Bagging, Random Forest und Boosting

Seminar Maschinelles Lernen

Simone Horstmann

24. Mai 2019

Universität zu Köln

Voraussetzungen

- Mittlerer quadratischer Fehler (MSE)

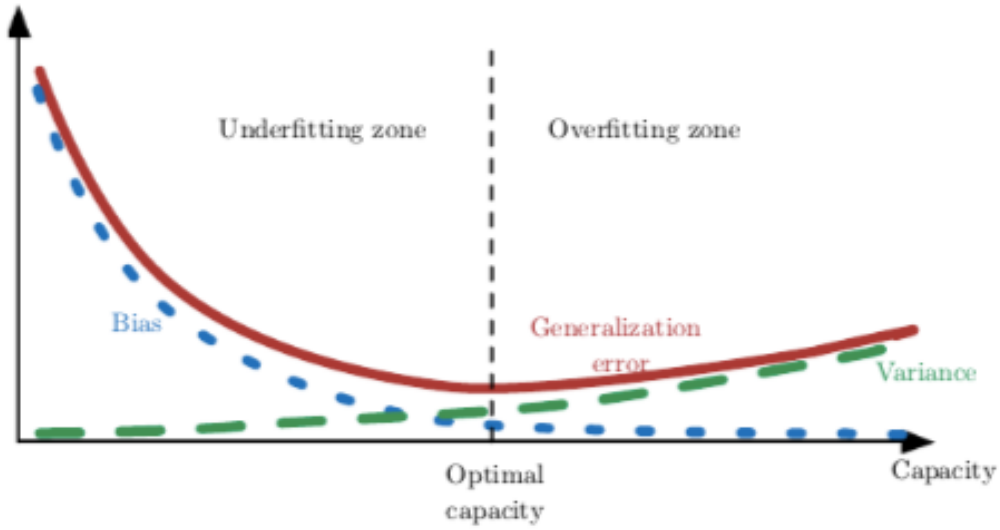
$$MSE(x_0) = Bias^2(\hat{y}_0) + Var_\tau(\hat{y}_0) + \sigma^2$$

  wobei σ^2 die Varianz des Fehlerterms darstellt
- Die Varianz ist ein Maß dafür, wie weit die Vorhersagewerte von ihrem Erwartungswert abweichen:

$$Var_\tau(\hat{y}_0) = E_\tau[\hat{y}_0 - E_\tau(\hat{y}_0)]^2$$

- Der Bias ist ein Maß für die Abweichung der Vorhersagewerte $\hat{y}_0$ von den wahren Werten der Funktion f(x):

$$Bias^2(\hat{y}_0) = [f(x_0) - E_\tau(\hat{y}_0)]^2$$



- Kapazität: Anzahl der Aspekte der Daten, welche im Modell berücksichtigt werden
- Mit steigender Kapazität passt sich das Modell an die Trainingsdaten an $\Rightarrow$ Bias sinkt $\Rightarrow$ Overfitting
- Modell mit geringerer Kapazität kann wahren Zusammenhang nicht mehr erfassen $\Rightarrow$ Varianz sinkt $\Rightarrow$ Underfitting

⇒ **'Model averaging'-Methoden** sollen Generalisierungsfehler minimieren.

- Bagging und Random Forest Methoden senken die Varianz bei gleichbleibendem Bias
- Beim Boosting wird ein Modell konstruiert, das durch höhere Kapazität den Bias senkt

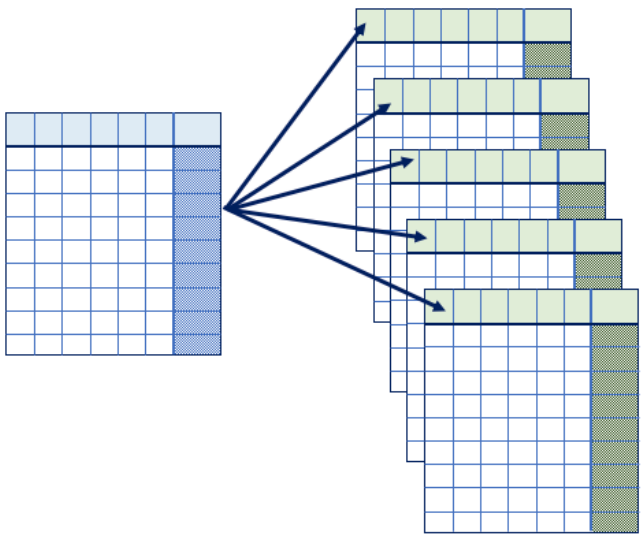
Inhaltsverzeichnis

Bagging, auch genannt Bootstrap aggregation

- erweitert ein Modell zu mehreren Modellen
- dafür werden aus dem Trainingsdatensatz sogenannte Bootstrap samples erzeugt
- auf diese wird dasselbe Modell, d.h. der gleiche Trainingsalgorithmus und die gleiche Zielfunktion angewandt

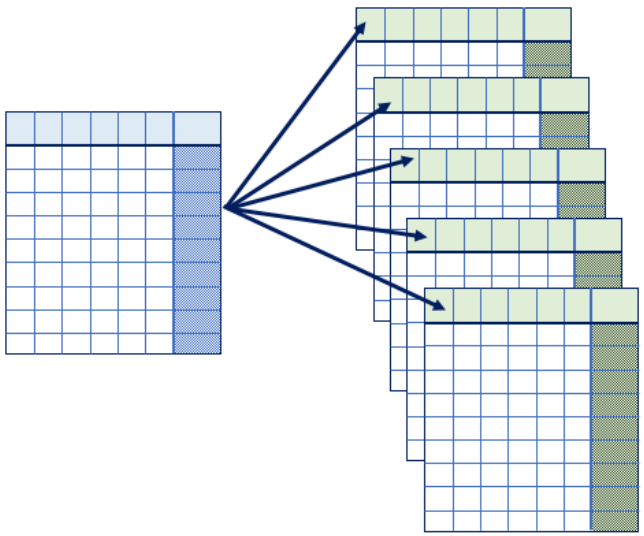
$\Rightarrow$ Vorhersage des ursprünglichen Modells $\hat{f}(x)$ wird durch die Vorhersage des Kollektivs der 'bagged' Modelle $\hat{f}_{bag}(x)$ ersetzt

Bagging - Funktionsweise



1. Aus dem Trainingsdatensatz Z werden B Bootstrap samples Z^{*b} mit b=1,2,...,B generiert
2. Für jedes Bootstrap sample wird ein Vorhersagemodell generiert, wobei immer derselbe Trainingsalgorithmus verwendet wird

Bagging - Funktionsweise



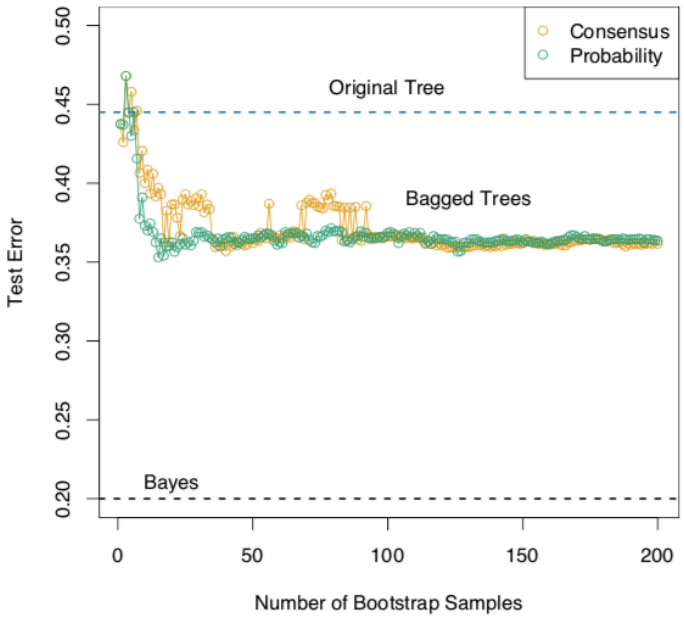
Die Vorhersage jedes Bootstrap samples $\hat{f}^{*b}(x)$ fließt mit gleichem Gewicht in die Gesamtvorhersage ein:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

Bei Klassifikationsbäumen ergibt sich die Vorhersage durch Mehrheitsvotum aller B Bäume:

$$\hat{G}_{bag}(x) = argmax_k \hat{f}_{bag}(x)$$

Bewertung der Bagging-Methode



Durch Bagging lässt sich die Varianz reduzieren.
Bei gleichbleibendem Biaswert der Bäume (gleicher Trainingsalgorithmus) reduziert sich somit der quadratische Fehler.

Besonders bei sogenannten instabilen Methoden kann Bagging zu einer Verbesserung führen:

Ein Entscheidungsbaum ist aufgrund seiner Hierachie instabil, d.h. eine kleine Veränderung der Daten kann schon eine Fehlklassifizierung bewirken.

$\rightarrow$ Durch Anwendung der Baggingmethode erhält man B Bäume, die alle auf verschiedenen Datensätzten Z^{*b} basieren

$\rightarrow$ Bäume bestehen aus unterschiedlichen Variablen und besitzen verschieden viele Knotenpunkte

$\rightarrow$ Die Bäume werden nicht alle denselben Fehler wie der ursprüngliche Baum erzeugen

$\Rightarrow$ Reduzierung von Varianz & Testfehler

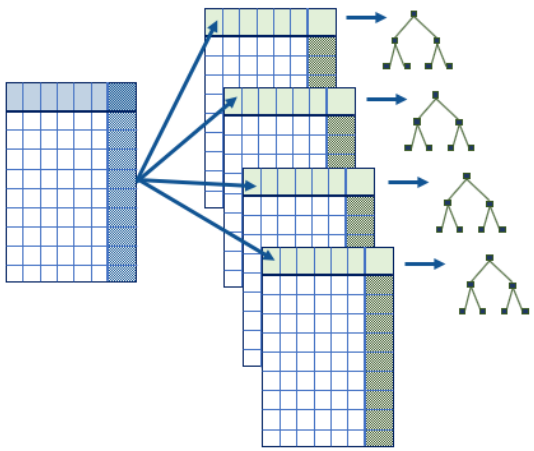
Inhaltsverzeichnis

Die Random Forest Methode

- ist eine Modifikation der Bagging-Methode
- Trainingsdatensatz wird ebenfalls in sogenannte Bootstrap samples überführt
- Entscheidungsbaummodell wird zu B Bäumen, basierend auf den B Bootstrap samples erweitert

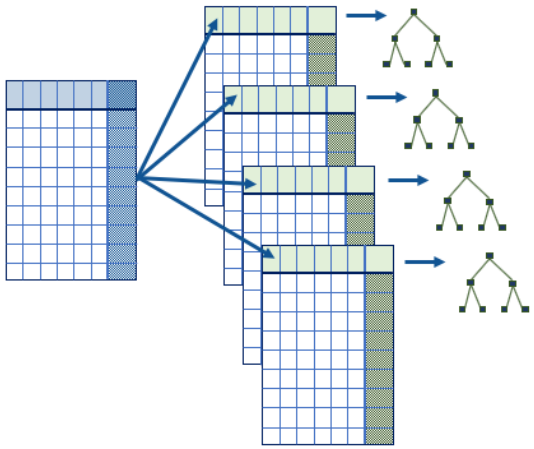
$\Rightarrow$ Darüber hinaus wird die Generierung der Bäume so modifiziert, dass diese möglichst wenig korrelieren

Random Forest - Funktionsweise



1. Von b=1 bis B
   - 1.1 Erstelle Bootstrap sample Z^* der Größe N
   - 1.2 Erstelle Entscheidungsbaum anhand des Bootstrap sample, durch rekusives Anwenden der folgenden Schritte auf jeden Endknoten, bis die Minimumknotengröße n_{min} erreicht ist:
     - 1.2.1 Wähle aus den p Variablen m zufällig aus
     - 1.2.2 Wähle aus den m Variablen die beste für den Knotenpunkt aus
     - 1.2.3 Teile den Knoten in zwei Kinderknoten auf
2. Gib die Gesamtheit aller B Bäume zurück $\{T_b\}_1^B$

Random Forest - Funktionsweise



Bei Regression wird über die Vorhersagewerte jedes Baumes gemittelt:

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Bei Klassifizierung erhält man die Klasse durch Mehrheitsvotum:

$$\hat{C}_{rf}^{B}(x) = majorityvote\{\hat{C}_b(x)\}_1^B$$

Die Varianz kann durch den sogenannten De-Korrelationseffekt gesenkt werden:

Die Varianz für genau einen Targetpunkt x ist gegeben durch:

$$Var\hat{f}_{rf}(x) = \rho(x)\sigma^2$$

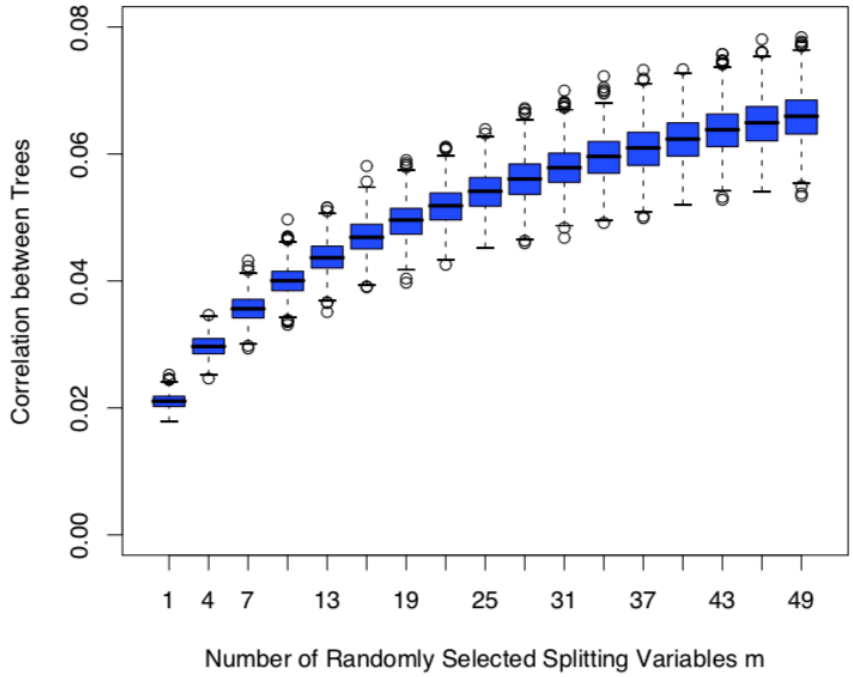
wobei

$$\rho(x) = corr[T(x;\Theta_1(Z))T(x;\Theta_2(Z))]$$

die Korrelation zweier zufällig gewählter Bäume im Random Forest und

$$\sigma^2(x) = VarT(x;\Theta(Z))$$

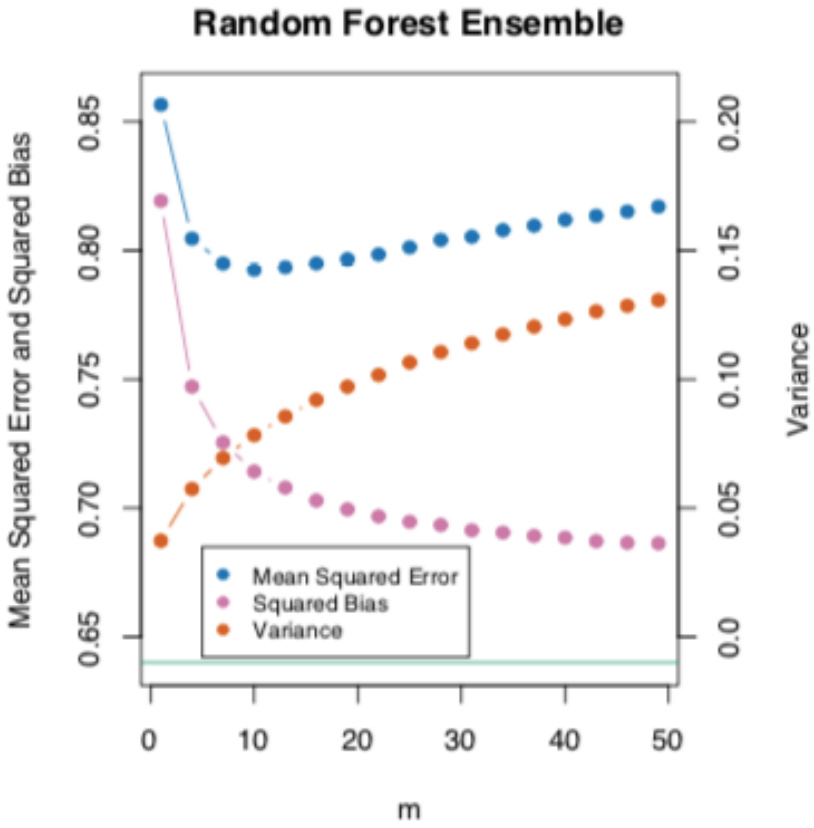
die Stichprobenvarianz eines zufällig gewachsenen Baumes ist.



Anzahl der Variablen m, aus denen an jedem Knotenpunkt ausgewählt werden kann, sinkt wobei die m Variablen zufällig gewählt sind

$\Rightarrow$ Wahrscheinlichkeit, dass identische Bäume erstellt werden, sinkt

$\Rightarrow$ Korrelation der Bäume sinkt



m wird kleiner und reduziert die Korrelation

- $\Rightarrow$ reduziert die Varianz
- $\Rightarrow$ bei (fast) gleichbleibendem Biaswert
- $\Rightarrow$ reduziert MSE bzw. den Test Fehler

Out-of-bag sample

Ist eine Beobachtung $z_i = (x_i, y_i)$ nicht im Bootstrap sample enthalten, kann sie genutzt werden, um genau die Bäume zu testen, welche aus dem bootstrap sample generiert wurden

Inhaltsverzeichnis

Die Boosting Methode

- konstruiert ein Modell mit höherer Kapazität und
- senkt dadurch den Bias
- Die Grundidee von Boosting besteht darin, aus vielen sogenannten 'weak learners' ein Kollektiv zu erstellen das die Modellvorhersage verbessert
- Boosting basiert dabei vorwiegend auf Entscheidungsbäumen

Boosting - Funktionsweise anhand der AdaBoost Methode

Die AdaBoost Methode erstellt iterativ eine Folge von Entscheidungsbäumen, mit je einem Knotenpunkt & einer Entscheidungsvariable

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Wie stark die Entscheidung des Baumes in die Gesamtentscheidung einfließen wird, wird durch sein **Stimmgewicht** α_m beschrieben.

→ α_m berechnet sich durch die **Fehlerquote der Variable** err_m und die **Stichproben-Gewichte** w_i

Boosting - Funktionsweise anhand der AdaBoost Methode

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Das **Stichproben-Gewicht** w_i ist zu Beginn $\frac{1}{N}$ und wird in jeder Iteration erneuert

$\rightarrow$ w_i berechnet sich durch die Stimmgewichtung des vorherigen Baumes α_{m-1}

Boosting - Funktionsweise anhand der AdaBoost Methode

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Das Gewicht von falsch kategorisierten Stichproben wird erhöht, indem es mit dem Faktor e^{α_m} skaliert wird.

Boosting - Funktionsweise anhand der AdaBoost Methode

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Der nächste Klassifikator wird dann anhand der neuen Stichprobengewichte ausgewählt.

$\Rightarrow$ Hatte ein Klassifikator also eine hohe Fehlerquote, werden die falsch klassifizierten Inputdaten die Wahl des nächsten Klassifikators stärker beeinflussen.

Boosting - Funktionsweise anhand der AdaBoost Methode

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

$\Rightarrow$ Die Bäume in einem von AdaBoost generierten Modell sind also abhängig voneinander und haben unterschiedliche Gewichtung bei der Modellvorhersage : $G(x) = \sigma(\sum \alpha_m G_m(x))$

Boosting Allgemein

- Iterativ werden einzelne Entscheidungsbäume erstellt, sodass das Modell am Ende aus der Summe aller Bäume besteht:

$$f_M(x) = \sum_{m=1}^{M} T(x, \theta_m)$$

  → um einen neuen Baum $\hat{\theta}_m$ zu definieren, muss immer wieder folgende Gleichung gelöst werden:

$$\hat{\theta}_m = \underset{\theta_m}{argmin} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \theta_m))$$

  → da hierbei auch $f_{m-1}(x_i)$, der Vorhersagewert des vorigen Baumes in Betracht gezogen wird, sind die Bäume abhängig

- die Entscheidungen der 'weak learner' fließen nicht gleichwertig, sondern gewichtet in die Gesamtentscheidung ein

Die Idee von Gradient Boosting

Eine mögliche Lösung der Gleichung ist ein Regressionsbaum der die Residuen $y_i - f_{m-1}(x_i)$ bestmöglich vorhersagt:

$\Rightarrow$ jeder Baum wird dann von den Fehlvorhersagen des vorhergehenden Baumes beeinflusst

Inhaltsverzeichnis

Übersicht der 3 Methoden

Alle drei Methoden zielen darauf den Generalisierungs- bzw Testfehler zu senken:

- Bagging und Random Forest reduzieren die Varianz des Modells
- Boosting reduziert den Biaswert des Modells

Die drei Methoden basieren auf der Grundidee von Entscheidungsbäumen:

- Bagging kann für verschiedene Trainingsalgorithmen angewandt werden
- Random Forest und Boosting erweitern immer ein Entscheidungsbaummodell

Inhaltsverzeichnis

Literatur

T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Verlag, 2009.

Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org