

# ① Support Vector Machine (Machine Learning I)

zugrundeliegender Gedanke: Occam's Razor

- Löse nie ein Problem komplizierter als nötig, denn die einfachste richtige Erklärung ist die Beste

Lernen ist Schätzen einer Abbildung (Hypothese)

Lernproblem definiert durch  $X \times Y$  (Lernbeispiele:  $(\vec{x}_1, y_1) \dots (\vec{x}_n, y_n) \in X \times Y$ )

$\{\text{Attr}_1, \dots, \text{Attr}_n\} \times \{\text{true}, \text{false}\}$  - Konzeptlernen

$\mathbb{R}^N \times \{\text{Klasse}_1, \dots, \text{Klasse}_n\}$  - Klassifikation

$\mathbb{R}^N \times \mathbb{R}$  - Regression

Wiederholung: Minimierung des Testfehlers (VC-Dimension)

- reale Fehler  $E(h_a)$  soll minimiert werden beim Lernen (Wunsch)  
↳ kann meistens nicht berechnet werden → Schätzung

- Schätzung als empirischen Fehler  $E_{\text{emp}}(h_a)$

- nach Theorie von Vapnik: obere Schranke für realem Fehler als Summe vom emp. Fehler und Term für Kapazität einer Lernmaschine (VC)

$$\rightarrow E(h_a) \leq \approx E_{\text{emp}}(h_a) + \sqrt{\dots \frac{VC(h_a)}{N} \dots}$$

Beim Lernen wird obere Schranke versucht zu minimieren

D.h. -> nur LMs, deren Kapazität geeignet ist (nicht zu hoch)  
-> verschiedene Modelle einsetzen, die emp. F. minimiert  
-> z.B. Beispiele finden

## VC-Dim.

Kardinalität der größten Punktmenge, die der Attribut trennen kann

Lösungsansatz für Fehlerminimierung: structural Risk Minimization („Metagradienten“)

Folge:  
realer Fehler wird nicht minimiert!

- Lernmaschine so konstruieren, dass sie in verschiedenen Hypothesenräumen „sucht“

- Hypothesenräume sollen dabei eine Ordnung haben, bei der die VC-Dimension steigend ist:  $H^1 \subset H^2 \subset \dots \subset H^n$ ,  $VC(h_a^1) \leq VC(h_a^2) \dots$

→ wenn das gelingt: Iteration durch Hypothesenräume → Gesamtminimum des Ausdrucks:

$$\min_{H^n} (E_{\text{emp}}(h_a) + \sqrt{\dots \frac{VC(h_a)}{N} \dots}) \text{ finden}$$

→ funktioniert häufig gut, z.B. bei „verbundenen“ neuronalen Netzen wo Neuronen hinzugefügt und VC-Dim. erhöht werden kann (Cascade Correlation), außerdem Topologie bekannt

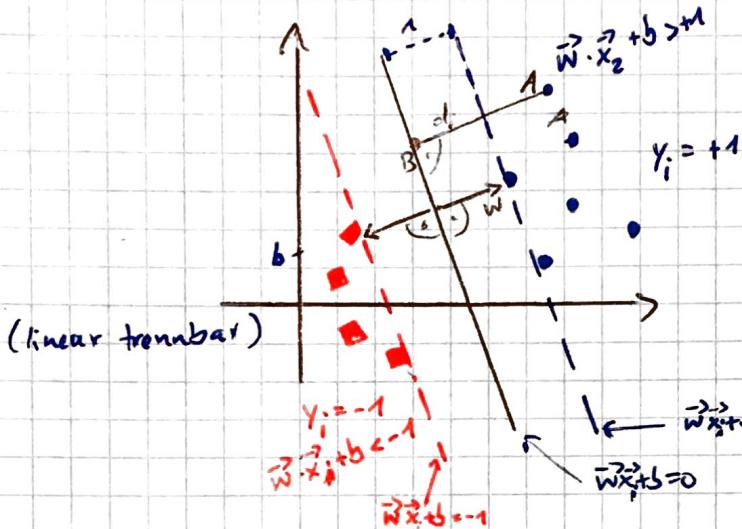
→ weitere Methode, die eine solche Ordnung ermöglicht: SVM

(2)

## Support Vector Machine

Trenne zwei Mengen mit einer besten Trennebene, die größten Randbereich zwischen den Klassen hat (= höchste Generalisierung)

### Einleitung zu Randbereiche



### Beispiel Margin = 1

Punkte auf blauer und roter "Grenze":

$$\vec{w}^T \vec{x}_1 + b = 1$$

$$\vec{w}^T \vec{x}_2 + b = -1$$

$$\Rightarrow \vec{w}(\vec{x}_1 - \vec{x}_2) = 2$$

$$\Rightarrow \frac{\vec{w}}{\|\vec{w}\|} \cdot (\vec{x}_1 - \vec{x}_2) = \frac{2}{\|\vec{w}\|}$$

Abstand  
zur zwei Klassen = 2  
wenn Margin = 1

### Bedingungen:

$$\begin{cases} \vec{w}^T \vec{x}_i + b > 1 & \text{wenn } y_i = +1 \\ \vec{w}^T \vec{x}_i + b < -1 & \text{wenn } y_i = -1 \end{cases}$$

Wie wird  $d_i$  bestimmt?

$\vec{w}$  Einheitsvektor der in die  
selbe Richtung wie  $\vec{w}$  zeigt  
 $A$ : repräsentiert  $x_i$

$\Rightarrow B$  liegt auf Grenze, erfüllt:  
 $\vec{w}^T (\vec{x}_i - d_i \cdot \frac{\vec{w}}{\|\vec{w}\|}) + b = 0$

$$d_i = \left( \frac{\vec{w}}{\|\vec{w}\|} \right)^T \vec{x}_i + \frac{b}{\|\vec{w}\|}$$

(gilt für  $y_i = 1$ )

$$\text{Allgemein: } d_i = y_i \left( \left( \frac{\vec{w}}{\|\vec{w}\|} \right)^T \vec{x}_i + \frac{b}{\|\vec{w}\|} \right) \quad (1)$$

Der kleinste Margin wird damit wie folgt berechnet:

$$d = \min_{i=1 \dots m} d_i$$

(durchsuche die Abstände  $d_i$  für die Punkte 1 ... m)

### Maximalen Margin finden:

$$\max \left( \frac{1}{\|\vec{w}\|} \right) = \min \left( \|\vec{x}\|^2 \right)$$

Ansatz:  $\max d$ ,

$$\text{also: } y_i (\vec{w}^T \vec{x}_i + b) \geq d \quad (i=1, \dots, m \text{ und } \|\vec{w}\|=1) \quad (1)$$

Aber: schwer zu lösen, denn  $\|\vec{w}\|=1$  (Annahme) ist nicht konvex

Transformiere zu konvexer minimierbarer Funktion (nutze Tatsache, dass  $w$  und  $b$  beliebig skaliert werden können  $\Rightarrow$  margin wird auch skaliert):

$$(\text{wähle Margin}=1): \text{maximiere } \frac{2}{\|\vec{w}\|^2} = \text{minimiere } \frac{1}{2} \|\vec{w}\|^2$$

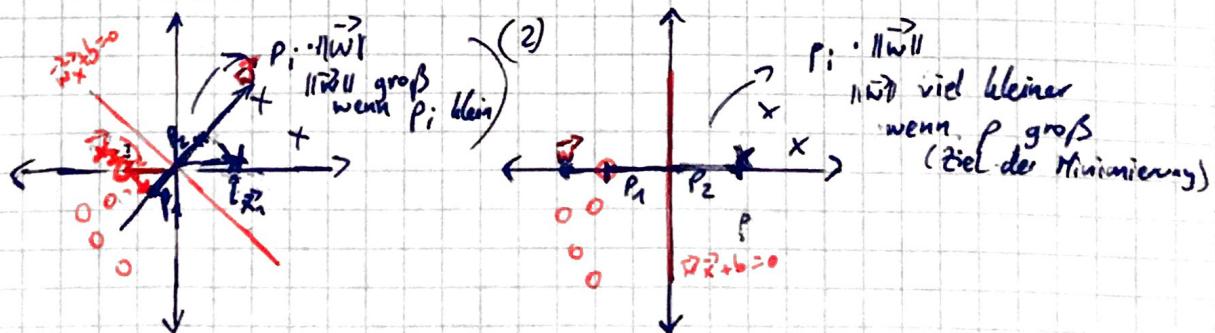
$$\text{Es ergibt sich: } \min \frac{1}{2} \|\vec{w}\|^2$$

$$\text{damit: } y_i (\vec{w}^T \vec{x}_i + b) \geq 1$$

$d=1$   
Achtung! Nicht  $\vec{w}$  (ktor), sondern die Norm von  $\vec{w}$   
wird minimiert, sondern Quadrieren nur für bessere Ableitbarkeit!  
(auch  $\frac{1}{2}$ )

Gegeben  $\vec{w}^T \vec{x}_i = p_i \cdot \|\vec{w}\|$ , Minimieren von  $\|\vec{w}\|$  hat zur Folge:

(Ange der  
Projektion der  $\vec{x}_i$ -Vektoren auf  $\vec{w}$   
multipliziert mit Norm  $\|\vec{w}\|$   
ist gleich Skalarpr.  
von  $\vec{w}^T$  und  $\vec{x}_i$ )



SVM wählt durch Minimierung (2) anstatt (1)  $\rightarrow$  größere Stützvektoren

# Optimale Hyperfläche mit Lagrange finden

Optimale Hyperfläche:  $\min_{i=1 \dots n} \frac{1}{2} \|\vec{w}\|^2$ , sodass:  $y_i(\vec{w}^T \vec{x}_i + b) \geq 1, i=1 \dots m$  (1)

Minimierung über Lagrange-Methode  
( $\vec{w}$  und  $b$  finden, die (1) minimieren)

$$L_p = L(\vec{w}, b, \alpha) = \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^m \alpha_i [y_i(\vec{w}^T \vec{x}_i + b) - 1]$$

Lagrange-Multiplikatoren ( $\alpha \geq 0$ )

Wende Sattelpunkt-Bedingungen an:

(1) Minimum von  $L$  bzgl.  $\vec{w}$  und  $b$

$\Rightarrow$  1. Ableitung = 0

$$\frac{\partial L}{\partial b} = \sum_{i=1}^m y_i \alpha_i = 0 \quad \frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^m \vec{x}_i y_i \vec{x}_i = 0$$

$$\vec{w} = \sum_{i=1}^m \alpha_i y_i \vec{x}_i$$

(2) Maximum von  $L$  bzgl.  $\alpha_1 \dots \alpha_m$

$$\sum_{i=1}^m \alpha_i y_i = 0 \quad \alpha_i \geq 0 \quad i=1 \dots m$$

(1) Eingesetzt in  $L_p$  ergibt duale Lagrange-Gleichung

$$\vec{w} = W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \vec{x}_i \vec{x}_j \xrightarrow{\text{Skalarprodukt}}$$

$\Rightarrow$  Vorteil: nur noch von  $\alpha$  abhängig + einfacher zu lösen  
mit (2)  $W(\alpha)$  maximieren:  $\max_{\alpha} W(\alpha)$

II

Es ergeben sich aus der Maximierung von  $W(\alpha)$   
wenige  $\alpha > 0 \Rightarrow$  Support-Vektoren

- die meisten  $\alpha_i = 0$  (KKT dual complementarity condition)

$\rightarrow$  für  $x_i$  mit  $\text{margin} = 1$  sind die  $\alpha_i > 0 \Rightarrow$  Support-Vektor

- für die  $\alpha_i > 0$  lassen sich mit

$$\vec{w} = \sum_{i=1}^m \alpha_i y_i \vec{x}_i$$

die Support-Vektoren berechnen (linearkombination)  $\Rightarrow$  daraus  $\vec{w}^T \vec{x} + b$

bestimmbar  
(Support-Vektoren liegen orthogonal auf  $\vec{w}^T \vec{x} + b$ )

Annahme: Modell wurde trainiert, Schätzung für neuen Punkt  $x$ :

1) wenn linear SVM:  $y = 1$ , wenn  $\vec{w}^T \vec{x} + b > 1$  (ein Skalarprodukt aller Features)

2) wenn non-linear SVM: einsetzen  $\Rightarrow \vec{w}^T \vec{x} + b = \sum_{i=1}^m \alpha_i y_i \langle \vec{x}_i, \vec{x} \rangle + b \Rightarrow$  nur noch Skalarprodukt der Train samples

und neuer  $x \Rightarrow$  nur für sehr wenige

(Support-Vektor) da meist  $\alpha = 0$

$\Rightarrow$  d.h. wenn linear SRF, dot product des  $\vec{w}$ -Vektors und  $x$

und wenn non-linear, dot product der support vectors u.  $x$

Verbesserung: nicht fordern, dass alle Daten korrekt klassifiziert werden  $\rightarrow$  Soft Margin

$\Rightarrow$  d.h. wenn linear SRF, dot product des  $\vec{w}$ -Vektors und  $x$

$\Rightarrow$  Änderung der Randbedingungen:  $y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \gamma_i; \gamma_i \geq 0$

$\Rightarrow$  Minimiere nun:  $\min_{\vec{w}, b, \gamma_i} \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^m \gamma_i$

(C ist Regularisierungsparameter)

- C groß  $\rightarrow$  wenig Fehlklassifikation

Bedingungen: 1.)  $y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \gamma_i; \gamma_i \geq 0$

(Summe muss klein sein) keine Fehler

$\Rightarrow$  2.)  $\gamma_i \geq 0$

- C klein  $\rightarrow$  große Margins

(Summe darf größer sein)

$\Rightarrow$  Anhand der folgenden Bedingungen lässt sich feststellen, wo sich Vektoren  $\vec{x}_i$  befinden:  
(erneut viele Vektoren  $\vec{x}_i$  mit  $\alpha_i = 0$ )

$\alpha_i < C$   $\rightarrow$   $\vec{x}_i$  auf Rand (margin vector)

$\alpha_i = C$  und:

$\gamma_i > 1$  Fehlklassifikation ( $1 - \gamma_i < 0$ )

$0 < \gamma_i \leq 1$  richtig aber geringer Abstand (margin error)

$\gamma_i = 0$  margin vector

\*  $L_p$  hat nun noch zwei weitere Terme (C und  $\gamma_i$  [ergänzt])

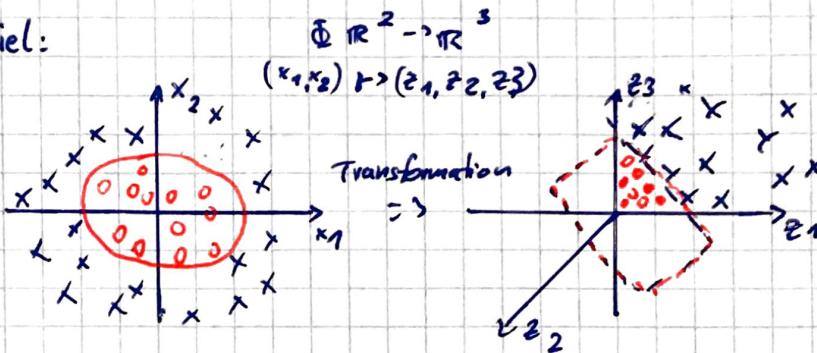
## Nichtlineare SVM mithilfe von Kernel-Methoden

Problem: Daten nicht linear separierbar (bei Klassifikation)

Lösung: transformieren der Daten in einen anderen (höherdimensionalen)

Raum:  $\Phi: \mathbb{R}^n \rightarrow \mathbb{R}^m$ ,  $n > m$  (meistens)

Beispiel:



neues Problem: Transformation und Rechnen in hochd. Räumen rechenintensiv  
neue Lösung:

### Kernel-Trick

Idee: für ein Kernelalgorithmus (hier: SVM), der in Form von Skalarprodukten  $\langle x, z \rangle$  seiner Eingabedaten (Vektoren)  $x$  und  $z$  beschrieben werden kann, ist es möglich,  $\langle x, z \rangle$  durch eine Kernelfunktion  $K(x, z)$  zu beschreiben, die Skalarprodukte in einem anderen Raum durchführt.

=> erlaubt das Arbeiten in hochd. Räumen, ohne explizites Berechnen von Koordinaten in diesen Räumen.

### Was es nicht tut:

Der Kernel-Trick erzeugt kein Mapping von Punkten eines niedrig-dimensionalen in ein hoch-dimensionalem Raum.

### Was es tut:

stellt eine Möglichkeit dar, Skalarprodukte von Punkten direkt in einem hochdimensionalen Raum zu berechnen, ohne das Mapping zu kennen.

=> Beispiel: sei  $\Phi(\vec{x})$  die Mapping-Funktion für Vektor  $\vec{x}$  eines niedrig-dim. Raums in einen Vektor  $\vec{x}'$  eines hoch-dim. Raums, z.B.

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \text{ und } \vec{x}' = \begin{pmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{pmatrix}$$

3-dim.

Um nur  $\Phi(\vec{x}) \cdot \Phi(\vec{z})$  zu berechnen, ist es notwendig:  
 -> 9 Mult. bzw. 18 Mult. für  $\vec{x}_i$  und  $\vec{x}_j$  durchzuführen  
 -> 9 Mult. und 8 Additionen für Skalarprodukt durchzuführen  
 => insgesamt 35 Operationen

Stattdessen nun die Kernelfunktion  $K(\vec{x}, \vec{z}) = (\vec{x} \cdot \vec{z})^2$

$$\left( \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \right)^2 = (x_1 z_1 + x_2 z_2 + x_3 z_3)^2$$

Operationen:  
 -> 3 Mult., 2 Additionen  
 ->  $\cdot$  = eine Multiplikation

Neben dem Vorteil des verminderten Rechenaufwands, erlaubt das nicht explizite Berechnen der Features auch Mappings in infinit-dimensionalen Räume (Gaussian Kernel)

5) Nicht jede Funktion  $\Phi: \mathbb{R}^n \mapsto \mathbb{R}^m$  mit  $n, m \in \mathbb{N}$  ist eine gültige Kernelfunktion (Muss „Mercer“-Theorem erfüllen)

### Beliebte Kernelfunktionen

- Skalarprodukt  $K(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y}$  (nicht höher-dimensional)
- Polynom (Vonk)  $K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + c)^d$
- Radial Basis-Funktion (RBF), Gaussian-Kernel  $K(\vec{x}, \vec{y}) = e^{-\frac{\|\vec{x} - \vec{y}\|^2}{2\sigma^2}}$  immer noch auf Basis von Skalarprodukten, denn:  
= "Similarity-Berechnung"  
$$K'(\vec{x}, \vec{y}) = e^{-\frac{\vec{x} \cdot \vec{y}}{\sigma^2}} = \sum_{n=0}^{\infty} \frac{(\vec{x} \cdot \vec{y})^n}{\sigma^n n!}$$
 bzw.
- Sigmoid  $K(\vec{x}, \vec{y}) = \tanh(K(\vec{x} \cdot \vec{y}) + \theta)$

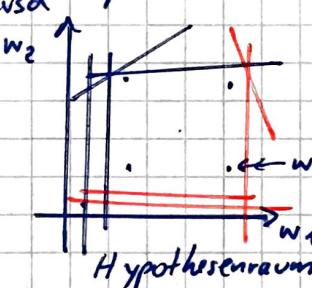
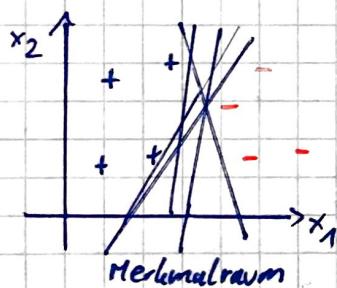
### SVM-Architektur (Ähnlichkeit zu einschichtigen neuronalen Netzen)

- Idee:
- Neuronen implementieren Kernelfunktion ( $k(x, x_i)$ )
  - Gewichte der Neuronen sind Zusammenfassung von  $\alpha_i$  und  $y_i$ :  $\lambda_i = \alpha_i y_i$
  - Klassifikatorfunktion  $f(x) = \text{sign}(\sum \lambda_i k(x, x_i) + b)$
  - NN gibt 0 oder 1 für sample aus
  - Support-Vektoren sind

### Version Space für SVM

Besondere Eigenschaft: Dualität von Merkmals- und Hypothesenraum

=> Punkte ( $\vec{x}_i$ ) im Merkmalsraum korrespondieren zu Hyperebenen im Hypothesenraum und vice versa



$$y_i(w^T x_i + b) \geq 1 \quad i = 1 \dots n$$

(bei korrekter Klassifikation,  
„version space duality“,  
Vapnik, 1998)

Sind genügend Datapunkte vorhanden, wird der Hypothesenraum so stark eingeschränkt, dass nur noch ein oder wenige  $w^*$ 's übrigbleiben => das erledigt Lagrange-Optimierung  
-> die Suche nach dem besten  $w^*$  hat auch im Hypothesenraum den größten Abstand zu allen Hyperebenen der Punkte hat:  $w^*$

Also: Lagrange-Optimierung schränkt Hypothesenraum durch Berücksichtigung immer weiterer Datapunkt zunehmend ein. Dann -> Beste Wahl der Trennebene (kleiner Fehler) mit Mittelpunkt der Hyperkugel

Mittelpunkt der Hyperkugel  $w^*$

⑥

# Support vector Machine (Machine Learning I)

## Multiple Klassifikation (k Klassen)

- Einer - gegen - Alle:

- Trainiere eine SVM für jede Klasse gegen alle Daten nicht angehörig der Klasse
- bei  $k$  Klassen  $k$  SVM's
- nach dem Training (Testzeit) führe Mehrheitsentscheidung durch, d.h. frage jede SVM: gehört  $x$  zu deiner Klasse?

- Einer - gegen - Einen:

- Trainiere eine SVM für jede Klasse gegen Daten einer anderen Klasse (blende dabei die anderen aus)
- bei  $k$  Klassen  $\frac{k(k-1)}{2}$  SVM's (mehr als E-g-A)
- erneut Mehrheitsentscheidung

- Mehrfachzugehörigkeit (Multiple)

- Trainiere eine SVM für jede Klasse gegen alle Daten ( $k$  SVM)
- ein Datenpunkt kann mehreren Klassen angehören
- anderes Mehrheitsentscheidungsverfahren: muss multiple Zugehörigkeit berücksichtigen

- Multi-Class SVM (nach Watkins)

- ein Datenpunkt kann auch hier mehreren Klassen angehören
- kein Abstimmungsverfahren, Zugehörigkeit über Klassifikationsvektor für jeden Datenpunkt bestimmbar ( $k$  - Einträge in Vektor)

## Gewichtete SVM

- je ein  $C$ -Faktor (Regularisierungsparameter) für positive u. negative Klasse:

$$\min \frac{1}{2} \|\vec{w}\|^2 + C_+ \left( \sum_{y_i=1, i=1}^m \xi_i \right) + C_- \left( \sum_{y_i=-1, i=1}^m \xi_i \right)$$

→ Intuitiv: stellt Schäfte der Trennwände ein, z.B.

$$(C_-, C_+) = (0.1, 10) \rightarrow \text{neg. Klasse hat unschärferen Rand als positive}$$

## Dichte-Träger Schätzung mit SVM

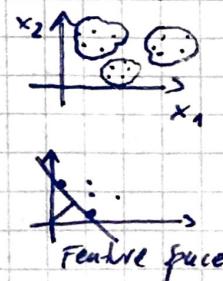
- gesucht wird eine Funktion  $f$ , die für eine kleine Region, welche die meisten Lernbeispiele enthält, den Wert  $> 0$  und sonst Wert 0 oder  $< 0$  annimmt
- ähnlich zu unüberwachtem Lernen (Clustering)

Idee: - Trennung der Lernbeispiele vom Ursprung des transformierten Merkmalsraums

=> nicht-linearer Schätzter

- Optimierung mit modifizierter Problemdefinition:

$$\min \frac{1}{2} \|\vec{w}\|^2 + \dots$$



## Weitere Kernel - Methoden

### Kernel Perceptron

- Anwendung des Perceptron - Algorithmus mithilfe einer Kernel - Funktion (und damit im höherdimensionalen Raum)
- Wiederholung: Perceptron - Algorithmus implementiert binären Klassifikator mit der Kombination mehrerer Gewichte mit einem Eingabe - Featurevektor  
 $\Rightarrow$  Skalarprodukt:  $\sum_{i=1}^n w_i \cdot x_i$  bzw. Hyperebene  $h(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x})$
- Update der  $w_i$  ist error-driven ( $\vec{w}_{t+1} = \vec{w}_t + y_i \cdot \vec{x}_i$ ) mit  $y = \text{sgn}(\vec{w} \cdot \vec{x}_i)$   $y_i \in \{-1, 1\}$
- Beschreibung Perceptron Gewichte als eine Linearkombination der training samples:  $w = \sum_{i=1}^n \alpha_i \cdot y_i \cdot \vec{x}_i$  ( $\alpha_i$ : Anzahl  $x_i$ : Fehlklassifiziert)
- Anpassung Perceptron - Algo: 1)  $y = \text{sgn}(w \cdot x) \Rightarrow y = \text{sgn}(\sum_{i=1}^n \alpha_i \cdot y_i \cdot \langle x_i, x \rangle)$   
 2) Skalarprodukt  $\langle x_i, x \rangle$  mit beliebiger Kernel - Fkt. ersetzen  
 3) prüfe ab ob korrekt oder falsch klassifiziert,  
 erhöhe  $\alpha$  wenn falsch  
 $\Rightarrow w$  wird nicht mehr geupdated, Gewicht ist nur noch  $\alpha$  (neuer Gewicht)

$$y = \text{sgn} \sum_{i=1}^n \alpha_i \cdot y_i \cdot (x_i \cdot x)$$

"new"  $\alpha$  "new sample"  
 "label"  $y_i$  "new sample"  
 "new"  $w$  "new sample"

Prinzip:

- verschmelze Kernel - Methodik mit Perceptron - Algorithmus
- integriere hierfür die Kernel - Methodik, dass ein Klassifikator die samples  $x_i$  mit einem Gewicht assoziiert und mit für neue samples  $\text{sgn} \sum_{i=1}^n \alpha_i \cdot y_i \cdot k(x_i, x)$  berechnet, in den Perceptron - Algo
- wie? betrachte  $w$  - Vektor als Linearkombination der  $n$  training samples:  
 $w = \sum_{i=1}^n \alpha_i \cdot y_i \cdot \vec{x}_i$  ( $\alpha_i$ : Anzahl „Fehlschläge“)
- bringe  $w$  in  $y = \text{sgn}(w \cdot x)$  hinein  $\Rightarrow y = \text{sgn} \sum_{i=1}^n \alpha_i \cdot y_i \cdot (x_i \cdot x)$   
 ( $\alpha$  ist dazu da die Updates von  $w$  basierend auf altem  $w$  zu approximieren)  
 $\Rightarrow$  führe Algorithmus auf Linearkombination zurück, sodass beim Verschmelzen ein Skalarprodukt entsteht

## SVM for Regression

- zu approximierende Funktion kann wieder durch eine Menge von Parametern beschrieben werden
- neue Optimierungskriterien definieren (wie bspw. bei Klassifikation:  $y_i (\vec{w}^\top \vec{x}_i + b) \geq 1 - \xi$ )
- Wenn nichtlineare Regression gewünscht  $\rightarrow$  Kernel - Trick Transformation

## Vor- und Nachteile von SVM

- ⊕ Optimierungsproblem ist konvex  $\rightarrow$  findet optimale Hyperebene (keine lokalen Minima)  
 $\hookrightarrow$  findet damit optimale VC-Dimension
- ⊕ kann gut mit hochdimensionalen Daten umgehen
- ⊕ Kernel ist austauschbar für unterschiedliche Anwendungen
- ⊕ generell schnelle Auswertung (Test Time):
  - linear SVM: nur Skalarprodukt berechnen, prüfen ob  $w^\top x + b \geq 1$  (Hyperebene liegt berechnet vor)
  - non-linear SVM: Skalarprodukt mit (wenigen) Support Vektoren (Hyperebene liegt wegen Kernel - Trick nicht berechnet vor)
- ⊕ Klassifikation, Regression, PCA
- ⊖ finden des optimalen Kernels oftmals nicht trivial
- ⊖ Parameter für Kernels notwendig
- ⊖ Speicheraufwand / Rechenaufwand (Train Time) kann bei sehr vielen samples hoch sein (viele Skalarprodukte)
- ⊖ Anzahl Support - Vektoren abhängig von Problem, Parametern, Train Samples (Feature space)
- ⊖ Vorverarbeitung der Daten kann nicht geleistet werden (z.B. anders bei „tiefen“ Lernern, z.B. NN)