Healthcare Data Analytics

Regression und Training von Modellen

Dr. Michael Strobel

25.04.2022

Inhalt

Letzte Woche

- Precision / Recall tradeoff
- Multi-Klassen Klassifikation

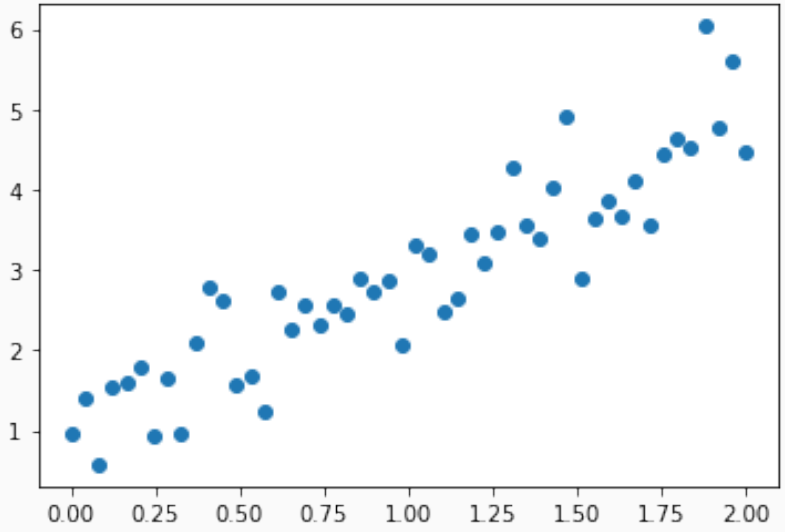
Diese Woche

- Regression
- Gradientenabstieg
- Training von Modellen

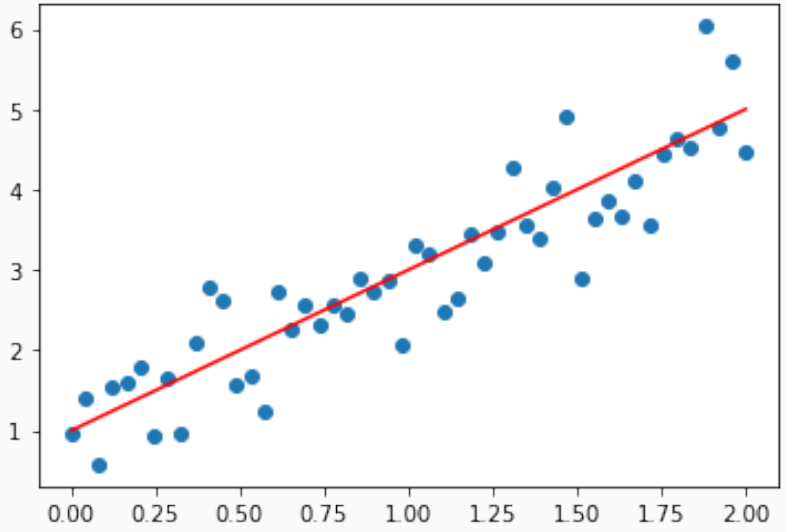
Intro

- Bis jetzt war für uns Machine Learning eine “Black-Box”
- Sie haben verschiedene Methoden benutzt wie z.B.
  - Decision Trees
  - Random Forests
  - Support Vector Machinen
  - ...
- Jetzt kümmern wir uns darum wie man diese Verfahren trainiert werden
- Beispielhaft schauen wir uns die *Lineare Regression* an

Lineare Regression - Beispiel



Lineare Regression - Beispiel



Features Verallgemeinerung

Bis jetzt haben wir angenommen, dass für jedes Feature $x_1, \ldots, x_n$ gilt dass $x_i \in \mathbb{R} \forall i$.

Ab jetzt nehmen wir an, dass jedes Feature $x_1, \ldots, x_n \in \mathbb{R}^m$ für $m \in \mathbb{N}$.

Wir definieren die *Featurematrix* $X \in \mathbb{R}^{n \times m}$ als

$$X = \begin{pmatrix} x_1^T \\ x_2^T \\ \ldots \\ x_n^T \end{pmatrix}$$

Beispiel:

$x_1 = (1, 2, 3)^T, x_2 = (4, 5, 6)^T$. Damit ist $n = 2$ und $m = 3$. Daher ist $X \in \mathbb{R}^{2 \times 3}$ und es gilt

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Lineare Regression - Definition

Gegeben seien features $x_1, \ldots, x_n \in \mathbb{R}^m$, $n, m \in \mathbb{N}$ eine target variable $y(x_1, \ldots, x_n) \in \mathbb{R}$. Wir definieren das *lineare Regressionsmodell* mit Hypothese h_α als

$$\hat{y} = \alpha_0 + \alpha_1 \cdot x_1 + ... + \alpha_N \cdot x_n$$

oder in Kurzschreibweise über die Matrix Multiplikation mit $\alpha = (\alpha_0, ..., \alpha_n)$ und

$$X = \begin{pmatrix} 1 & x_1^T \\ 1 & x_2^T \\ \ldots & \\ 1 & x_n^T \end{pmatrix}$$

$$\hat{y} = X\alpha \in \mathbb{R}$$

Damit kann nun ein Label vorhergesagt werden. Wie kommen wir jetzt an so ein α? Über Training!

Lineare Regression - Optimierungsproblem

Erinnern Sie sich an Ihre Numerik beziehungsweise Statistik Einführungvorlesung. Hier haben Sie die *lineare Ausgleichsrechnung* kennengelernt. Wenn Sie die "beste" Lösung $\hat{\alpha}$ für alle Traningsdaten erhalten wollen müssen Sie den MSE minimieren.

$$\hat{\alpha} := \min_{\alpha \in \mathbb{R}^n} \text{MSE}(\alpha) = \min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} ((X\alpha)^i - (\hat{y}_i)^i)^2$$

$$\Leftrightarrow \hat{\alpha} = \min_{\alpha \in \mathbb{R}^n} \|X\alpha - \hat{y}\|_2^2$$

Lineare Regression - Geschlossene Lösung

Wir wissen aus den genannten Einführungvorlesung, dass die geschlossene Lösung von $\hat{\alpha}$ wie folgt lautet

$$\hat{\alpha} = (X^T X)^{-1} X^T y$$

Was ist das Problem mit dieser Formel?

Lineare Regression - Geschlossene Lösung Beweis

Beweis an der Tafel

Lineare Regression - Geschlossene Lösung Praxis

Matrix $X^T X$ zu invertieren ist

- Teuer zu berechnen: naiv $O(n^3)$
- Anfällig für numerische Instabilität (hier kann SVD verwendet werden)
- Machine Learning Probleme sind meistens zu groß

Über Singulärwertzerlegung (SVD)

- SVD Lösung des Problems immer noch $O(n^2)$
- Für Machine Learning Algorithmen kann die Datenmenge immer noch zu groß werden

Daher wird in der Praxis hierfür ein *iterativer* und *approximativer* Algorithmus verwendet. Hier ist insbesondere der *Gradientenabstieg* zu nennen.

Gradientenabstieg - Idee

Gegeben sei eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$. Gesucht wird das Optimum

$$\min_{x \in \mathbb{R}^n} f(x)$$

Gradientenverfahren

Gegeben sei ein Startpunkt $x^0 \in \mathbb{R}^n$. Dann definiert sich das *Gradientenverfahren* mit der Iterationsvorschrift

$$x^{k+1} = x^k + \eta^k d^k, \quad k = 0, 1, \ldots$$

wobei $\eta^k > 0$ die *Schrittweite* bezeichnet und d^k eine *Abstiegsrichtung*. Im Kontext von Machine Learning nennt man die Schrittweite *learning rate*.

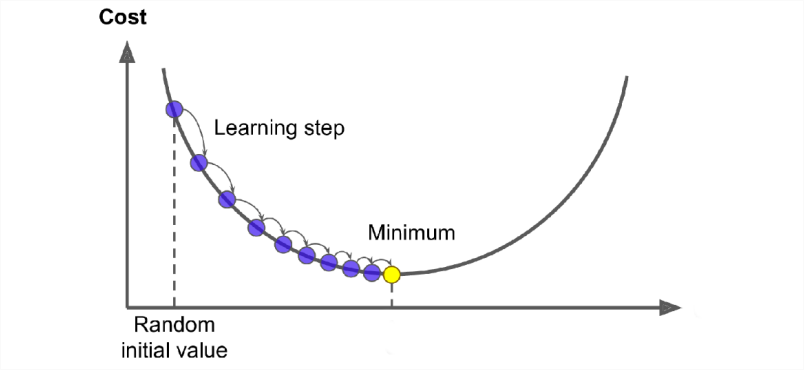
Meist wird d^k wie folgt gewählt: $d^k = -D^k \nabla f(x^k)$ wobei $D \in \mathbb{R}^{n \times n}$ wird. Für $D = I$ (Einheitsmatrix) bekommen wir das klassische Gradientenverfahren mit der *Richtung des steilsten Abstiegs*.

Gradientenabstieg

- Der Gradientenabstieg ist das meist genutzte Verfahren im Machine Learning
- Die Konvergenz, also das finden des Minimums, ist nicht trivial
- Probleme die Auftreten können
  - Schrittweite zu klein
  - Schrittweite zu groß
  - Lokale Minima / Wendepunkte
- Daten im Machine Learning sind normalerweise hochdimensional, daher ist Konvergenz des Gradientenabstiegs Forschungsthema
- Viele der Fortschritte im Machine Learning gehen auch auf Verbesserungen im Gradientenabstieg zurück

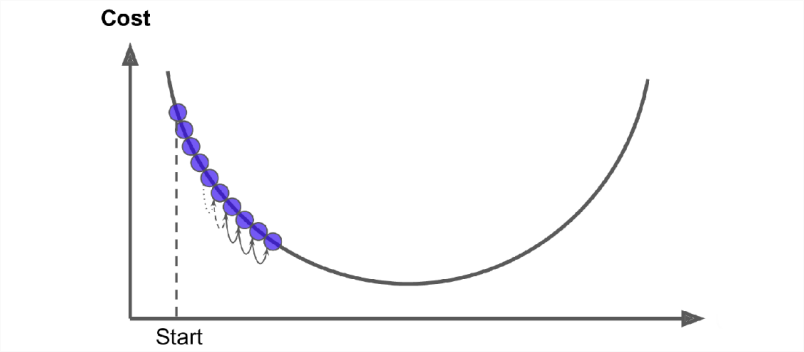
Gradientenabstieg - Visualisierung 1

Normaler Gradientenabstieg



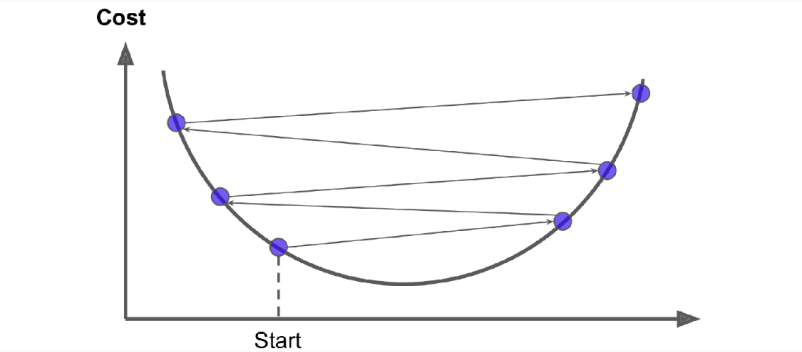
Géron, Aurélien. "Hands-on machine learning with scikit-learn and tensorflow"

Schrittweite zu gering



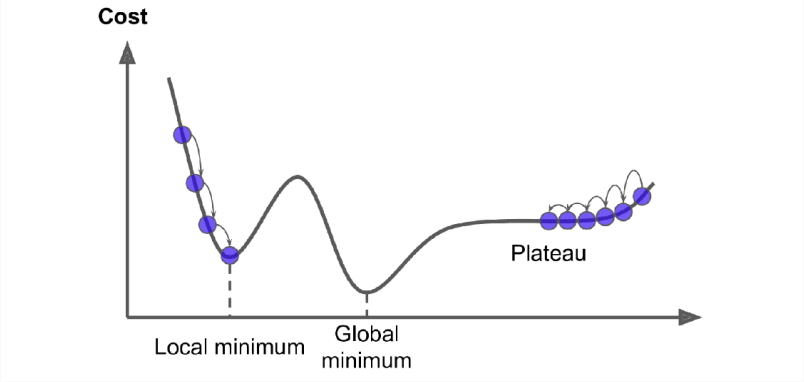
Géron, Aurélien. “Hands-on machine learning with scikit-learn and tensorflow”

Schrittweite zu hoch



Géron, Aurélien. “Hands-on machine learning with scikit-learn and tensorflow”

Lokales Minimum



Géron, Aurélien. “Hands-on machine learning with scikit-learn and tensorflow”

Gradientenabstieg - MSE

Für unser Beispiel der lineare Regression gilt dann

$$\nabla_\alpha \, \mathrm{MSE}(\alpha) = \frac{2}{m} X^T (X\alpha - y)$$

Und somit gilt für die Regression:

$$\alpha^{k+1} = \alpha^k - \eta^k \frac{2}{m} X^T (X\alpha - y), \quad k = 0, 1, \ldots$$

wobei wir α^0 z.B. mit $(1, \ldots, 1)$ initialisieren.

Wie wählt man am besten die Schritweite η^k für $k = 1, \ldots$?
Es gibt mehrere Möglichkeiten

- Konstant
- Heuristischer konstanter Wert basierenden auf Features / Beobachtungseinheiten
- Adaptiv z.B. Armijo Regel

Details können Sie bei Bedarf nachlesen z.B. in `Ulbrich, Michael, and Stefan Ulbrich. Nichtlineare Optimierung. Springer-Verlag, 2012.`

Gradientenabstieg - Verschiedene Ausprägungen

Neben der Schrittweitensteuerung ist die Auswahl der Trainingsdaten ein wichtiger Parameter, hier gibt es mehrere gängige Verfahren

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini Batch Gradient Descent

Gradientenabstieg - Batch Gradient Descent

Beim *Batch Gradient Descent* die gesamten Trainingsdaten **gesamte** auf einmal verarbeitet.

Der Gradient bei der Linearen Regression ist folgender:

$$\nabla_\alpha \text{MSE}(\alpha) = \frac{2}{m} X^T (X\alpha - y)$$

Daher wird in **jedem Schritt der gesamte Datensatz verarbeitet**.

Dies kann bei sehr großen Datensätzen sehr viel Aufwand bedeuten: Speicher- und CPU/GPU Zeit steigt linear mit der Anzahl der Beobachtungseinheiten.

Gradientenabstieg - Stochastic Gradient Descent

Beim *Stochastic Gradient Descent* wird nur **eine** Beobachtungseinheit auf einmal verarbeitet.

Speicher- und CPU/GPU Zeit sind hier **sehr gering**, aber das Konvergenzverhalten ist nicht optimal, d.h. das Optimum wird selten erreicht.

Die Features müssen **skaliert** werden, so dass alle Features in die Berechnung mit eingehen und der Abstieg nicht von wenigen Features mit großem Absolutbetrag dominiert werden.

Gradientenabstieg - Epoch / Online Learning / Out of Core

Definition: Epoch

Wir nennen das Training mit nur einem Teil der Trainingsdaten *Epoch*.

Definition: Online Learning

Wenn wir ein Modell *inkrementell* trainiert werden kann nennen wir dies *online learning*.

Definition: Out of Core

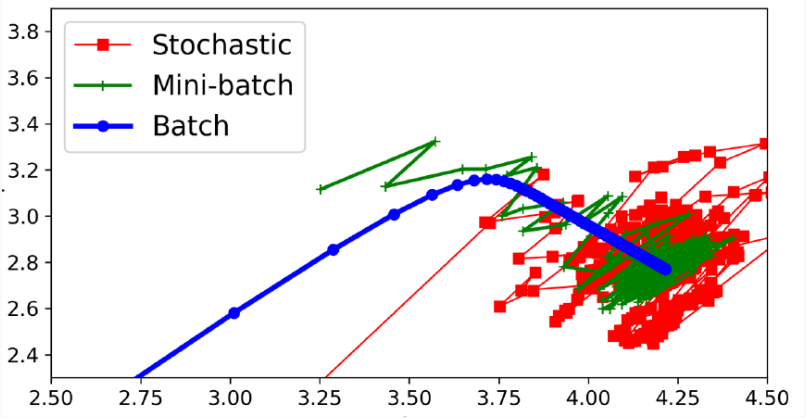
Wenn ein Modell auch trainiert werden kann, wenn der Datensatz der zum Training genutzt wird **nicht** in den Hauptspeicher des Computers geladen werden kann, nennen wir dies *out-of-core* learning.

Gradientenabstieg - Mini-Batch Gradient Descent

Kombiniert man diese Ideen erhält man den *Mini-Batch Gradient Descent*. Hier wird nur einer **Teilmenge** der Beobachtungseinheiten in jeder Epoch trainiert.

Speicher- und CPU/GPU Zeit sind hier **gering**, aber das Konvergenzverhalten liegt zwischen Stochastic Gradient Descent und Batch Gradient Descent.

Verschiedene Gradientenabstiegsmethoden im Vergleich



Géron, Aurélien. "Hands-on machine learning with scikit-learn and tensorflow"

Gradientenabstieg - Auswahl

Seien m die Anzahl der Beobachtungseinheit und n die Anzahl der Features, dann

Algorithm	Large m	Out-of-core support	Large n	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

Géron, Aurélien. "Hands-on machine learning with scikit-learn and tensorflow"

Referenzen

- Ulbrich, Michael, and Stefan Ulbrich. Nichtlineare Optimierung. Springer-Verlag, 2012.
- Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.