Data Mining und Maschinelles Lernen

Ensemble Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

“There ain’t no such thing as a free lunch.”

— Robert A. Heinlein in “The Moon Is a Harsh Mistress”, 1966

Ensemble Methoden treffen Vorhersage, in dem sie die Vorhersagen vieler individueller Lernen zu einer einzigen Vorhersage

Basierend auf Folien von Aaron Bobick, Beate Sick, Katharina Morik, Uwe Ligges, Johannes Fürnkranz, Emily Fox und Mapt. Danke fürs Offenlegen der Folien.

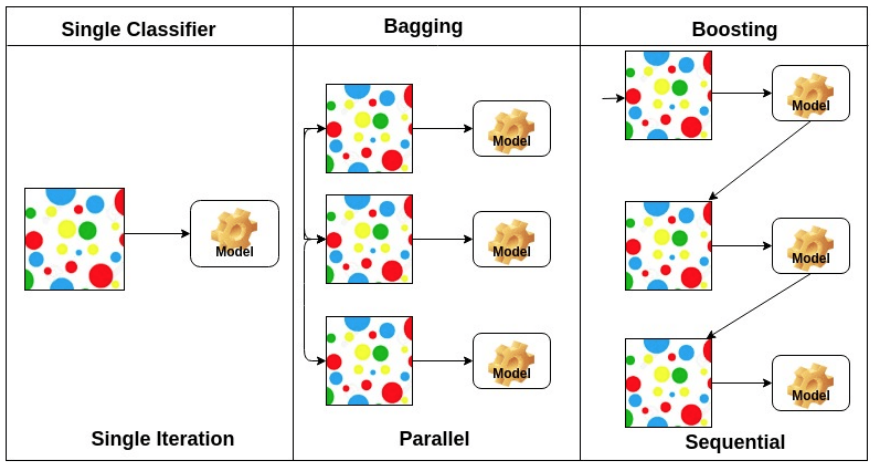
Unsere Ziele für heute

TECHNISCHE UNIVERSITÄT DARMSTADT

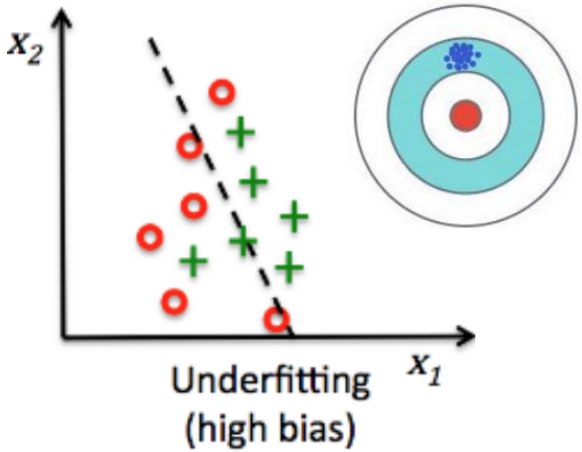
Heute wollen wir Ensemble-Methoden kennenlernen:

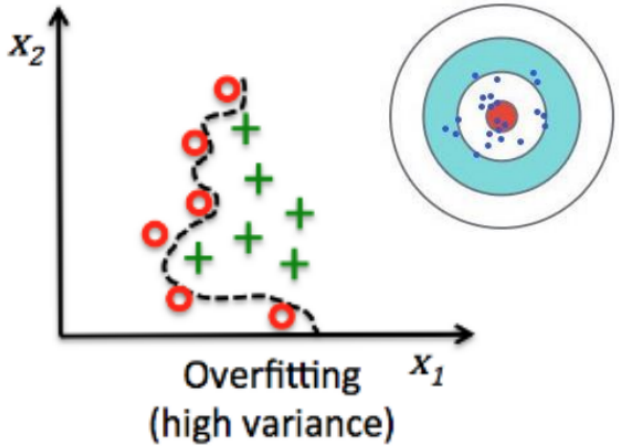
Gemeinsam können “schwache” Lerner mächtiger sein!

- Was sind Zufallswälder (Random Forrests)?
- Was ist Bagging?
- Was ist Boosting?



Ausgangspunkt: Bias und Varianz





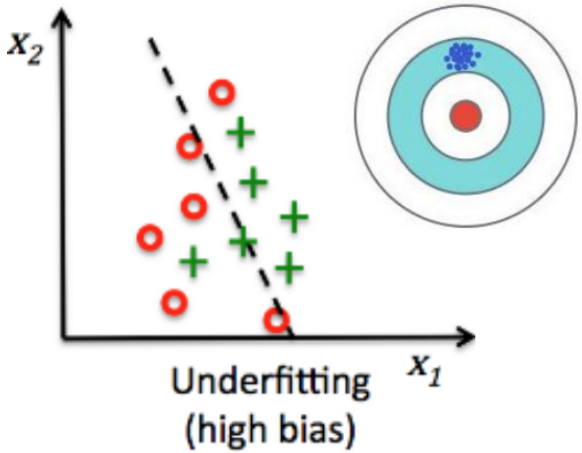
Ein unterangepasster Klassifikator ist

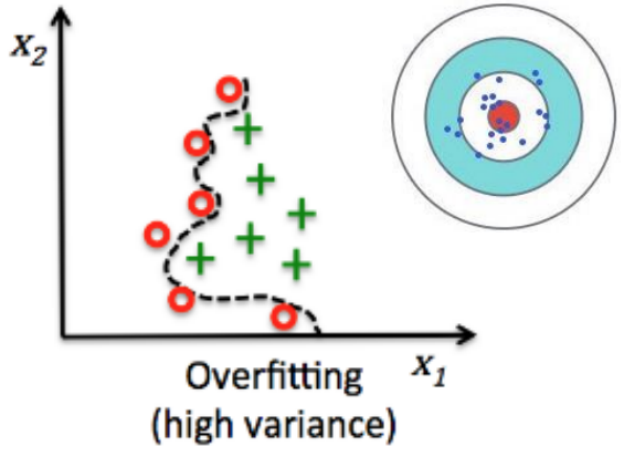
- nicht flexibel genug
- grosser Trainingsfehler und systematischer Testfehler (grosser Bias)
- Vorhersagen variieren kaum auf Testmenge (niedrige Varianz)

Ein überangepasster Klassifikator ist

- zu flexibel
- niedriger Trainingsfehler und nicht-systematischer Testfehler (niedriger Bias)
- Vorhersagen variieren stark auf Testmenge (grosse Varianz)

Ensemble-Methoden bekämpfen Unter- und Überanpassung





Bekämpfe Nachteile eine einzelnen Modells:

Adaptives Boosting (AdaBoost)

Bagging

Weiter Verbesserungen von Ensemble-Methoden:

Gradient Boosting (bekämpft Unter- und Überanpassung)

Random Forest (falls Baumverfahren benutzt werden)

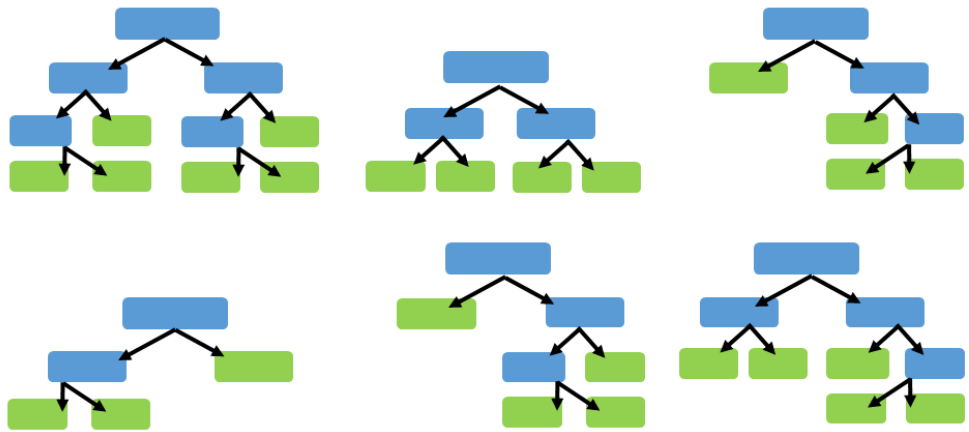
Zufallswälder (Random Forests)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zufallswälder klassifizieren auf der Basis von vielen Klassifkationsbäumen.
- Das Lernen der erforderlichen Klassifkationsbäume ist dabei zufällig.
- Deshalb spricht man von Zufallswäldern.

Wie lernen wir Zufallswälder?

TECHNISCHE UNIVERSITÄT DARMSTADT



Zufallswald (Random Forrest)

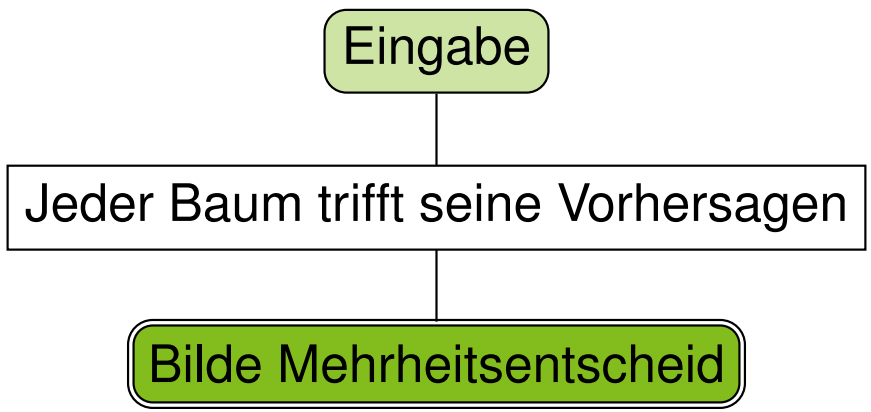
Ein Zufallswald besteht aus mehreren unkorrelierten Entscheidungsbäumen. Alle Entscheidungsbäume sind unter einer bestimmten Art von Randomisierung während des Lernprozesses gewachsen.

Wie treffen Zufallswälder (Random Forrests) Vorhersagen?

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorhersage Zufallswald

Für eine Klassifikation[1] darf jeder Baum in dem Wald eine Entscheidung treffen, und die Klasse mit den meisten Stimmen entscheidet die endgültige Klassifikation.



[1] Random Forests können auch zur Regression eingesetzt werden.

Wie lernen wir Zufallswälder? Wie viele Bäume?

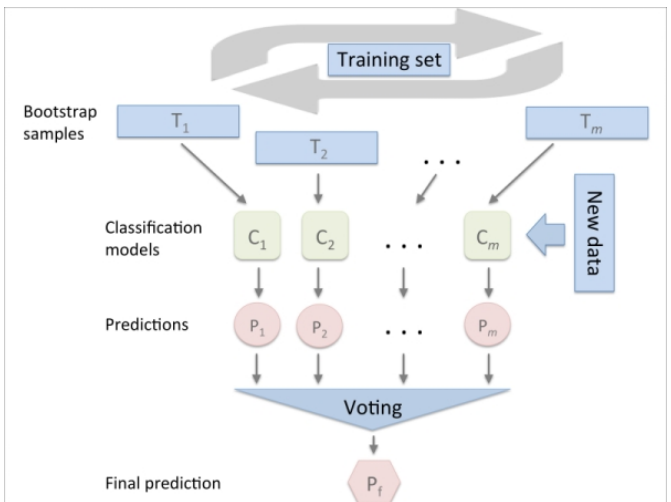
TECHNISCHE UNIVERSITÄT DARMSTADT

- Zuerst muss festgelegt werden, wie viele Bäume den Wald bilden sollen. Dabei wird die Klassifikationsgüte eines Waldes meistens umso besser, je mehr einzelne Bäume berechnet werden.
- Eine große Anzahl Bäume führt nicht zu Overfitting und ist deshalb zu bevorzugen.
- Grundsätzlich hängt die Anzahl der Bäume aber von verschiedenen Parametern ab, z.B. der Anzahl Merkmalen und der Anzahl Klassen.

Wie lernen wir Zufallswälder? Welche Daten pro Baum?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Bäume werden nicht aus allen zur Verfügung stehenden Daten bestimmt, sondern es wird für jeden Baum eine Stichprobe (mit Zurücklegen) aus den Beobachtungen gezogen, daher auch Zufallswald.
- Da hier mit vielen unterschiedlichen Stichproben gearbeitet wird, spricht man auch von bagging (Bootstrap aggregation).



Sample indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...

C_1 C_2 C_m

Wie lernen wir Zufallswälder? Welche Variablen? Pruning?

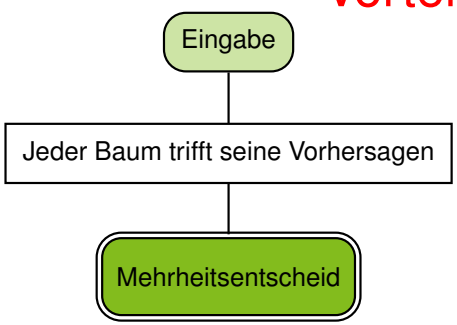
TECHNISCHE UNIVERSITÄT DARMSTADT

- Außerdem werden für die Bestimmung der Verzweigungsregeln nicht alle Variablen verwendet, sondern es wird zufällig eine (vorher festgelegte) Anzahl aus allen Variablen gezogen. Nur diese werden nach der besten Partition durchsucht.
- Jeder Baum sollte dabei so groß wie möglich sein (kein Pruning). Er soll auf der entsprechenden Stichprobe mit den gezogenen Variablen den kleinstmöglichen Wiedereinsetzungsfehler haben.

Wie lernen wir Zufallswälder? Demokratie im Wald!

TECHNISCHE UNIVERSITÄT DARMSTADT

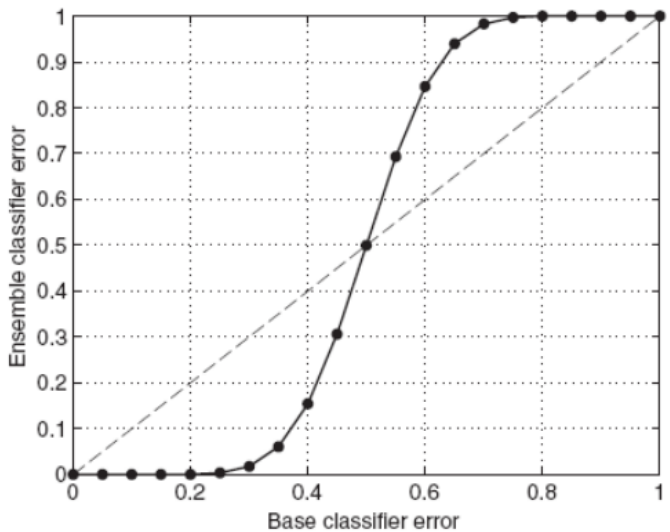
Ein neues Objekt wird von dem Zufallswald klassifiziert, indem es von jedem der berechneten Bäume einmal klassifiziert wird und dann der Klasse zugeordnet wird, die die meisten Bäume bevorzugt haben.



Vorteil und Nachteil gegenüber eines einzelnen Klassifikationsbaum:

+ Jede Variable, die einen Beitrag zur Klassentrennung liefert, wird irgendwann auch bei der Klassifikation verwendet.
– Verständlichkeit geht verloren, denn die Klassifikationsregel ist nicht mehr einfach ablesbar.

Warum sollte man das machen?

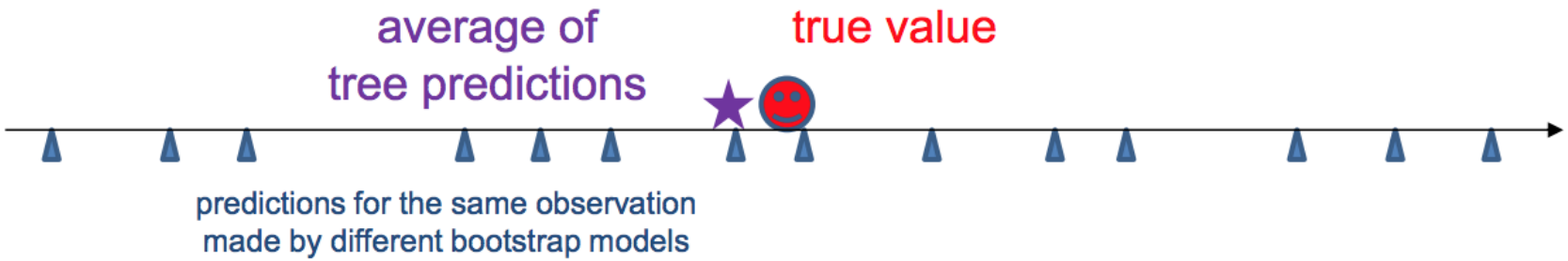


Ensembles sind dann besser, wenn jeder einzelner Klassifikator besser als der Zufall in der Vorhersage ist.

- Nehmen wir mal an, dass wir 25 unabhängige Klassifikatoren haben
- Wenn jeder Klassifikator eine Fehlerrate von $\epsilon = 0.35$ hat, dann ist die Wahrscheinlichkeit, dass das Ensemble (Mehrheitsentscheid, also $>$ 50%, also 13 oder mehr Klassifikatoren) einen Fehler macht,

$$P(\text{wrong prediction}) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Auch für Regression ist das gut!



Nehmen wir mal an, dass wir 25 flexible Regressionsmodelle haben

- Weil sie sehr flexibel sind, haben sie keinen oder nur einen geringen Bias
- Weil sie sehr flexibel sind, haben sie aber eine grosse Varianz

Nach dem Zentralen Grenzwertsatz bleibt der Erwartungswert der gleiche, aber die standard Abweichung wird um einen Faktor von $\sqrt{n}$ also hier von 5 reduziert.

OK, Verständlichkeit geht verloren, aber können wir trotzdem etwas von Zufallsbäumen lernen?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zufallswälder eignen sich besonders dann zur Klassifikation, wenn es mehrere Variablen gibt, die nur einen kleinen Beitrag zur Klassentrennung liefern.
- Der Beitrag zur Klassentrennung kann durch die Wichtigkeit einer Variablen bestimmt werden.
- Eine mögliche Berechnung dieser Wichtigkeit ergibt sich aus dem Gütemaß.

Wichtigkeit einer Variablen: $I := VG/NV$

VG = Summe der Verminderungen des Gütemaßes (z.B. Gini-Index) im gesamten Wald durch eine Variable
NV = Anzahl der Verzweigungen dieser Variable im Wald

Random Forest in R — randomForest

TECHNISCHE UNIVERSITÄT DARMSTADT

Funktion `randomForest()` in Paket `randomForest`, hier am Beispiel des Datensaetzes Pima Indianer

```
> library("randomForest")
> set.seed(20060911)
> pid_rf<-randomForest(diabetes~., data = pid_learn,
+ mtry = 3, ntree = 100, importance = TRUE,
+ keep.forest = TRUE)
```

Random Forest in R — randomForest

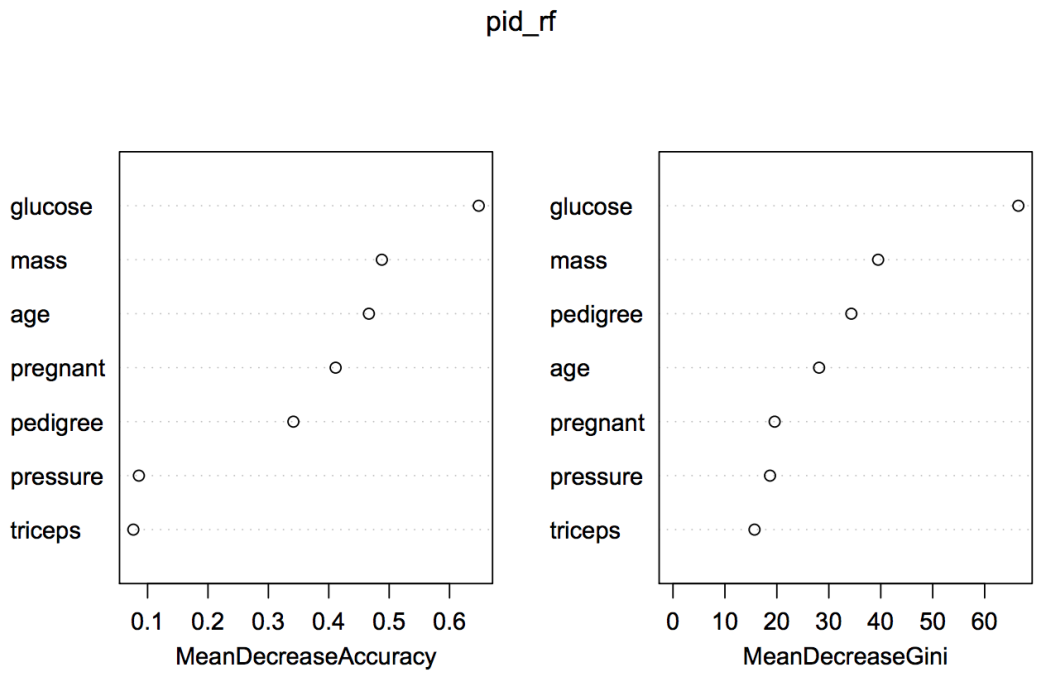
TECHNISCHE UNIVERSITÄT DARMSTADT

```
> pred_rf <- predict(pid_rf, newdata = pid_test)
> mc(pred_rf)

$table
pred
true neg pos
neg 134 24
pos 38 56

$rate
[1] 0.2460317
```

Random Forest in R — randomForest

TECHNISCHE UNIVERSITÄT DARMSTADT

```
> varImpPlot(pid_rf)
```



Random Forest in R — randomForest
Überwachung von Elektrobauteilen z.B. in einer Produktionsstrasse

TECHNISCHE UNIVERSITÄT DARMSTADT

Aus 16 Charakteristika einer Stromreihe und 5 Charakteristika einer Sensorreihe soll bestimmt werden, ob ein so genannter Lichtbogen vorliegt oder nicht. Dabei werden 2 Arten von Lichtbögen unterschieden (Serial Arc und Wet Arc). Es werden also 3 Klassen unterschieden. Lichtbögen können zu Kabelbränden führen, die gefährlich sind. Es liegen 1235 Beobachtungen vor.

Random Forest in R — randomForest
Überwachung von Elektrobauteilen z.B. in einer Produktionsstrasse

TECHNISCHE UNIVERSITÄT DARMSTADT

Zunächst müssen die Parameter des Verfahrens festgelegt werden:

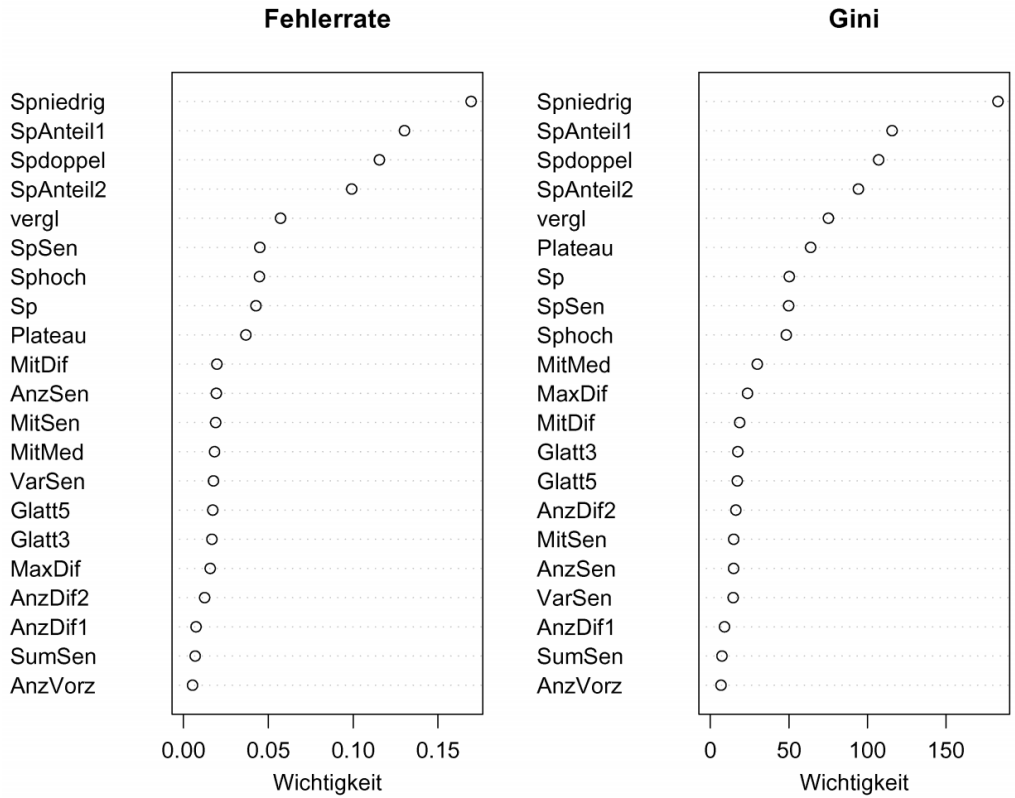
- Pro Baum werden genauso viele Beobachtungen verwendet wie insgesamt zur Verfügung stehen (1235), allerdings zufällig mit Zurücklegen gezogen.
- Pro Baum werden 3 zufällig bestimmte Variablen verwendet.
- Es werden 1000 Bäume verwendet.

Daraus ergibt sich eine 246-fach kreuzvalidierte Fehlerrate von 2.59% (hier nicht behandelt).

Random Forest in R — randomForest
Überwachung von Elektrobauteilen z.B. in einer Produktionsstrasse

TECHNISCHE UNIVERSITÄT DARMSTADT

Wichtigkeit einzelner Variablen



Random Forest in R — randomForest
Überwachung von Elektrobauteilen z.B. in einer Produktionsstrasse

TECHNISCHE UNIVERSITÄT DARMSTADT

Man erkennt, dass nur sehr wenige Variablen besonders wichtig sind, also stark zur Senkung der Unreinheit (gemessen mit dem Gini-Index) beitragen oder bei einer Permutation der Werte der entsprechenden Variablen die Fehlerrate stark ansteigen lassen.

Die anderen Variablen scheinen aber trotzdem einen kleinen Beitrag zur Klassentrennung zu liefern. Gerade in solchen Fällen wird die Verwendung von Zufallswäldern empfohlen, da alle Variablen an der Klassifikation beteiligt werden ohne Überanpassung an die Daten.

Bagging als Verallgemeinerung

TECHNISCHE UNIVERSITÄT DARMSTADT

Es wurde erwähnt, dass die Methode der Konstruktion von Zufallswäldern, nämlich viele Klassifikationsregeln zusammenfassend zu gewichten, als bagging bezeichnet wird. Diese Technik des bagging kann auch auf andere (u.U. auch gleichzeitig auf unterschiedliche) Klassifikationsverfahren angewandt werden.

Bagging als Verallgemeinerung

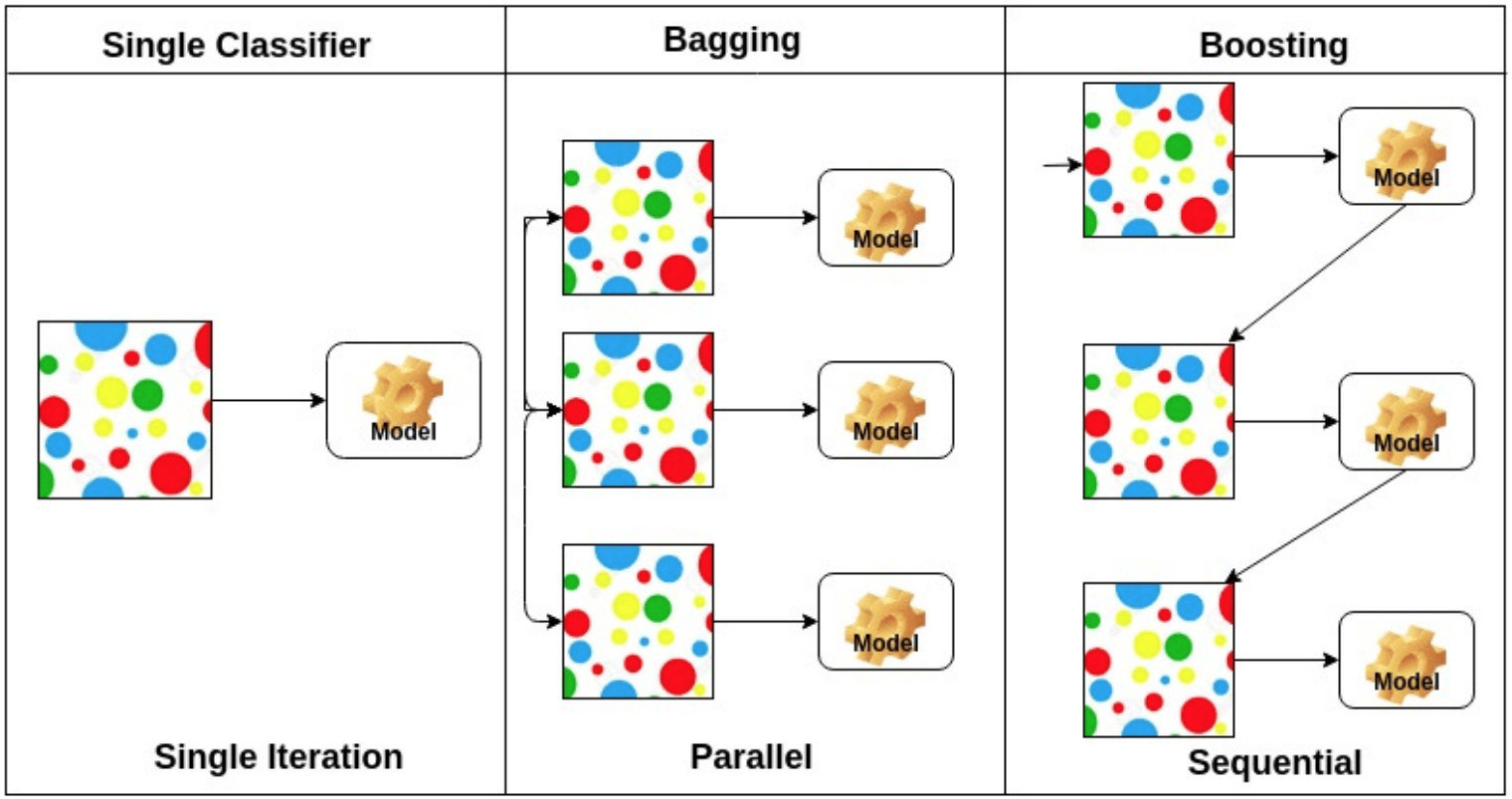
TECHNISCHE UNIVERSITÄT DARMSTADT

Bagging

Mit bagging (Bootstrap aggregation) wird die Sammlung vieler gleichartiger Lerner und deren gemeinsame Entscheidungsregel bezeichnet, wobei die Lerner aus Bootstrap-Stichproben sowohl der Beobachtungen als auch der Variablen eines Datensatzes generiert werden.

Die Lernen sind tpyischerweise von recht einfacher Struktur, z.B. Random Forests als Bagging Verfahren, die einzelne Bäume als Lernen verwenden.

Wo stehen wir?

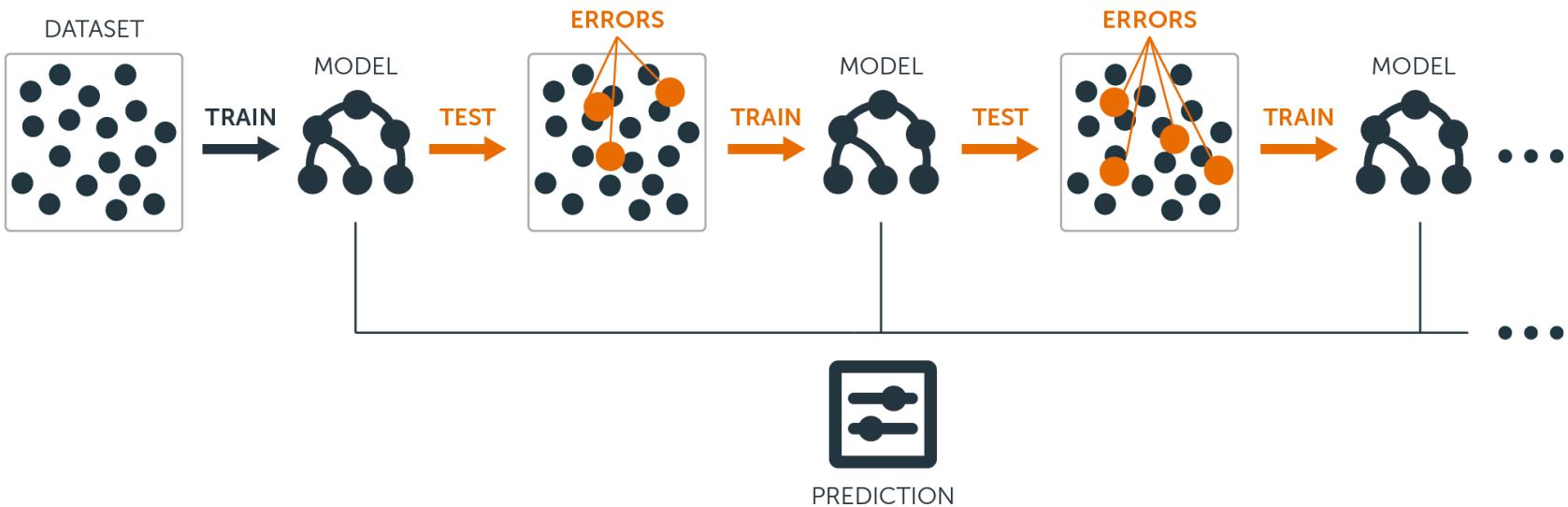


Was ist denn jetzt Boosting?

Boosting

TECHNISCHE UNIVERSITÄT DARMSTADT

Idee: Es ist wesentlich leichter ein paar Faustregeln für ein Problem zu finden sind, als eine generelle Regel, die das gesamte Problem wirklich löst.



Boosting — Ein intuitives Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Wir wollen beim Pferderennen auf den Gewinner tippen. Wir fragen eine Expertin. Ihr wird es schwer fallen, eine einzige Regel aufzustellen, anhand derer man auf den Gewinner schließen kann. Stattdessen nennt sie uns einige Faustregeln:

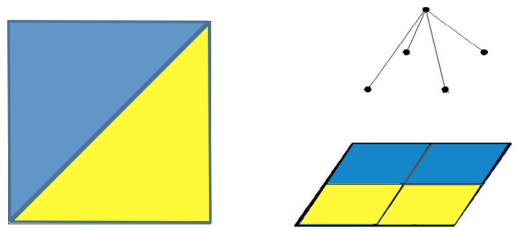
"Tipp auf das Pferd, das in der letzten Zeit oft gewonnen hat"
und
"Tipp auf das Pferd, das die besten Quoten hat", ...

Boosting

Jede Regel für sich alleine ist nur ein bisschen besser, als zufällig auf ein Pferd zu tippen. Kombinieren wir viele solche Fausregeln zu einer gewichteten Mehrheitsentscheidung, so erhält man u.U. einen mächtigen Klassifikator.

Boosting — Schwache und starke Lerner

TECHNISCHE UNIVERSITÄT DARMSTADT



- Boosting ist ein Verfahren, das eine effiziente Entscheidungsregel für ein Klassfikationsproblem aufstellt, indem es mehrere einfache Regeln kombiniert. Diese Regeln werden schwache Klassifikatoren (weak learner) oder Basisklassifikatoren genannt:
  - z.B. naive Bayes, logistische Regression, decisions stumps oder flache Entscheidungsbäume, etc.: typischerweise keine Überanpassung, können aber auch keine komplexen Vorhersagen treffen.
- Das Ergebnis von Boosting dagegen bezeichnen wir als starken Klassifikator (strong learner).

Boosting

TECHNISCHE UNIVERSITÄT DARMSTADT

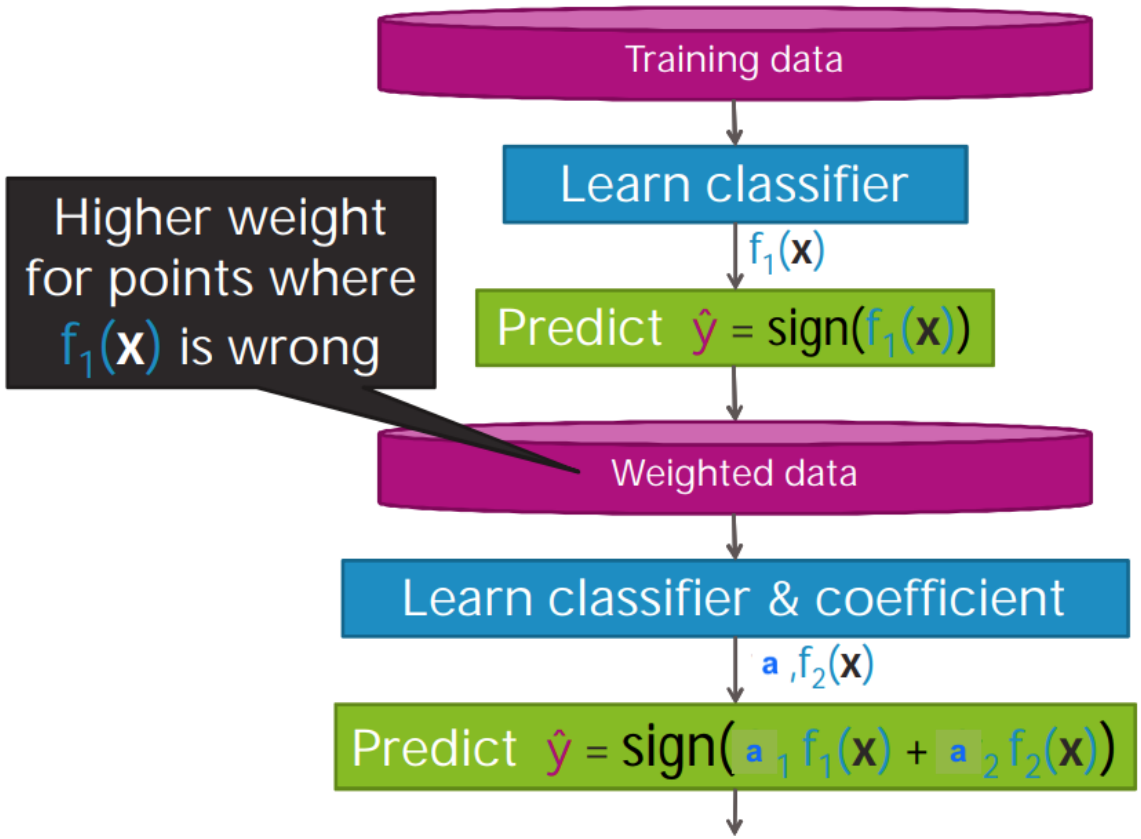
Idee (Shapire, 1989)

Nimm einen schwachen Lernen und trainiere ihn mehrmals — immer wieder neu gewichtet — auf den Trainingsdaten

In jeder Iteration t:

- Gewichte jede Trainigsbeispiel je nachdem “wie falsch” es klassifiziert wurde
- Trainiere den schwachen Lernen erneut auf den so gewichteten Bespielen
- Bestimme ein “Mitspracherecht” α_t des trainierten Weak-Learners

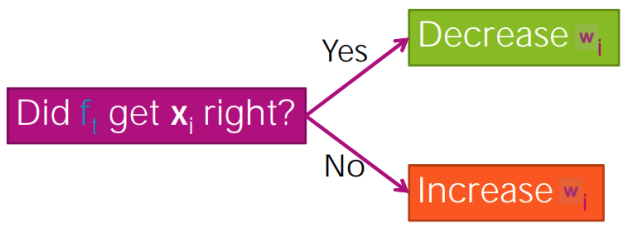
Boosting — Anschaulich



Boosting — AdaBoost für zwei Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Als Input ist ein Trainingdatensatz $(y_1, x_1), \ldots, (y_m, x_m)$ gegeben, wobei die x_is die Merkmale und die y_is die Klassenlabels sind . Hier also $y_i \in \{-1, +1\}$.
- Anfangs sind alle Beispiele gleich wichtig: $w_i = \frac{1}{m}$
- Wir trainieren einen schwachen Klassifikator, der den (durch die w_is) gewichteten Klassifiaktionsfehler minimiert.
- Der gewichteten Klassifiaktionsfehler sagt uns, wieviel Mitspracherecht α_t der Weak-Learner hat
- Nach jeder Trainingsrunde t gewichten wir Beispiel mit einem größerem Fehler mehr, alle Gewichte zusammen summieren sich aber noch zu 1.



Boosting — AdaBoost für zwei Klassen genauer

TECHNISCHE UNIVERSITÄT DARMSTADT

Algorithm:

1) Set $y_i \in \{-1,+1\}$ and start with identical weights $w_i = 1/n$
2) Repeat for $m = 1,2,...,M$:
   a) Fit the classifier $f_m(x) \in \{-1,+1\}$ using weights w_i
   b) Compute the weighted error $err_m = \sum_i w_i \cdot I[y_i \neq f_m(x_i)]$
   c) Compute the aggregation weight $\alpha_m = \log((1-err_m)/err_m)$
   d) Set $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot I[y_i \neq f_m(x_i)])$; normalize to $\sum_i w_i = 1$
3) Output $F_M(x) = sign \sum_{m=1}^{M} \alpha_m f_m(x)$

AdaBoost — Warum die Gewichte genau so?

TECHNISCHE UNIVERSITÄT DARMSTADT

- AdaBoost betrachtet das Exponential Loss $e^{-yf(x)}$, das eine obere Schranke für das 0-1 Loss ist.
- Wenn wir den m-ten Klassifikator im m-ten Schritt hinzufügen, haben wir als Zielfunktion

$$\begin{aligned} E &= \sum_i e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(\mathbf{x}_i) - \frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \\ &= \sum_i e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(\mathbf{x}_i)} \; e^{-\frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \end{aligned}$$

- Weil wir die ersten $m-1$ Terme festhalten, können wir sie mit als Konstante $w_i^{(m)}$ zusammenfassen und erhalten das Gewichtsupdate von AdaBoost

$$w_i^{(m)} \propto w_i^{(m-1)} e^{-\frac{1}{2} y_i \alpha_j f_{m-1}(\mathbf{x}_i)}$$

AdaBoost — Warum das Mitspracherecht genau so?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jetzt teilen wir nach korrekt und inkorrekt klassifizierten Beispielen auf

$$E = \sum_{i:f_m(\mathbf{x}_i)=y_i} w_i^{(m)} e^{-\frac{\alpha_m}{2}} + \sum_{i:f_m(\mathbf{x}_i)\neq y_i} w_i^{(m)} e^{\frac{\alpha_m}{2}}$$

- und schreiben um

$$E = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i) + e^{-\frac{\alpha_m}{2}} \sum_i w_i^{(m)}$$

- Dann leiten wir nach f_m ab und setzen den Gradient gleich 0

$$\frac{dE}{d\alpha_m} = \frac{\alpha_m}{2}\left(e^{\frac{\alpha_m}{2}} + e^{-\frac{\alpha_m}{2}}\right) \sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i) - \frac{\alpha_m}{2} e^{-\frac{\alpha_m}{2}} \sum_i w_i^{(m)} = 0$$

- Dann ergibt ein Lösen nach α

$$\begin{aligned} 0 &= e^{\frac{\alpha_m}{2}}\epsilon_m + e^{-\frac{\alpha_m}{2}}\epsilon_m - e^{-\frac{\alpha_m}{2}} \\ e^{\frac{\alpha_m}{2}}\epsilon_m &= e^{-\frac{\alpha_m}{2}}(1-\epsilon_m) \\ \frac{\alpha_m}{2} + \ln \epsilon_m &= -\frac{\alpha_m}{2} + \ln(1-\epsilon_m) \\ \alpha_m &= \ln \frac{1-\epsilon_m}{\epsilon_m} \end{aligned}$$

AdaBoost — Aber wie beachten wir die Gewichte beim Trainieren der Weak-Learners?

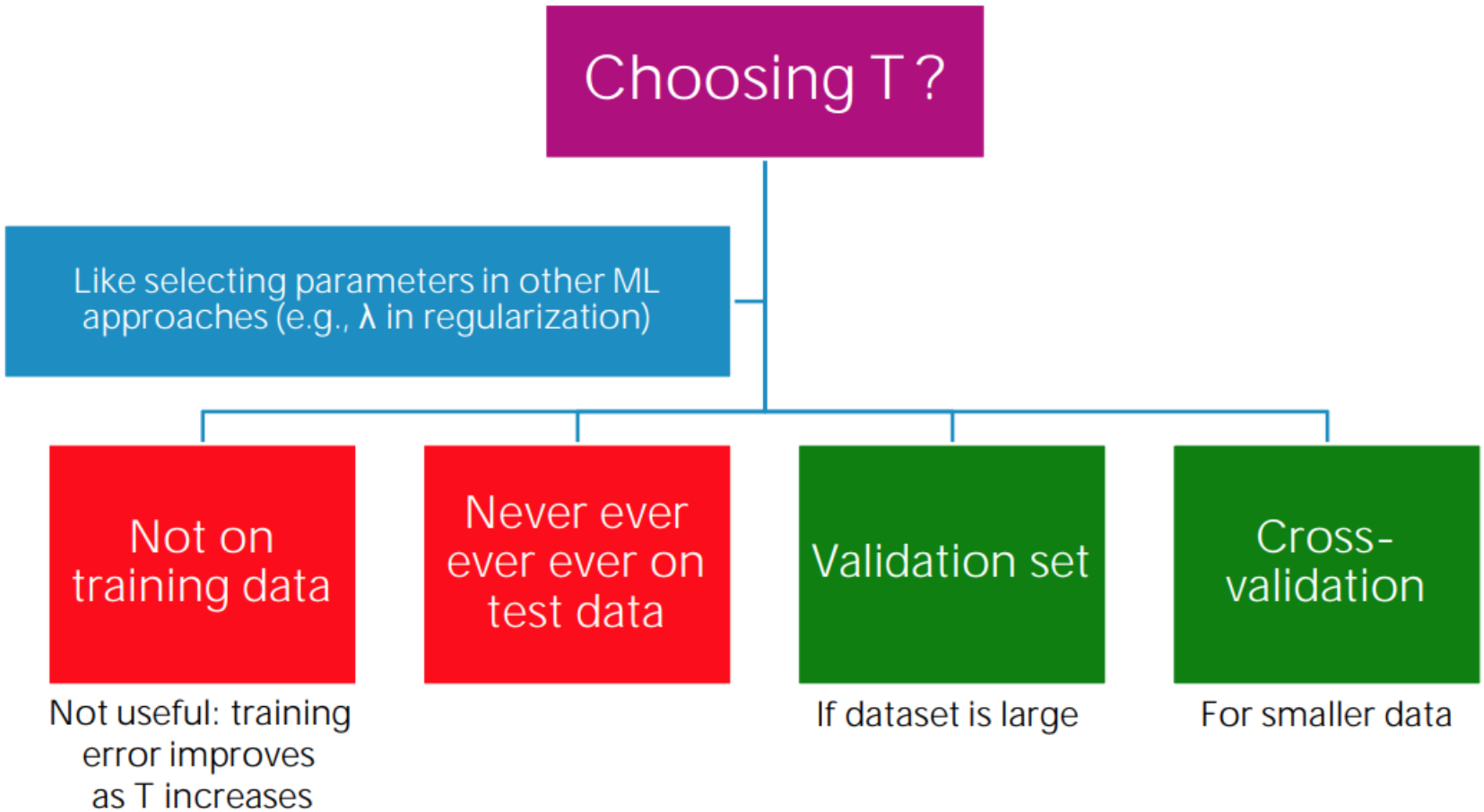
TECHNISCHE UNIVERSITÄT DARMSTADT

Wenn der Lerner nicht direkt mit gewichteten Trainingsbeispielen umgehen kann, dann ziehen wir Stichproben:

- Wir interpretieren die Gewichte als Wahrscheinlichkeiten
- Beispiele mit größerem Gewicht, haben eine größere Wahrscheinlichkeit als Stichprobe (mit Zurücklegen) gezogen zu werden

Funktioniert wenn, der der Weak-Learner mit unterschiedliche Anzahlen von identischen Trainingsbeispielen umgehen kann.

Boosting — Und wann stoppen wir das Lernen?



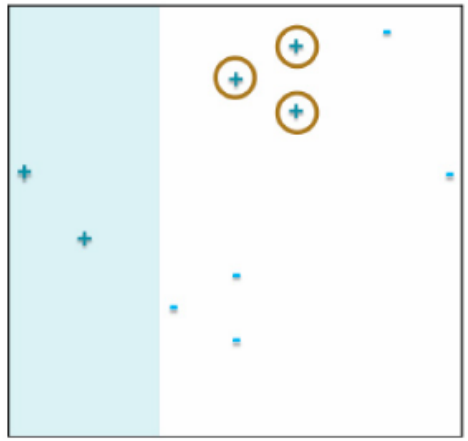
Beispiel AdaBoost für zwei Klassen

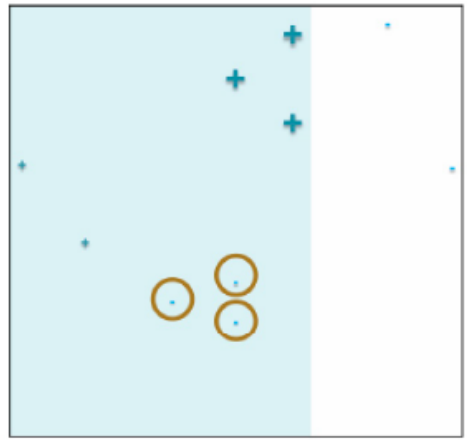
TECHNISCHE UNIVERSITÄT DARMSTADT

C_1

$\text{err}_1 = 0.30$

$\alpha_1 = 0.42$





C_2

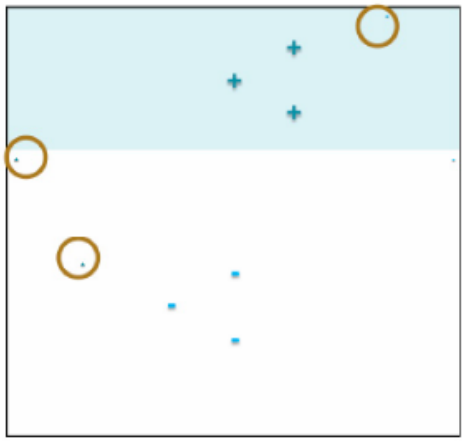
$\text{err}_2 = 0.21$

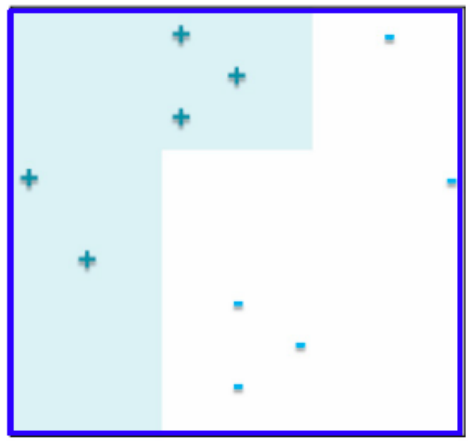
$\alpha_2 = 0.65$

C_3

$\text{err}_3 = 0.14$

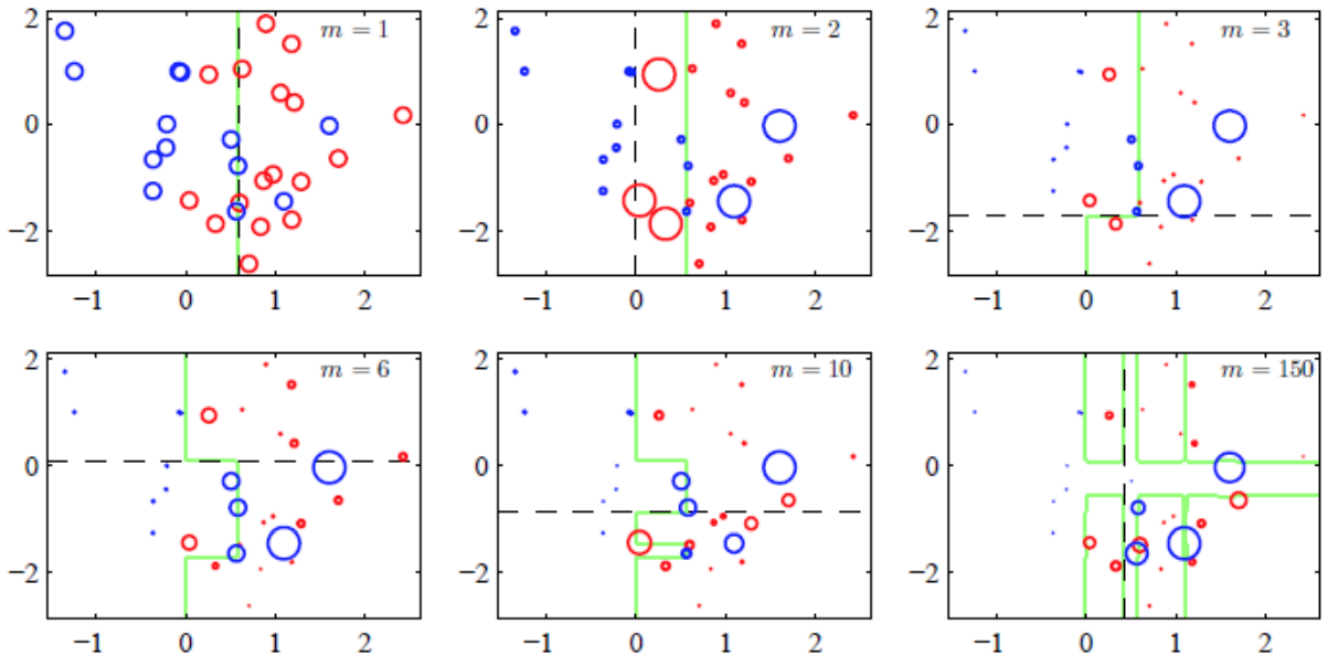
$\alpha_1 = 0.92$





$$C' = \sum_{m=1}^{3} \alpha_m \cdot C_m$$
$$= 0.42 \cdot C_1 + 0.65 \cdot C_2 + 0.92 \cdot C_3$$

Ein weiteres Beispiel für AdaBoost auf zwei Klassen

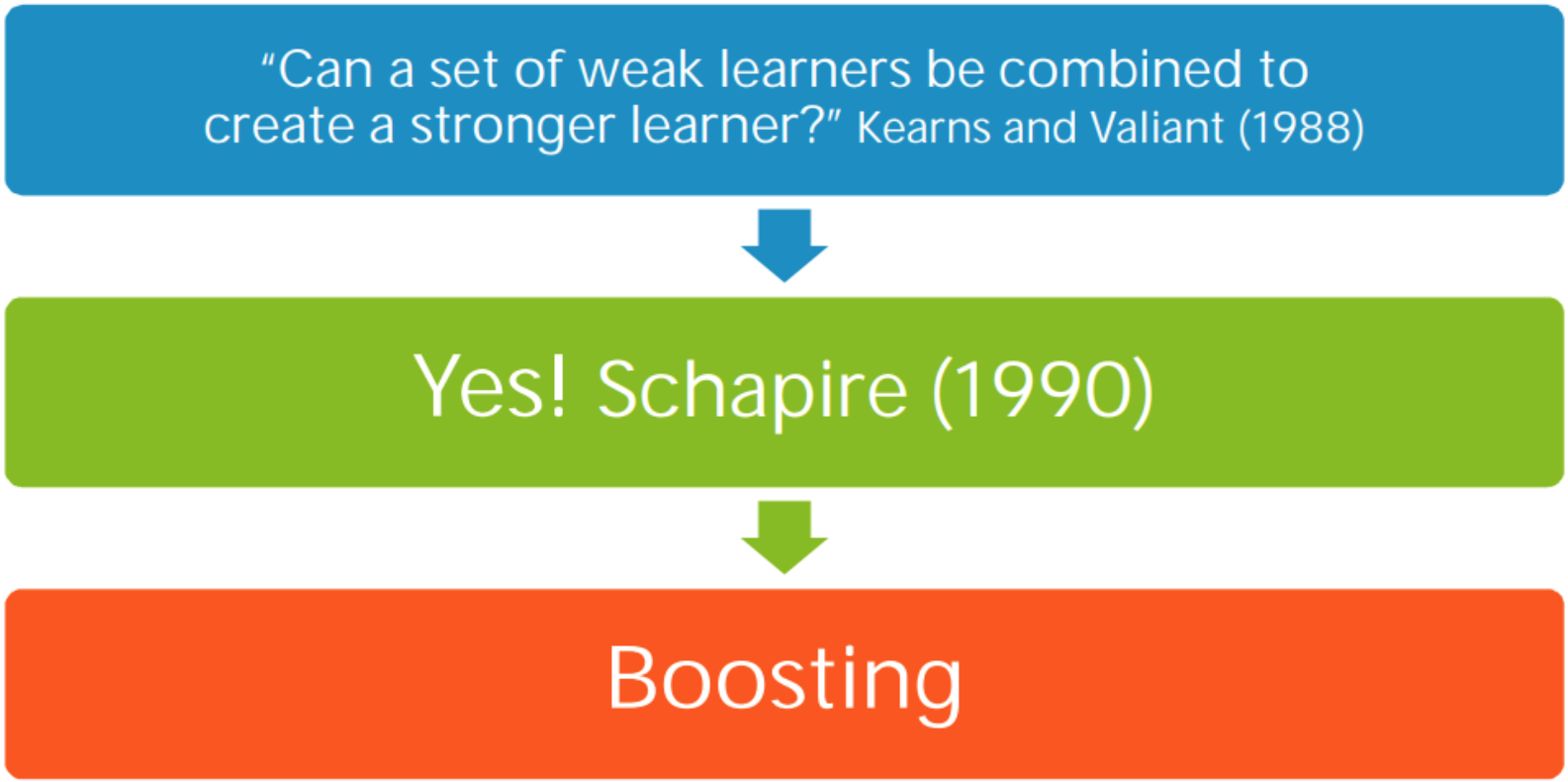


Circle = data points, circle size = weight.

Dashed line: Current weak learner. Green line: Aggregate decision boundary.

Die Boosting Frage

TECHNISCHE UNIVERSITÄT DARMSTADT



Und das funktioniert sehr oft (aber nicht immer)

TECHNISCHE UNIVERSITÄT DARMSTADT

Under some technical conditions...

Training error of boosted classifier → 0 as T→∞

Condition = At every t, can find a weak learner with weighted_error(f_t) < 0.5

Not always possible

Extreme example: No classifier can separate a +1 on top of -1

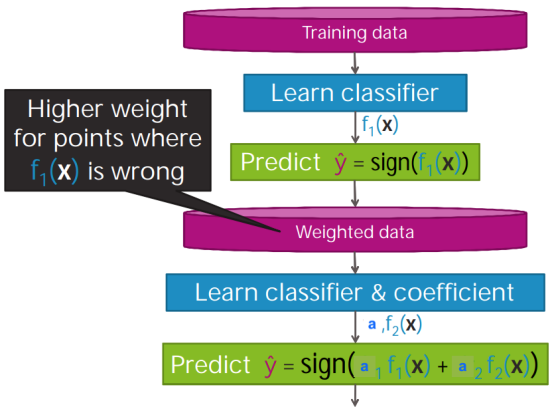
Nonetheless, boosting often yields great training error

Boosting versus Bagging

TECHNISCHE UNIVERSITÄT DARMSTADT

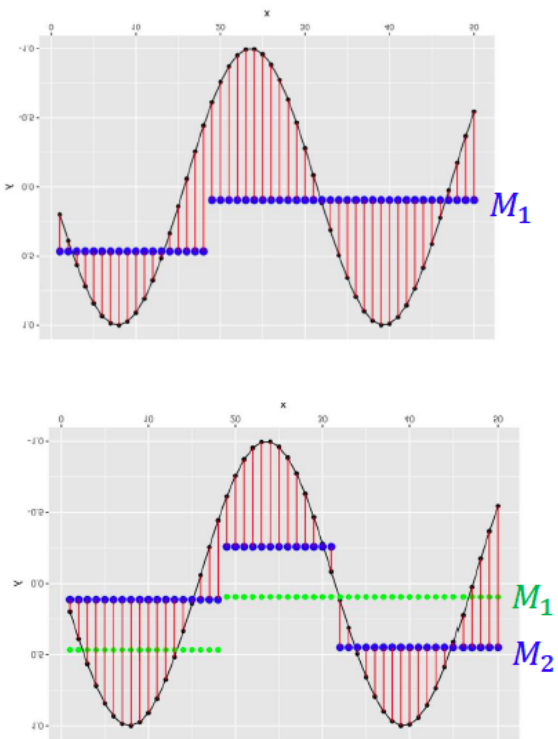
Boosting	Bagging
Gewichtung von Trainingsbeispielen	Ziehen von Stichproben
Das Gewicht eines schwachen Lerners hängt von der Vorhersagegüte ab	Gewicht ist für alle Lerner gleich

Gradienten Boosting — Warum schauen wir uns nicht den Gradient an, wenn wir ihn haben? Der sagt uns doch auch, wo wir falsch liegen.



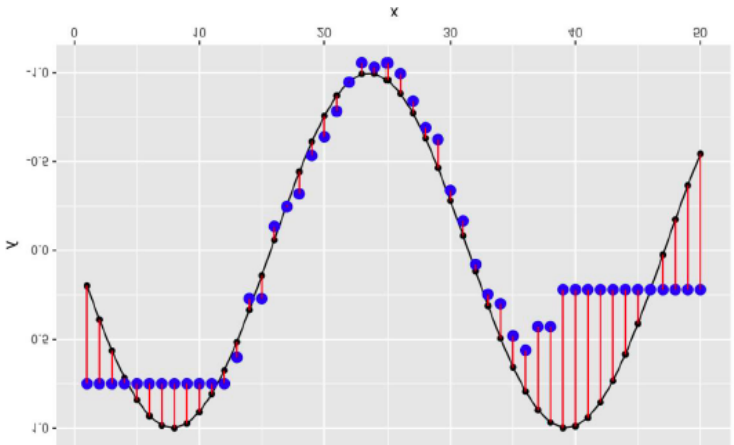
- Wir berechnen eine additives Modell $f(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$ (Ensemble) Schritt für Schritt von 1 bis M
- In jeder Iteration trainieren wir einen neuen schwachen Lernen, der die “Schwachstellen” der bisherigen schwachen Lerner ausmerzt.
- Gradienten Boosting benutzt dazu die Gradienten der Lossfunktion.
- Bei AdaBoost wurden die “Schwachstellen” durch stark-gewichtete Fehlklassifikationen bestimmt.

Gradienten Boosting — Regression als Beispiel



1. Trainiere einen flachen Regressionsbaum T_1 auf den Daten. Wir erhalten als erstes Model $M_1 = T_1$. Die “Schwachstellen” ergeben sich aus den *residuen* $= y - \hat{y}$.
2. Trainiere einen Regressionsbaum T_2 auf den Residuen. Wir erhalten als zweites Model $M_2 = M_1 + \gamma_2 T_2$, wobei γ eine Lernrate ist. Zur Regularisierung führen eine zweite Lernrate $\nu \in (0, 1)$ ein $M_2 = M_1 + \nu\gamma_2 T_2$
3. Trainiere einen weiteren Regressionsbaum auf den Residuen von M_2 ... und mache das so lange, bis die kombinierten Model gut die Daten darstellt.

Gradienten Boosting — Regression als Beispiel



- Wir führen also schrittweise immer mehr Modelle der Residuen ein.
- Das finale Model M ist das gewichtete Mittel der nacheinander gelernten Modelle:

$$M = M_1 + \nu \sum \gamma_i T_i$$

Je näher man die Lernrate ν zu 1 setzt, desto schneller das Lernen und desto größer die Gefahr einer Überanpassung.

Gradienten Boosting versus Lineare Regression

TECHNISCHE UNIVERSITÄT DARMSTADT

Bei der linearen Regression können wir die optimalen Parameter analytisch finden. Aber wir können auch den Gradienten der Lossfunktion bzgl. der Parameter pro Beispiel berechnen. Damit können wir iterative die optimalen Parameter schätzen.

Funktionaler Gradient

Gradienten Boosting folgt dieser Idee. Es approximiert aber den funktionalen Gradienten, also den Gradienten im Funktionenraum! Bei einem Squared Loss sind die Residuen gerade die negativen Gradienten der Lossfunktion:

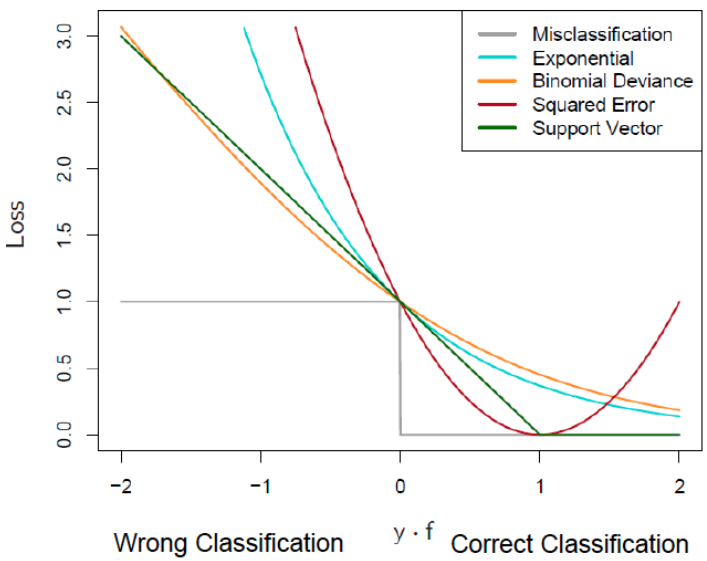
$$L = \frac{1}{2}\sum_{i=1}^{n} r_i^2 = \frac{1}{2}\sum_{i=1}^{n} \big(y_i - f(x_i)\big)^2$$

$$\frac{\partial L}{\partial f(x_i)} = -\big(y_i - f(x_i)\big) = r_i$$

Vorteil von Gradienten Boosting

TECHNISCHE UNIVERSITÄT DARMSTADT

Damit können wir andere Lossfunktionen betrachten und entsprechenden Boosting Verfahren herleiten.



- Fehlklassifikation: $L(y, F) = I[y = sign(F)]$
- Exponential/AdaBoost: $L(y, F) = \exp(-yF)$
- Binomial : $L(y, F) = \log\left(1 + \exp(-2yF)\right)$
- Quadratisch/L_2: $L(y, F) = (y - F)^2$
- SVM: $L(y, f) = y(1 - yF)$

Allgemeines Gradienten Boosting

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wähle Lossfunktion
- Trainieren ein schwaches Anfangsmodel $M = M_1$
- Wiederhole die folgenden Schritte bis zur Konvergenz:
  - Berechne negative Gradienten $-g(x_i) = -\frac{\partial Loss\left(y-i,m(x_i)\right)}{\partial M(x_i)}$
  - Trainiere ein Regressionsmodell T_k auf den negativen Gradienten $-g(x_i)$
  - Füge T_k zum Ensemble hinzu: $M = M_1 + \nu \sum \gamma_k T_k$

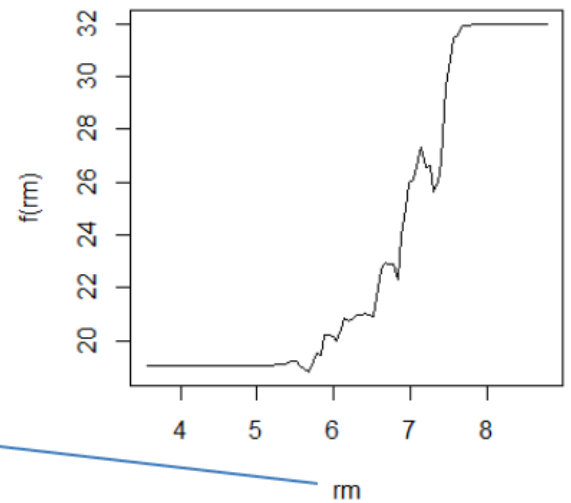
xgboost

Extreme Gradient Boosting (xgboost) eine sehr gute Realisierung von Gradient Boosting. Betont Regularisierung etwas mehr. Verteilte Berechnung möglich (10x schneller als auf einer einzelnen Maschine). War häufig Teil von Gewinnern bei Kaggle Wettbewerben.

Gradient Boosting in R

TECHNISCHE UNIVERSITÄT DARMSTADT

```
library(gbm)
library(MASS) # for boston housing data

#separating training and test data
train=sample(1:506,size=374)

Boston.boost=gbm(medv ~ . ,data = Boston[train,],
                 distribution = "gaussian",
                 n.trees = 10000,
                 shrinkage = 0.01, # aka learning rate
                 interaction.depth = 4)

# look at variable importance
summary(Boston.boost)
#   var         rel.inf
# lstat     lstat 36.0378370
# rm           rm 32.0817888
# dis         dis  9.1929237
# crim       crim  5.2662981
# nox         nox  3.9236955
# age         age  3.6299790
# black     black  3.3031968
# ptratio ptratio  2.6644378
# tax         tax  1.4270161
# rad         rad  0.7865713
# indus     indus  0.7627721
# chas       chas  0.7511395
# zn           zn  0.1723443

# partial dependency plots
plot(Boston.boost,i="rm")
# price increases with #rooms
```

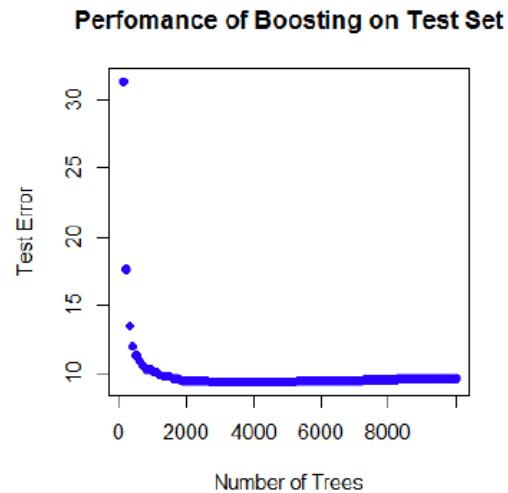


code credits: https://datascienceplus.com/gradient-boosting-in-r/

Gradient Boosting in R

TECHNISCHE UNIVERSITÄT DARMSTADT

```
# Test error as function of #trees

n.trees = seq(from=100 ,to=10000, by=100)

#Generating a Prediction matrix for each Tree
predmatrix<-predict(Boston.boost,Boston[-train,],
                    n.trees=n.trees, type="response")
dim(predmatrix)

#Calculating The Mean squared Test Error
test.error<-with(Boston[-train,],
                 apply((predmatrix-medv)^2,2,mean))
head(test.error)

#Plotting the test error vs number of trees

plot(n.trees , test.error ,
     pch=19,col="blue",
     xlab="Number of Trees",
     ylab="Test Error",
     main="Perfomance of Boosting on Test Set")
```



code credits: https://datascienceplus.com/gradient-boosting-in-r/

Gradient Boosting in R

TECHNISCHE UNIVERSITÄT DARMSTADT

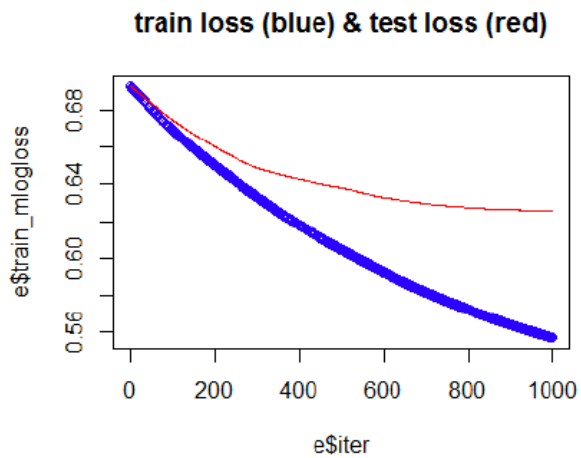
```
library(xgboost)
library(magrittr)
library(dplyr)
library(Matrix)

data <- read.csv("binary.csv", header = T)
names(data) # "admit" "gre"   "gpa"   "rank"
data$rank <- as.factor(data$rank)

# Partition data
set.seed(1234)
ind <- sample(2, nrow(data), replace = T, prob = c(0.8, 0.2))
train <- data[ind==1,]
test <- data[ind==2,]

# Create matrix - One-Hot Encoding for Factor variables
trainm <- sparse.model.matrix(admit ~ .-1, data = train)
head(trainm)
train_label <- train[,"admit"]
train_matrix <- xgb.DMatrix(data = as.matrix(trainm), label = train_label)

testm <- sparse.model.matrix(admit~.-1, data = test)
test_label <- test[,"admit"]
test_matrix <- xgb.DMatrix(data = as.matrix(testm), label = test_label)
```

code credits: https://drive.google.com/file/d/0B5W8CO0Gb2GGVUM4c2t6bnliQ1E/view
data: https://drive.google.com/file/d/0B5W8CO0Gb2GGVjRILTdWZkpJU1E/view
youtube: youtube: https://www.youtube.com/watch?v=woVTNwRrFHE&t=299s

Gradient Boosting in R

TECHNISCHE UNIVERSITÄT DARMSTADT

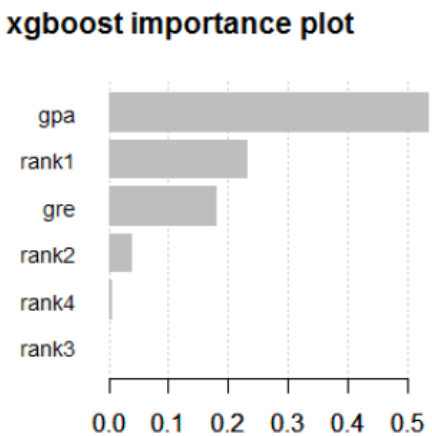
```
# Parameters
nc <- length(unique(train_label))
xgb_params <- list("objective" = "multi:softprob",
                   "eval_metric" = "mlogloss",
                   "num_class" = nc)
watchlist <- list(train = train_matrix, test = test_matrix)

# eXtreme Gradient Boosting Model
bst_model <- xgb.train(params = xgb_params,
                       data = train_matrix,
                       nrounds = 1000,
                       watchlist = watchlist,
                       eta = 0.001,
                       max.depth = 3,
                       gamma = 0,
                       subsample = 1,
                       colsample_bytree = 1,
                       missing = NA,
                       seed = 333)

# Training & test error plot
e <- data.frame(bst_model$evaluation_log)
plot(e$iter, e$train_mlogloss, col = 'blue')
lines(e$iter, e$test_mlogloss, col = 'red')

min(e$test_mlogloss)
e[e$test_mlogloss == 0.625217,]
```



Gradient Boosting in R

TECHNISCHE UNIVERSITÄT DARMSTADT

```
# Feature importance
imp <- xgb.importance(colnames(train_matrix),
                      model = bst_model)
print(imp)
xgb.plot.importance(imp)
title("xgboos importance plot")

# Prediction & confusion matrix - test data
p <- predict(bst_model, newdata=test_matrix)

pred <- matrix(p, nrow=nc) %>%
        t() %>%
        data.frame() %>%
        mutate(label = test_label,
               max_prob = max.col(., "last")-1)
               #find max pos in each row

table(Prediction=pred$max_prob, Actual=pred$label)
#             Actual
# Prediction  0  1
#          0 49 21
#          1  1  4
```



Was haben sie kennengelernt?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ensemble Methoden: Zusammen sind schwache Klassifiaktoren mächtig
- Eine der am meisten benutzten Techniken des Maschinellen Lernens. Wird von vielen Gewinnern von ML-Wettbewerben benutzt. Die meisten ML-Systeme in der Praxis basieren auf Ensemble Methoden
- Bagging und Boosting
- Sie können Algorithmen für Random Forests, AdaBoost und Gradient Boosting beschreiben
- Sie können die Konvergenzeigenschaften von AdaBoost diskutieren.
- Sie kennen die R packages `randomForest`, `gbm` und `xgboost`