Advanced Studies in Applied Statistics (WBL), ETHZ
Applied Multivariate Statistics
Spring 2018, Week 12

Lecturer: Beate Sick

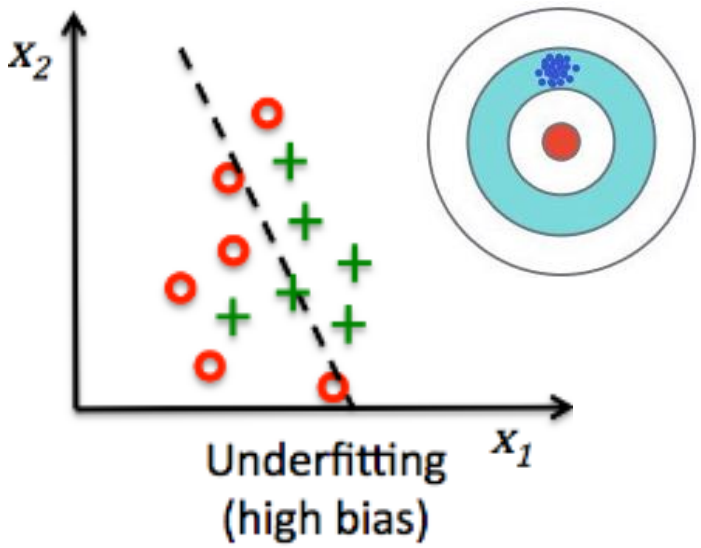
sickb@ethz.ch

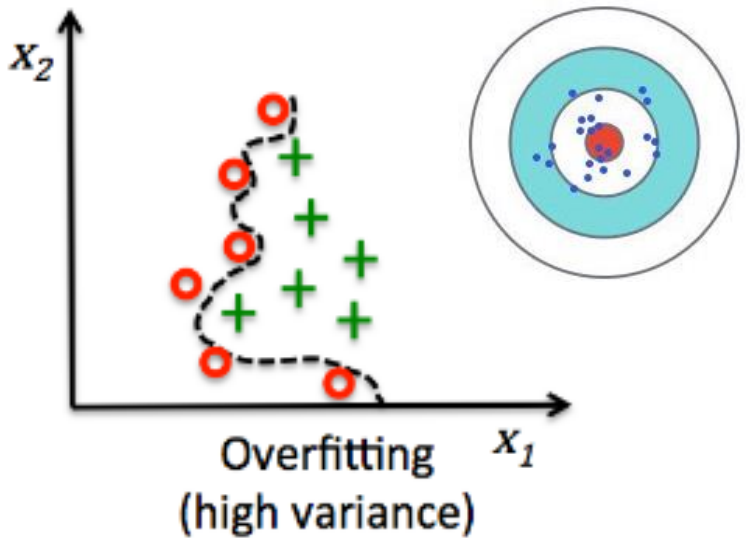
Remark: Much of the material have been developed together with Oliver Dürr for different lectures at ZHAW.

Topics of today

- **The concept of Bias and Variance of a classifier**
  - **Recap concepts of over- and under-fitting**
- **Bagging as ensemble method to reduce variance**
  - **Bagging**
  - **Random Forest**
- **Boosting as ensemble method to reduce bias**
  - **Adaptive Boosting**
  - **Gradient boosting**
- **How to get the best prediction model?**

The concept of Bias and Variance of a classification model





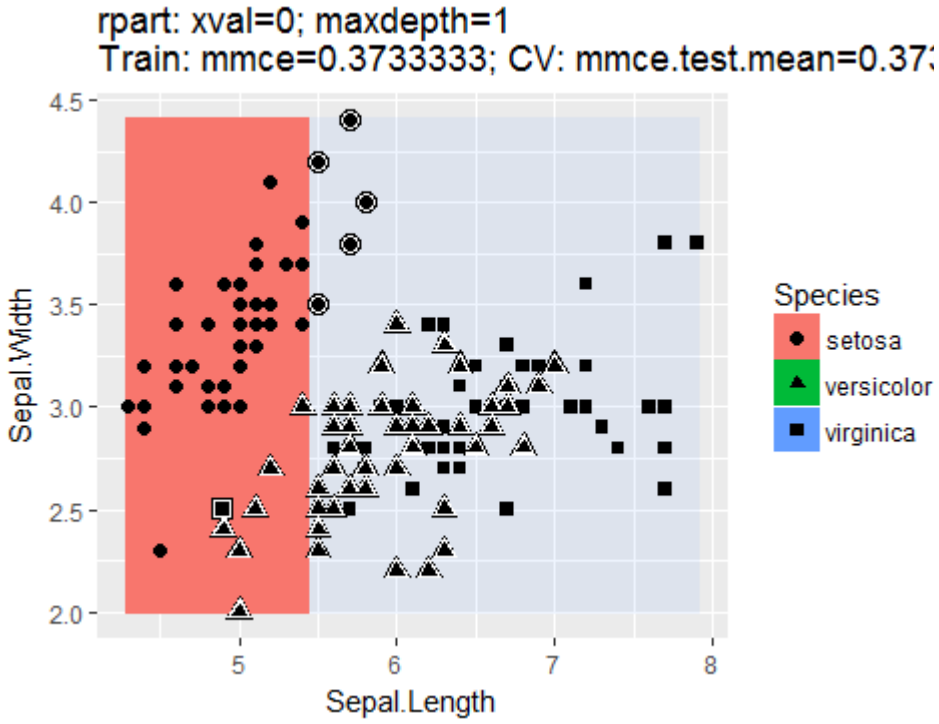
A underfitting classification model

- is not flexible enough
- quite many errors on train data and systematic test error (high bias)
- will not vary much if new train data is sampled from population (low variance)

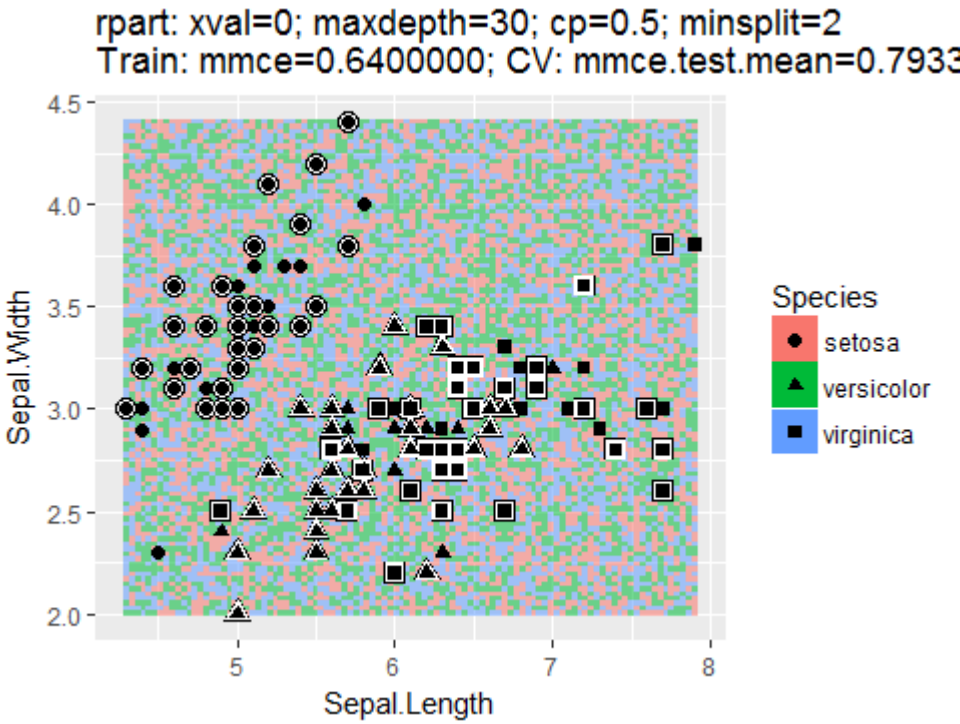
A overfitting classification model

- is too flexible for data structure
- few errors on train set and non-systematic test errors (low bias)
- will vary a lot if fitted to new train data (high variance)

Examples for underfitting or overfitting tree models

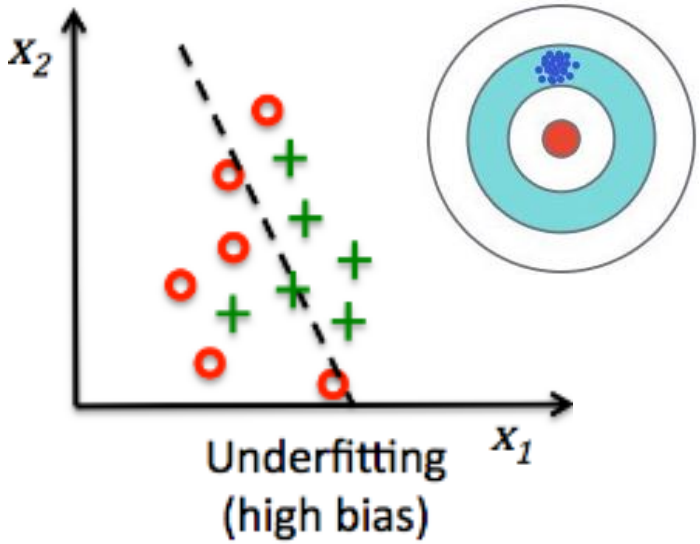


partitioning resulting
from an underfitting tree

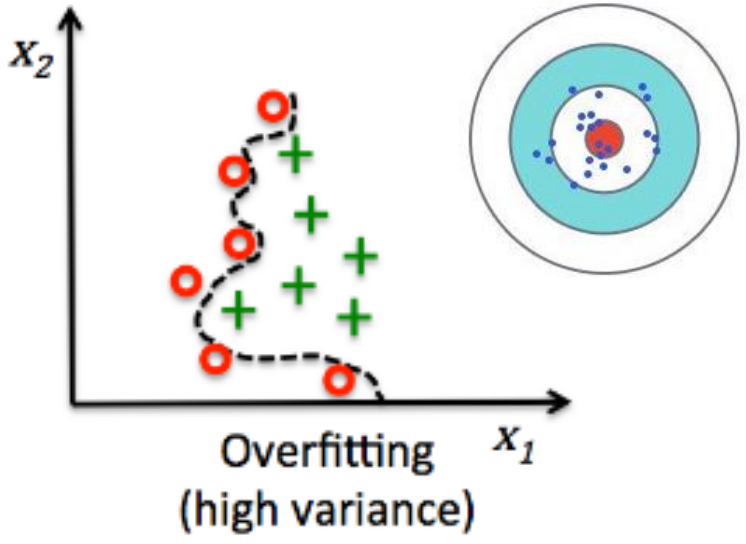


partitioning resulting
from an overfitting tree

Use ensemble methods to fight under and overfitting



Underfitting
(high bias)



Overfitting
(high variance)

fight the deficits of the single model by

Adaptive boosting

Bagging

improve ensemble approach further by

Gradient boosting
(fights under- and overfitting)

Random Forest
in case of tree models

Ensemble methods are the cure!

http://www.spiegel.de/spam/humor-fuer-leute-mit-humor-nel-ueber-schwarmintelligenz-a-842671.html

Bagging
&
Random Forest

Bagging as ensemble of parallel fitted models

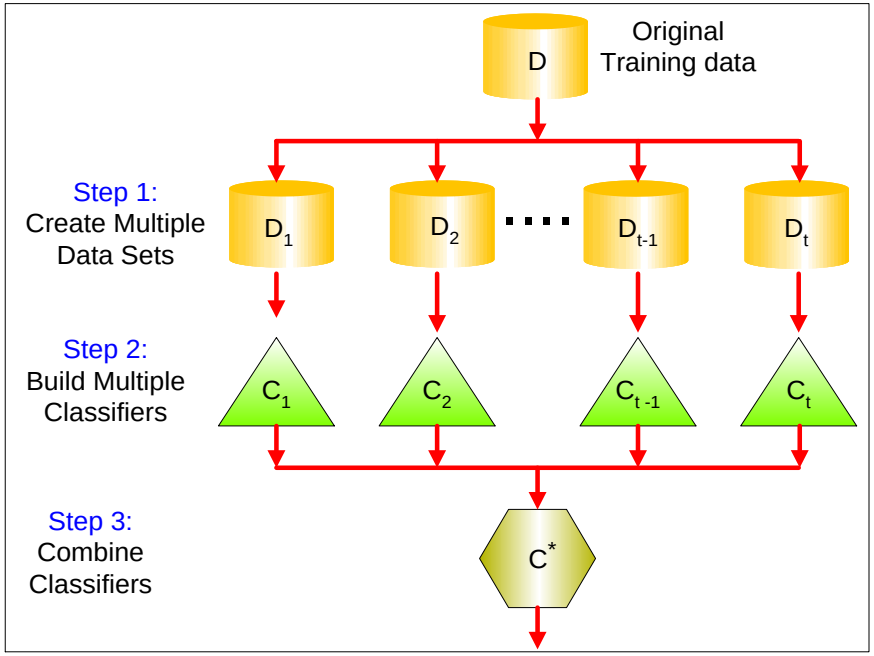
Each classifier tends to overfit its version of training data.

Bagging: bootstrapping and averaging

1) Fit flexible models on different bootstrap samples of train data

2) Minimize Bias by using flexible models and allow for overfitting

3) Reduce variance by averaging over many models



Remarks: highly non-linear estimators like trees benefit the most by bagging
If model does not overfit the data bagging does not help nor hurt.

Recap: Why does bagging help for overfitting classifiers?

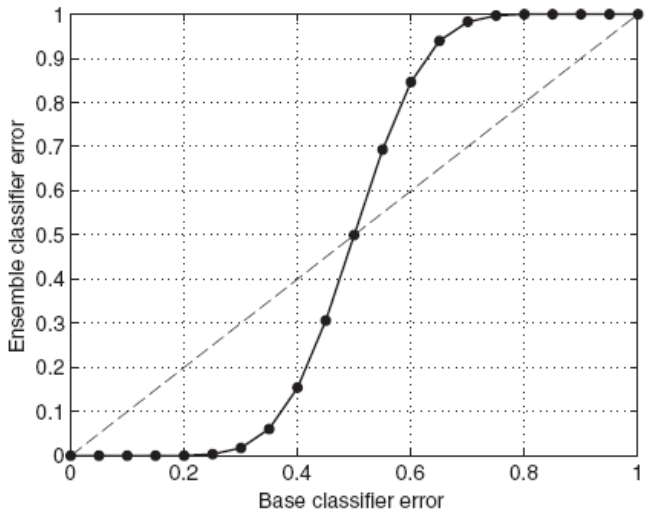
Suppose there are 25 base overfitting classifiers

Each classifier has error rate, $\varepsilon = 0.35$

Assume classifiers are independent

Probability that the ensemble classifier makes a wrong prediction (that is if >50%, here 13 or more classifiers out of 25 make a wrong prediction)

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

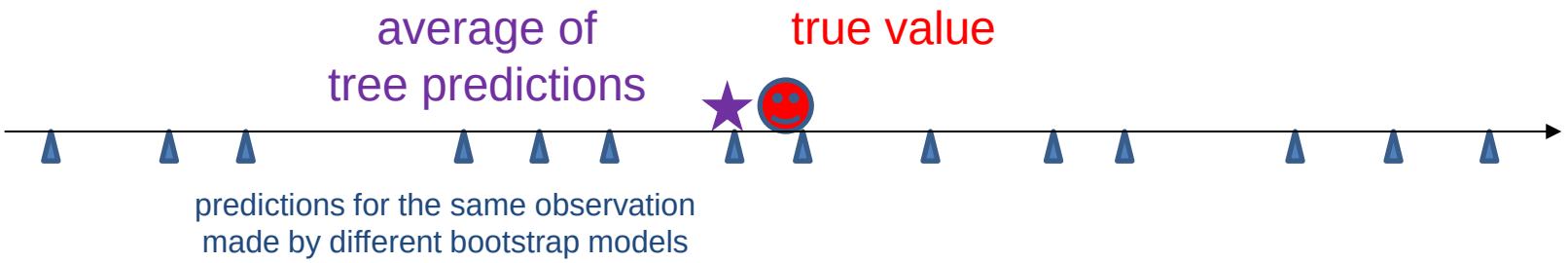


=> Ensembles are only better than one classifier, if each classifier is better than random guessing!

Source: Tan, Steinbach, Kumar

Recap: Why does bagging help for overfitting regression models?

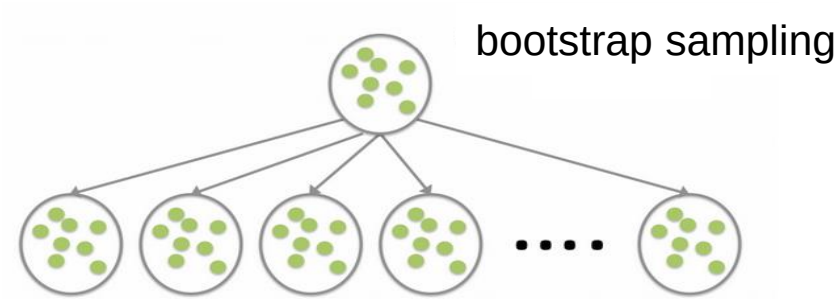
- Suppose there are n=25 very flexible regression models
  - All models are flexible -> the trees have no or only small bias
  - All models are flexible -> the trees have high variance
  - According to the Central Limit Theorem the average of the predictions of n regression models have the same expected value as the single model but a standard deviation which is reduced by a factor of $\sqrt{n} = \sqrt{25} = 5$.

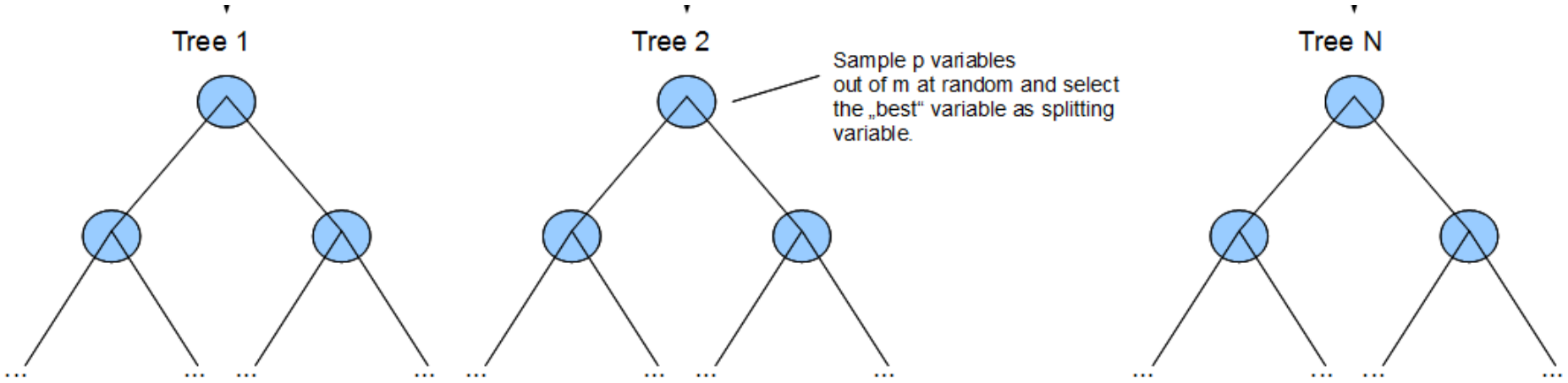


Recap: Random Forest improves Bagging idea further

1) take bootstrap sample

2) grow on each bootstrap sample a tree, but use the additional tweak to sample from the predictor sets at each split before choosing among these predictors -> uncorrelate trees



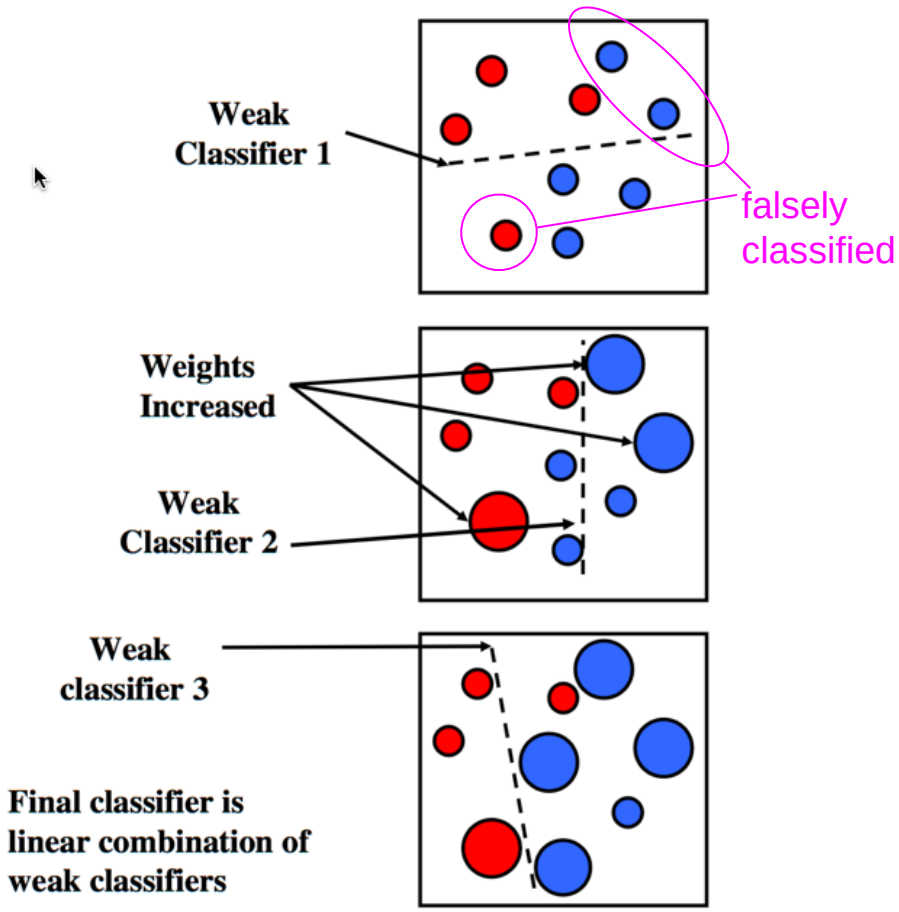


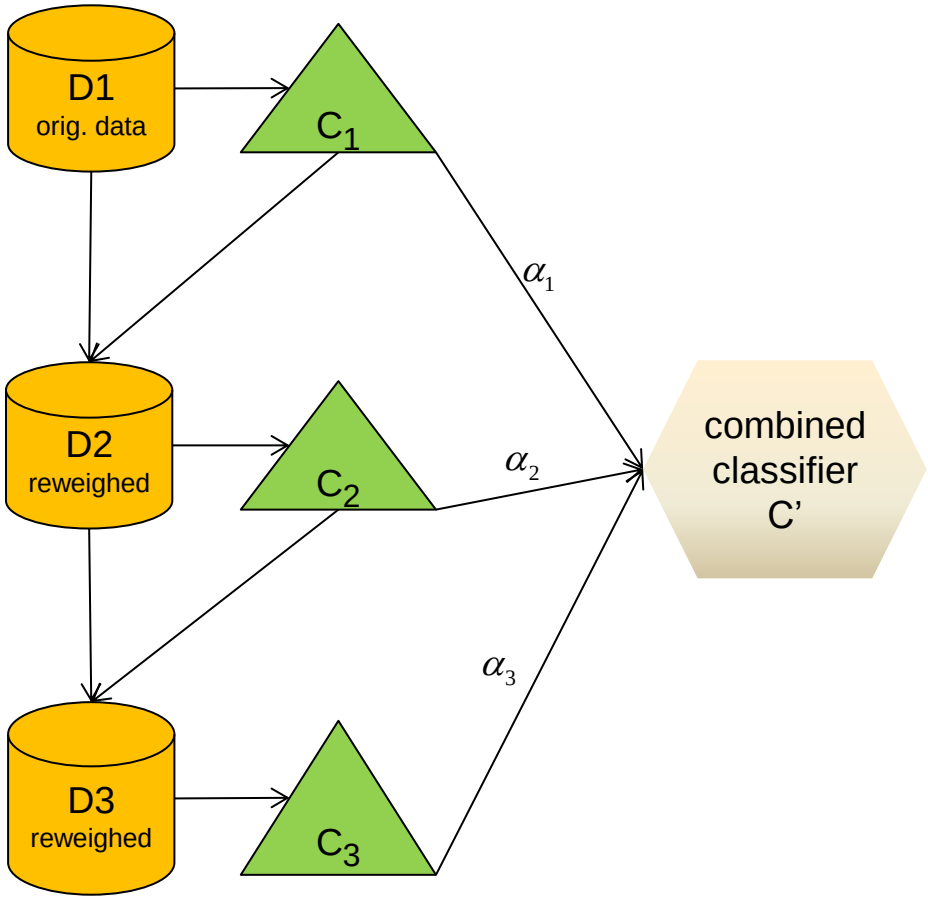
Adaptive Boosting

Adaptive Boosting as ensemble of sequentialy fitted models

We use in each step a simple (underfitting) model to fit the current version of the data.

After each step those observations get up-weighted, that were misclassified.

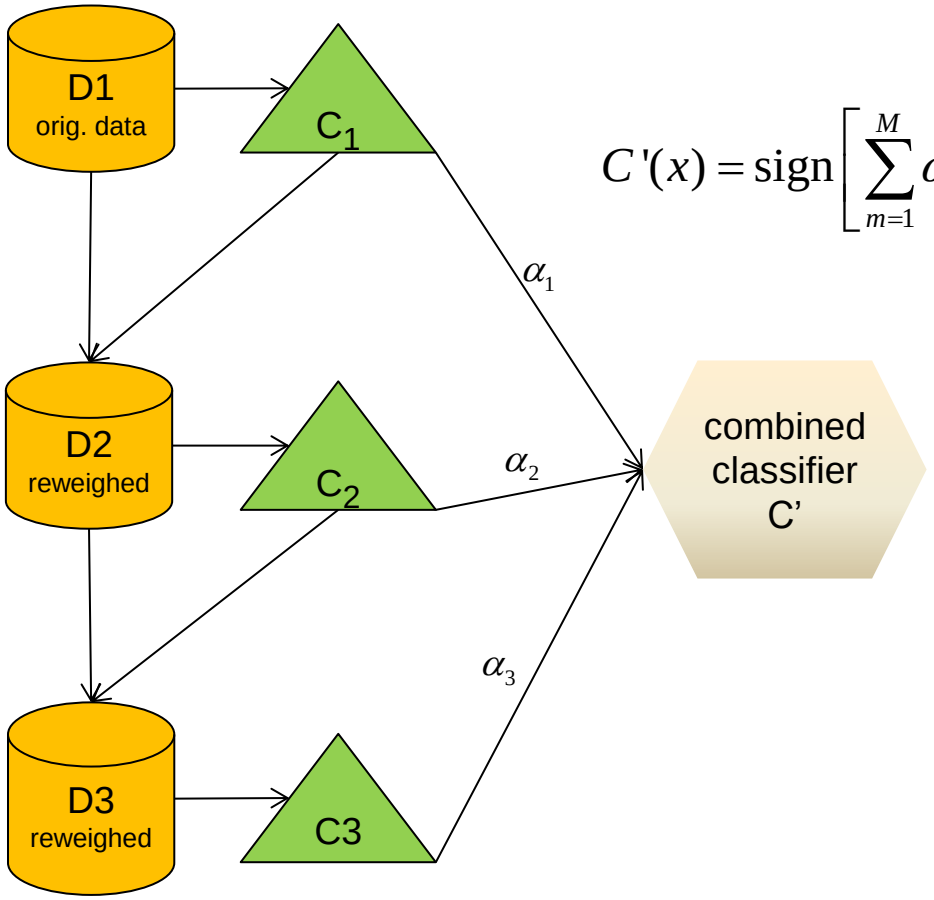




left fig credicts:
http://vinsol.com/blog/2016/06/28/computer-vision-face-detection/

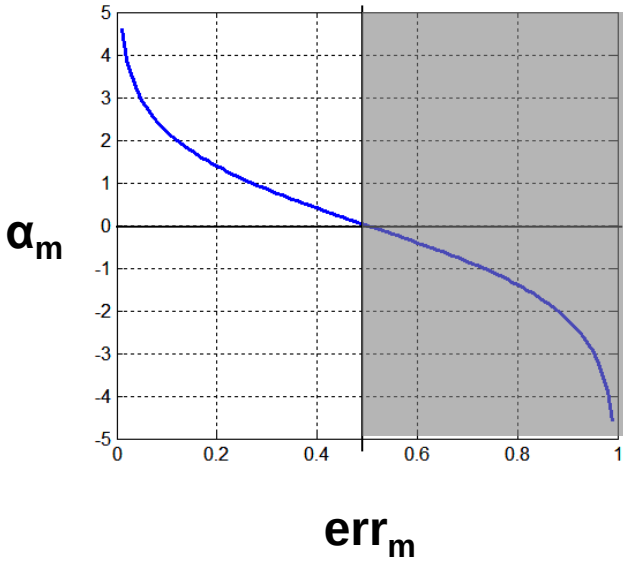
Adaptive Boosting as weighted average of sequential models

The final classifier C' is a weighted average of all sequential models C_m, where the model-weights α_m are given by the misclassification rate err_m of the model C_m taking into account the observation-weights of the used reweighted data set D_m.



$$C'(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m \cdot C_m(x)\right]$$

C_m error small ➔large weight α_m



Details of Ada Boost Algorithm

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 $F_M(x) = sign\sum_{m=1}^{M} \alpha_m f_m(x)$

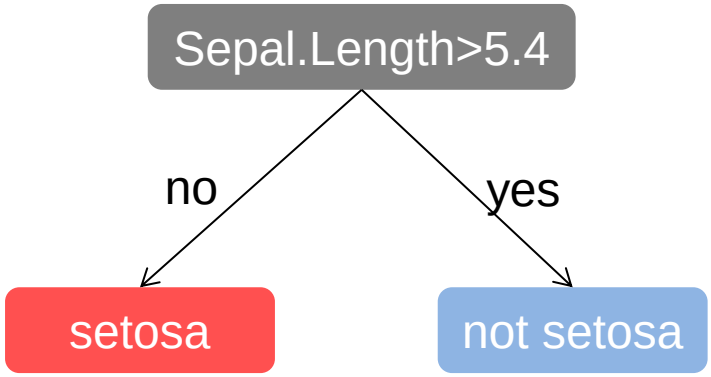
Remark: One can show (see ELS chapter 10.4, p.343) that the reweighting algorithm of AdaBoost is equivalent to optimizing an exponential loss.

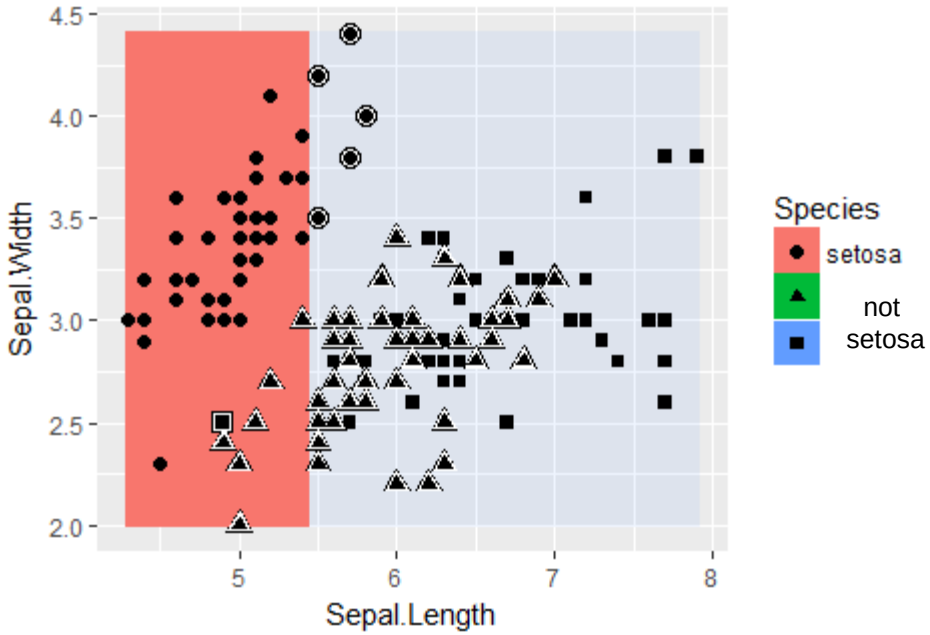
Ada Boost in simple words

- Fit an additive model $\sum_{m=1}^{M} \alpha_m \cdot C_m$ (ensemble) in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Adaboost,“shortcomings” are identified by high-weight data points.

Stumps are often used as simple tree models

Stumps have only one split and can therefore use only 1 feature.



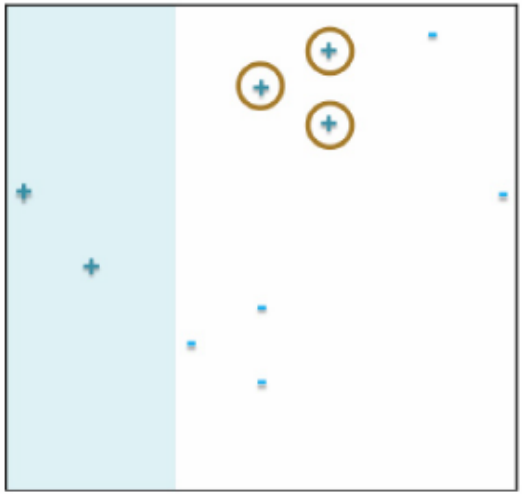


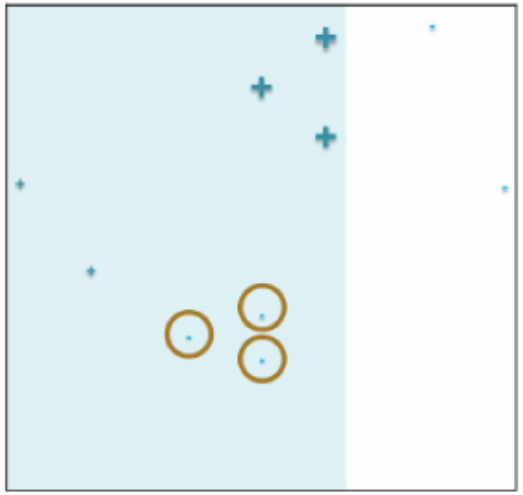
Adaptive boosting (Adaboost) relies often on "stumps" as underfitting tree classifiers

C_1

$\text{err}_1 = 0.30$

$\alpha_1 = 0.42$





C_2

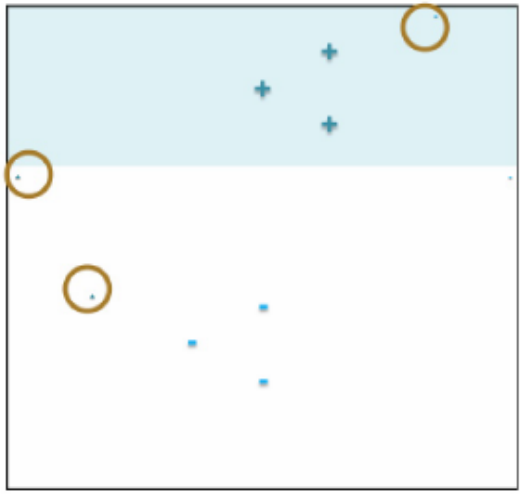
$\text{err}_2 = 0.21$

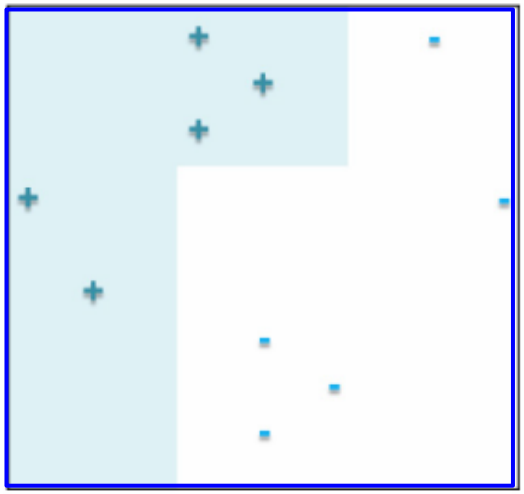
$\alpha_2 = 0.65$

C_3

$\text{err}_3 = 0.14$

$\alpha_1 = 0.92$





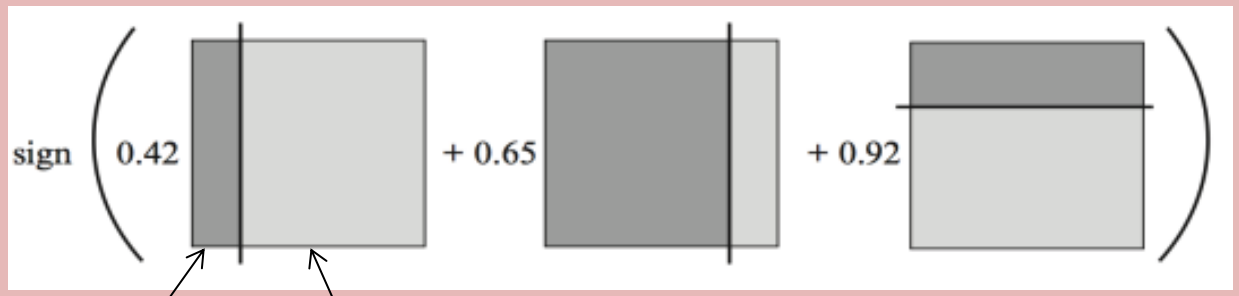
$$C' = \sum_{m=1}^{3} \alpha_m \cdot C_m$$

$$= 0.42 \cdot C_1 + 0.65 \cdot C_2 + 0.92 \cdot C_3$$

By averaging simple classifiers we can get a much more flexible classifier (which might overfit).

Example (You Turn)

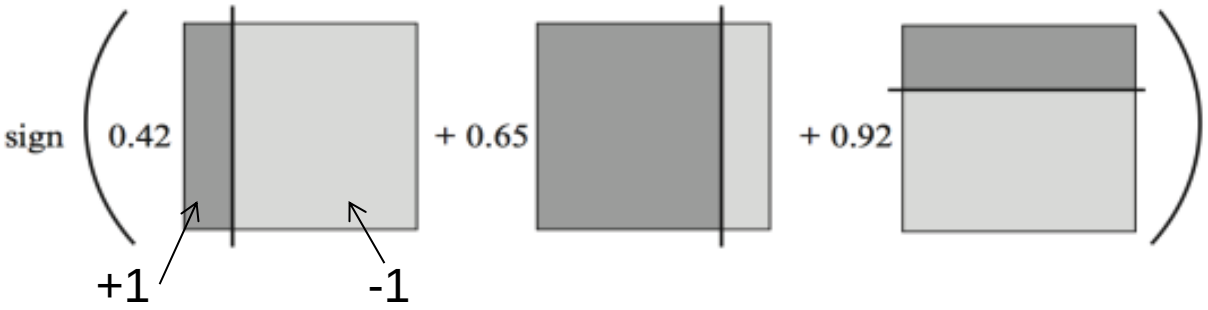
$$F_3(x) = \text{sign}\left[\sum_{m=1}^{3} a_m f_m(x)\right]$$

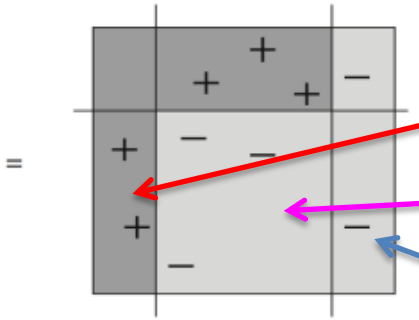


Example (You Turn)

Lösung

$$F_3(x) = \text{sign}\left[\sum_{m=1}^{3} a_m f_m(x)\right]$$

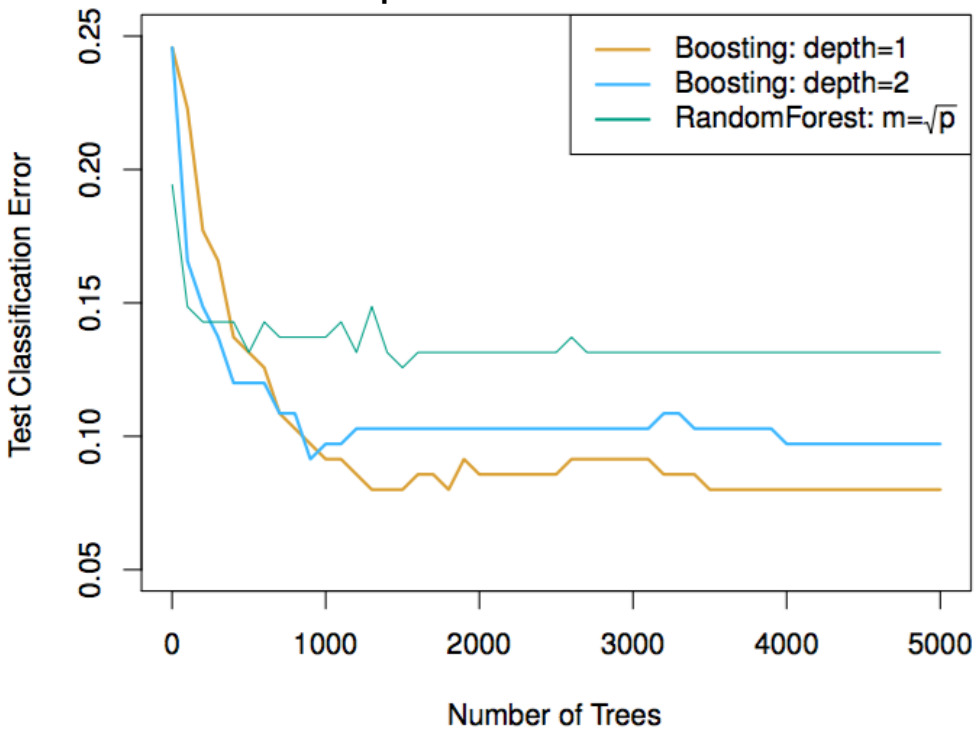




> 0.4+0.65-0.92
[1] 0.13 # +
> -0.42+0.65-0.92
[1] -0.69 # -
> -0.42-0.65-0.92
[1] -1.99 # -

Performance of Boosting / Diagnostic Setting

Look at a specific case



Boosting most frequently used with trees (not necessary).

Trees are typically only grown to a certain depth – often 1 or 2.

Diagnostic: Significant interaction if depth 2 works better than depth 1.

Gradient Boosting

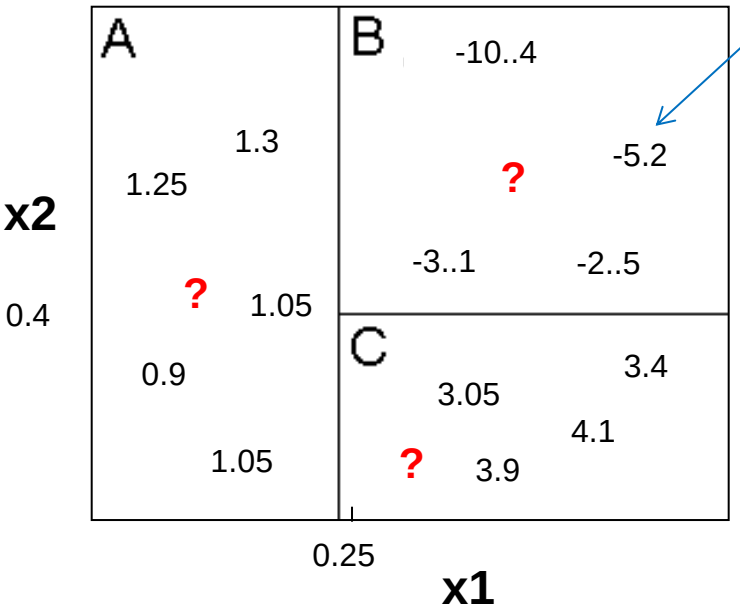
Where do we go? Gradient boosting in simple words

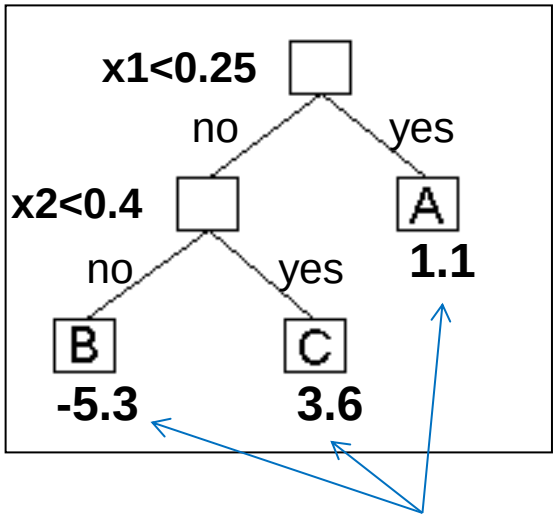
- Fit an additive model $\sum_{m=1}^{M} \alpha_m \cdot C_m$ (ensemble) in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Gradient Boosting, “shortcomings" are identified by **gradients of the loss**
- Recall: in Adaboost,“shortcomings” are identified by high-weight misclassified data points.

Recall: regression tree with 2 predictors

numbers indicate values
of **continuous outcome**
y

Here, we have 2 predictors:

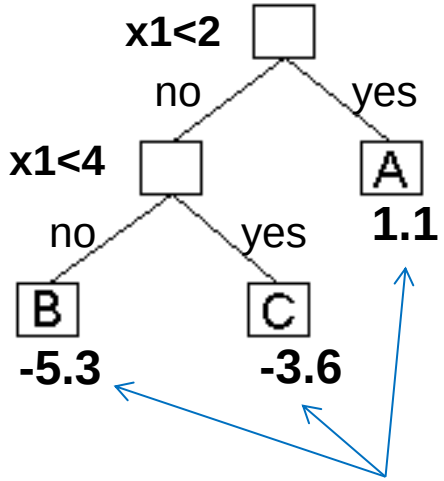




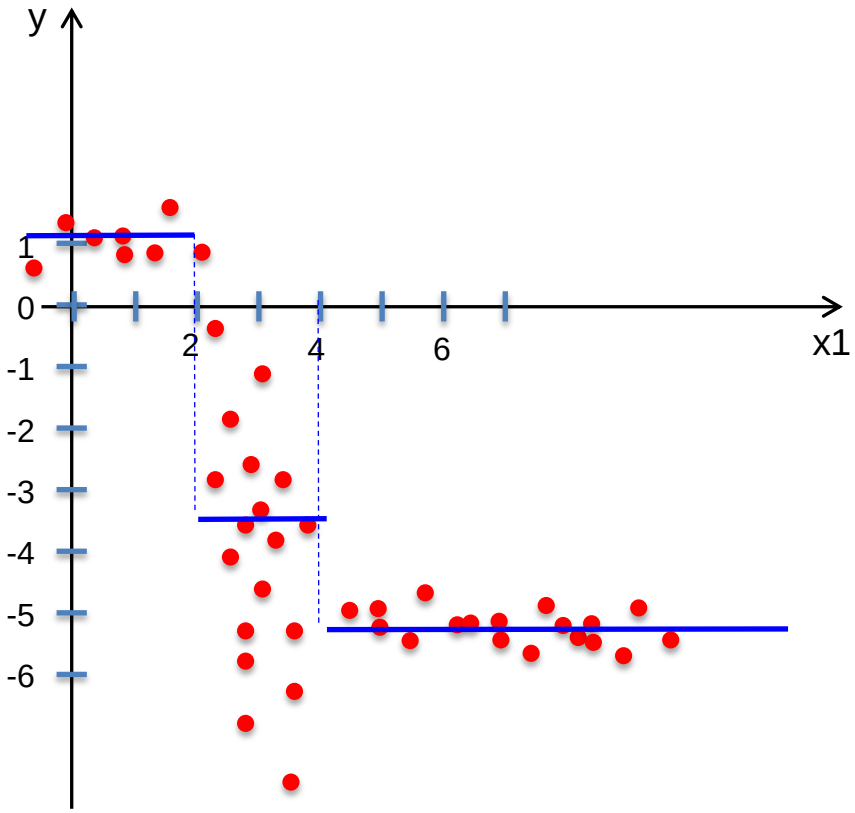
Per partition predict one outcome value
Given by the mean value of the observed data in this region

Score: MSE (mean squared error) $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$

Regression tree with 1 predictor

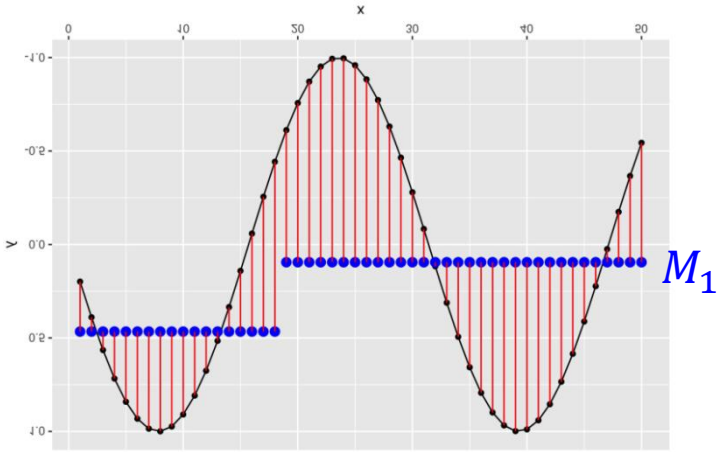


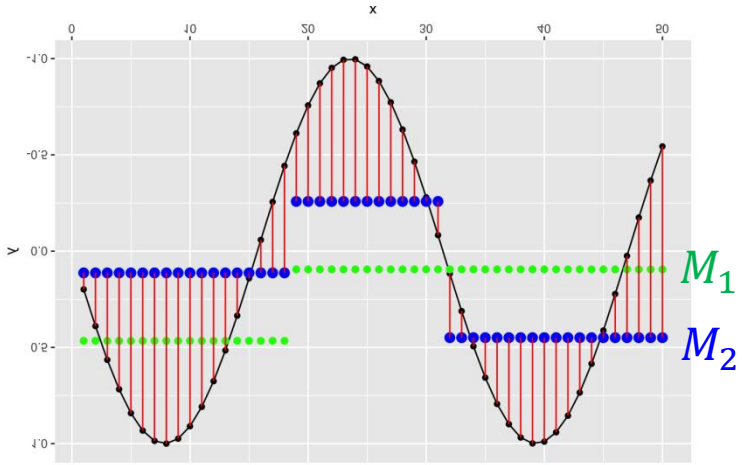
Per partition predict one outcome value
Given by the mean value of the observed data in this region



Start with a regression example of gradient boosting

First see in a simple example how it works and later see why this is gradient boosting





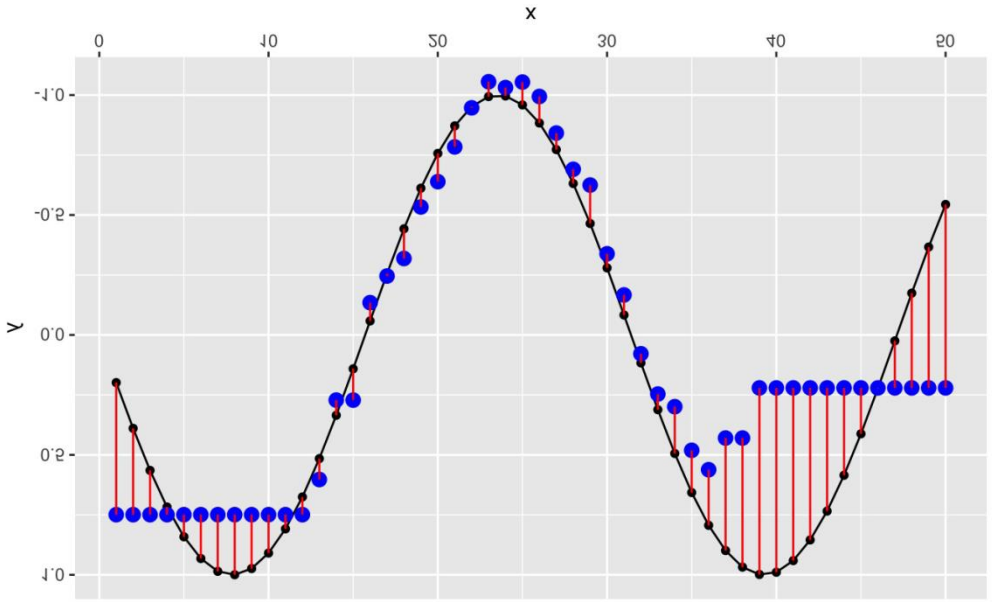
1) Fit a shallow regression tree T_1 to the data; the first model fits the data:
   $M_1 = T_1$
   the shortcomings of the model are given by the residuals$= y - \hat{y}$.

2) Fit a tree T_2 to the residuals; the second model is: $M_2 = M_1 + \gamma T_2$ where γ is optimized so that M_2 is best fit to data. We regularize the learning process by introducing a learning rate $\eta \in (0,1)$
   $$M_2 = M_1 + \eta\gamma T_2$$

3) Again fit a tree to the residuals … and continue until the combined model fits.

figure credits: dataCamp

Boosting model is weighted average of sequential models



final Model M:

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

update by adding stage-wise modeled residuals

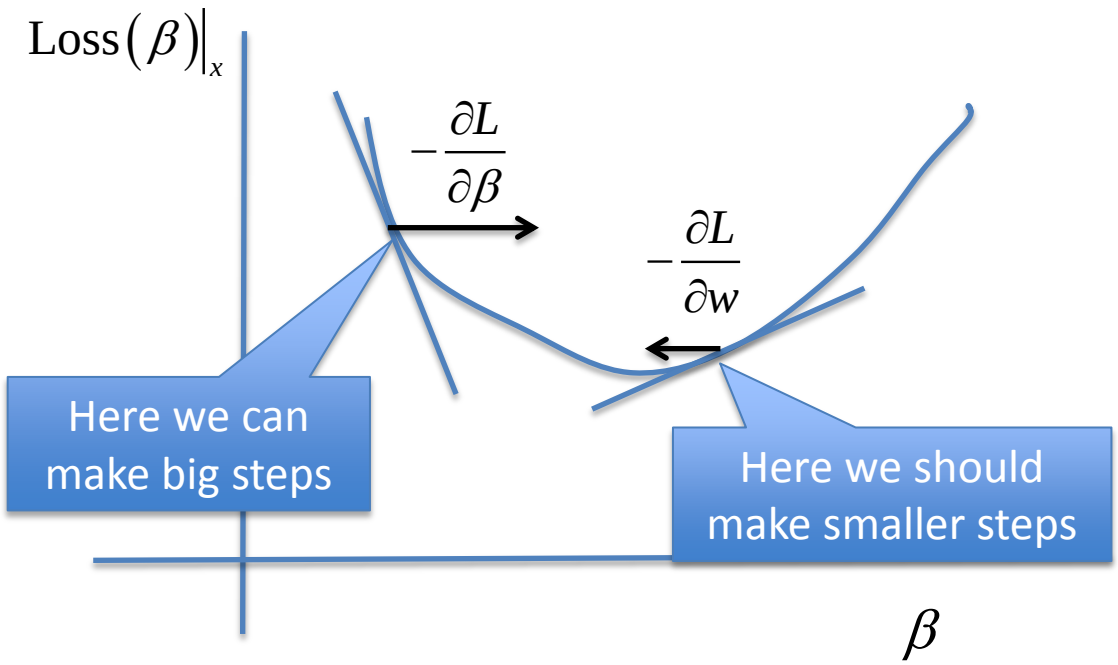
The closer the learning rate η is to 1 the faster is the learning and the higher the risk for overfitting.

Side track: Loss or Cost function in linear regression

We know that in linear Regression, we determine the parameter β_i such that the likelihood is maximized or (equivalently) the sum of the squared residuals is minimized

The loss function in linear regression is: $\text{Loss}(\mathbf{x}, \boldsymbol{\beta}) = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} \left(y_i - \left(\beta_0 + \beta_1 \cdot x_i \right) \right)^2$

There is a closed solution but we can also find minimum by gradient descent:



$-\frac{\partial L}{\partial \beta}$

- Points to the downward direction.
- The magnitude is proportional to the slope

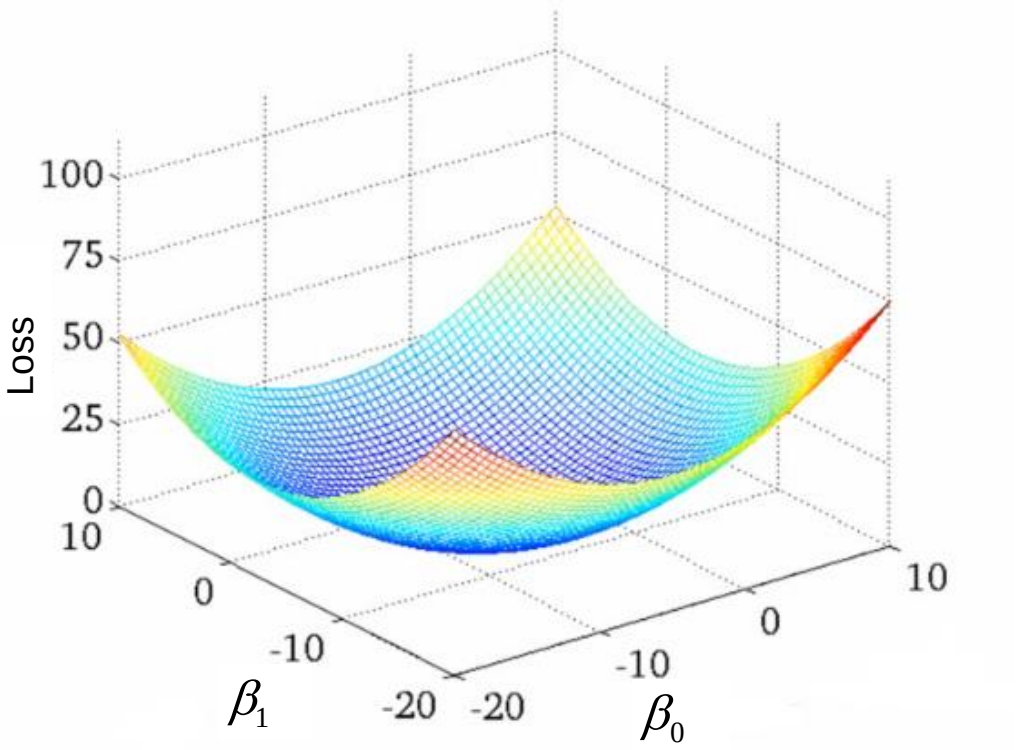
Iterative update

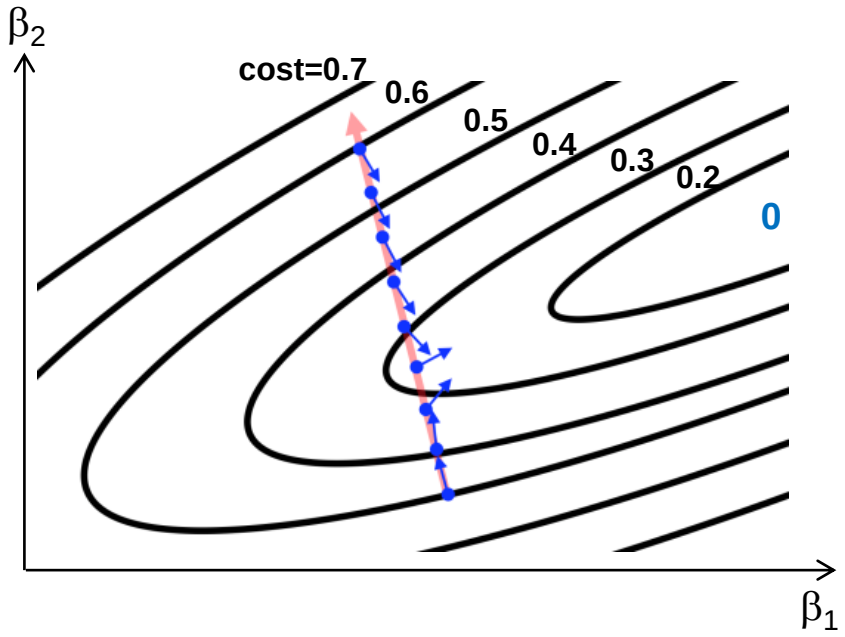
$$\beta^{(t)} = \beta^{(t-1)} - \eta \frac{\partial L(x, \beta^{(t-1)})}{\partial \beta}$$

step downhill

Loss or Cost landscape in case of convex 2D loss

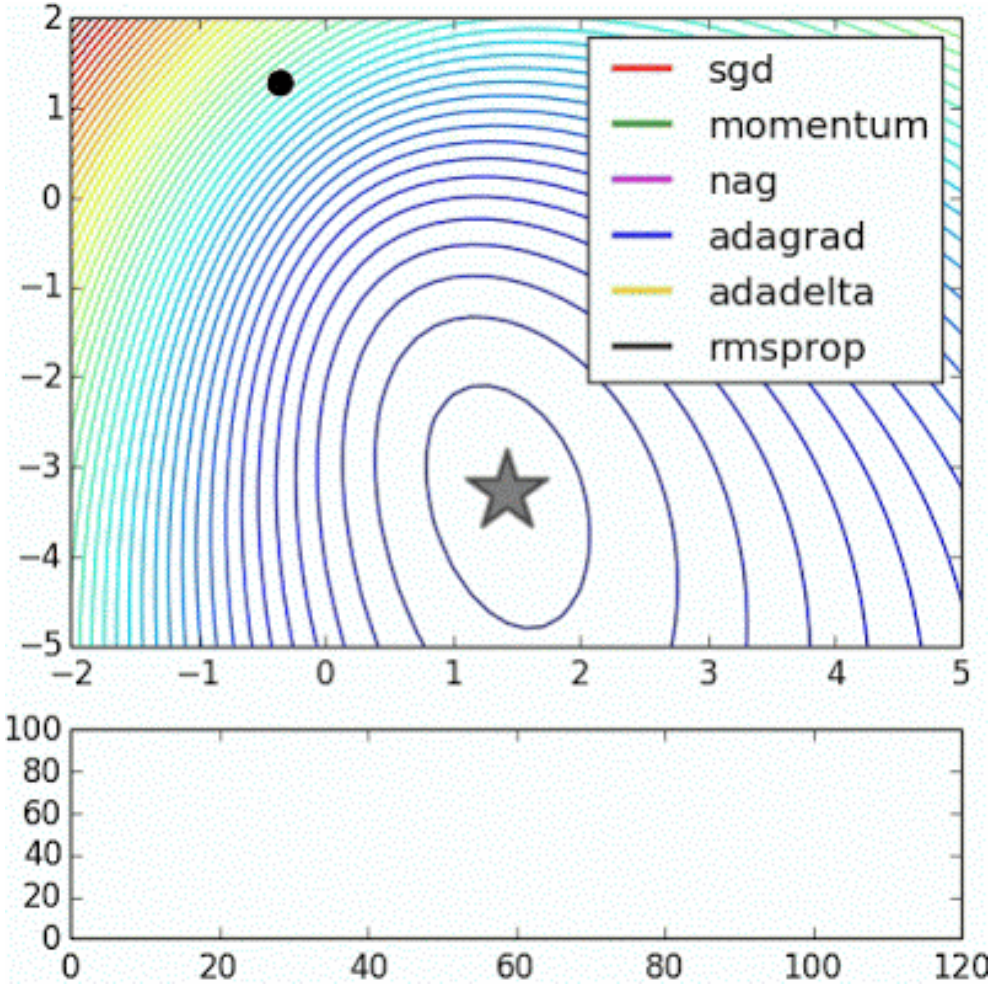
- 2 equivalent representations





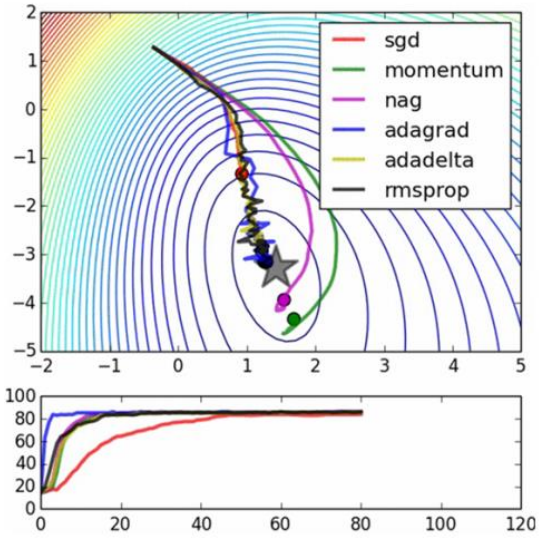
Remark: if we take too large steps, we could “step over the minimum”.
-> the learning rate aka shrinkage is one of the most important tuning parameter

Iteratively optimize the parameter to find best model



sgd:
(stochastic) gradient descent is most famous optimization method

Other methods are more efficient by adapting the learning rates $\eta^{(t)}$ for each parameter w_i based on the past couple of gradients.



Animation: http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Side track: Connection between gradients and residuals

We want to minimize the cost or loss function by adjusting the parameter and with that the fitted values $f(x_i)$ - notice for a given setting of parameter and x values $f(x_i)$ are just some numbers and we can treat them as parameters of the loss

the gradient $\frac{\partial C}{\partial f(x)}$ tells us (like the residual) in which direction the modeled value $f(x)$ should be changed to improve the fit.

$$\text{Loss or Cost C} = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$$

$$\frac{\partial C}{\partial f(x_i)} = -\left(y_i - f(x_i)\right) = -r_i$$

With a squared loss the residuals are the negative gradients g of the loss!

$$r_i = -\frac{\partial C}{\partial f(x_i)} = -g(x_i)$$

Formulate the boosting algorithm in terms of gradients

1) Fit a shallow regression tree T_1 to the data; the first model is: $M_1 = T_1$
   the shortcomings of the model are given by the negative gradients.

2) Fit a tree T_2 to the negative gradients; the second model is: $M_2 = M_1 + \eta\gamma T_2$
   where γ is optimized so that M_2 is best fit to data.

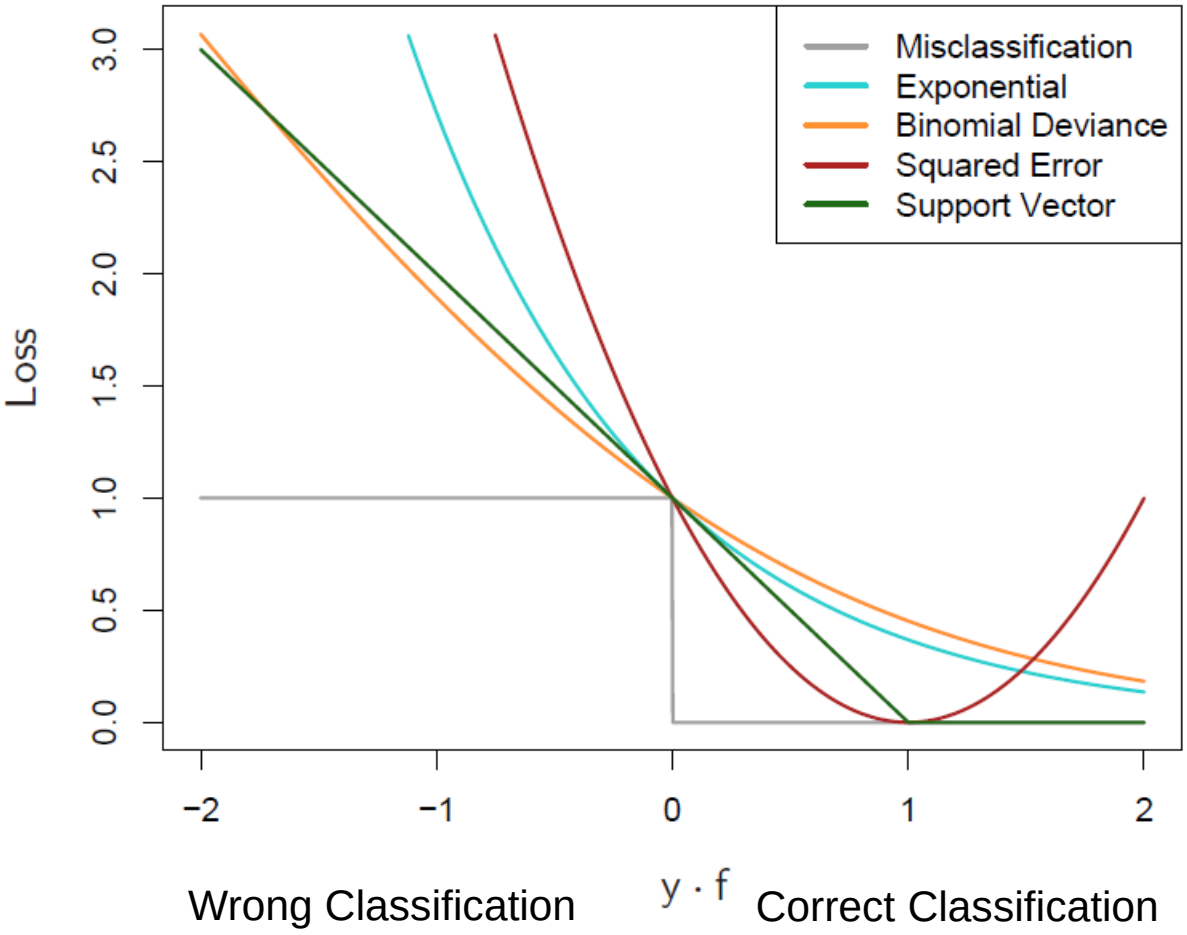
3) Again fit a tree to the negative gradients, continue until the combined model
   $M = M_1 + \eta\sum\gamma_i T_i$ fits.

update first fit M_1 by adding the stage-wise modeled negative gradients

So we are actually updating our model using gradient descent!

The benefit of formulating this algorithm using gradients is that it allows us to consider other loss functions and derive the corresponding algorithms in the same way.

Commonly used loss functions for gradient boosting



Misclassification

$$L(y,F) = I[y \neq sign(F)]$$

Exponential / AdaBoost

$$L(y,F) = \exp(-yF)]$$

Binomial Deviance

$$L(y,F) = \log(1+\exp(-2yF))$$

Quadratic / L2-Boost

$$L(y,F) = (y-F)^2$$

SVM

$$L(y,\mathrm{F}) = y \cdot (1 - y \cdot F)$$

Remark: One can show (see ELS chapter 10.4, p.343) that optimizing the exponential loss is equivalent to the reweighting algorithm of AdaBoost.

General gradient boosting procedure

We pick a problem specific differential Loss function.

We start with a initial (underfitting) model $M = M_1$

Iterate and do at each stage the following until converge:

a) calculate negative gradients

remember that for a given setting of parameter and x values $M(x_i)$ are just some numbers and we can treat them as parameters of the loss and the gradient tells us (like the residual) in which direction this modeled value should be changed to improve the fit.

$$-g(x_i) = -\frac{\partial\ \mathrm{Loss}\left(\mathrm{y}_i, \mathrm{M}(x_i)\right)}{\partial M(x_i)}$$

b) fit a model T to the negative gradients $-g(x_i)$

$$M = M_1 + \eta \sum \gamma_k T_k$$

c) Get an updated model by adding a fraction of T

Remarks: GB can easily overfit and needs to be regularized. GB based on trees cannot extrapolate

Historical View on boosting methods

- Adaptive Boosting (adaBoost) *Algorithm*:
  - Freund & Schapire, 1990-1997
  - An *algorithmic* method based on iterative data reweighting for two class classification.

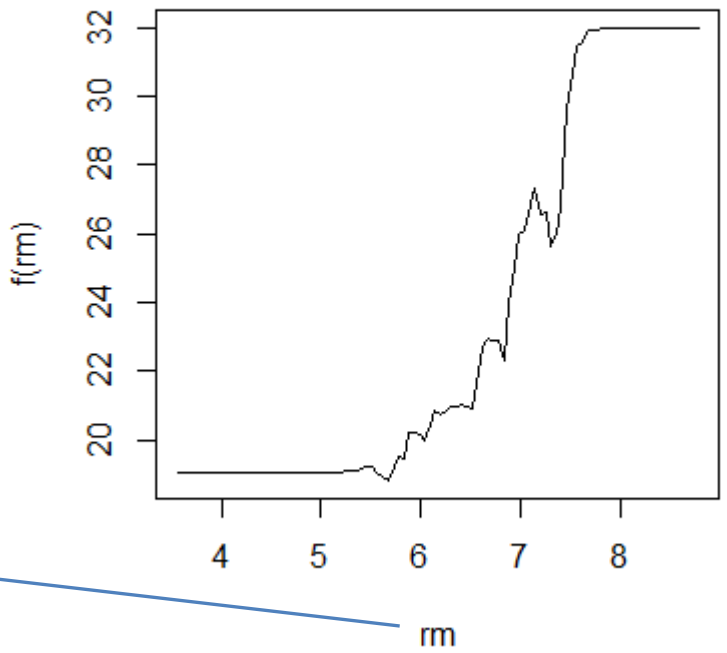
- Gradient Boosting (GB):
  - Breiman (1999) Sound theoretical framework using iterative minimization of a loss function (opening the door to > 2 class classification and regression)…
  - Friedman/Hastie/Tibshirani (2000) Generalization to a variety of loss functions

- Extreme Gradient Boosting (xgboost - a particular implementation of GB)
  - Tianqi Chen (2014 code, 2016 published arXiv://1603.02754)
  - Theory similar to GB (often trees), but more emphasis on regularisation
  - Much better implementation (distributed, 10x faster on single machine)
  - Often used in Kaggle Competitions as part of the winning solution

Boosting in R

Gradient boosting in R's gbm package

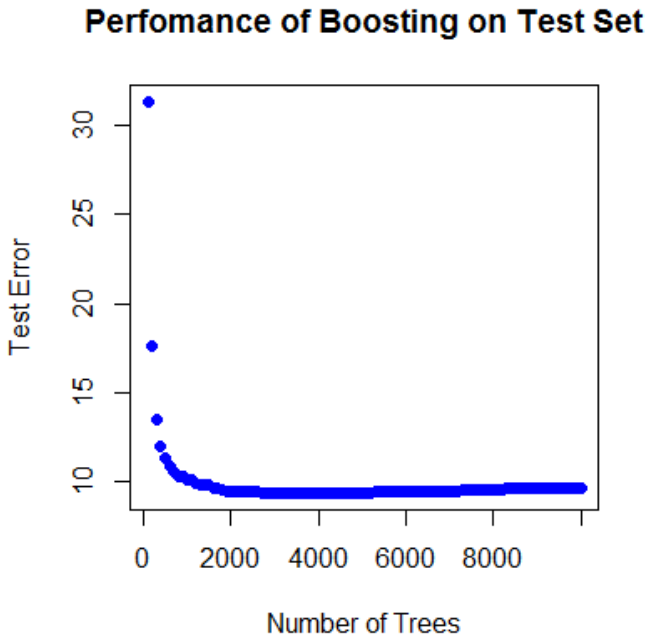
```
library(gbm)
library(MASS) # for boston housing data

#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/

Check on test error in gradient boosting in R's gbm

```
# 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/

How to tune an extreme gradient boosting model?

The (three) most important parameter for Tree Booster:

- eta aka learning rate: Default [default=0.3][range: (0,1)]
  can be lowered to fight overfitting
- gamma [default=0][range: (0,Inf)] minimum loss reduction required for further split. Can be increased to fight overfitting with shallow trees.
- max_depth maximum depth of a tree [default=6][range: (0,Inf)].
  Can (often should) be lowered to prevent overfitting.

- subsample subsample ratio of the training instance. Can be lowered to fight influence of outliers (and decorrelate trees).
- Cross validation should be used to tune hyper-parameter. Best via a multivariate-grid search.
- Watchlist is helpful for simple (univariate) tuning

https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/

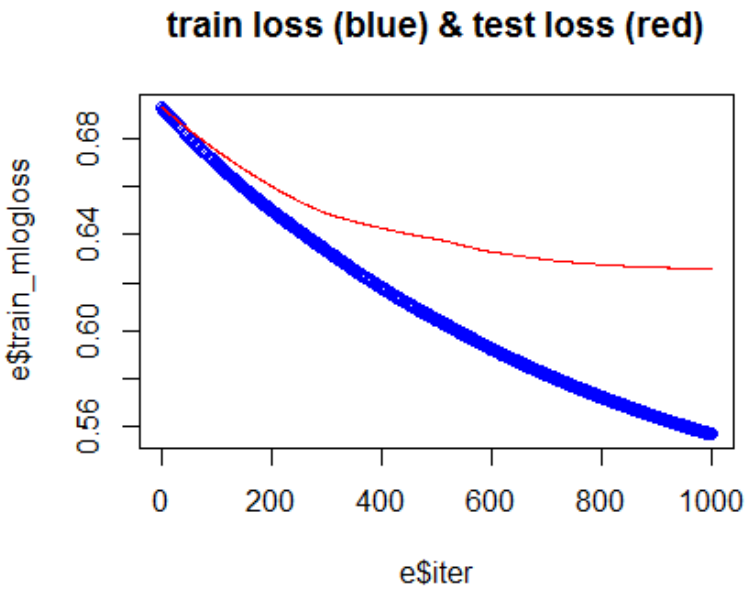
Extreme Gradient boosting in R's xgboost package

```
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

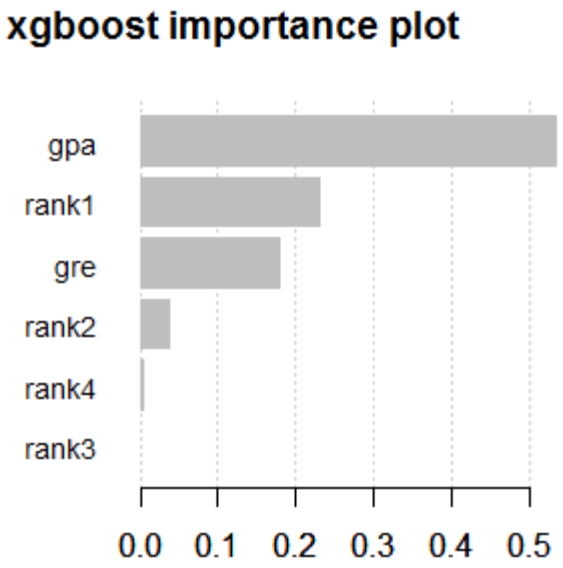
Extreme Gradient boosting in R's xgboost package

```
# 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,]
```

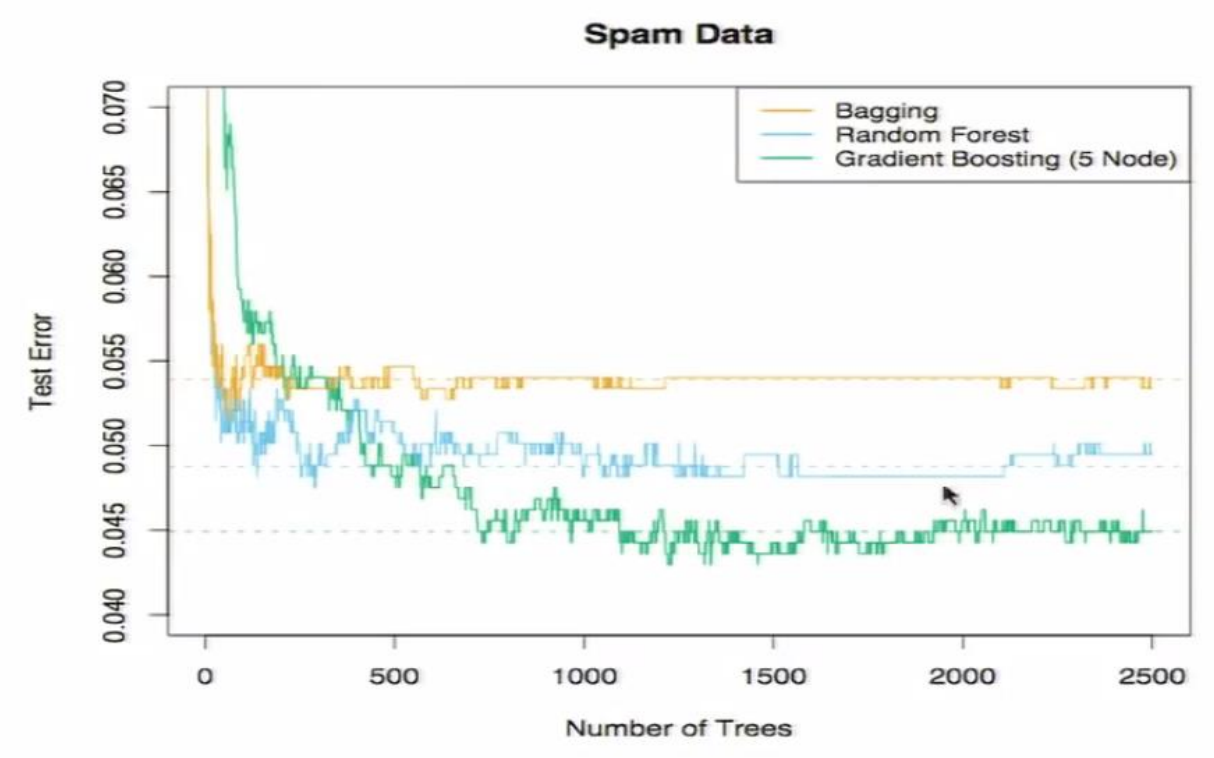


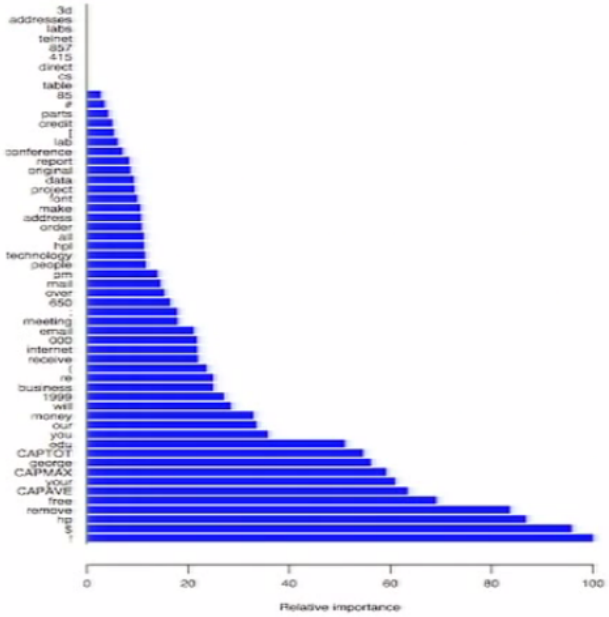
Extreme Gradient boosting in R's xgboost package

```
# 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
```



Comparison of bagging, RF, and gradient boosting





In many applications the test prediction performance increases from bagging to RF to gradient boosting

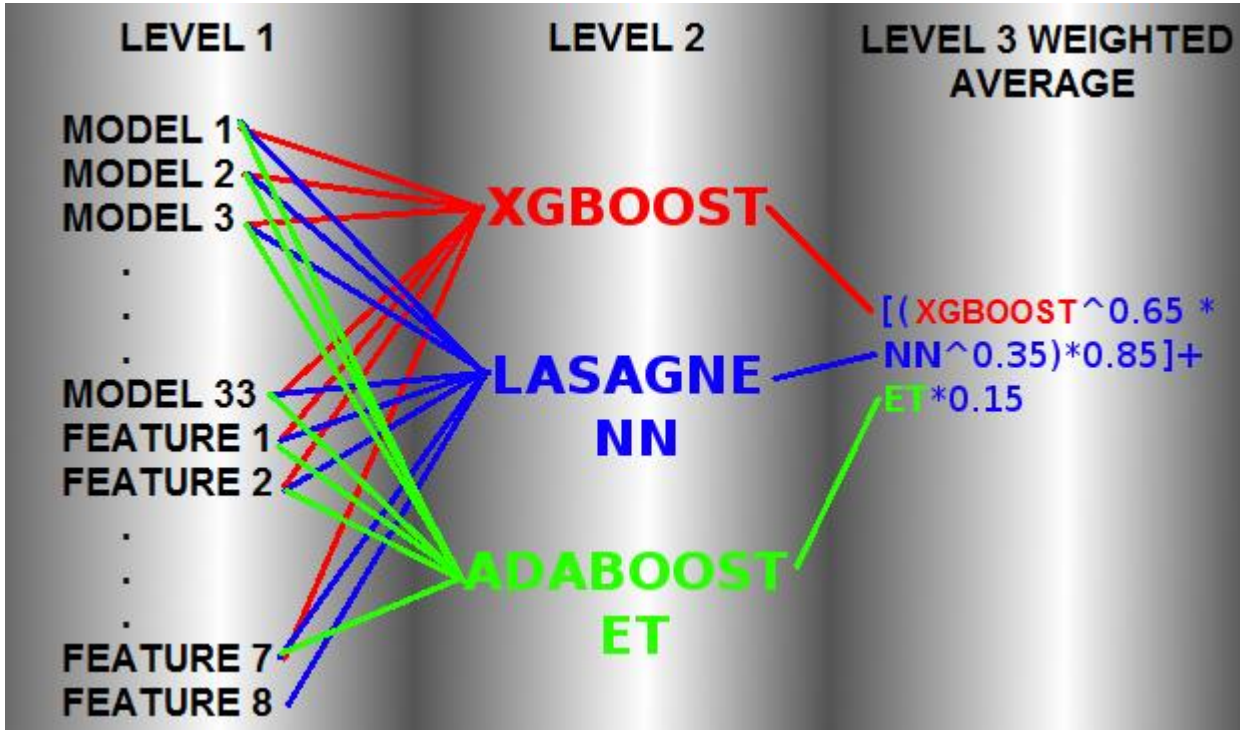
Both, RF and gbm yield variable importance plots

Remark: Often stumps perform better than deeper trees. A reason might be that additive model of stumps can fit very well quadratic boundaries in each coordinate, which is often a good and robust approximation (see Hastie 2014 talk).
With glmnet we can also do a post-procession of boosting or bagging and use lasso to pick the relevant trees.

How to construct the best performing model?

Join Forces, use different models including ensemble methods and learn how to combine their predictions for a joint prediction.

Winning solution for the Otto Challenge



Use outcome from models (e.g. xgboost) as meta features

See: https://www.kaggle.com/c/otto-group-product-classification-challenge/forums/t/14335/1st-place-winner-solution-gilberto-titericz-stanislav-semenov/79598#post79598

Summary on ensemble methods with focus on boosting

- Ensemble methods often improve prediction performance of individual models
- Bagging (bootstrapping and averaging) relies on averaging “independent” models and reduces variance of baseline model predictions (RF is an improved bagging method)
- Adaptive Boosting (synonyms: increase, go up, add to..) relies on step-wise improving the model by up-weighting miss-classified data from previous model.
- (extreme) Gradient Boosting (synonyms for boosting: increase, go up, add to..) relies on step-wise improving the model by adding modeled negative gradients resulting from combined previous model.
- Bagging (or RF) is easy to use and is a good bench mark.
- Boosting, especially xgboost shows often better performance but is not so easy to use since it has many hyper-parameter that need to be carefully tuned.

What have we done during this semester? Statistics, machine learning or data science?

Machine Learning:
Here we focus on algorithms that can learn from data.

Statistical Learning:
Branch of applied statistics that emerged in response to machine learning, emphasizing statistical models and assessment of uncertainty.

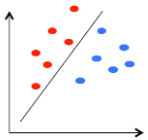
Data Science
Extraction of knowledge from data, using ideas from mathematics, statistics, machine learning, computer science, engineering, …

All of these are very similar – with different emphases.

Credits for these definitions: Trevor Hastie (2015)

What have we don during this semester?

Classifiers



K-Nearest-Neighbors (KNN)
Classification Trees
Linear discriminant analysis
Logistic Regression
Support Vector Machine (SVM)
Neural networks NN
…

Evaluation

Cross validation
Performance measures
Confusion matrices
ROC Analysis

Ensemble methods

Bagging
Random Forest
Boosting

Theoretical Guidance / General Ideas

Bayes Classifier
Concept of Bias and Variance trade-off
overfitting (high variance)
underfitting (high bias)

Feature Engineering

Feature expansion (kernel trick, NN)
Feature Selection (lasso, tree models…)

Wrapping up

- Multivariate data sets (p>>2) call for special methods.
- First step in most data analysis project: visualization, QC…
  - Outlier detection via robust PCA, χ^2 quantiles of MD^2 …
  - PCA, MDS for 2D visualization of rather small data (~located in 2D hyperplane)
  - Cluster analysis such as k-means, or hierarchical
  - t-SNE for 2D visualization of rather large data focusing on preserving close neighbors
- Supervised learning with “wide data sets” (p~10-100’000, n~10-1000)
  - SVM, Lasso, Ridge, LDA, knn, stepwise selection
- Supervised learning with “long data sets” (p~10-1000, n~500-100’000)
  - GLM, RF, Boosting
- Which method is best, depends on the (often unknown) data structure, therefor it is a good strategy to try to understand the data as good as possible before picking the method and to try different methods.