

Regression

Intelligente Systeme (SoSe 2017)

Dr.-Ing. Sabine Schumann & Jan Paul Assendorp

Stand: 13.05.2017



Gliederung

Einleitung

Lineare Regression

Batch Gradient Descent

Stochastic Gradient Descent

Locally Weighted Regression

Logistische Regression

Zusammenfassung



Gliederung

Einleitung

Lineare Regression

- Batch Gradient Descent

- Stochastic Gradient Descent

- Locally Weighted Regression

Logistische Regression

Zusammenfassung

Beispiel

*Kaufpreis einer
Immobilie in
Portland, Oregon*





Beispiel: Immobilienpreise

Dataset mit Immobilienpreisen aus Portland, Oregon

Living area (feet ²)	Price (1000\$)
2104	400
1600	330
2400	369
1416	232
3000	540
...	...

Wie lassen sich Preise für weitere Immobilien vorhersagen?



Beispiel: Immobilienpreise

Allgemeine Notation:

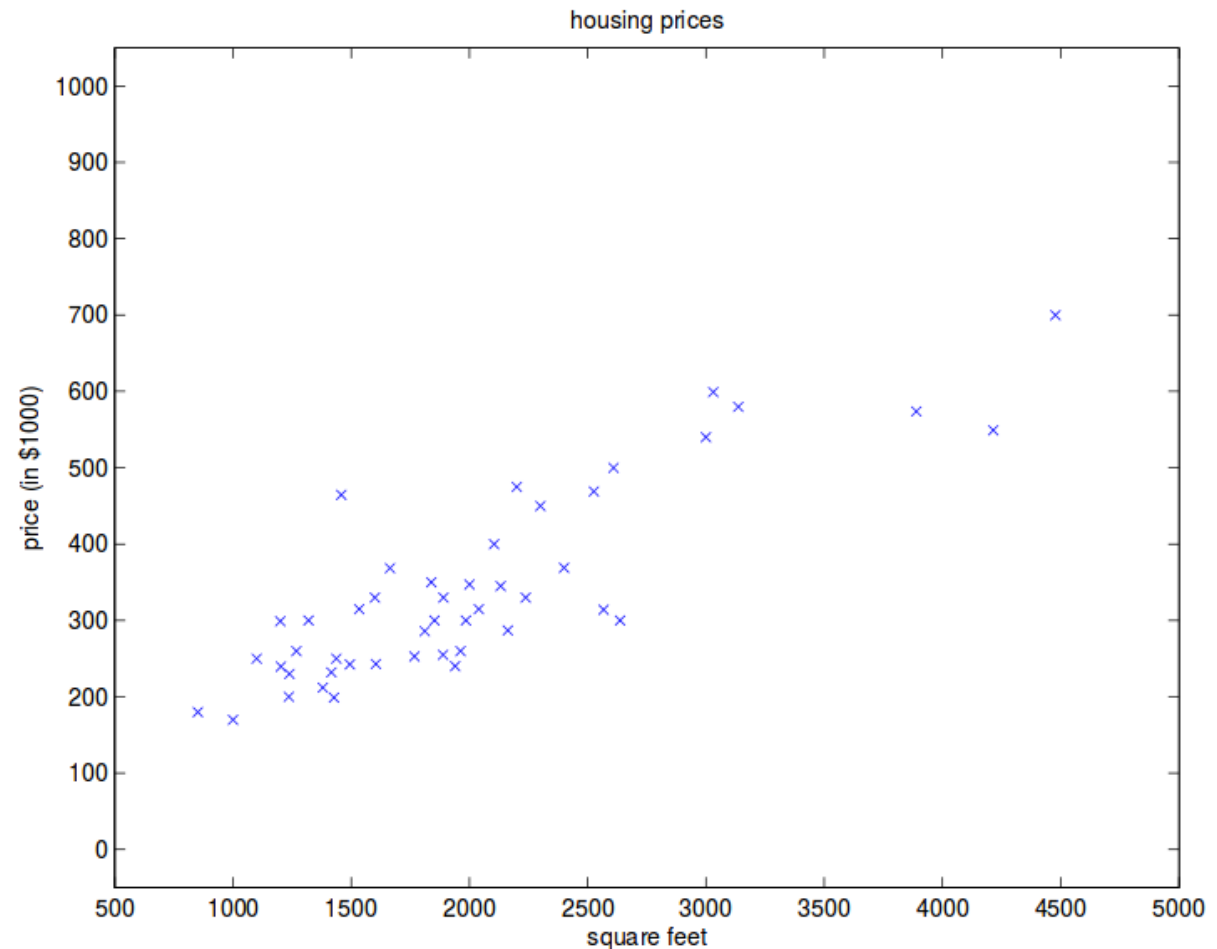
$x^{(i)}$: Input, hier Wohnraum

$y^{(i)}$: Zielvariable, hier Preis

Paar $(x^{(i)}, y^{(i)})$: Trainingsset

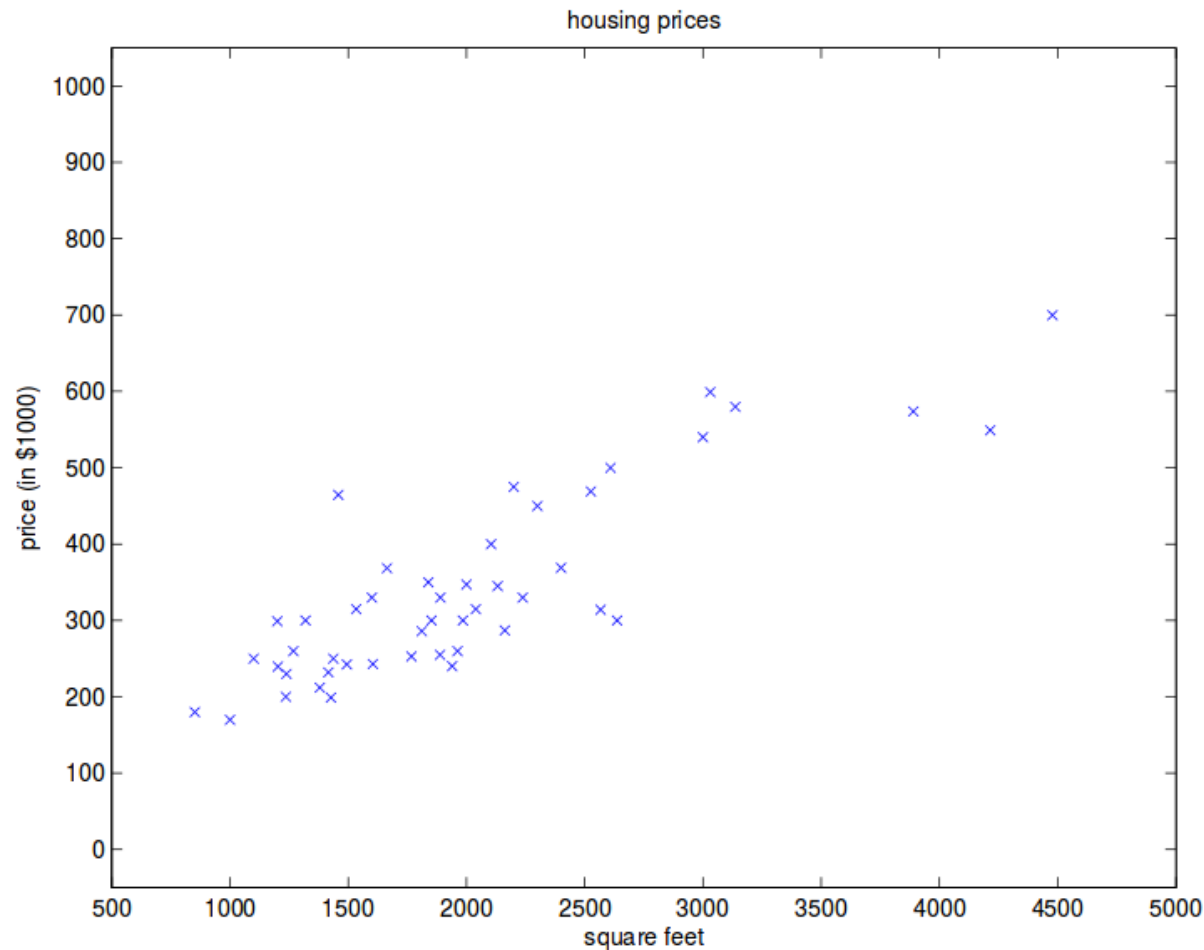
Ziel ist das Lernen einer Funktion:

$$h(x) \text{ mit } h: X \mapsto Y$$





Beispiel: Immobilienpreise



*Welche Art von
Machine-Learning-
Problem?*



Klassifikation vs. Regression

Im Allgemeinen lässt sich unterscheiden:

- Kontinuierliche Zielvariable *-> Regressions-Problem*
 $y \in \mathbb{R}$
- Zielvariable nimmt kleine Anzahl diskreter Werte oder Label an *-> Klassifikations-Problem*

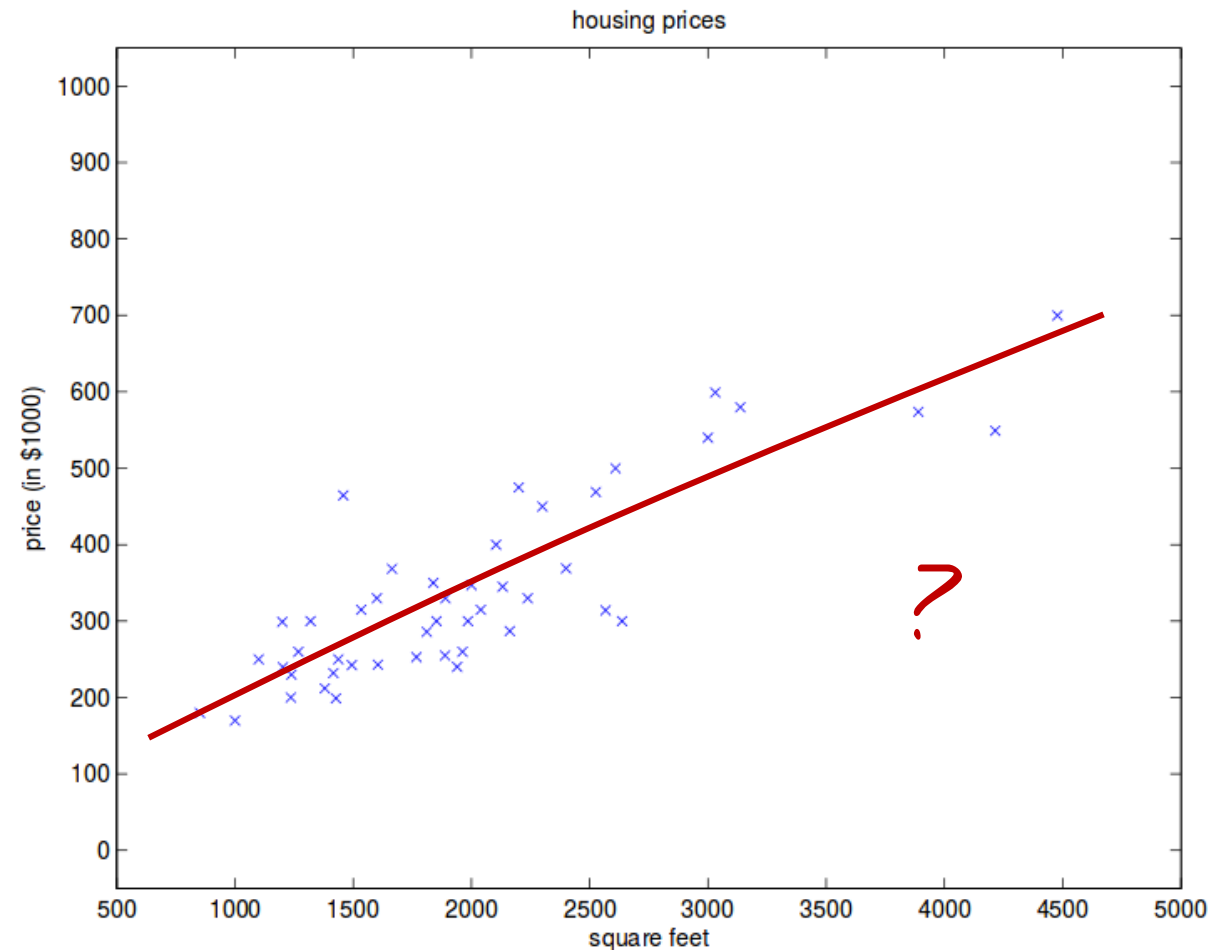
z.B. $y \in \langle gut, neutral, schlecht \rangle$



Beispiel: Immobilienpreise

Zielvariable (Preis) ist
kontinuierlich

→ Linear Regressions





Gliederung

Einleitung

Lineare Regression

Batch Gradient Descent

Stochastic Gradient Descent

Locally Weighted Regression

Logistische Regression

Zusammenfassung



Lineare Regression

Erweiterung des Beispiels: Immobilienpreise



Living area (feet ²)	# Bedrooms	Price (1000\$)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
...

x_1

x_2

y



Lösung eines supervised Learning Problems

1. Festlegen der **Input-Output**-Paare:
2. Auswahl der **Hypothese h**
3. Bestimmen einer **Error-Funktion J** zum Prüfen der Hypothese
4. Auswahl eines Algorithmus zur effektiven Suche der besten Hypothese



Lineare Regression

Definition der Hypothese h als lineare Funktion:

θ_0 : Bias Term θ_i : Parameter (auch Gewichte)

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_i x_i$$

$\theta_0 x_0$ mit $x_0 = 1$ als
Intercept Term

x_i : Ausprägungen eines Datensatzes



Lineare Regression

Definition der Hypothese h als lineare Funktion:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_i x_i$$

Damit gilt:

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x$$

-> Vereinfachung möglich durch $x_0 = 1$



Lineare Regression

Wie wird das Modell trainiert?

-> $h(x)$ annähern an y mittels Cost-Funktion

Cost-Funktion (SSE):

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

-> für alle Paare $(x^{(i)}, y^{(i)})$, wobei $x \in \mathbb{R}^n$



Lineare Regression

Erinnerung: Sum of Squared Errors (SSE)

*Faktor zur
Vereinfachung*

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Zielvariable

Annäherung

→ $J(\theta)$ misst Abweichung zwischen $h(x^{(i)})$ und $y^{(i)}$



Lösung: Closed Form

Lösung in geschlossener Form:

$$w = (X^T X)^{-1} X^T y$$

→ Geschlossene Form nur selten anwendbar

Beispiel: $m = 1,000,000, n = 100,000$

$$X^T X: \mathbb{R}^{n \times m} \times \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{n \times n}$$

*→ Matrix mit $100,000 * 100,000$ Elementen → $1 * 10^{10}$ Gleit-Kommazahlen (8 Bytes) benötigt 80 Gigabyte Hauptspeicher*



LMS Algorithmus (Least Mean Squares)

Suchalgorithmus zum finden optimaler Gewichte θ :

1. Zufällige initiale Werte für θ
2. Reduzierung der Error-Funktion $J(\theta)$ durch schrittweises Update der Gewichte θ mittels **Gradienten-Verfahren**
3. Abbruch bei **Konvergenz**



Gradienten-Verfahren (Gradient Descent)

Update der Gewichte mittels Gradient:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

*Learning Rate α : Geschwindigkeit
der Bewegung zum Optimum
(je Kleiner desto mehr Schritte)*

*Partielle Ableitung
von $J(\theta)$ nach θ_j*

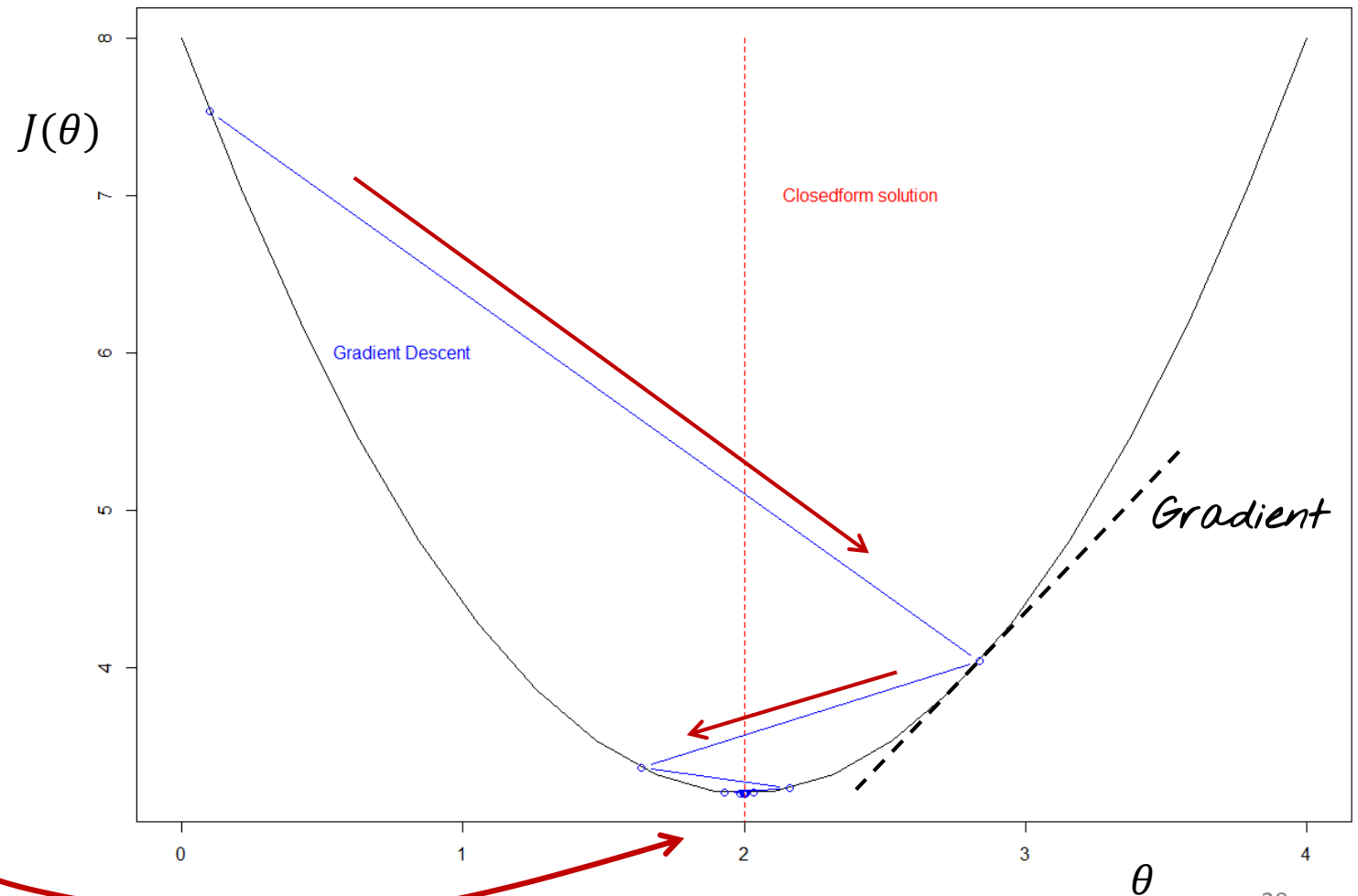


Gradienten-Verfahren (Gradient Descent)

Beispiel zu quadratischer Funktion

$$y = \frac{1}{2}(x - 2)^2 + 3,2$$

Globales Minimum $J_{\min}(\theta)$





Gradienten-Verfahren (Gradient Descent)

Update der Gewichte mittels Gradient:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

-> Verfahren simultan durchführen für alle θ_j mit $j = 0, \dots, n$



Beispiel: Gradienten-Verfahren

Gradient Descent für **einen** Datensatz (x, y) mit SSE Error-Funktion:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

*Wir suchen partielle Ableitung von $J(\theta)$
nach θ_j für einen Datensatz*

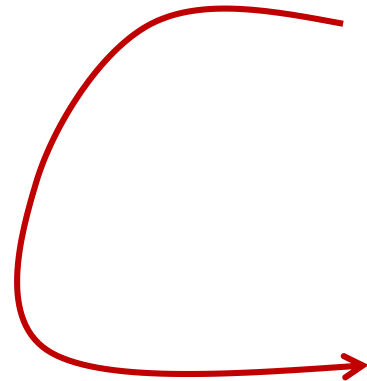


Beispiel: Gradienten-Verfahren

Gradient Descent für **einen** Datensatz (x, y) mit SSE Error-Funktion:

m Datensätze

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$




Ein Datensatz

$$J(\theta) = \frac{1}{2} (h_{\theta}(x) - y)^2$$



Beispiel: Gradienten-Verfahren

Gradient Descent für **einen** Datensatz (x, y) mit SSE Error-Funktion:

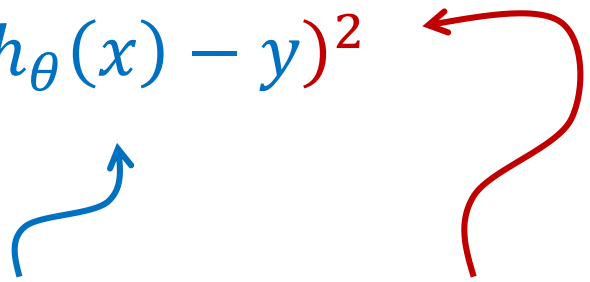

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2$$



Beispiel: Gradienten-Verfahren

Partielle Ableitung von $J(\theta)$:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2$$



$u = g(x), \quad z = f(u) = f(g(x))$

Erinnerung: Kettenregel

$$\frac{dz}{dx} = \frac{dz}{du} \frac{du}{dx} \rightarrow \text{Äußere mal innere Ableitung}$$



Beispiel: Gradienten-Verfahren

Mehrdimensionale Kettenregel: *x ist ein Vektor*

$$\frac{dz}{dx} = \frac{dz}{du} \frac{du}{dx} \quad \rightarrow \quad \frac{\partial z}{\partial x_j} = \sum_i \frac{\partial z}{\partial u_i} \frac{\partial u_i}{\partial x_j}$$

*Hier nur ein i
bzw. Datensatz*

$$\frac{\partial z}{\partial x_j} = \frac{\partial z}{\partial u} \frac{\partial u}{\partial x_j}$$



Beispiel: Gradienten-Verfahren

Kettenregel am Beispiel $J(\theta)$: $\frac{\partial z}{\partial x_j} = \frac{\partial z}{\partial u} \frac{\partial u}{\partial x_j} = \frac{\partial f(g)}{\partial g} \frac{\partial g(x)}{\partial x_j}$

$$f(g) = \frac{1}{2} g^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(x) - y)^2 = \frac{1}{2} * 2 * (h_{\theta}(x) - y) * \frac{\partial}{\partial \theta_j} (h_{\theta}(x) - y)$$

$$g(x) = h_{\theta}(x) - y$$

Vereinfachung durch Faktor



Beispiel: Gradienten-Verfahren

Partielle Ableitung der Error-Funktion:

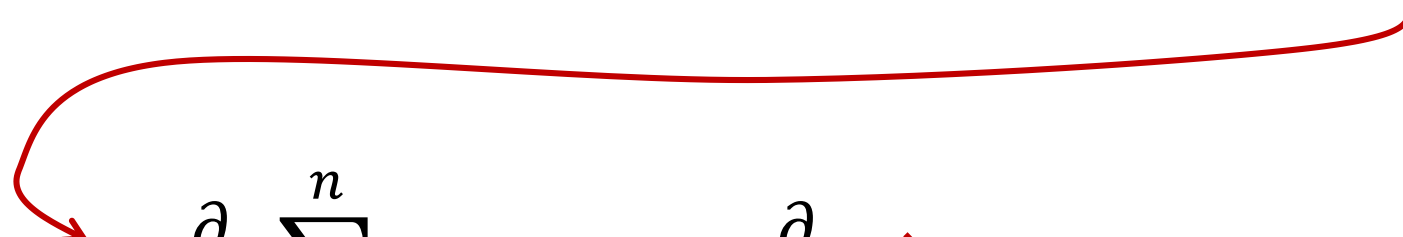
$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \cancel{\frac{1}{2}} * \cancel{2} * (h_{\theta}(x) - y) * \frac{\partial}{\partial \theta_j} (\textcolor{red}{h}_{\theta}(\textcolor{red}{x}) - y) \\ &= (h_{\theta}(x) - y) * \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{\textcolor{red}{n}} \textcolor{red}{\theta}_i \textcolor{red}{x}_i - y \right)\end{aligned}$$



Beispiel: Gradienten-Verfahren

Partielle Ableitung der Error-Funktion:

$$\frac{\partial}{\partial \theta_j} J(\theta) = (h_{\theta}(x) - y) * \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right)$$


$$\frac{\partial}{\partial \theta_j} \sum_{i=0}^n \theta_i x_i - y = \frac{\partial}{\partial \theta_j} (\cancel{\theta_0 x_0} + \dots + \theta_j x_j + \dots \cancel{\theta_n x_n}) = \frac{\partial}{\partial \theta_j} \theta_j x_j = \underline{\underline{x_j}}$$



Beispiel: Gradienten-Verfahren

Partielle Ableitung der Error-Funktion:

$$\frac{\partial}{\partial \theta_j} J(\theta) = (h_{\theta}(x) - y) * \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = (h_{\theta}(x) - y) x_j$$



Beispiel: Gradienten-Verfahren

Mit $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ $\frac{\partial}{\partial \theta_j} J(\theta) = (h_\theta(x) - y)x_j$

Gradient Descent *Partielle Ableitung von $J(\theta)$*

→ LMS (Least Mean Squares) Update Regel für einen Datensatz $(x^{(i)}, y^{(i)})$:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$



Batch Gradient Descent

Wie ist die Update-Regel für mehrere Trainingsdatensätze?

→ Batch Gradient Descent

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m \left(y^{(i)} - h_{\theta}(x^{(i)}) \right) x_j^{(i)}$$



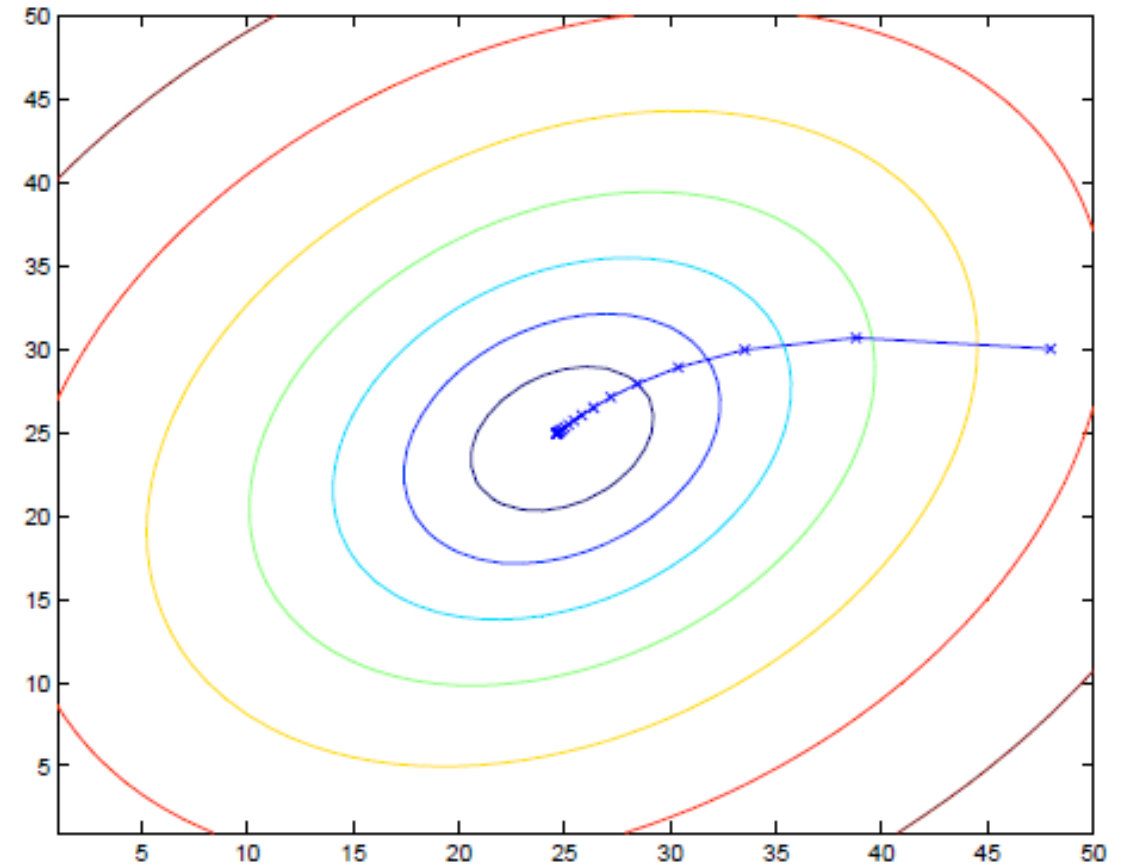
Für alle Parameter θ_j wiederholen bis Konvergenz



Visualisierung: Batch Gradient Descent

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Initialer Wert (48, 30)

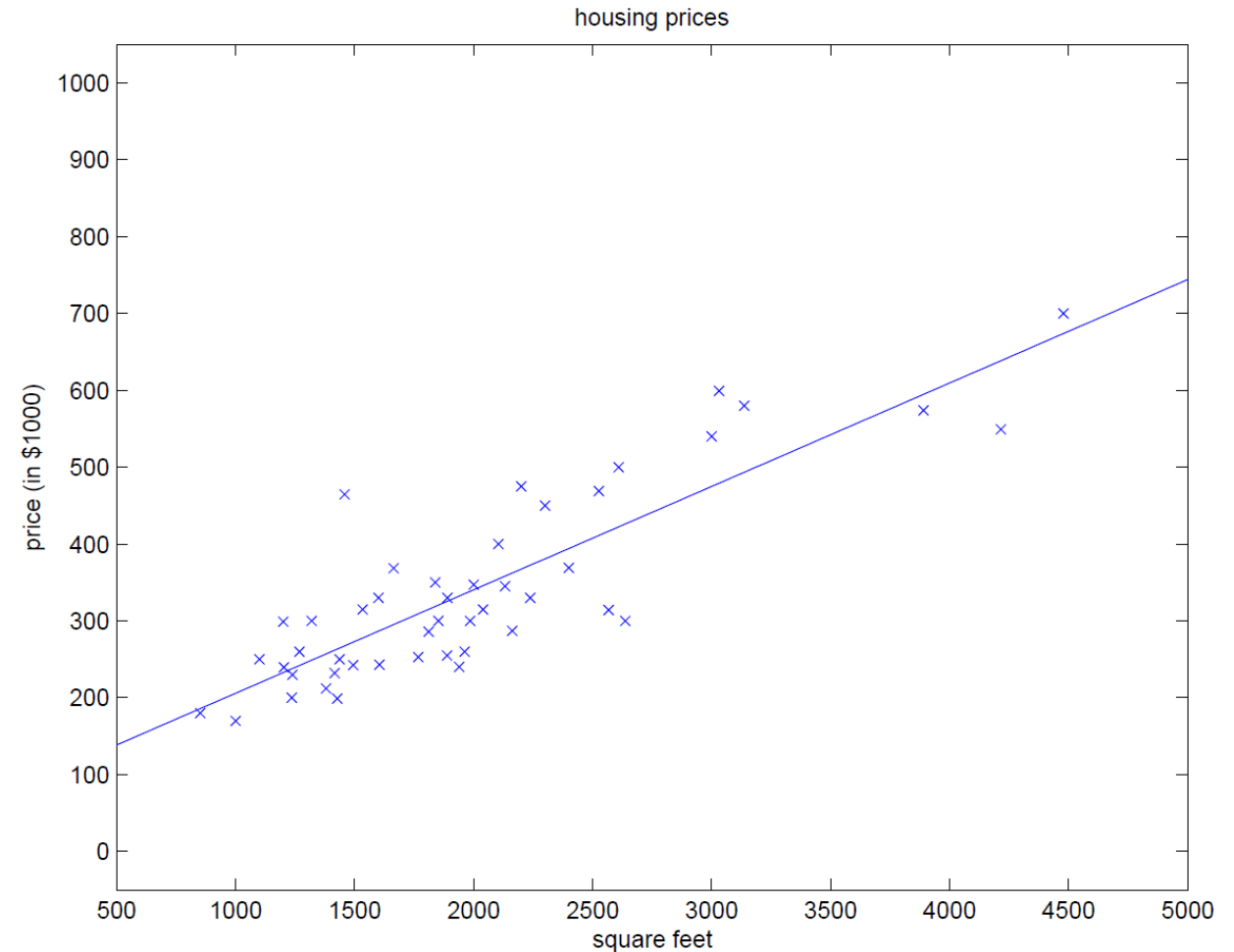




Ergebnis: Batch Gradient Descent

Approximierte Hypothese:

$$h(x) = 89.60 + 0,1392x_1 - 8,738x_2$$





Batch Gradient Descent

Loop bis Konvergenz {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m \left(y^{(i)} - h_{\theta}(x^{(i)}) \right) x_j^{(i)}, \text{ for every } j$$

}



Beispiel: Gradienten-Verfahren in

```
# Batch Gradient Descent
```

```
for (k in seq(max_iterations)) {  
  for(j in seq(n)){  
    w[j] <- w[j] + alpha * ( t(y - (X %*% w)) %*% X[,j] )  
  }  
}
```



Gliederung

Einleitung

Lineare Regression

Batch Gradient Descent

Stochastic Gradient Descent

Locally Weighted Regression

Logistische Regression

Zusammenfassung



Stochastic Gradient Descent (SGD)

- **Batch Gradient Descent** geht über das gesamte Trainingsset um Parameter θ_j zu aktualisieren
 - *langsam, wenn Datensatz RAM sprengt*
- **SGD** geht über das Trainingsset und verändert die Parameter gemäß der Gradienten des betrachteten Datensatzes i
 - *stochastisch durch Annäherung an Subset der Daten*
 - *optimiert durch mini-batches*
 - *übertritt lokales Optimum durch „noisy“ Gradient*



Stochastic Gradient Descent (SGD)

Loop bis Konvergenz {

for i = 1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_{\theta}(x^{(i)}) \right) x_j^{(i)} \quad (\text{for every } j).$$

}

}

Stochastic Gradient Descent (SGD) in



```
# Stochastic Gradient Descent (SGD)
```

```
for (k in seq(max_iterations)) {
```

```
  for (i in seq(m)) { # iterate over Datapoints
```

```
    w <- w + alpha * t((y[i] - (t(w) %*% X[i,]))) %*% X[i,])
```

```
  }
```

```
}
```




Gliederung

Einleitung

Lineare Regression

Batch Gradient Descent

Stochastic Gradient Descent

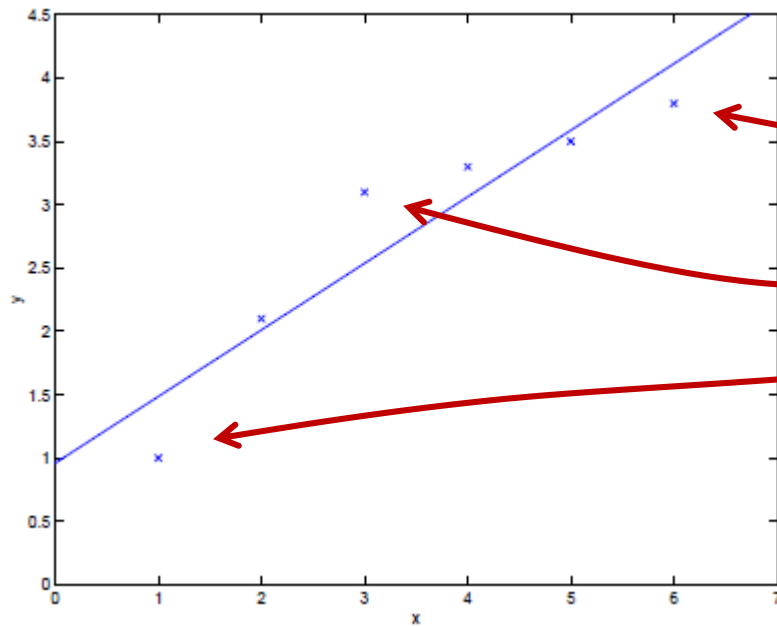
Locally Weighted Regression

Logistische Regression

Zusammenfassung



Approximation Performance

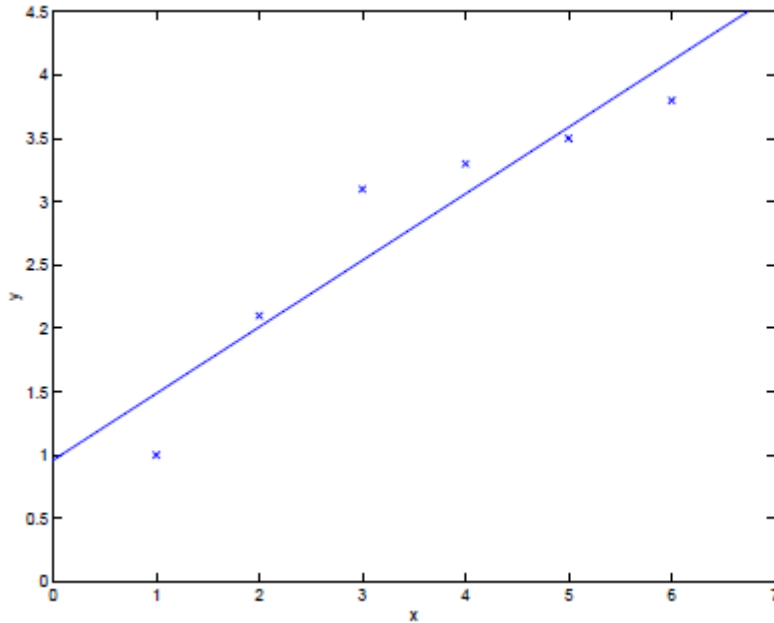


*Punkte nicht optimal
approximiert*

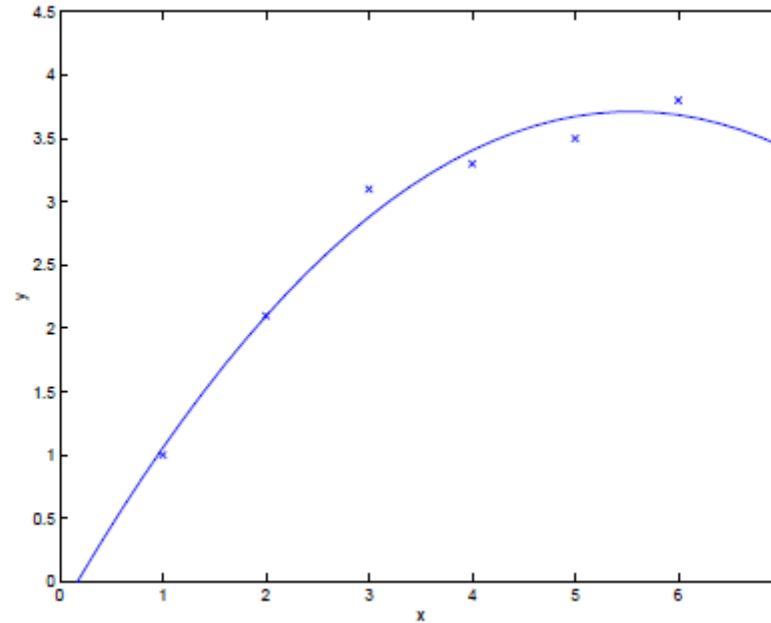
$$y = \theta_0 + \theta_1 x$$



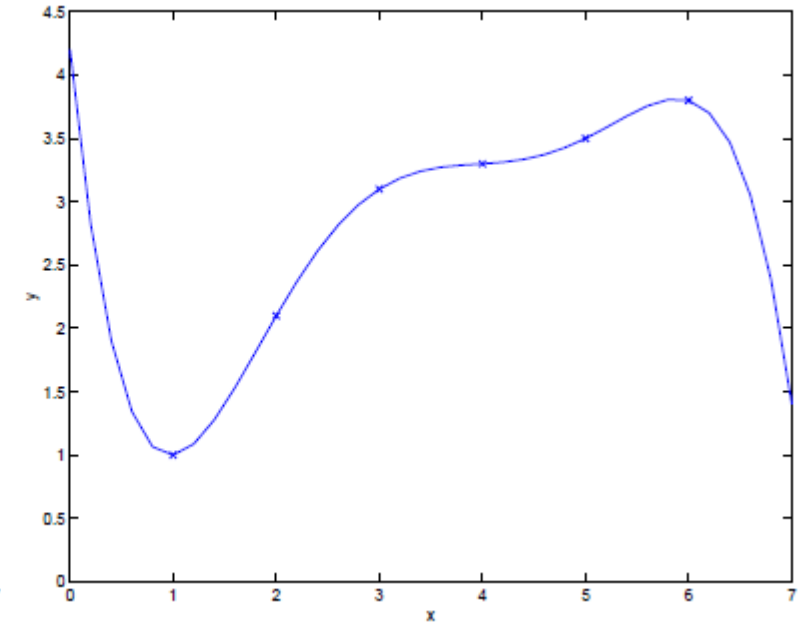
Approximation Performance



$$y = \theta_0 + \theta_1 x$$



$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$



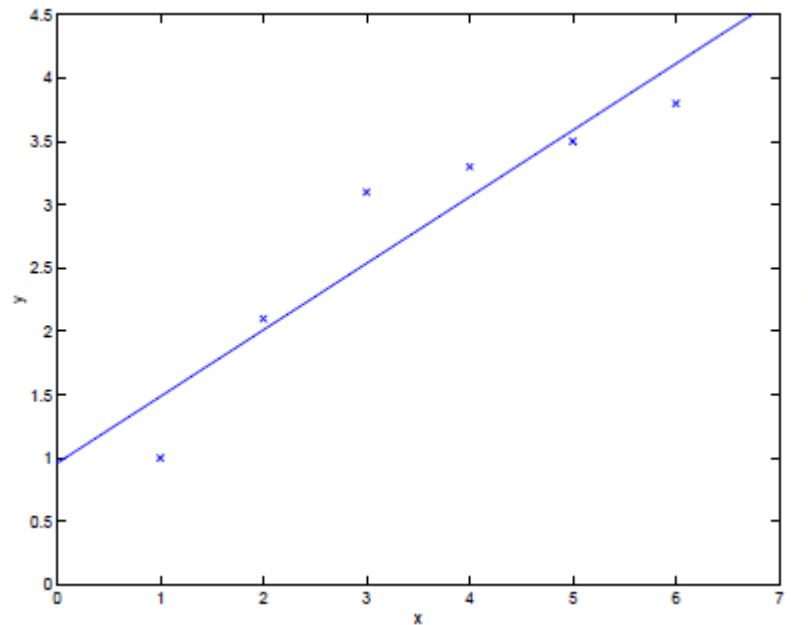
$$y = \sum_{j=0}^5 \theta_j x^j$$

→ Error Reduktion durch mehr Features

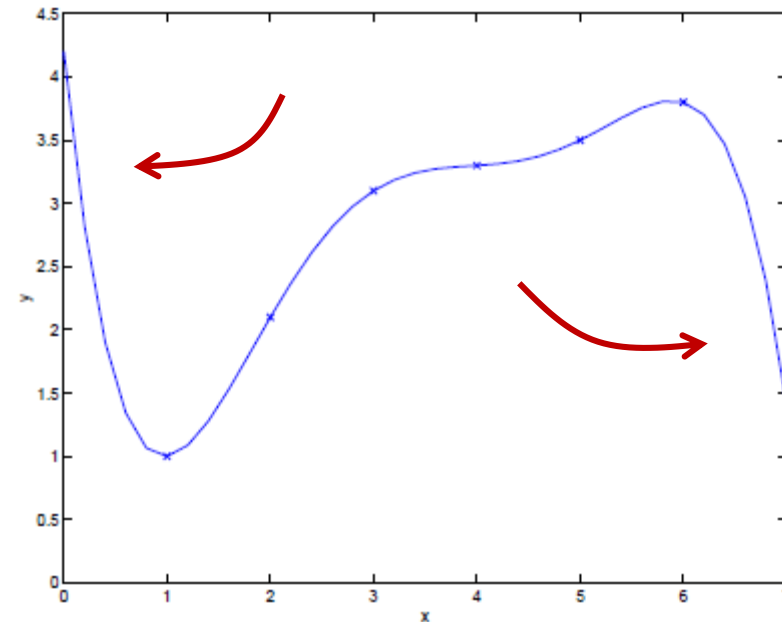


Approximation Performance

Sind mehr Feature wirklich besser?



→ Underfitting

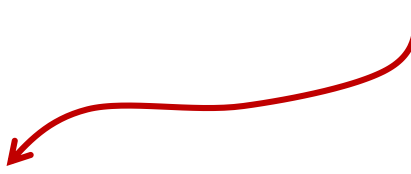


→ Overfitting



Locally Weighted Linear Regression (LWR)

- LWR Algorithmus wählt passende Features aus
→ *Voraussetzung: Genügend Trainingsdaten*
- Auswahl von θ , sodass $J(\theta)$ minimal wird mit $w^{(i)}$ als nicht-negative Gewichte mit

$$J(\theta) = \sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$$




Locally Weighted Linear Regression (LWR)

- Gewichte $w^{(i)}$ mit τ als Bandbreite, typisch wie folgt:

$$w^{(i)} = e^{\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)}$$

- Bandbreite definiert das Abfallen des Gewichtes $w^{(i)}$ bei größerer Differenz $|x^{(i)} - x|$, wobei x der aktuell betrachteten Datensatz ist

→ Trainingsdaten zum Zeitpunkt der Berechnung neuer Punkte nötig



Gliederung

Einleitung

Lineare Regression

Batch Gradient Descent

Stochastic Gradient Descent

Locally Weighted Regression

Logistische Regression

Zusammenfassung



Anwendungsbeispiel

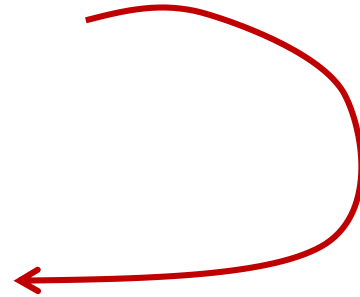


Binäre Klassifikation von Spam-E-mails:

0: negative Klasse

1: positive Klasse

→ Zielvariable: $y \in \{0, 1\}$



*Zielvariable ist
Wahrscheinlichkeit
für Klasse Spam*



Logistische Regression

Hypothese

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

mit Sigmoid-Funktion

$$g(z) = \frac{1}{1 + e^{-z}}$$



Logistische Regression

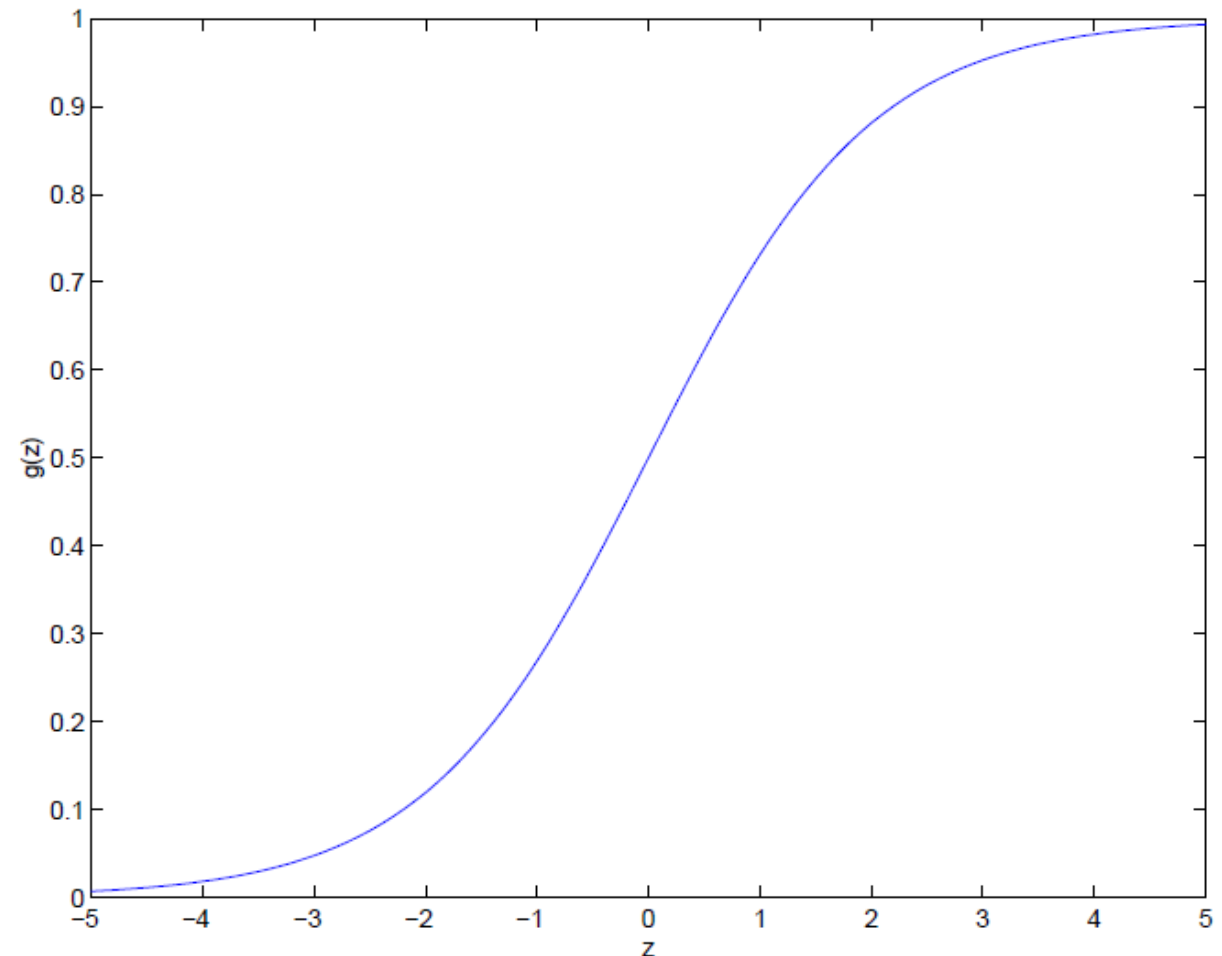
Sigmoid-Funktion

→ *logistische Funktion*

$$z \rightarrow \infty = 1$$

$$z \rightarrow -\infty = 0$$

→ Einfach differenzierbar





Logistische Regression

Ableitung der Sigmoid-Funktion

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = \frac{d}{dz} \frac{1}{1 + e^{-z}}$$

$$= \frac{1}{(1 + e^{-z})^2} (e^{-z})$$

$$= \frac{e^{-z}}{(1 + e^{-z})^2}$$

Kettenregel $\frac{df(u)}{dz} = \frac{df}{du} \frac{du}{dz}$

$$f(u) = u^{-1} \rightarrow \frac{df(u)}{du} = -u^{-2}$$

$$u(z) = 1 + e^{-z} \rightarrow \frac{du(z)}{dz} = -e^{-z}$$



Logistische Regression

Ableitung der Sigmoid-Funktion

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = \frac{e^{-z} + 1 - 1}{(1 + e^{-z})^2} \quad \rightarrow \text{Erweitern mit Faktor (Summe 0)}$$

$$= \frac{1 + e^{-z}}{(1 + e^{-z})^2} - \frac{1}{(1 + e^{-z})^2}$$

$$= \frac{1}{1 + e^{-z}} - \frac{1}{(1 + e^{-z})^2} = \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}} \right)$$



Logistische Regression

Ableitung der Sigmoid-Funktion

$$g(z) = \frac{1}{1 + e^{-z}}$$

mit $g(z)$

$$g'(z) = \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}} \right)$$

$$g'(z) = g(z)(1 - g(z))$$

Ableitung der logistischen Funktion



Logistische Regression

Annahmen bzgl. der Wahrscheinlichkeiten:

1: positive Klasse

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

0: negative Klasse

$$P(y = 0 \mid x; \theta) = 1 - h_{\theta}(x)$$

*Wahrscheinlichkeit von y unter der Bedingung x ,
parametrisiert durch θ (Keine Zufalls-Variable)*

Zusammengefasst:

$$p(y \mid x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$



Logistische Regression

Likelihood L der Parameter θ

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^m \left(h_{\theta}(x^{(i)}) \right)^{y^{(i)}} \left(1 - h_{\theta}(x^{(i)}) \right)^{1-y^{(i)}} \end{aligned}$$

Likelihood als Produkt der Wahrscheinlichkeiten über alle m Datensätze





Logistische Regression

Anstatt **Likelihood** $L(\theta)$ ist es einfacher die **Log-Likelihood** $l(\theta)$ zu maximieren:

$$l(\theta) = \log L(\theta) = \log \prod_{i=1}^m \left(h_{\theta}(x^{(i)}) \right)^{y^{(i)}} \left(1 - h_{\theta}(x^{(i)}) \right)^{1-y^{(i)}}$$

$$= \sum_{i=1}^m \log \left(h_{\theta}(x^{(i)}) \right)^{y^{(i)}} \left(1 - h_{\theta}(x^{(i)}) \right)^{1-y^{(i)}}$$

*-> mit $\log(a * b * \dots) = \log(a) + \log(b) + \dots$*



Logistische Regression

Log-Likelihood $l(\theta)$: \rightarrow mit $\log(a * b * \dots) = \log(a) + \log(b) + \dots$

$$\begin{aligned} l(\theta) &= \sum_{i=1}^m \log \left(h_{\theta}(x^{(i)}) \right)^{y^{(i)}} + \log \left(1 - h_{\theta}(x^{(i)}) \right)^{1-y^{(i)}} \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log \left(1 - h(x^{(i)}) \right) \end{aligned}$$

\rightarrow mit $\log(a^b) = b * \log(a)$



Logistische Regression

Wie lässt sich die Log-Likelihood **maximieren**?

- Updates der Parameter: $\theta := \theta + \alpha \nabla_{\theta} l(\theta)$ *(Gradient Ascent)*



- Berechnen des Gradienten $\frac{\partial}{\partial \theta_j} l(\theta)$

Alternative: minimieren der negativen Log-Likelihood



Logistische Regression

Berechnung des Gradienten der Log-Likelihood:

$$\frac{\partial}{\partial \theta_j} l(\theta) = \frac{\partial}{\partial \theta_j} \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log (1 - h(x^{(i)}))$$

zunächst nur für einen Datensatz (x, y)

$$\frac{\partial}{\partial \theta_j} l(\theta) = \frac{\partial}{\partial \theta_j} (y \log h(x) + (1 - y) \log(1 - h(x)))$$



Logistische Regression

Berechnung des Gradienten der Log-Likelihood:

$$h_{\theta}(x) = g(\theta^T x)$$

$$\frac{\partial}{\partial \theta_j} l(\theta) = \frac{\partial}{\partial \theta_j} (y \log h_{\theta}(x) + (1 - y) \log(1 - h_{\theta}(x)))$$

Kettenregel $\rightarrow \frac{\partial f(g)}{\partial \theta_j} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial \theta_j}$

$$\frac{\partial}{\partial \theta_j} l(\theta) = \frac{\partial}{\partial g(\theta^T x)} (y \log g(\theta^T x) + (1 - y) \log(1 - g(\theta^T x))) \frac{\partial}{\partial \theta_j} g(\theta^T x)$$



Logistische Regression

Berechnung des Gradienten der Log-Likelihood:

$$\frac{\partial}{\partial \theta_j} l(\theta) = \frac{\partial}{\partial g(\theta^T x)} (y \log g(\theta^T x) + (1 - y) \log(1 - g(\theta^T x))) \frac{\partial}{\partial \theta_j} g(\theta^T x)$$

mit Ableitungsregel $\frac{d}{dx} \log(x) = \frac{1}{x}$ und $\frac{d}{dx} \log(1 - x) = -\frac{1}{1 - x}$

$$\frac{\partial}{\partial \theta_j} l(\theta) = \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x)$$



Logistische Regression

Berechnung des Gradienten der Log-Likelihood:

$$\frac{\partial}{\partial \theta_j} l(\theta) = \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x)$$

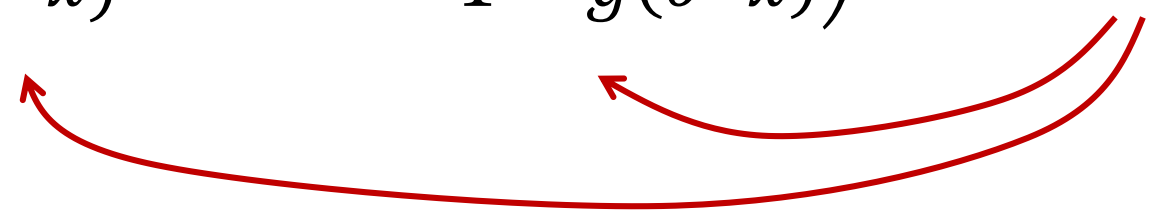
$$\frac{\partial}{\partial \theta_j} g(\theta^T x) = \frac{\partial g(z)}{\partial z} \frac{\partial z}{\partial \theta_j} = g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x$$

Erinnerung $g'(z) = g(z)(1 - g(z))$ $= x_j$



Logistische Regression

Berechnung des Gradienten der Log-Likelihood:

$$\frac{\partial}{\partial \theta_j} l(\theta) = \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x) (1 - g(\theta^T x)) x_j$$


$$\frac{\partial}{\partial \theta_j} l(\theta) = (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) * x_j$$



Logistische Regression

Berechnung des Gradienten der Log-Likelihood:

$$\frac{\partial}{\partial \theta_j} l(\theta) = (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) * x_j$$

$$\frac{\partial}{\partial \theta_j} l(\theta) = (y - y * g(\theta^T x) - g(\theta^T x) + y * g(\theta^T x)) * x_j$$

$$\frac{\partial}{\partial \theta_j} l(\theta) = (y - g(\theta^T x)) * x_j$$



Logistische Regression

Damit ergibt sich für logistische Regression die folgende Update-Regel für Stochastic Gradient Ascent (SGA):

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_{\theta}(x^{(i)}) \right) x_j^{(i)}$$

→ Die Regel ist identisch zur linearen Regression!



Regularization

- Einfache Methode zum Verhindern von **Overfitting**
- Hinzufügen von extra Bias zum Verhindern extremer Gewichte

L2-Regularization: $\frac{\lambda}{2} \|\theta\|^2 = \frac{\lambda}{2} \sum_{j=1}^m \theta_j^2$ *Bias-Term θ_0 ausgeschlossen*

→ Cost-Funktion $J(\theta) = l(\theta) = \left[\sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log (1 - h(x^{(i)})) \right] + \frac{\lambda}{2} \sum_{j=1}^n \theta_j^2$

λ steuert Balance zwischen Overfitting und Kleinen Gewichten



Logistische Regression in

```
# Sigmoid Function
sigmoid <- function(x) 1/(1+exp(-x))

# Logistic Regression Hypothesis
h <- function(x,w){
  sigmoid(t(w) %*% x)
}
```



Logistische Regression in

```
# Stochastic Gradient Descent (SGD)

for (k in seq(max_iterations)) {

  for (i in seq(m)) { # iterate over Datapoints
    w <- w + alpha * t((y[i] - h(X[i,],w)) %*% X[i,])
  }

}
```

Logistische Regression in der Praxis

Beispiel: Google's Vorhersage der Schwierigkeit der Parkplatzsuche

- Integriert in Google Maps (USA)
- Croudsourcing + Machine Learning
- Logistische Regression
 - Einfach und weit verbreitet
 - Einfluss der Feature gut interpretierbar

→ Ergebnis leicht verifizierbar





Gliederung

Einleitung

Lineare Regression

Batch Gradient Descent

Stochastic Gradient Descent

Locally Weighted Regression

Logistische Regression

Zusammenfassung



Zusammenfassung

Regression beschreibt ein einfaches Modell zur Vorhersage von Zielvariablen

- Kontinuierliche Zielvariable → *lineare Regression*
- Diskrete Zielwerte → *logistische Regression*



Regression in

```
# Example Formula
```

```
formula <- myData$y ~ myData$x1 + myData$x2 + myData$x3
```

```
# Linear Regression with Linear Model Function (lm)
```

```
fit <- lm(formula)
```

```
# Logistic Regression with General Linear Model Function (glm)
```

```
fit <- glm(formula, family = binomial("logit"))
```


Regression in



```
from sklearn import linear_model
```

```
# Create linear regression object
```

```
model = linear_model.LinearRegression()
```

```
# Training with dataset X_train and target y
```

```
model.fit(X_train,y)
```

```
# Prediction of target variables for dataset X_test
```

```
prediction = model.predict(X_test)
```



Quellen

- Andrew Ng: *Stanford CS229 Lecture Notes*:
<http://cs229.stanford.edu/notes/cs229-notes1.pdf>
- Jürgen Cleve, Uwe Lämmel: *Data Mining*, 2014
- *Scikit-Learn* Dokumentation: <http://scikit-learn.org/stable/documentation.html>
- Google Research: [Using Machine Learning to predict parking difficulty](#), 2017