

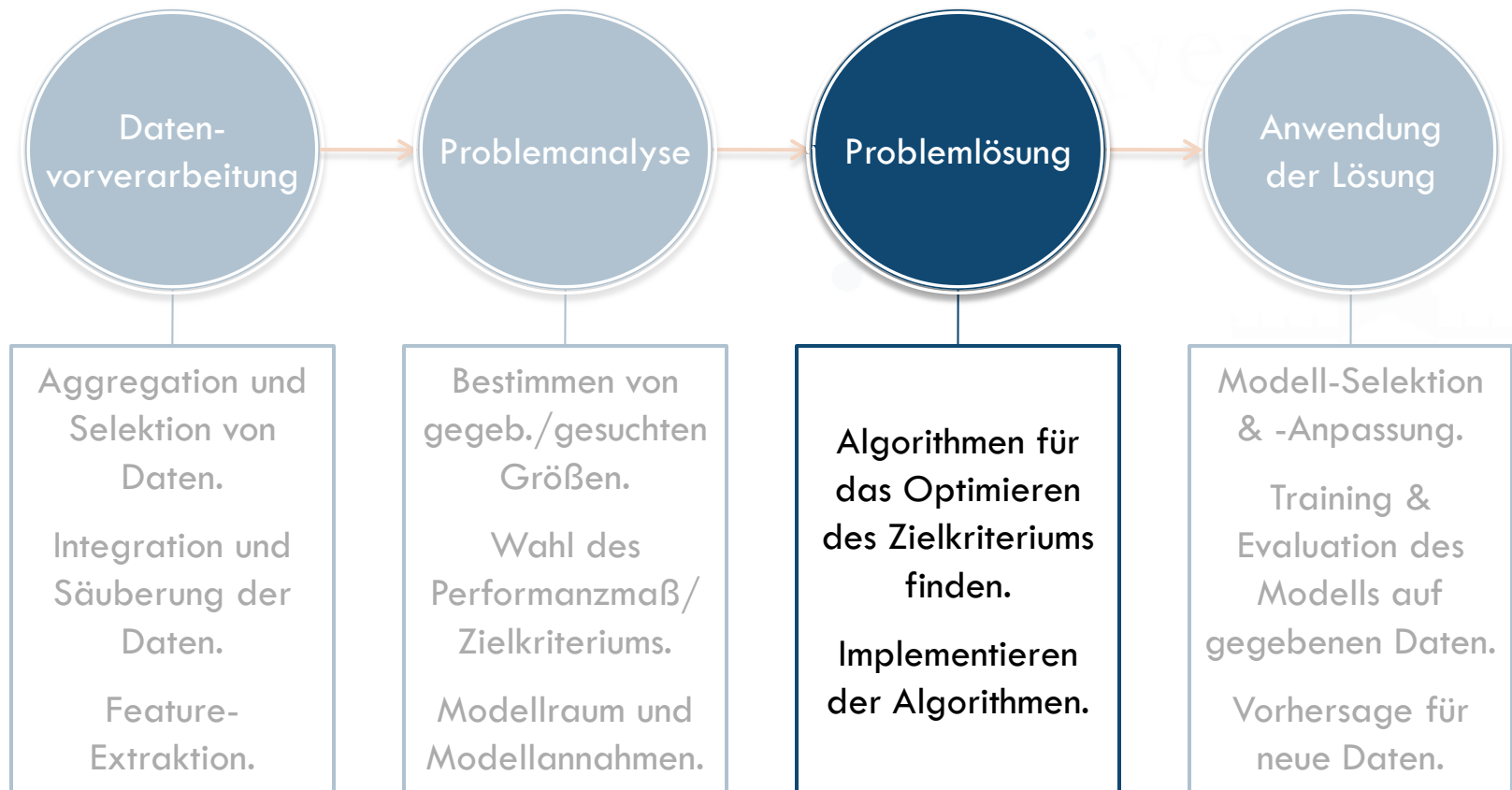


INTELLIGENTE DATENANALYSE IN MATLAB

Überwachtes Lernen: Lineare Modelle

Überblick

□ Schritte der Datenanalyse:



Überwachtes Lernen

Problemstellung



- Gegeben: Trainingsdaten mit bekanntem Zielattributen (gelabelte Daten).
- Eingabe: Instanz (Objekt, Beispiel, Datenpunkt, Merkmalsvektor) = Vektor mit Attribut-Belegungen.
- Ausgabe: Belegung des/der Zielattribut(e).
 - Klassifikation: Nominaler Wertebereich des Zielattributs.
 - Ordinale Regression: Ordinaler Wertebereich des Zielattributs.
 - Regression: Numerischer Wertebereich des Zielattributs.
- Gesucht: Modell $f : \mathbf{x} \mapsto y$.

Überwachtes Lernen

Arten von Modellen

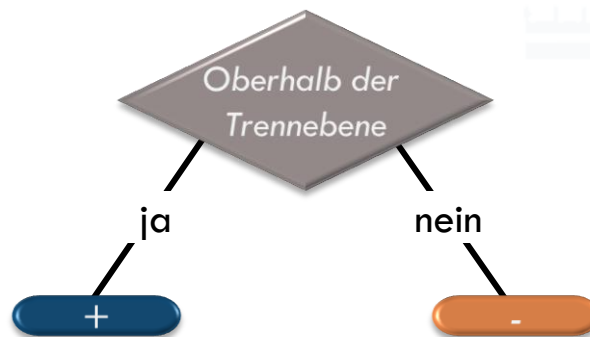
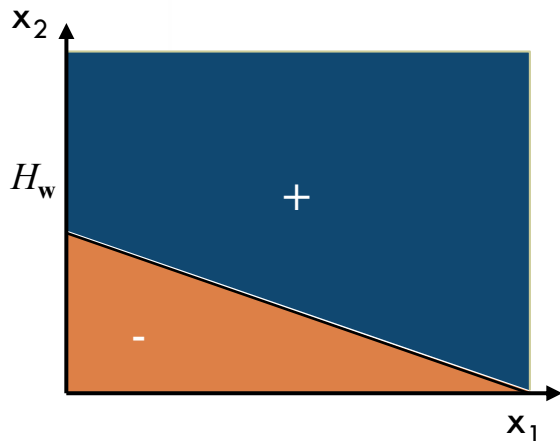


- Entscheidungsbäume/Regelsysteme:
 - Klassifikations-, Regressions-, Modellbaum.
- **Lineare Modelle:**
 - Trennebenen, Regressionsgerade.
- Nicht-lineare Modelle, linear in den Parametern:
 - Nicht-lineare Datentransformation + lineares Modell.
 - Kernel-Modell.
 - Probabilistisches Modell.
- Nicht-lineare Modelle, nicht-linear in den Parametern:
 - Neuronales Netz.

Lineare Modelle

Lineare Modelle für Klassifikation

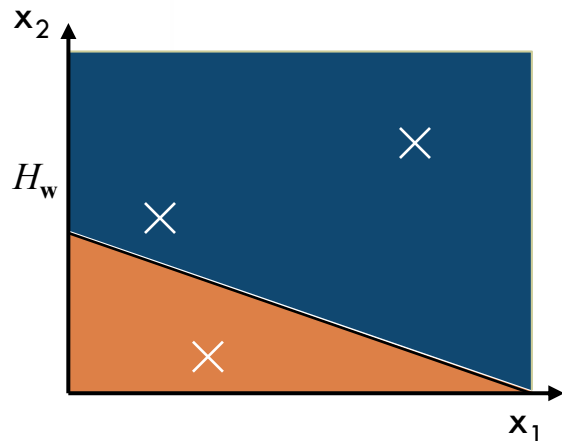
- Instanzen (Vektoren mit m Einträgen) liegen im m -dimensionalen Eingaberaum.
- Dieser wird in durch Trenngerade (Hyperebene H_w) halbiert.



Lineare Modelle

Lineare Modelle für Klassifikation

- Instanzen (Vektoren mit m Einträgen) liegen im m -dimensionalen Eingaberaum.
- Dieser wird in durch Trenngerade (Hyperebene H_w) halbiert.

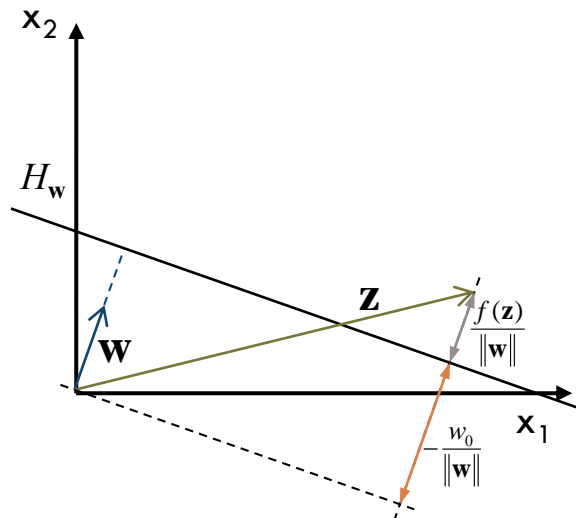


ID	x_1	x_2	y
1	0,9	3,0	+
2	3,0	4,1	+
3	1,2	0,3	-

Lineare Modelle

Lineare Modelle für Klassifikation

- Hyperebene $H_{\mathbf{w}}$ ist durch Normalenvektor \mathbf{w} und Verschiebung w_0 gegeben: $H_{\mathbf{w}} = \{\mathbf{x} \mid f(\mathbf{x}) = \mathbf{x}^T \mathbf{w} + w_0 = 0\}$
- Entscheidungsfunktion: $y(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$



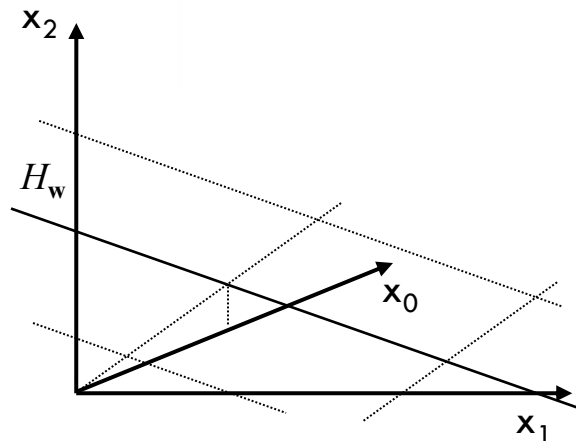
Normalenvektor \mathbf{w}
 Zu klassifizierender Punkt \mathbf{z}
 Zielfunktionswert $f(\mathbf{z})$
 Verschiebung w_0

Lineare Modelle

Lineare Modelle für Klassifikation

- Umformulierung mit zusätzlichem, konstanten Eingabeattribut $x_0=1$:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w} + w_0 = w_0 + [x_1 \quad x_2 \quad \dots \quad x_m] \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = \underbrace{[1 \quad x_1 \quad x_2 \quad \dots \quad x_m]}_{\text{Erweiterte Instanz}} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

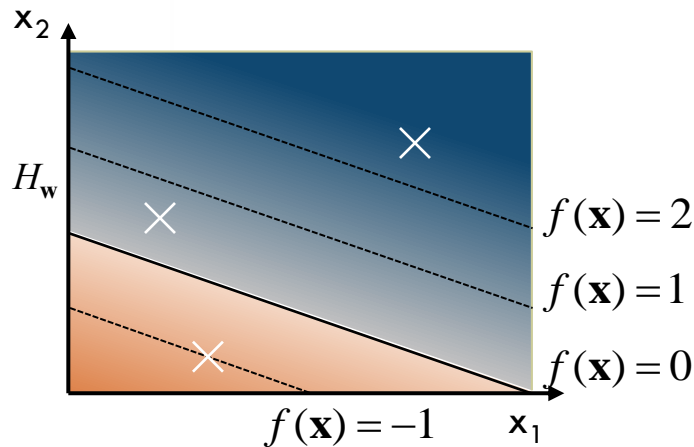


Aus Hyperebene im m -dimensionalen Eingaberaum wird Hyperebene im $(m+1)$ -dimensionalen Eingaberaum.

Lineare Modelle

Lineare Modelle für Regression

- Abstand zur Regressionsgerade (Hyperebene H_w) liefert numerischen Wert.
- Entscheidungsfunktion: $y(\mathbf{x}) = f(\mathbf{x})$



ID	x_1	x_2	y
1	0,9	3,0	0,5
2	3,0	4,1	2,5
3	1,2	0,3	-1,0

Lineare Modelle

Besonders geeignet ...

- Für kleine bis sehr große Klassifikations- & Regressionsprobleme.
- Falls unverrauschte Daten linear separierbar.
- Falls Interpretierbarkeit der Entscheidung nicht notwendig.
- Für Daten mit überwiegend numerischen Attributen.
- Falls schnelles & einfaches Verfahren gewünscht.

Lernen linearer Modelle

Problemstellung

- Gegeben: n Trainingsinstanzen \mathbf{x}_i mit Zielattribut y_i .

$$\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_n] = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} \quad \mathbf{y} = [y_1 \quad y_2 \quad \cdots \quad y_n]$$

- Gesucht: Parametervektor \mathbf{w} der Klassifikations-/Regressionsfunktion $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$.
 - Maximum Likelihood (ML)-Schätzer für Parameter.
 - Maximum A-Posteriori (MAP)-Schätzer für Parameter.

Lernen linearer Modelle

Ansatz

- ML-Schätzer = minimieren einer Verlustfunktion:

$$\begin{aligned}\mathbf{w}_{ML} &= \arg \max_{\mathbf{w}} p(\mathbf{X}, \mathbf{y} | \mathbf{w}) = \arg \max_{\mathbf{w}} \prod_{i=1}^n p(y_i | \mathbf{x}_i, \mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \sum_{i=1}^n l(f_{\mathbf{w}}(\mathbf{x}_i), y_i)\end{aligned}$$

- MAP-Schätzer = minimieren einer regularisierten Verlustfunktion:

$$\begin{aligned}\mathbf{w}_{MAP} &= \arg \max_{\mathbf{w}} p(\mathbf{w} | \mathbf{X}, \mathbf{y}) = \arg \max_{\mathbf{w}} \left(\prod_{i=1}^n p(y_i | \mathbf{x}_i, \mathbf{w}) \right) p(\mathbf{w}) \\ &= \arg \min_{\mathbf{w}} \sum_{i=1}^n l(f_{\mathbf{w}}(\mathbf{x}_i), y_i) + \Omega(\mathbf{w})\end{aligned}$$

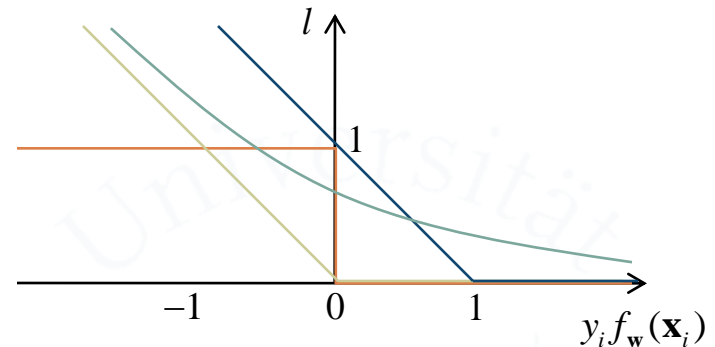
Lernen linearer Modelle

Verlustfunktion für binäre Klassifikation $y_i \in \{-1, +1\}$

□ Binary loss:

$$l_{0/1}(f_w(\mathbf{x}_i), y_i) = \begin{cases} 1 & \text{if } \text{sign}(f_w(\mathbf{x}_i)) \neq y_i \\ 0 & \text{if } \text{sign}(f_w(\mathbf{x}_i)) = y_i \end{cases}$$

$$= \begin{cases} 1 & -y_i f_w(\mathbf{x}_i) > 0 \\ 0 & -y_i f_w(\mathbf{x}_i) \leq 0 \end{cases}$$



□ Perceptron loss:

$$l_p(f_w(\mathbf{x}_i), y_i) = \begin{cases} -y_i f_w(\mathbf{x}_i) & -y_i f_w(\mathbf{x}_i) > 0 \\ 0 & -y_i f_w(\mathbf{x}_i) \leq 0 \end{cases} = \max(0, -y_i f_w(\mathbf{x}_i))$$

□ Hinge loss:

$$l_h(f_w(\mathbf{x}_i), y_i) = \begin{cases} 1 - y_i f_w(\mathbf{x}_i) & 1 - y_i f_w(\mathbf{x}_i) > 0 \\ 0 & 1 - y_i f_w(\mathbf{x}_i) \leq 0 \end{cases} = \max(0, 1 - y_i f_w(\mathbf{x}_i))$$

Binary loss ist nicht konvex
 \Rightarrow schwer zu minimieren!

□ Logistic loss:

$$l_{\log}(f_w(\mathbf{x}_i), y_i) = \log(1 + e^{-y_i f_w(\mathbf{x}_i)})$$

Lernen linearer Modelle

Verlustfunktion für Regression

□ Absolute loss:

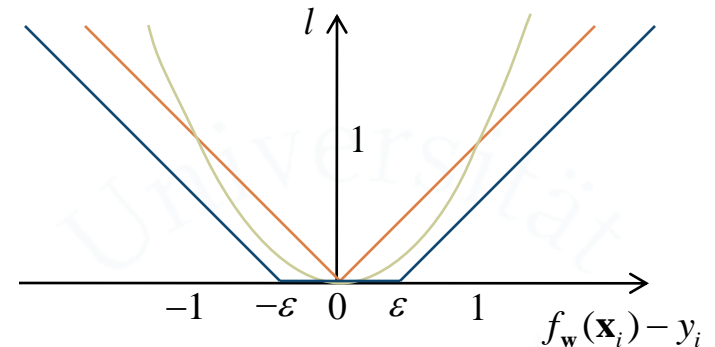
$$l_a(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = |f_{\mathbf{w}}(\mathbf{x}_i) - y_i|$$

□ Squared loss:

$$l_s(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = |f_{\mathbf{w}}(\mathbf{x}_i) - y_i|^2$$

□ ε -Insensitive loss:

$$l_{\varepsilon}(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} |f_{\mathbf{w}}(\mathbf{x}_i) - y_i| - \varepsilon & |f_{\mathbf{w}}(\mathbf{x}_i) - y_i| - \varepsilon > 0 \\ 0 & |f_{\mathbf{w}}(\mathbf{x}_i) - y_i| - \varepsilon \leq 0 \end{cases} = \max(0, |f_{\mathbf{w}}(\mathbf{x}_i) - y_i| - \varepsilon)$$



Lernen linearer Modelle

Regularisierer

- Idee: Entscheidung basierend auf so wenig wie möglich Attributen treffen.
 - Wenig Einträge im Parametervektor (Gewichte) ungleich Null:

$$\Omega_0(\mathbf{w}) \propto \|\mathbf{w}\|_0 = \text{Anzahl } w_j \neq 0$$

Ω_0 ist nicht konvex
 \Rightarrow schwer zu minimieren!

- Geringe Manhattan-Norm:

$$\Omega_1(\mathbf{w}) \propto \|\mathbf{w}\|_1 = \sum_{j=1}^m |w_j|$$

- Geringe (quadratische) euklidische Norm:

$$\Omega_2(\mathbf{w}) \propto \|\mathbf{w}\|_2^2 = \sum_{j=1}^m w_j^2$$

Lernen linearer Modelle

Lösen der Minimierungsaufgabe

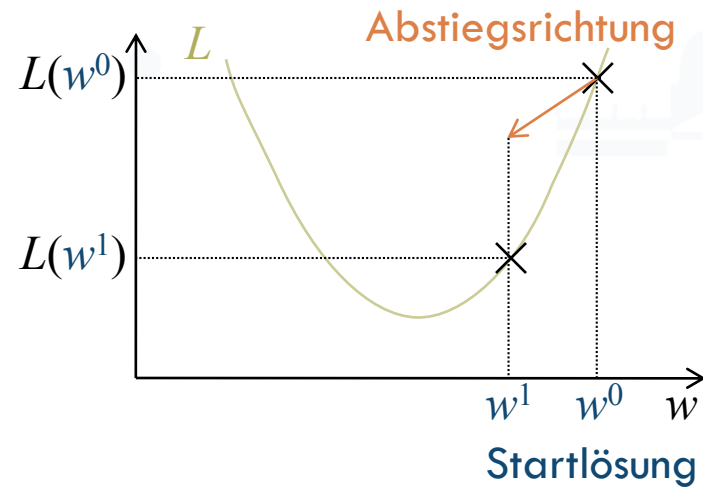
- Ziel: Minimierung von empirischen Verlust + Regularisierer: $L(\mathbf{w}) = \sum_{i=1}^n l(\mathbf{x}_i^T \mathbf{w}, y_i) + \Omega(\mathbf{w})$
- Analytische Lösung (nur selten möglich).
 - ▣ Beispiel: *Ridge Regression*.
- Numerische Lösung der primalen OA.
 - ▣ Beispiel: *Perceptron*.
- Numerische Lösung der dualen OA.
 - ▣ Beispiel: *duale Support Vector Machine (SVM)*.

Lernen linearer Modelle

Numerische Lösungsansätze



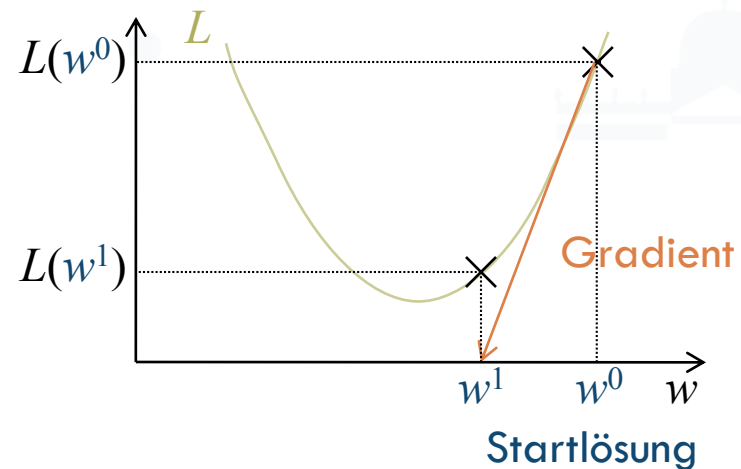
- Greedy-Suche (z.B. Abstiegsrichtung + Linesearch).



Lernen linearer Modelle

Numerische Lösungsansätze

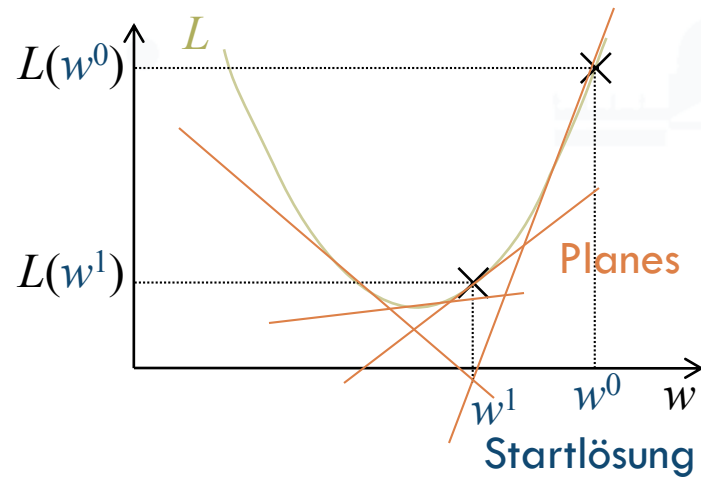
- Greedy-Suche (z.B. Abstiegsrichtung + Linearch).
□ Gradientenabstieg (z.B. Newton-Verfahren).



Lernen linearer Modelle

Numerische Lösungsansätze

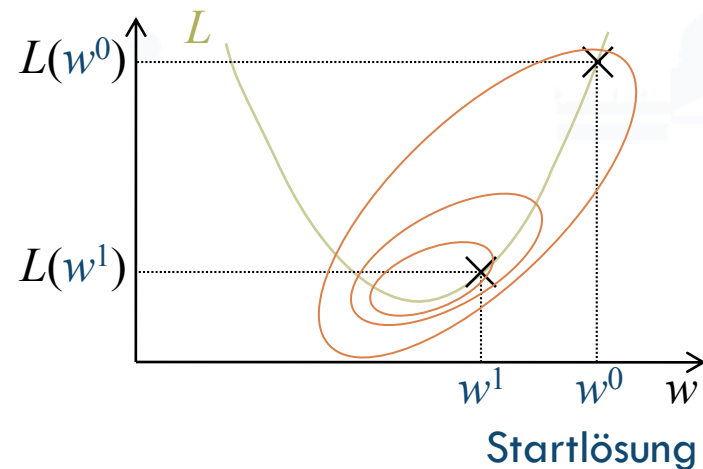
- Greedy-Suche (z.B. Abstiegsrichtung + Linesearch).
- Gradientenabstieg (z.B. Newton-Verfahren).
- Cutting-Plane-Verfahren.



Lernen linearer Modelle

Numerische Lösungsansätze

- Greedy-Suche (z.B. Abstiegsrichtung + Linesearch).
- Gradientenabstieg (z.B. Newton-Verfahren).
- Cutting-Plane-Verfahren.
- Innere-Punkt-Verfahren.



Lernen linearer Modelle

Beispiel: Ridge Regression ($y_i \in \mathbb{R}$)

□ Verlustfunktion:

$$l_s(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = |\mathbf{x}_i^T \mathbf{w} - y_i|^2$$

□ Regularisierer:

$$\Omega_2(\mathbf{w}) = \lambda \sum_{j=1}^m |w_j|^2$$

□ Analytische Lösung:

$$L(\mathbf{w}) = \sum_{i=1}^n |\mathbf{x}_i^T \mathbf{w} - y_i|^2 + \lambda \sum_{j=1}^m |w_j|^2$$

Quadratische OA ohne Nebenbedingungen

$$= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

$$= (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

$$= \mathbf{w}^T (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y}$$

Nach \mathbf{w} ableiten und Ableitung Null setzen

$$\nabla L(\mathbf{w}) = 2(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} - 2\mathbf{X}^T \mathbf{y} = 0$$

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Lernen linearer Modelle

Beispiel: Perceptron ($y_i \in \{-1, +1\}$)

- Verlustfunktion: $l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} -y_i \mathbf{x}_i^T \mathbf{w} & -y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & -y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$
- Regularisierer: 0
- Numerische Lösung des primalen Problems:

$$L(\mathbf{w}) = \sum_{i=1}^n l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i).$$

- Stochastischer Gradientenabstieg.

Gradient

$$\nabla L(\mathbf{w}) = \sum_{i=1}^n \frac{\partial l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i)}{\partial \mathbf{w}}$$

i-ter Stochastischer Gradient

$$\frac{\partial l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i)}{\partial \mathbf{w}} = \begin{cases} -y_i \mathbf{x}_i & -y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ \mathbf{0} & -y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$$

Lernen linearer Modelle

Beispiel: Perceptron ($y_i \in \{-1, +1\}$)

- Verlustfunktion: $l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} -y_i \mathbf{x}_i^T \mathbf{w} & -y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & -y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$
- Regularisierer: 0
- Algorithmus (Gradientenabstieg):

Perceptron (*Instanzen* (\mathbf{x}_i, y_i))

Setze $\mathbf{w} = \mathbf{0}$

DO

$$\mathbf{w} = \mathbf{w} - \sum_{i=1}^n \frac{\partial l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i)}{\partial \mathbf{w}}$$

WHILE \mathbf{w} verändert

RETURN \mathbf{w}

Lernen linearer Modelle

Beispiel: Perceptron ($y_i \in \{-1, +1\}$)

- Verlustfunktion: $l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} -y_i \mathbf{x}_i^T \mathbf{w} & -y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & -y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$
- Regularisierer: 0
- Algorithmus (Stochastischer Gradientenabstieg):

Perceptron (*Instanzen* (\mathbf{x}_i, y_i))

Setze $\mathbf{w} = \mathbf{0}$

DO

FOR $i = 1 \dots n$

$$\mathbf{w} = \mathbf{w} - \frac{\partial l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i)}{\partial \mathbf{w}}$$

WHILE \mathbf{w} verändert

RETURN \mathbf{w}

Lernen linearer Modelle

Beispiel: Perceptron ($y_i \in \{-1, +1\}$)

- Verlustfunktion: $l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} -y_i \mathbf{x}_i^T \mathbf{w} & -y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & -y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$
- Regularisierer: 0
- Algorithmus (Stochastischer Gradientenabstieg):

Perceptron (*Instanzen* (\mathbf{x}_i, y_i))

Setze $\mathbf{w} = \mathbf{0}$

DO

FOR $i = 1 \dots n$

$$\mathbf{w} = \mathbf{w} + \begin{cases} y_i \mathbf{x}_i & -y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & -y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$$

WHILE \mathbf{w} verändert

RETURN \mathbf{w}

Lernen linearer Modelle

Beispiel: Perceptron ($y_i \in \{-1, +1\}$)

- Verlustfunktion: $l_p(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} -y_i \mathbf{x}_i^T \mathbf{w} & -y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & -y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$
- Regularisierer: 0
- Algorithmus (Stochastischer Gradientenabstieg):

Perceptron (*Instanzen* (\mathbf{x}_i, y_i))

Setze $\mathbf{w} = \mathbf{0}$

DO

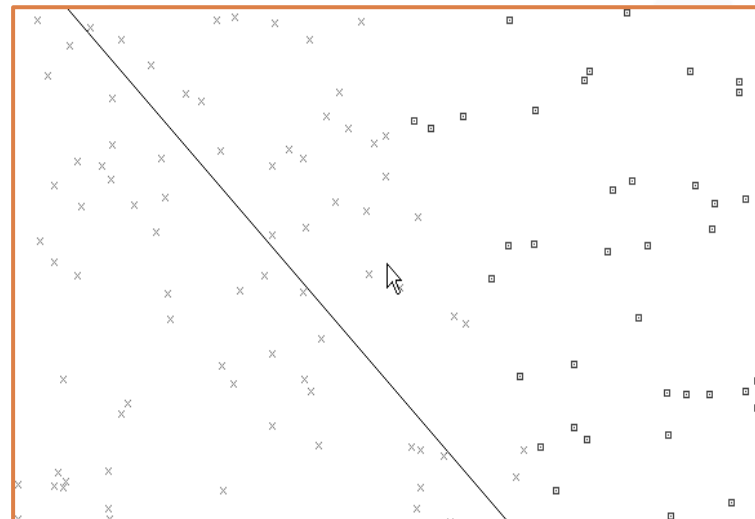
FOR $i = 1 \dots n$

IF $y_i \mathbf{x}_i^T \mathbf{w} \leq 0$ THEN

$\mathbf{w} = \mathbf{w} + y_i \mathbf{x}_i$

WHILE \mathbf{w} verändert

RETURN \mathbf{w}



Lernen linearer Modelle

Beispiel: Support Vector Machine ($y_i \in \{-1, +1\}$)

- Verlustfunktion: $l_h(f_{\mathbf{w}}(\mathbf{x}_i), y_i) = \begin{cases} 1 - y_i \mathbf{x}_i^T \mathbf{w} & 1 - y_i \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & 1 - y_i \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$
- Regularisierer: $\Omega_2(\mathbf{w}) = \frac{1}{2\lambda} \sum_{j=1}^m |w_j|^2$

- Numerische Lösung des primalen Problems

$$L(\mathbf{w}) = \sum_{i=1}^n \xi_i + \frac{1}{2\lambda} \sum_{j=1}^m |w_j|^2 \quad \text{mit } 1 - y_i \mathbf{x}_i^T \mathbf{w} \leq \xi_i, \quad 0 \leq \xi_i$$

bspw. mittels Subgradientenansatz, z.B. Pegasos.

- Lösen der dualen (quadratischen) OA

$$L(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\alpha} - \mathbf{y}^T \boldsymbol{\alpha} \quad \text{mit } \mathbf{1}^T \boldsymbol{\alpha} = 0, \quad 0 \leq y_i \alpha_i \leq \lambda$$

bspw. mittels QP-Solver, z.B. QuadProg.

Lernen linearer Modelle

Allgemein: Regularized Empirical Risk Minimization

□ Gegeben:

- Konvexe, ableitbare Verlustfunktion l mit Ableitung $l' = \frac{\partial l(z, y)}{\partial z}$.
- Konvexer, ableitbarer Regularisierer Ω mit Ableitung Ω' .

□ Algorithmus (Gradientenabstieg mit dyn. Schrittweite):

RegERM(*Instanzen* (\mathbf{x}_i, y_i))

Setze $k = 0, \mu^0 = 1, \mathbf{w}^0 = \mathbf{0}$

DO

$$\mathbf{g}^k = \sum_{i=1}^n l'(\mathbf{x}_i^T \mathbf{w}^k, y_i) \mathbf{x}_i + \Omega'(\mathbf{w}^k)$$

IF $k > 0$ THEN

$$\mu^k = \mu^{k-1} (\mathbf{g}^{k-1^T} \mathbf{g}^{k-1}) / ((\mathbf{g}^{k-1} - \mathbf{g}^k)^T \mathbf{g}^{k-1})$$

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \mu^k \mathbf{g}^k$$

$k = k + 1$

WHILE $\|\mathbf{w}^k - \mathbf{w}^{k-1}\| > \varepsilon$

RETURN \mathbf{w}^k

Barzilai-Borwein-
Verfahren

Zusammenfassung

- Lineare Modelle geeignet für sehr große Klassifikations- & Regressionsprobleme.
 - Besonders geeignet für numerische, (nahezu) linear separierbare Daten.

- Lernen von linearen Modellen:
 - Analytisch (z.B. Ridge Regression).
 - Numerisch bspw. durch Gradientenabstieg (z.B. primale SVM).

- Gefundene Lösung global optimal für konvexe Verlustfunktion & Regularisierer.