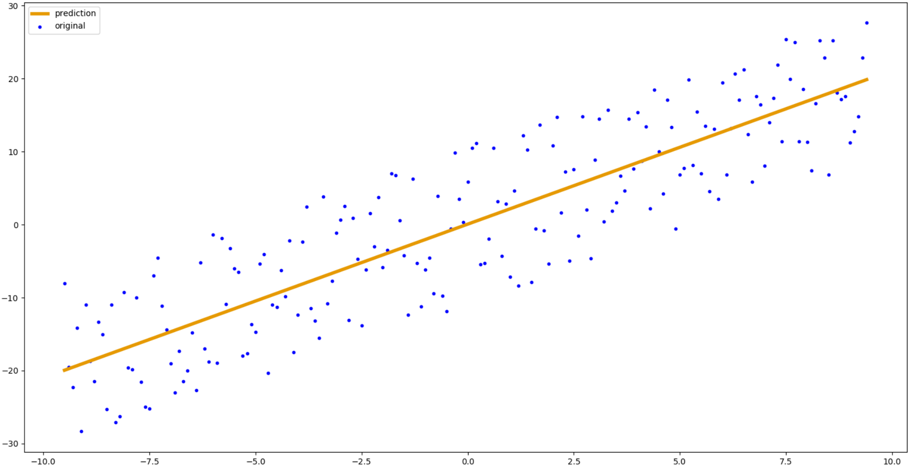
4. KI Übung: Lineare Regression und Gradient Descent

Matthias Tschöpe, Kunal Oberoi

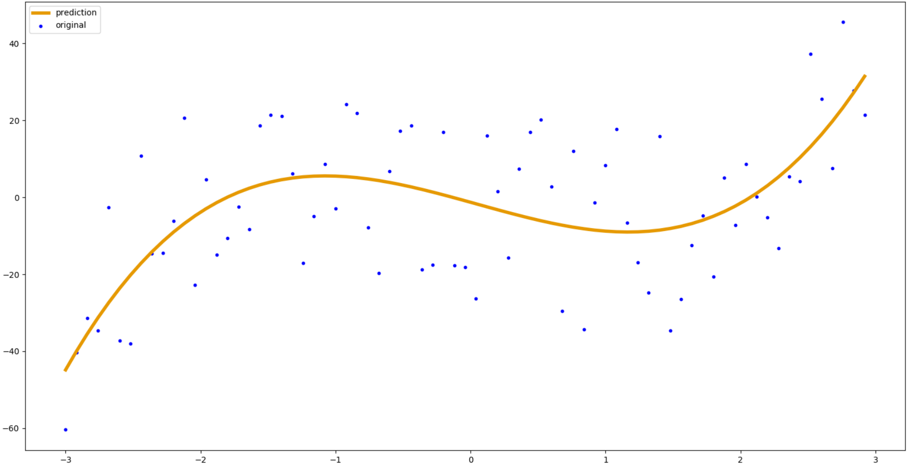
11. November 2019

Letzte Woche: Einfache lineare Regression



Heute: Lineare Regression

lineare unser Model ist eine Lineare Abbildung (wir können dennoch nicht-lineare Funktionen approximieren).



Idee der linearen Regression

Idee:

<u>Gegeben:</u>
Ein **Input-Datensatz** $\mathcal{X} := \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$, mit $x^{(i)} \in \mathbb{R}$, von Datenpunkten (oft auch **Samples** genannt) und ein **Label-Datensatz** $\mathcal{Y} := \{y^{(1)}, y^{(2)}, \dots, y^{(n)}\}$, wobei $y^{(i)} \in \mathbb{R}$ das **Label** (manchmal auch Ground Truth genannt) von $x^{(i)}$ ist.

<u>Gesucht:</u>
Eine Polynomfunktion $f : \mathbb{R} \to \mathbb{R}$ vom Grad d so, dass für alle $x^{(i)} \in \mathcal{X}$ gilt:

$$f(x^{(i)}) := \widehat{y}^{(i)} = \sum_{j=0}^{d} w_j \left(x^{(i)}\right)^j \approx y^{(i)} \tag{1}$$

und der Fehler zwischen $\widehat{y}^{(i)}$ und $y^{(i)}$ minimal ist. Das heißt, wir möchten die Faktoren w_j mit $0 \leq j \leq d$ approximieren. Als nächstes bringen wir die Gleichung (1) in Matrix-Vektor-Form.

Problemformulierung in Matrix-Vektor-Form

Dazu definieren wir zunächst:

$$X := \begin{bmatrix} 1 & x^{(1)} & \left(x^{(1)}\right)^2 & \cdots & \left(x^{(1)}\right)^d \\ 1 & x^{(2)} & \left(x^{(2)}\right)^2 & \cdots & \left(x^{(2)}\right)^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x^{(n)} & \left(x^{(n)}\right)^2 & \cdots & \left(x^{(n)}\right)^d \end{bmatrix} \quad w := \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \quad y := \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \tag{2}$$

Wenn $y^{(i)} = \widehat{y}^{(i)}$ (d.h. unsere Daten sind nicht verrauscht) und w ist optimal, dann gilt:

$$Xw = y \tag{3}$$

Dieses Problem können wir genau so lösen wie letzte Woche. Zur Erinnerung, für ein $\lambda > 0$ existiert stets die Lösung:

$$w = \left(X^T X + \lambda \cdot I_{(d+1)\times(d+1)}\right)^{-1} X^T y \tag{4}$$

1. Aufgabe

In dieser Aufgabe soll die allgemeine Form der Feature-Matrix X für Polynome d-ten Grades und die Berechnung der Funktionswerte $(\hat{y})$ implementiert werden (siehe Gleichungen 2 und 3).

(a) Erstellen Sie eine Array von Zahlen zwischen -2 und $+2$ mit der Schrittgröße 0.1 und speichern Sie diese in x ab.

(b) Implementieren Sie eine Methode `prepare_features(x_values, degree)` um X aus der Liste von x Werten zu berechnen.

(c) Berechnen Sie die Funktionswerte $(\hat{y})$, wenn

$$w := \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \tag{5}$$

Was ist der Polynomgrad d in diesem Fall?

Least Squares Method (Methode der kleinsten Quadrate)

Das Problem kann als Optimierungsaufgabe formuliert werden. y ist ein Vektor mit den Zielwerten (Labels) und $\widehat{y} =: Xw$ sind die vom Modell geschätzten Werte. Gesucht sind die Modelparameter w, die den Fehler zwischen y und $\widehat{y}$ minimieren.
Eine verbreitete Methode die Fehler zu bestimmten ist durch die mittlere quadratische Abweichung (mean squared error):

$$e =: \frac{1}{m} \sum_i (\widehat{y}_i - y_i)^2 \tag{6}$$

Es gibt andere, alternative Fehlerfunktionen (engl. Loss Function), wie z.B. die Euklidische Norm.

Aufgabe II

2. Aufgabe

In dieser Aufgabe soll die Methode $mse(y, \widehat{y})$ umgesetzt werden, so dass sie die mittlere quadratische Abweichung berechnet (Gleichung 6). Testen Sie die Methode mit

(a) $y =: [1, 1, 3, 5, 3, 2]$ und $\widehat{y} =: [1, 1, 3, 5, 3, 2]$

(b) $y =: [1, 1, 3, 5, 3, 2]$ und $\widehat{y} =: [2, 0, 3, 9, 1.4, 2.4]$

Welche Art von Optimierungsproblemen ist das?

$$\arg\min_{w \in \mathbb{R}^{d+1}} \underbrace{\frac{1}{2}\left\|y - Xw\right\|_2^2}_{=: f(w)} \tag{7}$$

(a) OP ohne Nebenbedingungen ($\rightsquigarrow$ ableiten, 0-setzen und nach w auflösen)

(b) konvex OP (da Hesse Matrix $\nabla^2 f(w)$ positiv semidefinit $\rightsquigarrow$ nachrechnen)

  (i) jedes lokale Minimum ist ein globales Minimum

  (ii) jeder stationäre Punkt w^* (d.h. $\nabla f(w^*) = 0$) ist ein globales Minimum

(c) quadratische Zielfunktion ($\rightsquigarrow f(w) = w^T X^T X w - y^T X w + \frac{1}{2} y^T y$)

Der Gradient $\nabla f(w)$ ist:

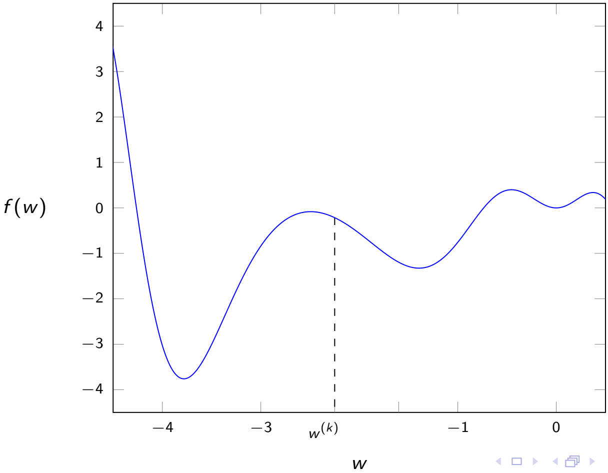
$$\nabla f(w) = X^T X w - y^T X \tag{8}$$

Kommt Ihnen Gleichung (8) bekannt vor? Jetzt fehlt noch ein Algorithmus, der stationäre Punkte findet. Dazu eignet sich z.B. **Gradient Descent**.

Gradient descent (I)

Gegeben ist die aktuell (in der $k-$ten Iteration) beste Position $w^{(k)}$. Wie finden wir die nächst (bessere) Position $w^{(k+1)}$?

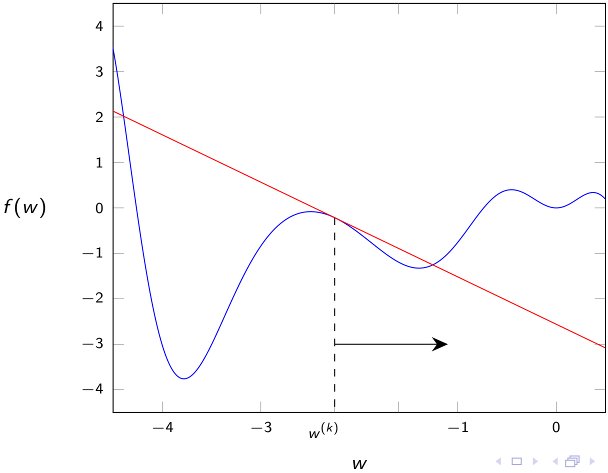
$$w^{(k+1)} := w^{(k)} \dots \tag{9}$$



Gradient descent (I)

Gegeben ist die aktuell (in der $k-$ten Iteration) beste Position $w^{(k)}$. Wie finden wir die nächst (bessere) Position $w^{(k+1)}$?

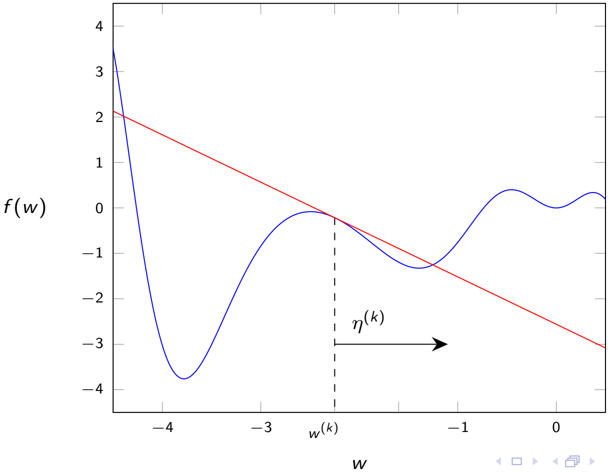
$$w^{(k+1)} := w^{(k)} \dots \tag{9}$$



Gradient descent (I)

Gegeben ist die aktuell (in der k−ten Iteration) beste Position $w^{(k)}$. Wie finden wir die nächst (bessere) Position $w^{(k+1)}$?

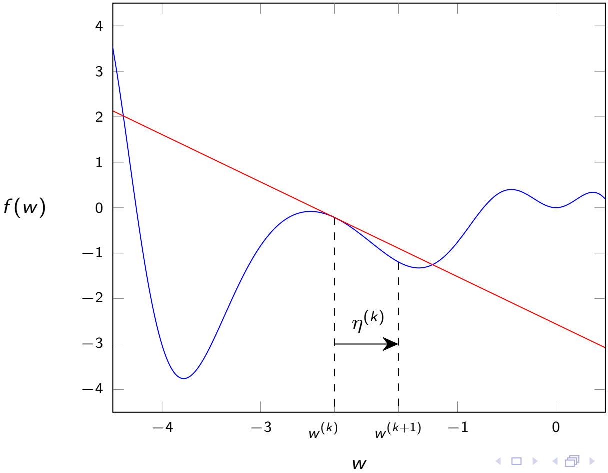
$$w^{(k+1)} := w^{(k)} - \qquad \nabla f(w^{(k)}) \tag{9}$$



Gradient descent (I)

Gegeben ist die aktuell (in der $k-$ten Iteration) beste Position $w^{(k)}$. Wie finden wir die nächst (bessere) Position $w^{(k+1)}$?

$$w^{(k+1)} := w^{(k)} - \eta^{(k)} \cdot \nabla f(w^{(k)}) \tag{9}$$



Gradient descent (II)

Algorithm 1 Gradient Descent

$\textsc{Gradient-Descent}(f, w^{(0)}, \tau, \eta^{(0)})$

Input: Zielfunktion $f \in C^1(\mathbb{R}^{d_w}, \mathbb{R})$, Startvektor $w^{(0)}$, Toleranz $\tau \geq 0$, Schrittlänge $\eta^{(0)}$.

Ouput: (fast) Stationärer Punkt $w^{(k)}$.

$k := 0$
while $\|\nabla f(w^{(k)})\|_2 > \tau$ **do**
    $w^{(k+1)} := w^{(k)} - \eta^{(k)} \cdot \frac{1}{n} \nabla f(w^{(k)})$
    $k := k + 1$
end while
return $w^{(k)}$

Gradient descent (III)

Pro:

- einfach zu verstehen
- leicht zu erweitern (Momentum, Nesterov)
- leicht zu verbessern (z.B. adaptive Lernrate)

Contra:

- langsam im Vergleich zu State-of-the-Art Optimierer

Aufgaben III

3. Aufgabe

In dieser Aufgabe soll der Gradient Descent Algorithmus implementiert und angewendet werden.

(a) Lesen Sie die Daten der Datei `XY2.csv` (oder `XY3.csv`) wie letzte Woche (mit der Datei `XY1.csv`) ein, und speichern Sie die 1. Spalte in *x_train* und die zweite in *y_train* ab.

(b) Plotten Sie die x-y Punktpaare. Welches Polynom-Grad könnte zu den Datenpunkten gut passen?

(c) Erstellen Sie die Feature-Matrix für *x_train* wie vorhin.

Aufgaben III

(d) Implementieren Sie die Methode für den Gradientenabstieg $sgd(x, y, w\_init, num\_iter, tol, lr)$, die versucht die Parameter zu optimieren. Die Methode soll entweder *num_iter* Iterationen durchführen, oder bereits früher abbrechen, wenn die Änderung der Fehler kleiner als die Toleranz (tol) ist.
Hinweis 1: Die sog. Learning-Rate (lr) beeinflusst ob und wie schnell die Methode zum Minimum konvergiert.
Hinweis 2: Gradient kann man entweder wie in Gleichung 8 oder wie folgt berechnen:

$$grad_j =: \sum_i (\widehat{y}_i - y_i) x_i^j \tag{10}$$

wobei $j = 0 ... d$.

(e) Visualisieren Sie Ihre Ergebnisse. (Input Daten vs. Geschätzte Funktion)

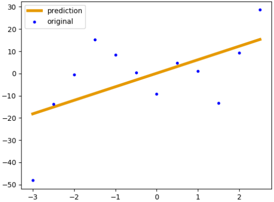
4. Aufgabe

Betrachten Sie die untenstehenden Bilder und beantworten Sie die folgenden Fragen.

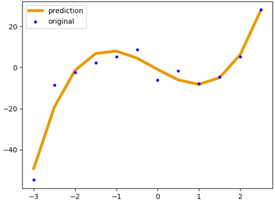
(a) Welche Funktion approximiert die gegebenen blauen Datenpunkte am besten?

(b) Welche Funktion predicted neue Datenpunkte am besten?

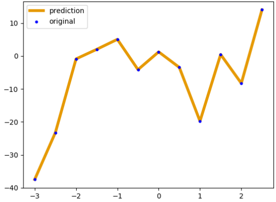
(c) Welches Bild zeigt Overfitting und welches Underfitting?



(a)



(b)



(c)

Quellen

1 Deep Learning: Das umfassende Handbuch (Ian Goodfellow et al)

2 Beyond SGD: Recent improvements of Gradient Descent Methods (Matthias Tschöpe)