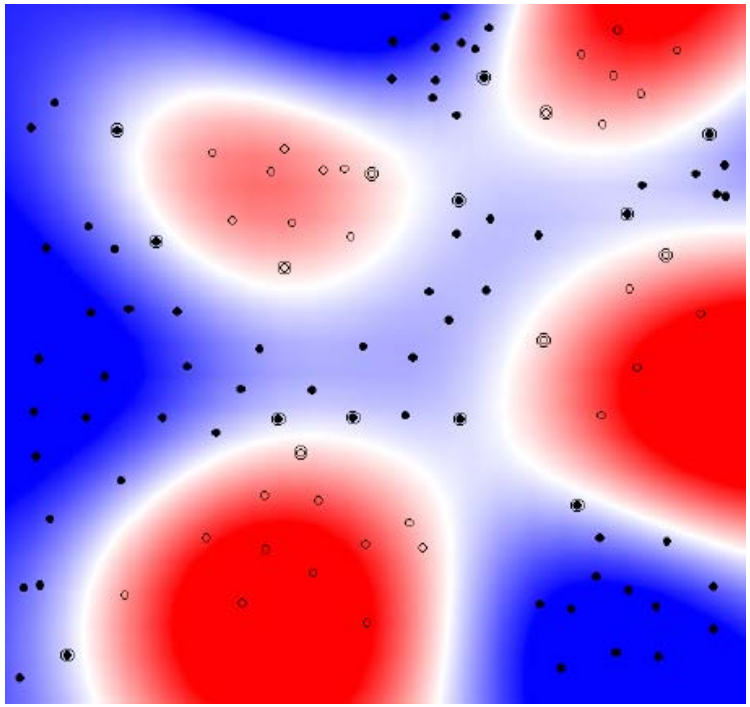
$$= \underbrace{\begin{pmatrix} \mathbf{x}_{i1}^2 & \mathbf{x}_{i2}^2 & \sqrt{2}\mathbf{x}_{i1}\mathbf{x}_{i2} & \sqrt{2}\,\mathbf{x}_{i1} & \sqrt{2}\mathbf{x}_{i2} & 1 \end{pmatrix}}_{\phi(\mathbf{x}_i)^{\mathrm{T}}} \underbrace{\begin{pmatrix} \mathbf{x}_{j1}^2 \\ \mathbf{x}_{j2}^2 \\ \sqrt{2}\mathbf{x}_{j1}\mathbf{x}_{j2} \\ \sqrt{2}\mathbf{x}_{j1} \\ \sqrt{2}\mathbf{x}_{j2} \\ 1 \end{pmatrix}}_{\phi(\mathbf{x}_j)}$$

Alle Polynome 2. Grades über Eingabeattribute

RBF-Kernel

- Kernel $k_{\mathrm{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma(\mathbf{x}_i - \mathbf{x}_j)^2\right)$

- Welcher Transformation entspricht das?



RBF-Kernel

- Kernel $k_{\text{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{2\sigma^2}\right)$
- Welcher Transformation entspricht das?
- $\varphi(\mathbf{x})$ hat keine geschlossene Form!
- $\varphi(\mathbf{x})$ hat unendlich viele Dimensionen.
  - Instanzen werden durch Ähnlichkeit zu allen anderen Instanzen repräsentiert.
  - Repräsentation hat eine Dimension für jeden Punkt.
- Macht alles nichts, Kernel ist leicht berechenbar.

Representer Theorem

- Gegeben $L = \langle (\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N) \rangle$
- Abbildung $\varphi(\mathbf{x})$ mit dazu analogem Kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)$
- Trennebenen $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}) + w_0$
- Die Trennebene $\mathbf{w}^*$, die $\|\mathbf{w}\|^2 + C \sum_i \max\{0, 1 - y_i(\mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}_i) + w_0)\}$ minimiert, lässt sich repräsentieren als
  - $f(\mathbf{x}) = \mathbf{w}^{*\mathrm{T}} \varphi(\mathbf{x}) + w_0 = \sum_{j=1}^{N} \alpha_j y_j k(\mathbf{x}, \mathbf{x}_j) + w_0$
- Primale Sicht:
  - Hypothese $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}) + w_0$ so viele Parameter $\mathbf{w}_i$ wie $\varphi(\mathbf{x})$ Dimensionen hat.
- Duale Sicht:
  - Hypothese $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_j y_j k(\mathbf{x}, \mathbf{x}_j) + w_0$ hat so viele Parameter α_j wie Beispiele existieren.

Representer Theorem

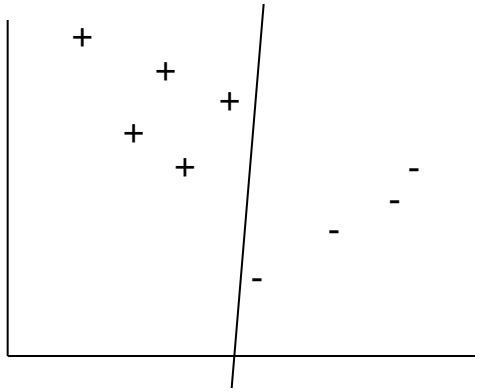
- Primale Sicht:
  - Hypothese $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}) + w_0$ so viele Parameter $\mathbf{w}_i$ wie $\varphi(\mathbf{x})$ Dimensionen hat.
  - Gut für viele Beispiele, wenige Attribute.
- Duale Sicht:
  - Hypothese $f(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j y_j k(\mathbf{x}, \mathbf{x}_j) + w_0$ hat so viele Parameter α_i wie Beispiele existieren.
  - Gut für wenige Beispiele, viele Dimensionen.
  - Repräsentation$\varphi(\mathbf{x})$ kann unendlich viele Dimensionen haben.
  - Kein Problem, solange $k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)$ effizient berechnet werden kann.

Duale Repräsentation

- Duale Sicht auf Hypothese:
  - Hypothese $f(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j y_j k(\mathbf{x}, \mathbf{x}_j) + w_0$ hat so viele Parameter α_i wie Beispiele existieren.
  - Kein Problem, solange $k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)$ effizient berechnet werden kann.
- Spezialfall linearer Kernel:
  - $\varphi(\mathbf{x}) = \mathbf{x}; \quad k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$
- Parametervektor ist Mischung aus Beispielen:
  - $$\begin{aligned} f(\mathbf{x}) &= \sum_{j=1}^{N} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}) + w_0 \\ &= \sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j^{\mathrm{T}} \mathbf{x} + w_0 \\ &= \underbrace{\left(\sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j \right)}_{\mathbf{w}}^{\mathrm{T}} \mathbf{x} + w_0 \end{aligned}$$

Kernel-Perzeptron

- Perzeptron-Klassifikation:
  - $f(\mathbf{x}_i) = \sum_{j=1}^{m} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i) + w_0$
- Perzeptron: für alle Beispiele muss gelten
  - $y_i \left(\sum_{j=1}^{m} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i) + w_0 \right) > 0$
  - = Beispiel liegt auf der richtigen Seite der Ebene.



Kernel-Perzeptron-Algorithmus

- Perzeptron-Klassifikation
  - $f(\mathbf{x}_i) = \sum_{j=1}^{m} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i) + w_0$
- Perzeptron-Trainingsalgorithmus:
- Solange noch Beispiele $(\mathbf{x}_i, y_i)$ mit der Hypothese inkonsistent sind, iteriere über alle Beispiele
  - Wenn $y_i \left(\sum_{j=1}^{m} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i) + w_0 \right) \leq 0$ dann $\alpha_i \leftarrow \alpha_i + 1$

    $w_0 \leftarrow w_0 + y_i$
- Beispiele tauchen nur in Kernel $k(\mathbf{x}_j, \mathbf{x}_i)$ auf.
- $\phi(x)$ braucht nicht berechnet zu werden, nur Kernel-Matrix.

Kernel-Support Vector Machine

- Primal:
  - Minimiere $\min |\mathbf{w}| \rightarrow \min |\mathbf{w}|^2 \rightarrow \min \frac{1}{2} |\mathbf{w}|^2$
  - Mit Nebenbedingungen $y_i(\mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}_i) + w_0) \geq 1$
- Lagrange-Multiplikatoren für Nebenbedingungen.
  - $L(\mathbf{w}, w_0, \alpha) = \frac{1}{2} |\mathbf{w}|^2 - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}_i) + w_0) - 1]$
- Minimum: Ableitungen von L nach $\mathbf{w}$, w_0 null setzen.

Kernel-Support Vector Machine

- Primal:
  - Minimiere $\min |\mathbf{w}| \rightarrow \min |\mathbf{w}|^2 \rightarrow \min \frac{1}{2} |\mathbf{w}|^2$
  - Mit Nebenbedingungen $y_i(\mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}_i) + w_0) \geq 1$
- Lagrange-Multiplikatoren für Nebenbedingungen.
  - $L(\mathbf{w}, w_0, \alpha) = \frac{1}{2} |\mathbf{w}|^2 - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}_i) + w_0) - 1]$
- Minimum: Ableitungen von L nach $\mathbf{w}$, w_0 null setzen.
  - $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \varphi(\mathbf{x}_i)$
  - $-\sum_{i=1}^{N} \alpha_i y_i = 0$

Relation zwischen primalen und dualen Parametern, Representer Theorem.

Kernel-Support Vector Machine

- Lagrange-Ausdruck
  - $$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2} |\, \mathbf{w} \,|^2 - \sum_{i=1}^{N} \alpha_i [y_i (\mathbf{w}^{\mathrm{T}} \varphi(\mathbf{x}) + w_0) - 1]$$
- Nullstelle der Ableitung eingesetzt:
  - $$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2} \left(\sum_{i=1}^{N} \alpha_i y_i \varphi(\mathbf{x}_i) \right)^{\mathrm{T}} \left(\sum_{i=1}^{N} \alpha_i y_i \varphi(\mathbf{x}_i) \right)$$

$$- \sum_{j=1}^{N} \alpha_j \left(y_j \left(\sum_{i=1}^{N} \alpha_i y_i \varphi(\mathbf{x}_i) \right)^{\mathrm{T}} \varphi(\mathbf{x}_j) \right) + \sum_{j=1}^{N} \alpha_j - w_0 \sum_{i=1}^{N} \alpha_i y_i$$

$$= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)$$

$$= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

Kernel-Support Vector Machine

- Optimierungskriterium der Kernel-SVM:
  - $$L(\mathbf{w}, w_0, \alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$
- Optimierung über Parameter α.
- Lösung mit QP-Solver.
- O(N^2).

Kernel-SVM

- Primales und duales Optimierungsproblem haben dieselbe Lösung.
  - $\mathbf{w} = \sum_{\mathbf{x}_i \in SV} \alpha_i y_i \varphi(\mathbf{x}_i)$
- Klassifikation mit dualer SVM:
  - $\mathbf{x} \mapsto sign\left(\sum_{\mathbf{x}_i \in SV} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + w_0\right)$
- Primale SVM:
  - Lösung ist Vektor **w** im Raum der Attribute.
- Duale SVM:
  - Gleiche Lösung repräsentiert als Gewichte α_i der Beispiele.

Andere Problemstellungen

- Bisher: binäre Klassifikation.
- Jetzt:
  - Multiklassen-Klassifikation.
  - Taxonomien, Klassenbeschreibungen.
  - Sequenz-Lernen, strukturierte Ausgaben.
  - Ranking, Ordinale Regression.

Multiklassen-Klassifikation

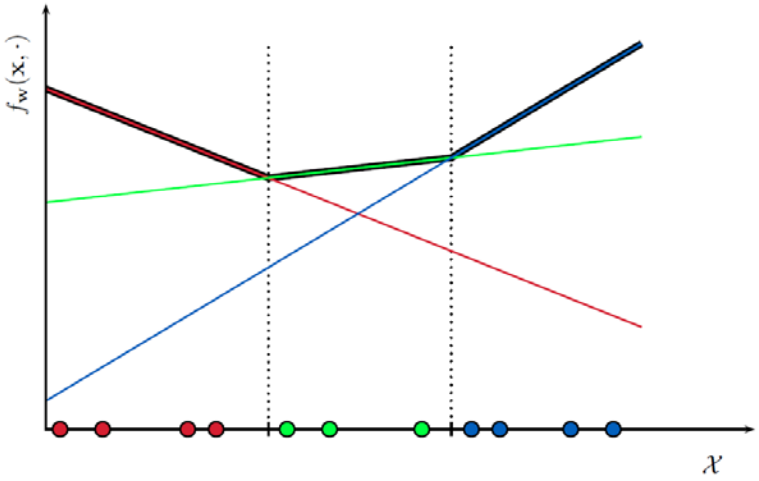
- Binäre Klassifikation:
  - $\mathbf{x} \mapsto \{-1,+1\}$
  - Lineare Klassifikation: $\mathbf{x} \mapsto sign(\mathbf{w}\mathbf{x} + w_0)$
- Multiklassen-Klassifikation:
  - $\mathbf{x} \mapsto y$
  - Endliche Menge von Klassen-Labels, $y \in Y$
- Ansatz:
  - Statt $\mathbf{x} \mapsto sign(\mathbf{w}\mathbf{x} + w_0)$ jetzt
  - $\mathbf{x} \mapsto \arg\max_y f(\mathbf{x}, y)$

Multiklassen SVM

- Klassifikation bei mehr als zwei Klassen:
  - $y^* = \arg\max_y f(\mathbf{x}, y) = \arg\max_y \left\langle \mathbf{w}_y, \mathbf{x} \right\rangle$
- Gewichtsvektors für jede mögliche Klasse

- Optimierungsproblem:
  - $\min \sum_{i=1}^{k} \|\mathbf{w}_i\|^2 + C \sum_{i=1}^{n} \xi_i$
  - Mit Nebenbedingungen

$$\forall y \neq y_i : f(\mathbf{x}_i, y_i) \geq f(\mathbf{x}_i, y) + 1 - \xi_i$$



- Originalreferenz:

J.Weston, C.Watkins: „ *Support vector machines for multi-class pattern recognition*”. 1999

Feature-Mapping

- Verschiedene Gewichtsvektoren kann auch als unterschiedliche Abschnitte eines Gewichtsvektors gesehen werden

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_k \end{pmatrix}$$

- Gemeinsame Repräsentation von Ein- und Ausgabe:

$$\Phi(\mathbf{x}, y = 2) = \begin{pmatrix} \mathbf{0} \\ \varphi(\mathbf{x}) \\ \vdots \\ \mathbf{0} \end{pmatrix}$$

Feature-Mapping

- Klassifikation bei mehr als zwei Klassen:
  - $y^* = \arg\max_y f(\mathbf{x}, y)$
- Multiklassen-Kernel:
  - $f(\mathbf{x}, y) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, y)$
  - $\Lambda(y) = \begin{pmatrix} [[y = y_1]] \\ \dots \\ [[y = y_k]] \end{pmatrix}$
  - $\Phi(\mathbf{x}, y) = \varphi(\mathbf{x}) \otimes \Lambda(y) = \begin{pmatrix} \varphi(\mathbf{x})[[y = y_1]] \\ \dots \\ \varphi(\mathbf{x})[[y = y_k]] \end{pmatrix}$

Multiklassen SVM

- Klassifikation bei mehr als zwei Klassen:
  - $y^* = \arg\max_y f(\mathbf{x}, y)$
- Multiklassen-Kernel:
  - $f(\mathbf{x}, y) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, y)$
  - $\Phi(\mathbf{x}, y) = \varphi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \otimes \begin{pmatrix} [[y = y_1]] \\ [[y = y_2]] \\ [[y = y_3]] \end{pmatrix} =$
  - $k(\mathbf{x}_i, y_i, \mathbf{x}_j, y_j) = \Phi(\mathbf{x}_i, y_i)^{\mathrm{T}} \Phi(\mathbf{x}_j, y_j)$

    $=$

Multiklassen SVM

- Klassifikation bei mehr als zwei Klassen:
  - $y^* = \arg\max_y f(\mathbf{x}, y)$
- Multiklassen-Kernel:
  - $f(\mathbf{x}, y) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, y)$
  - $\Phi(\mathbf{x}, y) = \varphi(\mathbf{x}) \otimes \Lambda(y) = \begin{pmatrix} \varphi(\mathbf{x})[[y = y_1]] \\ \dots \\ \varphi(\mathbf{x})[[y = y_k]] \end{pmatrix}$
  - $k(\mathbf{x}_i, y_i, \mathbf{x}_j, y_j) = \Phi(\mathbf{x}_i, y_i)^{\mathrm{T}} \Phi(\mathbf{x}_j, y_j)$
    $= \underbrace{[[y_i = y_j]]}_{\substack{1, \text{wenn } y_i = y_j \\ 0, \text{sonst}}} k(\mathbf{x}_i, \mathbf{x}_j)$

Klassifikation mit Informationen über Klassen

- Klassen weisen Merkmale auf:
  - $y^* = \arg\max_y \; f(\mathbf{x}, y)$
  - $f(\mathbf{x}, y) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, y), \; \Phi(\mathbf{x}, y) = \varphi(\mathbf{x}) \otimes \Lambda(y)$
  - $$k(\mathbf{x}_i, y_i, \mathbf{x}_j, y_j) = \Phi(\mathbf{x}_i, y_i)^{\mathrm{T}} \Phi(\mathbf{x}_j, y_j)$$
    $$= \left(\varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)\right)\left(\Lambda(y_i)^{\mathrm{T}} \Lambda(y_j)\right)$$
    $$= k(\mathbf{x}_i, \mathbf{x}_j) \quad \underbrace{\left(\Lambda(y_i)^{\mathrm{T}} \Lambda(y_j)\right)}_{\text{Korrespondenz zwischen Klassen}}$$

Klassifikation mit Informationen über Klassen

- Klassen weisen Merkmale auf:
  - $y^* = \arg\max_y f(\mathbf{x}, y)$
  - $f(\mathbf{x}, y) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, y),\ \Phi(\mathbf{x}, y) = \varphi(\mathbf{x}) \otimes \Lambda(y)$
  - $$k(\mathbf{x}_i, y_i, \mathbf{x}_j, y_j) = \Phi(\mathbf{x}_i, y_i)^{\mathrm{T}} \Phi(\mathbf{x}_j, y_j)$$
    $$= \left(\varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)\right)\left(\Lambda(y_i)^{\mathrm{T}} \Lambda(y_j)\right)$$
    $$= k(\mathbf{x}_i, \mathbf{x}_j) \quad \underbrace{\left(\Lambda(y_i)^{\mathrm{T}} \Lambda(y_j)\right)}_{\text{Korrespondenz zwischen Klassen}}$$
- Korrespondenz der Klassen $\Lambda(y_i)\Lambda(y_j)$:
  - Z.B. inneres Produkt der Klassenbeschreibungen.
  - $\Lambda(y_i)$: TF-Vektor über alle Wörter.

Multiklassen SVM

- Klassifikation bei mehr als zwei Klassen:
  - $y^* = \arg\max_y \; f(\mathbf{x}, y)$
  - *f* bekommt jetzt zwei Parameter.
- Gemeinsame Merkmale von Ein- und Ausgabe:
  - $f(\mathbf{x}, y) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, y)$
- Gleicher Ansatz für Multiklassen, Sequenz- und Struktur-Lernen, Ranking.

Multiklassen SVM

- $\mathbf{x}$ kodiert z.B. ein Dokument
- $y = 2$ kodiert Klasse

$$\Phi(\mathbf{x}, y) = \begin{pmatrix} [\![y=1]\!]\mathbf{x} \\ [\![y=2]\!]\mathbf{x} \\ [\![y=3]\!]\mathbf{x} \\ [\![y=4]\!]\mathbf{x} \\ [\![y=5]\!]\mathbf{x} \\ [\![y=6]\!]\mathbf{x} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \\ \mathbf{x} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{x} \\ \vdots \end{pmatrix} \begin{matrix} (1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \end{matrix}$$

Multiklassen SVM
Beispiel

- $\mathbf{x}$ kodiert z.B. ein Dokument
- $y = 2$ kodiert Klasse

$$\Phi(\mathbf{x}, y) = \begin{pmatrix} [\![y=1]\!]\mathbf{x} \\ [\![y=2]\!]\mathbf{x} \\ [\![y=3]\!]\mathbf{x} \\ [\![y=4]\!]\mathbf{x} \\ [\![y=5]\!]\mathbf{x} \\ [\![y=6]\!]\mathbf{x} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{x} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \end{pmatrix} \begin{matrix} (1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \end{matrix}$$

Klassifikation mit Taxonomien

- Klassen in Baumstruktur:
  - $y^* = \arg\max_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$
  - $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, \mathbf{y})$
  - $\mathbf{y} = (y^1, \ldots, y^d)$
  - $\Lambda(\mathbf{y}) = \begin{pmatrix} \Lambda(y^1) \\ \ldots \\ \Lambda(y^d) \end{pmatrix}$
  - $\Phi(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x}) \otimes \Lambda(\mathbf{y}) = \varphi(\mathbf{x}) \otimes \begin{pmatrix} \Lambda(y^1) \\ \ldots \\ \Lambda(y^d) \end{pmatrix} = \varphi(\mathbf{x}) \otimes \begin{pmatrix} [[y^1 = v_1^1]] \\ \ldots \\ [[y^1 = v_n^1]] \\ \ldots \\ [[y^d = v_1^d]] \\ \ldots \\ [[y^d = v_n^d]] \end{pmatrix}$

Klassifikation mit Taxonomien

- Klassen in Baumstruktur:
  - $y^* = \arg\max_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$
  - $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, \mathbf{y})$
  - $\mathbf{y} = (y^1, \ldots, y^d)$

  - $$k(\mathbf{x}_i, \mathbf{y}_i, \mathbf{x}_j, \mathbf{y}_j) = \Phi(\mathbf{x}_i, \mathbf{y}_i)^{\mathrm{T}} \Phi(\mathbf{x}_j, \mathbf{y}_j)$$
    $$= \left(\varphi(\mathbf{x}_i)^{\mathrm{T}} \varphi(\mathbf{x}_j)\right)\left(\Lambda(\mathbf{y}_i)^{\mathrm{T}} \Lambda(\mathbf{y}_j)\right)$$
    $$= k(\mathbf{x}_i, \mathbf{x}_j) \sum_{l=1}^{d} \Lambda(\mathbf{y}_i^l)^{\mathrm{T}} \Lambda(\mathbf{y}_j^l)$$

Klassifikation mit Taxonomien: Dekoding

- Klassen in Baumstruktur:
  - $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, \mathbf{y})$
  - $y^* = \arg\max_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$

$$= \arg\max_y \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{y}_i, \mathbf{x}, \mathbf{y})$$

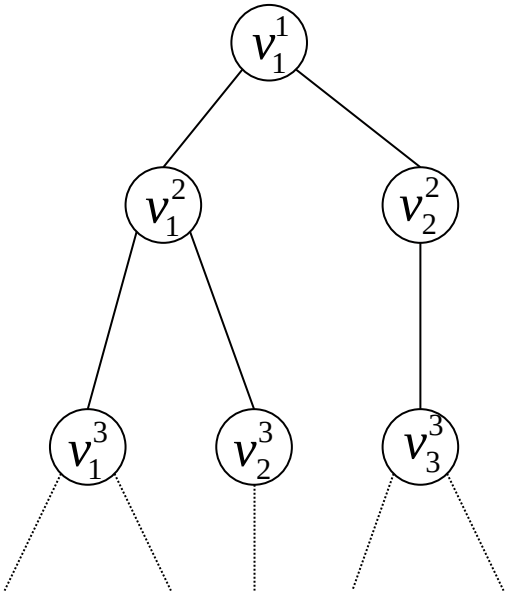
$$= \arg\max_y \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x})(\Lambda(\mathbf{y}_i)\Lambda(\mathbf{y}))$$

$$= \arg\max_y \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x})\left(\sum_{l=1}^{d} \Lambda(\mathbf{y}_i^l)\Lambda(\mathbf{y}^l)\right)$$

Klassifikation mit Taxonomien
Beispiel

- $\mathbf{x}$ kodiert z.B. ein Dokument
- $\mathbf{y} = \left(v_1^1, v_2^2, v_3^3\right)^T$ ist ein Pfad z.B. in einem Themenbaum

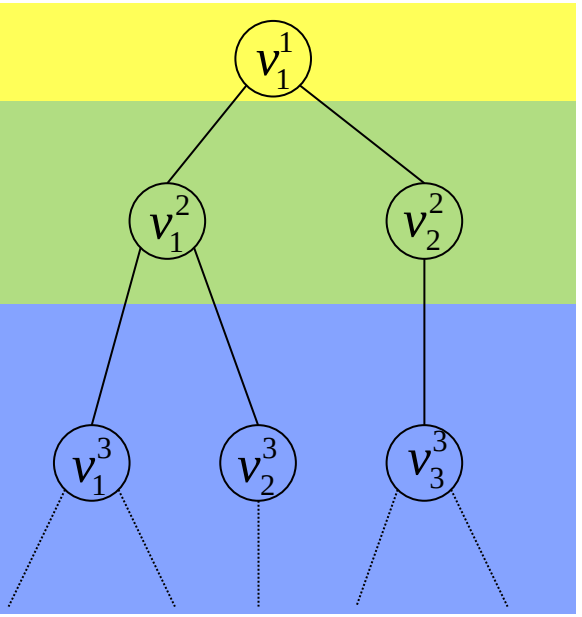
$$\Phi\left(\mathbf{x}, \mathbf{y}\right) = \begin{pmatrix} [\![y^1 = v_1^1]\!]\mathbf{x} \\ [\![y^2 = v_1^2]\!]\mathbf{x} \\ [\![y^2 = v_2^2]\!]\mathbf{x} \\ [\![y^3 = v_1^3]\!]\mathbf{x} \\ [\![y^3 = v_2^3]\!]\mathbf{x} \\ [\![y^3 = v_3^3]\!]\mathbf{x} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \\ \mathbf{x} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{x} \\ \vdots \end{pmatrix}$$



Klassifikation mit Taxonomien
Beispiel

- $\mathbf{x}$ kodiert z.B. ein Dokument
- $\mathbf{y} = \left(v_1^1, v_2^2, v_3^3\right)^T$ ist ein Pfad z.B. in einem Themenbaum

$$\Phi(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} [\![y^1 = v_1^1]\!]\mathbf{x} \\ [\![y^2 = v_1^2]\!]\mathbf{x} \\ [\![y^2 = v_2^2]\!]\mathbf{x} \\ [\![y^3 = v_1^3]\!]\mathbf{x} \\ [\![y^3 = v_2^3]\!]\mathbf{x} \\ [\![y^3 = v_3^3]\!]\mathbf{x} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \\ \mathbf{x} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{x} \\ \vdots \end{pmatrix}$$



Strukturierte Ein-/Ausgaben

- Ausgaberaum Y beinhaltet komplexe Objekte
- Darstellung als Kombination von binären Vorhersageproblemen nicht klar
- Beispiele:
  - Wortart- und Eigennamenerkennung
  - Natural Language Parsing
  - Sequence Alignment
  - …

Strukturierte Ein-/Ausgaben

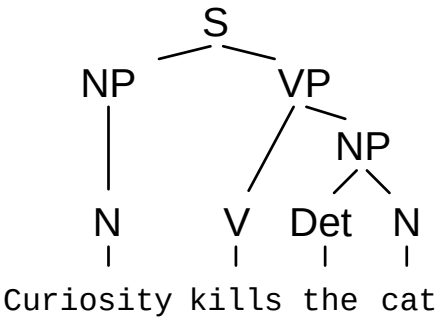
- Sequentielle Ein-/Ausgaben
  - Wortarterkennung

    x= “Curiosity kills the cat.” → **y**= <Noun, Verb, Determiner, Noun>
  - Eigennamenerkennung, Informationsextraktion:
  - **x**= “Barbie meets Ken.” → **y**= <Person, -, Person>

Strukturierte Ein-/Ausgaben

- Natural Language Parsing

x= “Curiosity kills the cat.” →



Strukturierte Ein-/Ausgaben

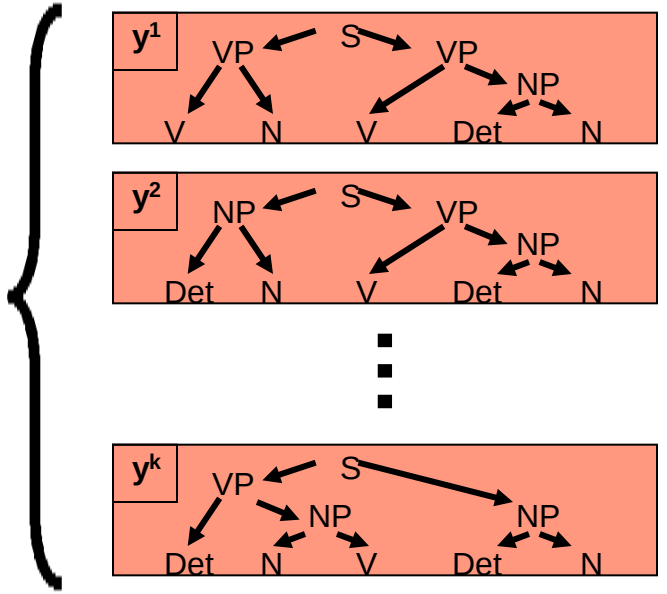
- Sequence Alignment
  - Gegeben seien zwei Sequenzen
  - Vorhersage ist ein Alignment

$$\mathbf{x} = \begin{matrix} s=(\text{ABJLHBNJYAUGAI}) \\ t=(\text{BHJKBNYGU}) \end{matrix} \rightarrow \begin{matrix} \text{AB-JLHBNJYAUGAI} \\ \text{| || || |||} \\ \text{BHJK-BN-YGU} \end{matrix}$$

Strukturierte Ein-/Ausgaben

- Ausgaberaum Y beinhaltet komplexe Objekte
- Mehrstufenverfahren propagieren Fehler
- Warum ist das kein einfaches Multiklassen-Problem?

The dog chased the cat



Strukturierte Ein-/Ausgaben

- Ausgaberaum Y beinhaltet komplexe Objekte
- Mehrstufenverfahren propagieren Fehler
- Warum ist das kein einfaches Multiklassen-Problem?
  - Exponentielle Anzahl von Parametern
  - Effektive Vorhersage
  - Effektives Lernen

Feature-Mapping: reduziert die Anzahl der Parameter für jede Klasse

Problemspezifische Kodierung

Schnittebenenalgorithmus

Lernen mit strukturierten Ausgaben

- Beispiel: POS-Tagging (Wortarterkennung)
- Satz **x**=“Curiosity kills the cat”
- Gewünscht:

$$\textbf{argmax}_{\mathbf{y}}\ \mathbf{w}^{T}\Phi(\,\mathbf{x}\,,\,\mathbf{y}\,) = \langle N,V,Det,N\rangle$$

- Explizit:

$$\mathbf{w}^{T}\Phi(\mathbf{x},\langle N,V,Det,N\rangle) \geq \mathbf{w}^{T}\Phi(\mathbf{x},\langle N,N,N,N\rangle)$$

$$\mathbf{w}^{T}\Phi(\mathbf{x},\langle N,V,Det,N\rangle) \geq \mathbf{w}^{T}\Phi(\mathbf{x},\langle N,N,N,V\rangle)$$

$$\mathbf{w}^{T}\Phi(\mathbf{x},\langle N,V,Det,N\rangle) \geq \mathbf{w}^{T}\Phi(\mathbf{x},\langle N,N,V,N\rangle)$$

$$\mathbf{w}^{T}\Phi(\mathbf{x},\langle N,V,Det,N\rangle) \geq \mathbf{w}^{T}\Phi(\mathbf{x},\langle N,V,N,N\rangle)$$

$$\vdots \qquad\qquad \vdots$$

Lernen mit strukturierten Ausgaben

- Optimierungsproblem
  - $\min \|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i$
  - Mit Nebenbedingungen

    $\forall i = 1 \ldots n \forall \bar{\mathbf{y}} \neq \mathbf{y}_i : \mathbf{w}^T \left(\Phi\left(\mathbf{x}_i, \mathbf{y}_i\right) - \Phi\left(\mathbf{x}_i, \bar{\mathbf{y}}\right) \right) \geq 1 - \xi_i$

    $\forall i = 1 \ldots n : \xi_i \geq 0$

- Iteratives Training.
  - Negative Constraints werden hinzugefügt, wenn beim Training Fehler auftritt.

- Originalreferenz:

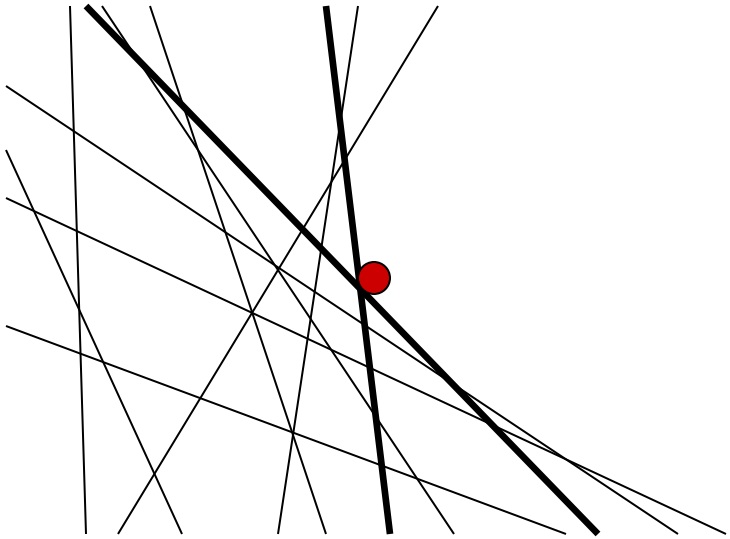
  I.Tsochantaridis, T.Hofmann, T.Joachims, Y. Altun: „*Support Vector Machine Learning for Interdependent and Structured Output Spaces*“. 2004

Lernen mit strukturierten Ausgaben

- Optimierungsproblem
  - $\min \|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i$
  - Mit Nebenbedingungen

$$\forall i = 1 \ldots n \forall \bar{\mathbf{y}} \neq \mathbf{y}_i : \mathbf{w}^T \left(\Phi\left(\mathbf{x}_i, \mathbf{y}_i\right) - \Phi\left(\mathbf{x}_i, \bar{\mathbf{y}}\right) \right) \geq 1 - \xi_i$$

$$\forall i = 1 \ldots n : \xi_i \geq 0$$



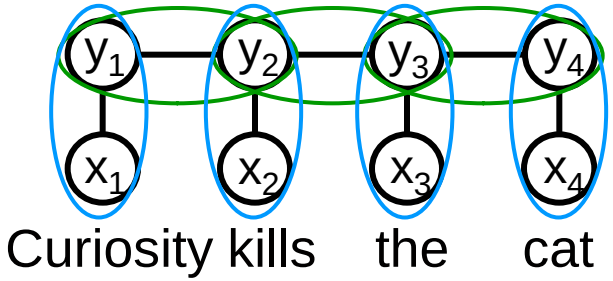
Lernen mit strukturierten Ausgaben
Schnittebenenalgorithmus

- Gegeben: $L = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$
- Wiederhole bis alle Sequenzen korrekt vorhergesagt werden.
  - Iteriere über alle Beispiele ($\mathbf{x_i}$, $\mathbf{y_i}$).
    - Bestimme $\bar{\mathbf{y}} = \arg\max_{\mathbf{y} \neq \mathbf{y}_i} \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}_i, \mathbf{y})$
    - Wenn $\mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}_i, \mathbf{y}_i) < \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}_i, \bar{\mathbf{y}}) + 1$ (Margin-Verletzung) dann füge Constraint $\mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}_i, \mathbf{y}_i) \geq \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}_i, \bar{\mathbf{y}}) + 1 - \xi_i$ dem Working Set hinzu.
    - Löse Optimierungsproblem für Eingabe $\mathbf{x_i}$, Ausgabe $\mathbf{y_i}$, und negative Pseudo-Beispiele $\bar{\mathbf{y}}$ (working set).
- Liefere **w** zurück.

Strukturierte Ausgaberäume

- Allgemeines Framework
- Anpassen von
  - Verlustfunktion
  - Feature-Mapping
  - Dekoder
    - $\arg\max_{\mathbf{y} \neq \mathbf{y}_i} \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}_i, \mathbf{y})$
    - ggf. mit Verlustfunktion
- Implementation:
  - http://www.cs.cornell.edu/People/tj/svm_light/svm_struct.html

Beispiel: Sequentielle Ein-/Ausgaben
(Feature-Mapping)



- Attribut für jedes Paar benachbarter Labels y_t und y_{t+1}.
  - $\varphi_{123}(y_t,y_{t+1})$ = [[y_t="Noun" $\wedge$ y_{t+1}="Verb"]]
- Attribut für jedes Paar aus Eingabe und Ausgabe.
  - $\bar{\varphi}_{234}(x_t,y_t)$ = [[y_t="Noun" $\wedge$ x_t = "cat"]]

- Label-label: $\sum_t \varphi_i(y_t,y_{t+1})$.
- Label-observation $\sum_t \bar{\varphi}_i(x_t,\bar{y}_t)$.

- Joint feature representation $\Phi(\mathbf{x},\mathbf{y})=\sum_t(\ldots,\varphi_{123}(y_t,y_{t+1}),\ldots,\bar{\varphi}_{234}(x_t,y_t),\ldots)^T$
- Weight vector $\mathbf{w}=(\ldots,w_{123},\ldots,w_{234},\ldots)^T$

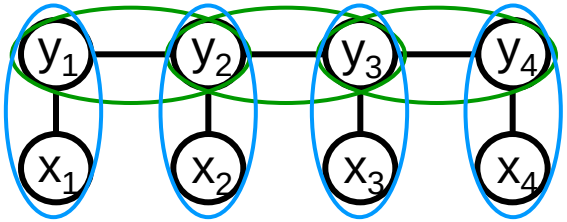
Beispiel: Sequentielle Ein-/Ausgaben
(Dekodierung)

- Um eine Sequenz zu klassifizieren, muss
  - $\mathbf{y}^* = \arg\max_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$

  berechnet werden.
- Das argmax geht über alle möglichen Sequenzen (exponentiell viele in der Länge).
- $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^{\mathrm{T}} \Phi(\mathbf{x}, \mathbf{y})$ summiert über Merkmale benachbarter Label-Paare und Merkmale von x_i-y_i-Paaren.
- Die Summanden unterscheiden sich nur dort, wo sich auch die y-Sequenzen unterscheiden.
- Mit dynamischer Programmierung kann argmax in linearer Zeit berechnet werden (Viterbi).

Beispiel: Sequentielle Ein-/Ausgaben (Dekodierung)

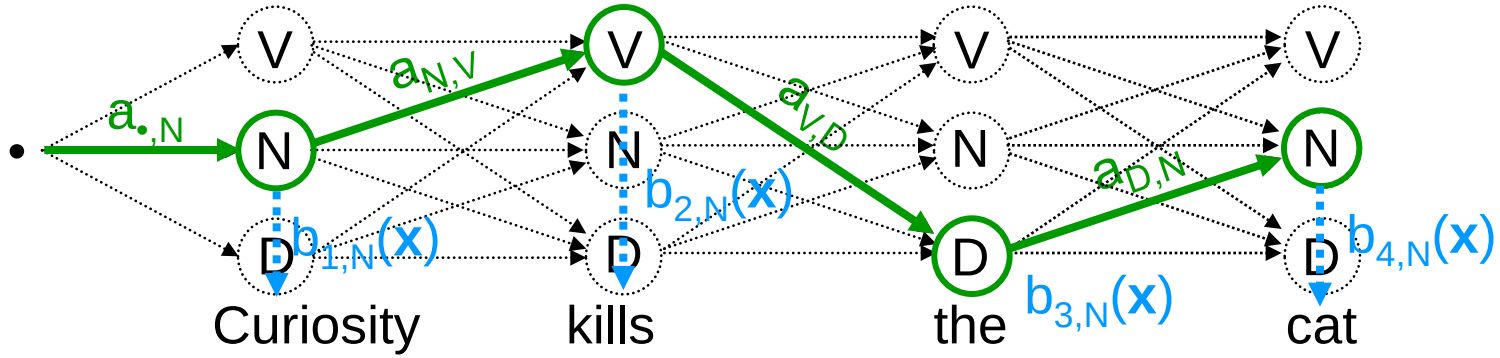
- Joint feature representation $\Phi(\mathbf{x},\mathbf{y})=\sum_t(\ldots,\varphi_{123}(y_t,y_{t+1}),\ldots,\varphi_{234}(x_t,y_t),\ldots)^T$



$\varphi_{123}(y_t,y_{t+1})$ = [[y_t="Noun" $\wedge$ y_{t+1}="Verb"]]

$\bar{\varphi}_{234}(x_t,y_t)$ = [[y_t="Noun" $\wedge$ x_t = "John"]]

- Decode $\mathbf{argmax}_{\mathbf{y}}\ \mathbf{w}^T\Phi(\mathbf{x},\mathbf{y})$ efficiently with transition matrix $A=\{a_{\sigma,\tau}\}$ and observation matrix $B_{\mathbf{x}}=\{b_{t,\sigma}(\mathbf{x})\}$, $\sigma,\tau \in \{\bullet,N,V,D\}$:



- E.g., $a_{\sigma,\tau}=\sum_i \sum_{\bar{y}\neq yi} \alpha_i(\bar{\mathbf{y}}) \sum_t [[y_{i,t}=\sigma \wedge y_{i,t+1}=\tau]] - [[\bar{y}_t= \sigma \wedge \bar{y}_{t+1}=\tau]]$.
- Auch bekannt unter HM SVM (uses 2-Best Viterbi decoding).

Viterbi-Algorithmus

- Definition: $a_{\sigma\tau} = S(y_t = \sigma, y_{t+1} = \tau)$

  …Score, der in $\mathbf{w}^T\Phi(\mathbf{x},\mathbf{y})$ erzeugt wird, wenn $y_t=\sigma$ und $y_{t+1}=\tau$.

- Definition: $b_\sigma(x_t) = S(y_t = \sigma, x_t)$

  …Score, der in $\mathbf{w}^T\Phi(\mathbf{x},\mathbf{y})$ erzeugt wird, wenn $y_t=\sigma$ und Wort x_t erscheint.

- Gesucht:
  - $\arg\max_{\mathbf{y}} \mathbf{w}^T\Phi(\mathbf{x},\mathbf{y}) = \arg\max_{y_1,\ldots,y_N} S(y_1,\ldots,y_N,x_1,\ldots,x_N)$

Viterbi-Algorithmus, Theorem

- $\delta_t(\sigma) = \max_{y_1,\ldots,y_{t-1}} S(y_1,\ldots,y_{t-1},y_t=\sigma,\mathbf{x}_1,\ldots,\mathbf{x}_t \mid \lambda)$
- Theorem:

  $\delta_{t+1}(\tau) = \left(\max_\sigma \delta_t(\sigma) + a_{\sigma\tau}\right) + b_\tau(\mathbf{x}_{t+1})$
- Beweis:

$$\begin{aligned}
\delta_{t+1}(\tau) &= \max_{y_1,\ldots,y_t} S(y_1,\ldots,y_t,y_{t+1}=\tau,x_1,\ldots,x_{t+1}) \\
&= \max_{y_1,\ldots,y_t} S(y_1,\ldots,y_t,x_1,\ldots,x_t) + S(y_{t+1}=\tau \mid y_t=\sigma,\ldots) + S(x_{t+1} \mid y_{t+1},\ldots) \\
&= \max_\sigma (\max_{y_1,\ldots,y_{t-1}} S(y_1,\ldots,y_t=\sigma,x_1,\ldots,x_t) + a_{\sigma\tau}) + b_\tau(x_{t+1}) \\
&\left(\max_\sigma \delta_t(\sigma) + a_{\sigma\tau}\right) + b_\tau(x_{t+1})
\end{aligned}$$

- Lineare Bestimmung des höchst-scorenden Pfades mit dynamischer Programmierung.

Strukturierte Ausgaberäume
Sequentielle Ein-/Ausgaben

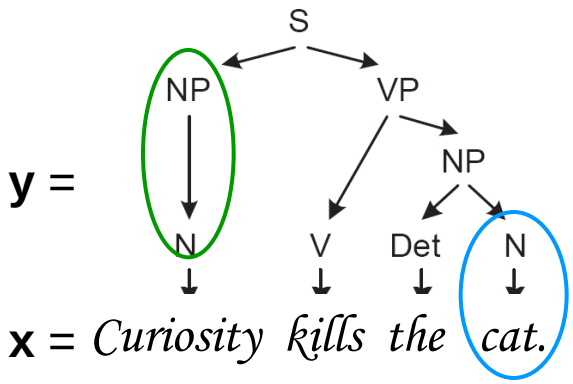
- Dekoderierung: Viterbi-Algorithmus
- Feature-Mapping:
  - Einträge für benachbarte Zustände

$$\varphi_{123}(y_t, y_{t+1}) = [[y_t = \text{“Noun”} \wedge y_{t+1} = \text{“Verb”}]]$$

  - Einträge für Beobachtungen in einem Zustand

$$\varphi_{234}(x_t, y_t) = [[y_t = \text{“Noun”} \wedge x_t = \text{“John”}]]$$

Strukturierte Ausgaberäume
Natural Language Parsing



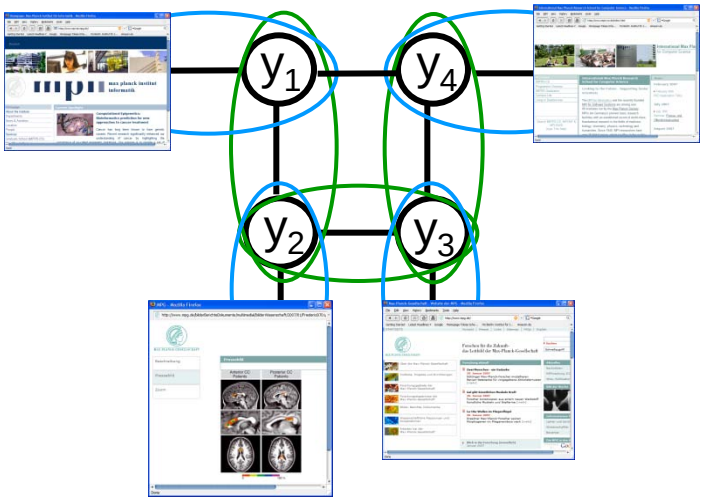
$$\varphi_{NP\to N}(y) = \sum_{p\in \mathbf{y}} [[\, y_p = NP\to N \,]]$$

$$\varphi_{N\to\text{"cat"}}(\mathbf{x},y) = \sum_{p\in \mathbf{y}} [[\, y_p = N\to\text{"cat"} \,]]$$

$$\Phi(\mathbf{x},\mathbf{y}) = \begin{pmatrix} 1 & S \to NP,VP \\ 1 & NP \to N \\ 0 & VP \to V \\ 1 & VP \to V,NP \\ \vdots & \\ 0 & N \to ate \\ 1 & N \to cat \end{pmatrix}$$

- Joint feature representation: $\Phi(\mathbf{x},\mathbf{y})=(\ldots,\varphi_{NP\to N}(y_p),\ldots,\varphi_{N\to\text{"cat"}}(\mathbf{x},y_p),\ldots)^T$
- Weight vector $\mathbf{w} = (\ldots,w_{NP\to N},\ldots,w_{N\to\text{"cat"}},\ldots)^T$
- Dekodierung mit dynamischer Programmierung
  - CKY-Parser, $O(n^3)$.

Strukturierte Ausgaberäume
Kollektive Klassifikation



- Attribut für jedes Paar benachbarter Labels y_t und y_{t+1}.
  - $\varphi_{123}(y_i,y_j)$
- Attribut für jedes Paar aus Eingabe und Ausgabe.
  - $\bar{\varphi}_{234}(x_i,y_i) = \bigwedge [[y_t=\text{“Research Institute”}, x_t = \text{“}$  $\text{”}]]$

- Dekoder: Message Passing Algorithmus.

Ranking, Ordinale Regression

- Instanzen sollen in die richtige Reihenfolge gebracht werden.
  - z.B. Relevanz-Ranking von Suchergebnissen.
  - z.B. Relevanz-Ranking von Produktempfehlungen.
- Beispiele sind Paare:
  - $L = \langle (f(\mathbf{x}_i) \geq f(\mathbf{x}_j)), \ldots \rangle$
  - = $\mathbf{x}_I$ soll im Ranking vor $\mathbf{x}_J$ stehen

Ranking, Ordinale Regression

- Relevanz-Ranking von Suchergebnissen.
  - Webseiten $\mathbf{x}_l$, Suchanfrage q.
  - Gemeinsame Merkmalsrepräsentation $\Phi(\mathbf{x}_l,q)$ von Webseite und Suchanfrage.
  - $\Phi(\mathbf{x}_l,q)$ kann Vektor verschiedener Merkmale der Übereinstimmung von $\mathbf{x}_l$ und q sein.
  - z.B. Anzahl übereinstimmende, Wörter; Übereinstimmungen in H1-Umgebung, PageRank, …
- Beispiele sind Paare:
  - $L = \langle (f(\mathbf{x}_i, q) \geq f(\mathbf{x}_j, q)), \ldots \rangle$
  - = $\mathbf{x}_l$ soll für Anfrage q im Ranking vor $\mathbf{x}_J$ stehen.

Ranking, Ordinale Regression

- Relevanz-Ranking von Suchergebnissen.
- Beispiele sind Paare:
  - $L = \langle (f(\mathbf{x}_i, q) \geq f(\mathbf{x}_j, q)), \ldots \rangle$
  - = $\mathbf{x}_I$ soll für Anfrage q im Ranking vor $\mathbf{x}_J$ stehen.
- Beispiele können z.B. aus Klick-Daten gewonnen werden.
- Ein Benutzer stellt Anfrage q, bekommt Ergebnisliste, klickt dann auf i-tes Listenelement x_i.
- Verwirft damit Listenelemente 1..i-1 implizit.
- Für diesen Benutzer und Anfrage q hätte x_i an erster Stelle stehen sollen.

Ranking, Ordinale Regression

- Relevanz-Ranking von Suchergebnissen.
- Beispiele sind Paare:
  - $L = \langle (f(\mathbf{x}_i, q) \geq f(\mathbf{x}_j, q)), \ldots \rangle$
- Ein Benutzer stellt Anfrage q, bekommt Ergebnisliste, klickt dann auf i-tes Listenelement x_i.
- Für diesen Benutzer und Anfrage q hätte x_i an erster Stelle stehen sollen.
- Daraus ergeben sich Beispiele:
  - $L_q = \langle (f(\mathbf{x}_i, q) \geq f(\mathbf{x}_1, q)), \ldots, ((f(\mathbf{x}_i, q) \geq f(\mathbf{x}_{i-1}, q)) \rangle$

Ranking, Ordinale Regression

- Gegeben: Beispiele
  - $L = \left\langle \begin{array}{l} (f(\mathbf{x}_{1i_1}, q_1) \geq f(\mathbf{x}_{11}, q_1)), \ldots, ((f(\mathbf{x}_{1i_1}, q_1) \geq f(\mathbf{x}_{1i_1-1}, q_1)), \\ \ldots, \\ (f(\mathbf{x}_{mi_m}, q_m) \geq f(\mathbf{x}_{m1}, q_m)), \ldots, ((f(\mathbf{x}_{mi_m}, q_m) \geq f(\mathbf{x}_{mi_m-1}, q_m)) \end{array} \right\rangle$
- Löse
  - min ½ $|\mathbf{w}|^2$ + C $\sum_J \sum_i \xi_{Ji}$
    s.d. $\forall j \; \forall i \leq i_J$ $\mathbf{w}^T(\Phi(\mathbf{x}_{Ji_1}, \mathbf{q}_J) - \Phi(\mathbf{x}_{Ji_J}, \mathbf{q}_J)) \geq 1 - \xi_{Ji}$
    $\forall j \; \forall i$ $\xi_{Ji} \geq 0.$

Subgradientenverfahren
Perzeptron

zufällig geordnet

- Gegeben: $L = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$
- Initialisiere $\mathbf{w}^0 := \mathbf{0}$, $t := 0$
- Wiederhole
  - Iteriere über alle Trainingsbeispiele $(\mathbf{x}_i, y_i)$
    - Setze $t := t + 1$
    - Setze $\mathbf{d} := \begin{cases} -y_i \mathbf{x}_i & , \text{wenn } y_i \left\langle \mathbf{w}^{t-1}, \mathbf{x}_i \right\rangle \leq 0 \\ 0 & , \text{sonst.} \end{cases}$
    - Setze $\mathbf{w}^t := \mathbf{w}^{t-1} - \mathbf{d}$

  bis $\left\| \mathbf{w}^t - \mathbf{w}^{t-n} \right\|^2 = 0$
- Liefere $\mathbf{w}^t$ zurück

Subgradientenverfahren
Support-Vector-Machine

zufällig geordnet

- Gegeben: $L = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, $C > 0$
- Initialisiere $\mathbf{w}^0 := \mathbf{0}$, $t := 0$
- Wiederhole
  - Iteriere über alle Trainingsbeispiele $(\mathbf{x}_i, y_i)$
    - Setze $t := t + 1$
    - Setze $\mathbf{d} := \begin{cases} n^{-1}\mathbf{w}^{t-1} - C \cdot y_i \mathbf{x}_i & , \text{wenn } y_i \left\langle \mathbf{w}^{t-1}, \mathbf{x}_i \right\rangle < 1 \\ n^{-1}\mathbf{w}^{t-1} & , \text{sonst.} \end{cases}$
    - Setze $\mathbf{w}^t := \mathbf{w}^{t-1} - \frac{1}{t+1}\mathbf{d}$

  bis $\left\| \mathbf{w}^t - \mathbf{w}^{t-n} \right\|^2 \leq \varepsilon$
- Liefere $\mathbf{w}^t$ zurück

Subgradientenverfahren

Strukturierte Support-Vector-Machine

zufällig geordnet

- Gegeben: $L = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$, $C > 0$
- Initialisiere $\mathbf{w}^0 := \mathbf{0}$, $t := 0$
- Wiederhole
  - Iteriere über alle Trainingsbeispiele $(\mathbf{x}_i, \mathbf{y}_i)$
    - Setze $t := t + 1$
    - Setze $\mathbf{d} := \begin{cases} n^{-1}\mathbf{w}^{t-1} - C \cdot \delta\Phi_i & , \text{wenn } \left\langle \mathbf{w}^{t-1}, \delta\Phi_i \right\rangle < 1 \\ n^{-1}\mathbf{w}^{t-1} & , \text{sonst.} \end{cases}$
    - Setze $\mathbf{w}^t := \mathbf{w}^{t-1} - \frac{1}{t+1}\mathbf{d}$

  bis $\left\| \mathbf{w}^t - \mathbf{w}^{t-n} \right\|^2 \leq \varepsilon$
- Liefere $\mathbf{w}^t$ zurück

$$\delta\Phi_i := \Phi(\mathbf{x}_i, \mathbf{y}_i) - \Phi(\mathbf{x}_i, \bar{\mathbf{y}})$$

$$\bar{\mathbf{y}} := \arg\max_{\mathbf{y} \neq \mathbf{y}_i} \left\langle w, \Phi(\mathbf{x}_i, \mathbf{y}) \right\rangle$$

Subgradientenverfahren
Stochachtic Gradient Descent

zufällig geordnet

- Gegeben: $L = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$, Lernrate $\alpha > 0$
- Initialisiere $\mathbf{w}^0 := \mathbf{0}$, $t := 0$
- Wiederhole
  - Iteriere über alle Trainingsbeispiele $(\mathbf{x}_i, \mathbf{y}_i)$
    - Setze $t := t + 1$
    - Setze $\mathbf{d} := \nabla_i J_P(\mathbf{w})$

      Subgradient der Zielfunktion
    - Setze $\mathbf{w}^t := \mathbf{w}^{t-1} - \alpha \mathbf{d}$

  bis $\left\| \mathbf{w}^t - \mathbf{w}^{t-n} \right\|^2 \leq \varepsilon$
- Liefere $\mathbf{w}^t$ zurück

Halbüberwachtes Lernen

- Trainingsdaten $(\mathbf{x}_i, \mathbf{y}_i)$-Paare.
- Eingabe $\mathbf{x}_i$ meist kein Problem, Ausgabe $\mathbf{y}_i$ teuer.
- Halbüberwachtes Lernen:
  - Wenige $(\mathbf{x}_i, \mathbf{y}_i)$-Paare.
  - Viele Eingaben $\mathbf{x}_i$.

Halbüberwachtes Lernen

- Verschiedene Mechanismen, mit denen Informationen aus ungelabelten Daten genutzt werden.
  - Transduktion
  - Graph-Laplacians
  - Co-Lernen

Halbüberwachtes Lernen

- Verschiedene Mechanismen, mit denen Informationen aus ungelabelten Daten genutzt werden.
  - Transduktion
    - Gelabelte Beispiele: auf der richtigen Seite der Trennebene mit maximalen Margin.
    - Ungelabelte Beispiele: auf irgendeiner Seite der Trennebene, aber mit maximalen Margin.
  - Graph-Laplacians
  - Co-Lernen

Halbüberwachtes Lernen

- Verschiedene Mechanismen, mit denen Informationen aus ungelabelten Daten genutzt werden.
  - Transduktion
  - Graph-Laplacians
    - Cluster-Annahme: die Ebene sollte nicht durch Gebiete mit hoher Beispiel-Dichte hindurchgehen.
    - Wenn zwei Beispiele ähnlich sind, müssen auch die Funktionswerte ähnlich sein.
  - Co-Lernen

Halbüberwachtes Lernen

- Verschiedene Mechanismen, mit denen Informationen aus ungelabelten Daten genutzt werden.
  - Transduktion
  - Graph-Laplacians
  - Co-Lernen
    - Gelabelte Beispiele: Lerne zwei möglichst unabhängige Hypothesen (Aufteilen der Feature).
    - Ungelabelte Beispiele: Maximiere Konsens zwischen Hypothesen.

Lineare Klassifikatoren

- Das Perzeptron.
- Lineare Klassifikatoren.
- Kernel-Maschinen.
- Multiklassenfall,
- Lernen von Sequenzen, Strukturen.
- Halbüberwachtes Lernen.