# Data Mining: Ensemble Learning

Business Analytics Practice
Winter Term 2015/16
Stefan Feuerriegel

# Today's Lecture

## Objectives

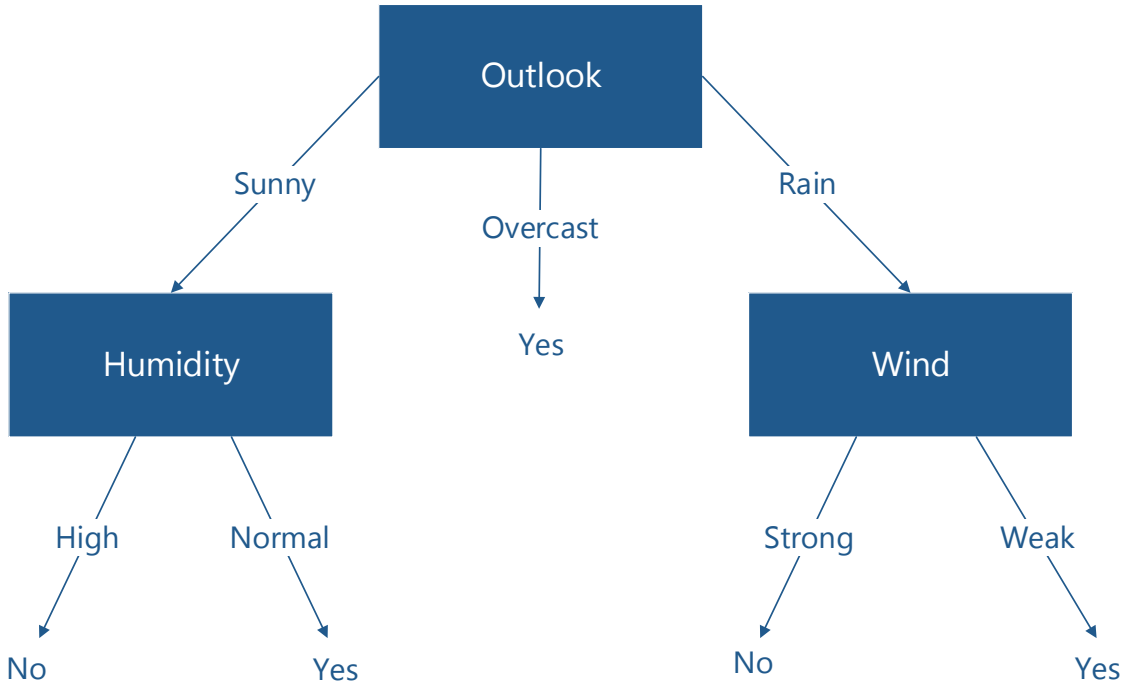
1. Creating and pruning decision trees
2. Combining an ensemble of trees to form a Random Forest
3. Understanding the idea and usage of Boosting and AdaBoost

# Outline

# Outline

# Decision Trees

- Flowchart-like structure in which nodes represent tests on attributes
- End nodes (leaves) of each branch represent class labels
- Example: Decision tree for playing tennis



# Decision Trees

- Issues
  - How deep to grow?
  - How to handle continuous attributes?
  - How to choose an appropriate attributes selection measure?
  - How to handle data with missing attributes values?
- Advantages
  - Simple to understand and interpret
  - Requires only few observations
  - Best and expected values can be determined for different scenarios
- Disadvantages
  - Information Gain criterion is biased in favor of attributes with more levels
  - Calculations become complex if values are uncertain and/or outcomes are linked

# Decision Trees in R

- Loading required libraries `rpart`, `party` and `partykit`

```
library(rpart)
library(party)
library(partykit)
```

- Accessing credit scores

```
library(caret)
data(GermanCredit)
```

- Split data into index subset for training (20 %) and testing (80 %) instances

```
inTrain <- runif(nrow(GermanCredit)) < 0.2
```

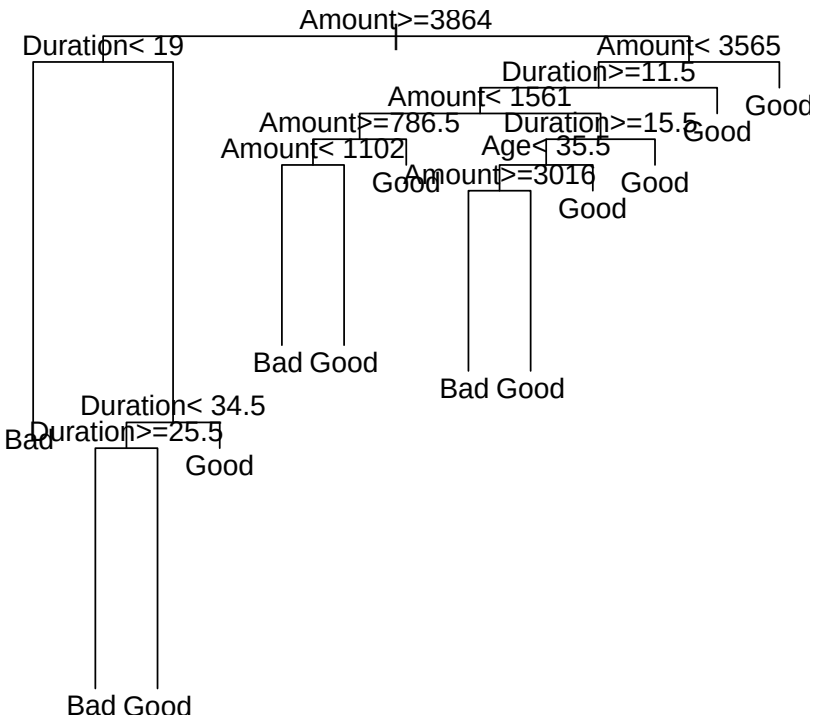
- Building a decision tree with `rpart(formula, method="class", data=d)`

```
dt <- rpart(Class ~ Duration + Amount + Age,
            method="class", data=GermanCredit[inTrain,])
```

# Decision Trees in R

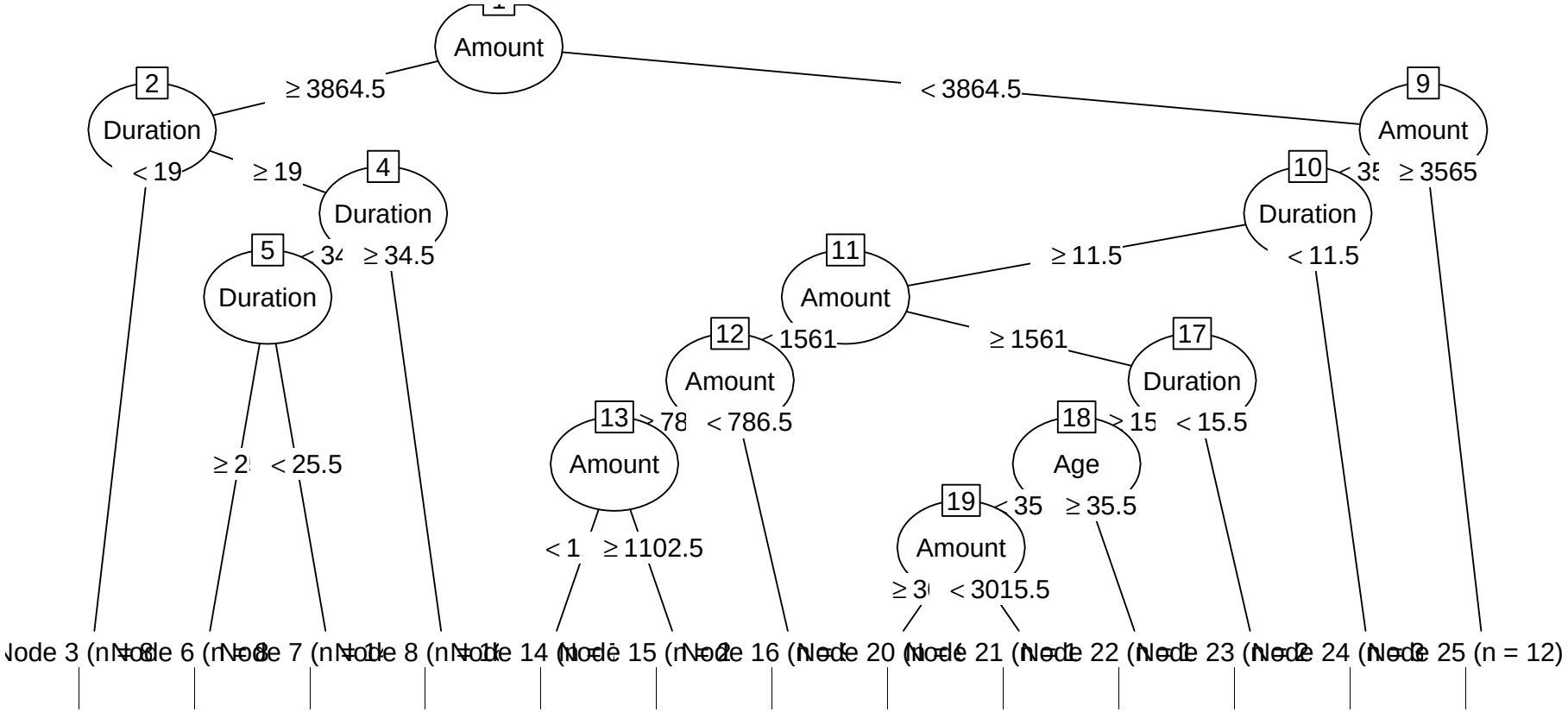
- Plot decision tree using `plot(dt)` and `text(dt)`

```
plot(dt)
text(dt)
```



# Drawing Decision Trees Nicely

```r
plot(as.party(dt))
```



# Complexity Parameter

```
printcp(dt)


##
## Classification tree:
## rpart(formula = Class ~ Duration + Amount + Age, data = GermanCredit[inTrain,
##     ], method = "class")
##
## Variables actually used in tree construction:
## [1] Age      Amount   Duration
##
## Root node error: 58/200 = 0.29
##
## n= 200
##
##         CP nsplit rel error  xerror    xstd
## 1 0.051724      0   1.00000 1.00000 0.11064
## 2 0.034483      2   0.89655 0.96552 0.10948
## 3 0.012931      4   0.82759 1.08621 0.11326
## 4 0.010000     12   0.72414 1.13793 0.11465
```

- Rows show results for trees with different numbers of nodes
- Cross-validation error in column `xerror`
- Complexity parameter in column `CP`, similar to number of nodes

# Pruning Decision Trees

- Reduce tree size by removing nodes with little predictive power
- Aim: Minimize cross-validation error in column `xerror`

```
m <- which.min(dt$cptable[, "xerror"])
```

- Index with smallest complexity parameter

```
m

## 2
## 2
```

- Optimal number of splits

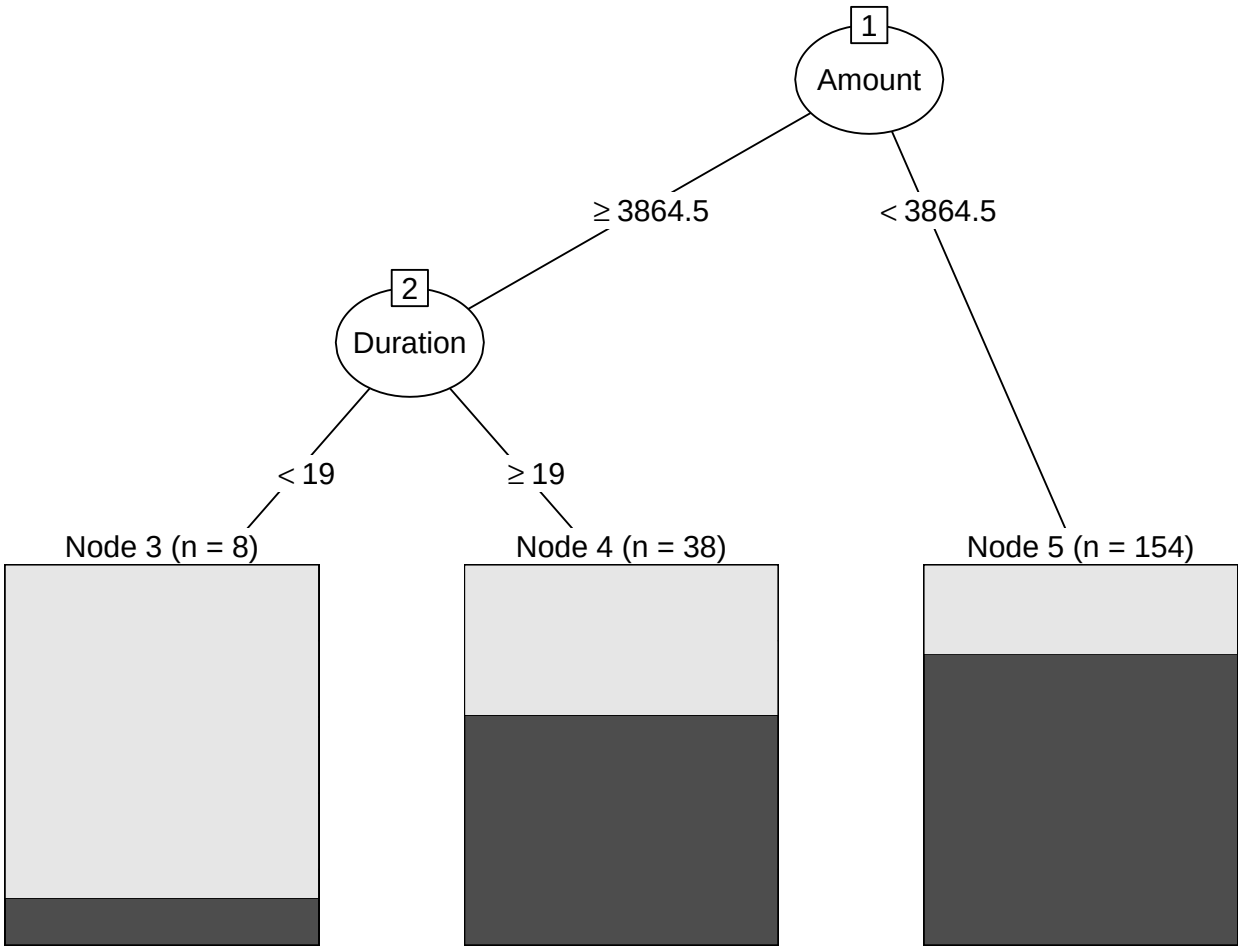
```
dt$cptable[m, "nsplit"]

## [1] 2
```

- Choose corresponding complexity parameter

```
dt$cptable[m, "CP"]

## [1] 0.03448276
```

# Pruning Decision Trees

```
p <- prune(dt, cp = dt$cptable[which.min(dt$cptable[, "xerror"]), "CP"])
plot(as.party(p))
```



# Prediction with Decision Trees

- `predict(dt, test, type="class")` predicts classes on new data `test`

```
pred <- predict(p, GermanCredit[-inTrain,], type="class")
pred[1:5]

##    2    3    4    5    6
## Good Good Good Good Good
## Levels: Bad Good
```

- Output: predicted label in 1st row out of all possible labels (2nd row)
- Confusion matrix via `table(pred=pred_classes, true=true_classes)`

```
# horizontal: true class; vertical: predicted class
table(pred=pred, true=GermanCredit[-inTrain,]$Class)

##       true
## pred   Bad Good
##   Bad   20   28
##   Good 280  671
```

# Outline

# Ensemble Learning

- Combine predictions of multiple learning algorithms $\rightarrow$ ensemble
- Often leads to a better predictive performance than a single learner
- Well-suited when small differences in the training data produce very different classifiers (e. g. decision trees)
- Drawbacks: increases computation time, reduces interpretability

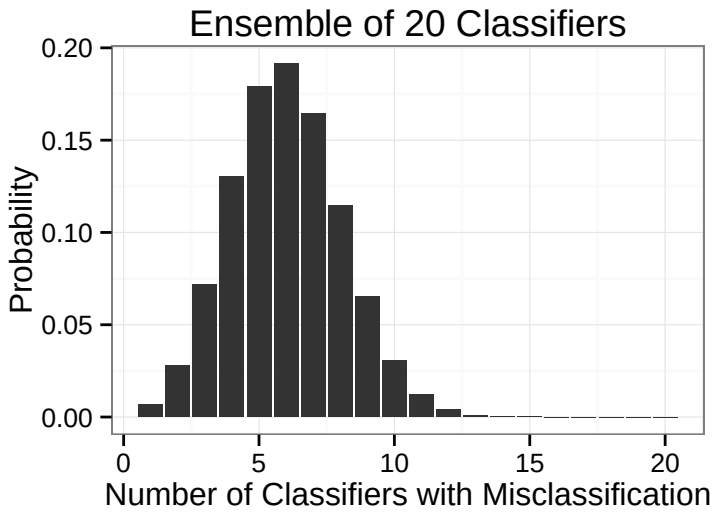
**Reasoning**

- Classifiers $C_1, \ldots, C_K$ which are independent, i. e. $\text{cor}(C_i, C_j) = 0$
- Each has an error probability of $P_i < 0.5$ on the training data
- Then an ensemble of classifiers should have an error probability lower than each individual $P_i$

# Example: Ensemble Learning

- Given $K$ classifiers, each with the same error probability $P_\varepsilon = 0.3$
- Probability that exactly $L$ classifiers make an error is

$$\binom{K}{L} P_\varepsilon^L (1 - P_\varepsilon)^{K-L}$$




- For example, the probability of 10+ classifiers making an error is 0.05
- Only if $P_\varepsilon > 0.5$, the error rate of the ensemble increases

# Ensemble Learning

→ Various methods exist for ensemble learning

**Constructing ensembles:** methods for obtaining a set of classifiers

- Bagging (also named Bootstrap Aggregation)
- Random Forest
- Cross-validation (covered as part of resampling)

→ Instead of different classifiers, train same classifier on different data
→ Since training data is expensive, reuse data by subsampling

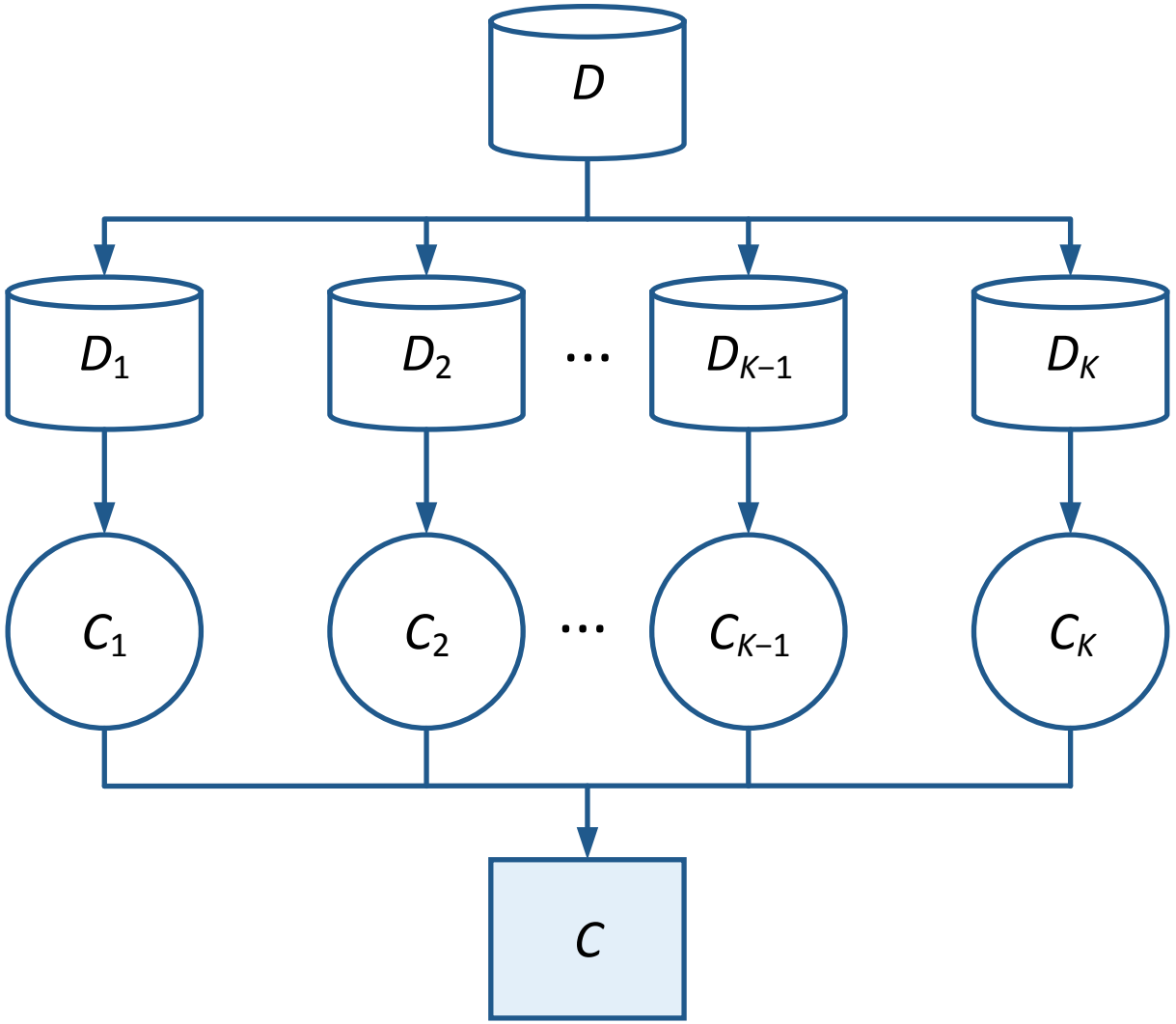
**Combining classifiers:** methods for combining different classifiers

- Stacking
- Bayesian Model Averaging
- Boosting
- AdaBoost

# Bagging: Bootstrap Aggregation

- Meta strategy design to accuracy of machine learning algorithms
- Improvements for unstable procedures
  $\rightarrow$ Neural networks, trees and linear regression with subset selection, rule learning (opposed to $k$-NN, linear regression, SVM)
- Idea: Reuse the same training algorithm several times on different subsets of the training data
- When classifier needs random initialization (e. g. $k$-means), very these across each run

# Bagging: Bootstrap Aggregation




Original training data

Step 1:
Create separate datasets

Step 2:
Train multiple classifiers

Step 3:
Combine classifiers

# Bagging: Bootstrap Aggregation

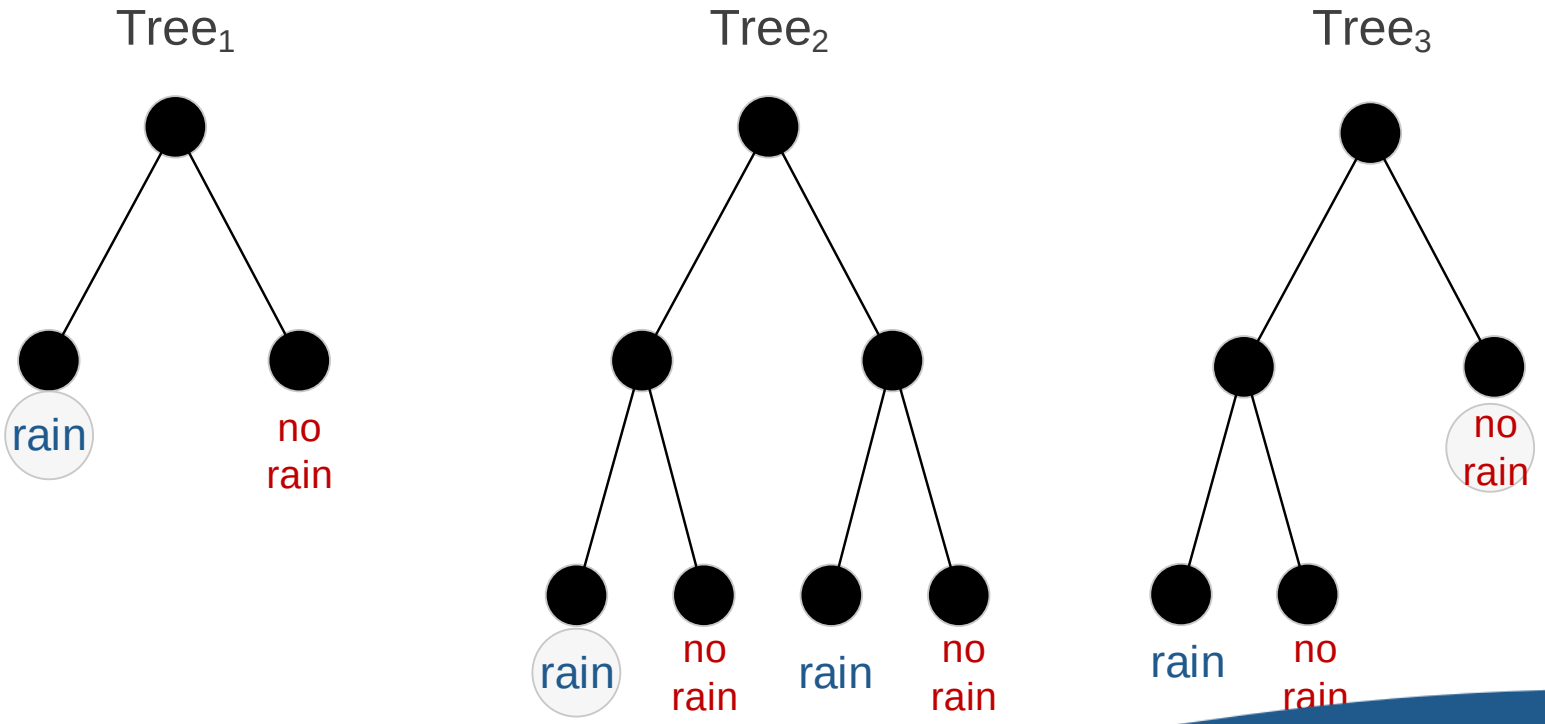
**Algorithm**

- Given training set $D$ of size $N$
- Bagging generates new training sets $D_i$ of size $M$ by sampling with replacement from $D$
- Some observations may be repeated in each $D_i$
- If $M = N$, then on average 63.2 % (Breiman, 1996) of the original training dataset $D$ is represented, the rest are duplicates
- Afterwards train classifier on each $C_i$ separately

# Outline

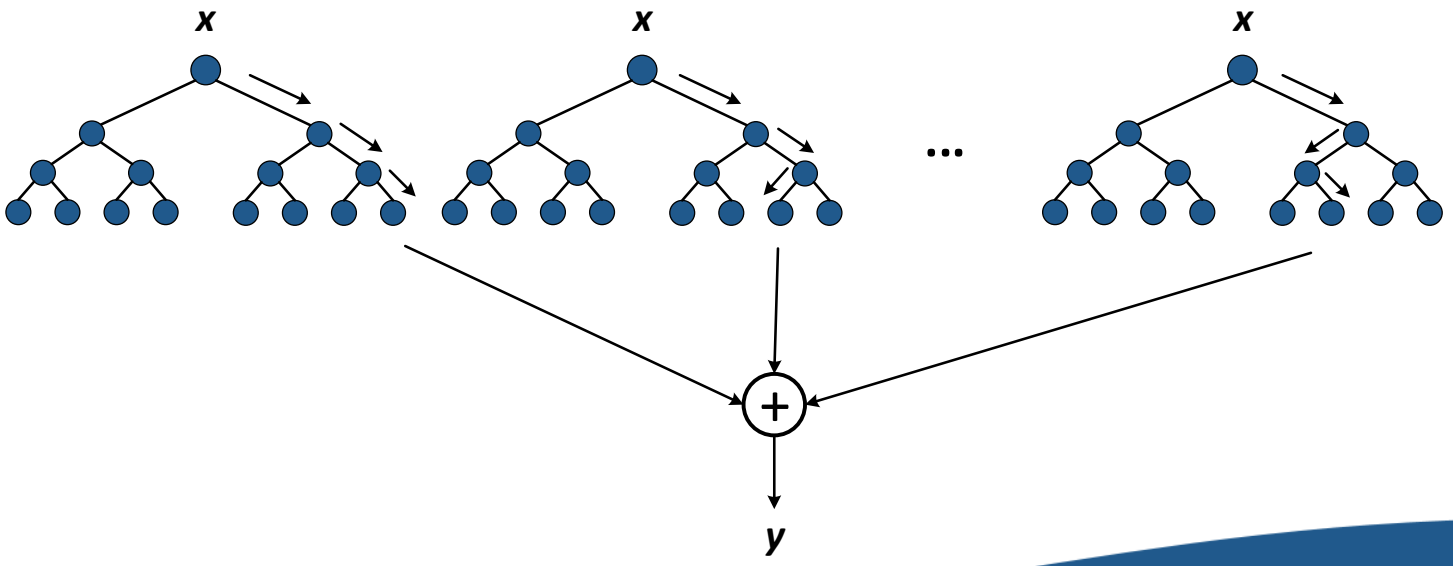
# Random Forests

- Random Forests are an ensemble learning method for classification and regression
- It combines multiple individual decision trees by means of bagging
- Overcomes the problem of overfitting decision trees



# Random Forest: Algorithm

1. Create many decision trees by bagging
2. Inject randomness into decision trees
   a. Tree grows to maximum size and is left unpruned
      - Deliberate overfitting: i. e. each tree is a good model on its own
   b. Each split is based on randomly selected subset of attributes
      - Reduces correlation between different trees
      - Otherwise, many trees would just select the very strong predictors
3. Ensemble trees (i. e. the random forest) vote on categories by majority



# Random Forest: Algorithm

1. Split the training data into $K$ bootstrap samples by drawing samples from training data with replacement
2. Estimate individual trees $t_i$ to the samples
3. Every regression tree predicts a value for unseen data
4. Averaging those predictions by

$$\hat{y} = \frac{1}{K} \sum_{i=1}^{K} t_i(\boldsymbol{x})$$

with $\hat{y}$ as the response vector and $\boldsymbol{x} = [x_1, \ldots, x_N]^T \in X$ as the input parameters.

# Advantages and Limitations

- Increasing the number of trees tends to decease the variance of the model without increasing the bias
- Averaging reveals real structure that persists across datasets
- Noisy signals of individual trees cancel out

**Advantages**

- Simple algorithm that learns non-linearity
- Good performance in practice
- Fast training algorithm
- Resistant to overfitting

**Limitations**

- High memory consumption during tree construction
- Little performance gain from large training data

# Random Forests in R

- Load required library `randomForest`

```
library(randomForest)
```

- Load dataset with credit scores

```
library(caret)
data(GermanCredit)
```

- Split data into index subset for training (20 %) and testing (80 %) instances

```
inTrain <- runif(nrow(GermanCredit)) < 0.2
```

# Random Forests in R

- Learn random forest on training data with `randomForest(...)`

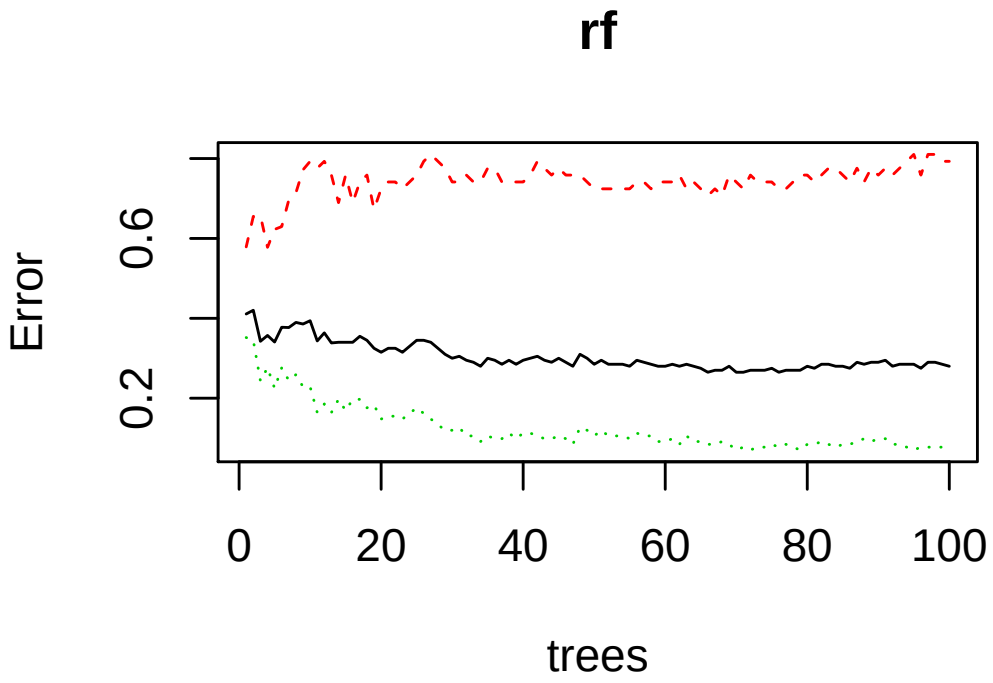
```
rf <- randomForest(Class ~ .,
                   data=GermanCredit[inTrain,],
                   ntree=100)
```

- Options to control behavior
  - `ntree` controls the number of trees (default: 500)
  - `mtry` gives number of variables to choose from at each node
  - `na.action` specifies how to handle missing values
  - `importance=TRUE` calculates variable importance metric

# Random Forest in R

- Plot estimated error across the number of decision trees

```
plot(rf)
```




Dotted lines represent corresponding error of classes and solid black line represents overall error

# Random Forest in R

- Calculate confusion matrix

```
rf$confusion

##       Bad Good class.error
## Bad    12   46  0.79310345
## Good   10  132  0.07042254
```

- Predict credit scores for testing instances

```
pred <- predict(rf, newdata=GermanCredit[-inTrain,])
table(pred=pred, true=GermanCredit$Class[-inTrain])

##        true
## pred    Bad Good
##   Bad    97   29
##   Good  203  670
```

# Variable Importance

- Coefficients normally tell the effect, but not its relevance
- Frequency and position of where variables appear in decision trees can be used for measuring variable importance
- Computed based on the corresponding reduction of accuracy when the predictor of interest is removed
- Variable importance is

$$VI^{(t)}(\boldsymbol{x}) = \frac{\sum_{i=1}^{K} I(y_i = \hat{y}_i^{(t)})}{K} - \frac{\sum_{i=1}^{K} I(y_i = \hat{y}_i^{(t)} \text{ learned from permuted } \boldsymbol{x})}{K}$$

  for tree $t$, with $y_i$ being the true class and $\hat{y}_i^{(t)}$ the predicted class
- A frequent alternative is the Gini importance index

# Variable Importance in R

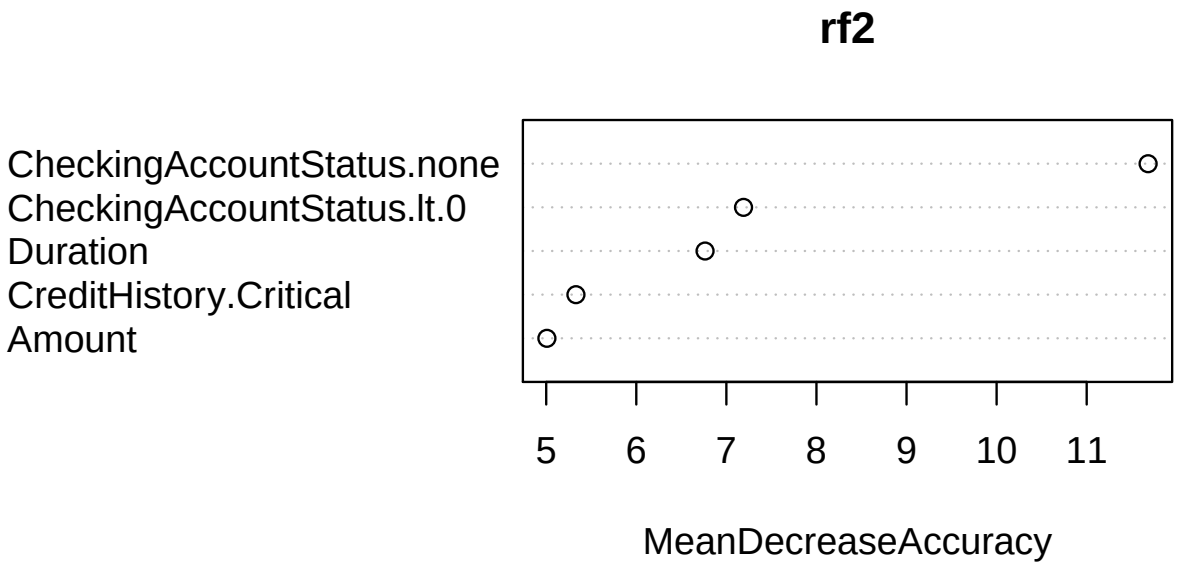
- Learn random forest and enable the calculation of variable importance metrics via `importance=TRUE`

```
rf2 <- randomForest(Class ~ .,
                    data=GermanCredit, #with full dataset
                    ntree=100,
                    importance=TRUE)
```

# Variable Importance in R

- Plot variable importance via `varImpPlot(rf, ...)`

```
varImpPlot(rf2, type=1, n.var=5)
```




- `type` choose the importance metric (= 1 is the mean decrease in accuracy if the variable would be randomly permuted)
- `n.var` denotes number of variables

# Outline

# Boosting

- Combine multiple classifiers to improve classification accuracy
- Works together with many different types of classifiers
- None of the classifier needs extremely good, only better than chance
  $\rightarrow$ Extreme case: decision stumps

$$y(\boldsymbol{x}) = \begin{cases} 1, & x_i \geq \theta \\ 0, & \text{otherwise} \end{cases}$$

- Idea: train classifiers on a subset of the training data that is most informative given the current classifiers
- Yields sequential classifier selection

# Boosting

**High-level algorithm**

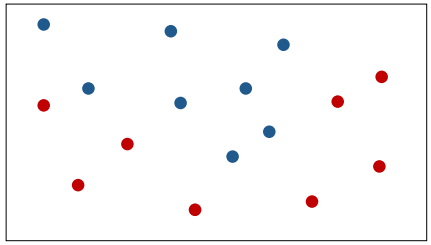
1. Fit a simple model to a subsample of the data
2. Identify misclassified observations, i. e. that are hard to predict
3. Focus subsequent learners on these samples and get them right
4. Combine weak learners to form a complex predictor

**Application:** spam filtering

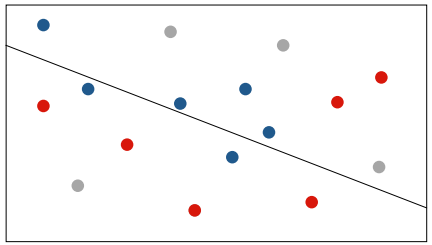
- First classifier: distinguish between emails from contacts and others
- Subsequent classifiers: focus on examples wrongly classified as spam (i. e. emails from others) and find words/phrases appearing in spam
- Combine to final classifier that predicts spam accurately

# Boosting: Example

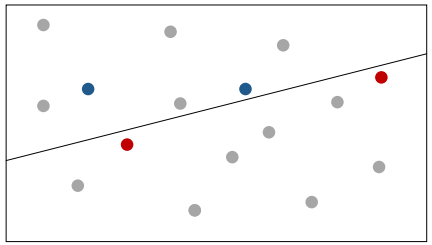
Given training data $D$ from which we sample without replacement



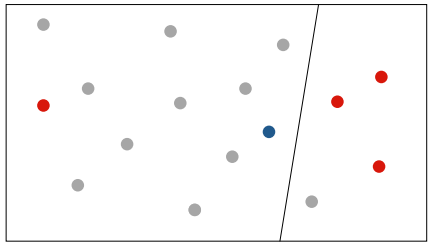
1. Sample $N_1 < N$ training examples $D_1$ from $D$
   - Train weak classifier $C_1$ on $D_1$



2. Sample $N_2 < N$ training examples $D_2$ from $D$, half of which were misclassified by $C_1$
   - Train weak classifier $C_2$ on $D_2$

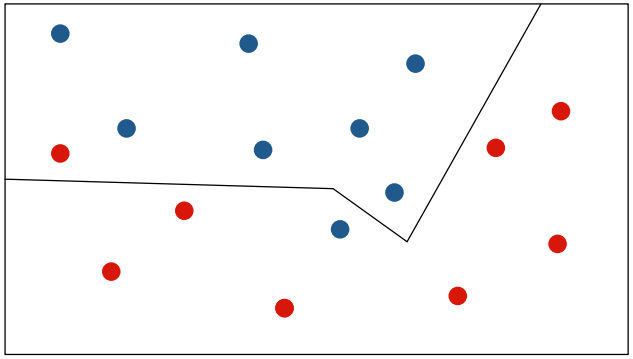


3. Identify all data $D_3$ in $D$ on which $C_1$ and $C_2$ disagree
   - Train weak classifier $C_3$ on $D_3$



# Boosting: Example

4. Combine $C_1$, $C_2$, $C_3$ to get final classifier $C$ by majority vote
   - E. g. on the missclassified red point, $C_1$ voted for red but $C_2$ and $C_3$ voted for blue



## Optimal number of samples $N_i$

- Reasonable guess $N_1 = N_2 = N_3 \Rightarrow \frac{N_1}{3}$ but problematic
  - Simple problem: $C_1$ explains most of the data and $N_2$ and $N_3$ are small
  - Hard problem: $C_1$ explains a small part and $N_2$ is large
- Solution: run boosting procedure several times and adjust $N_1$

# Boosting in R

- Load the required packages `mboost`

```
library(mboost)
```

- Fit a generalized linear model via `glmboost(...)`

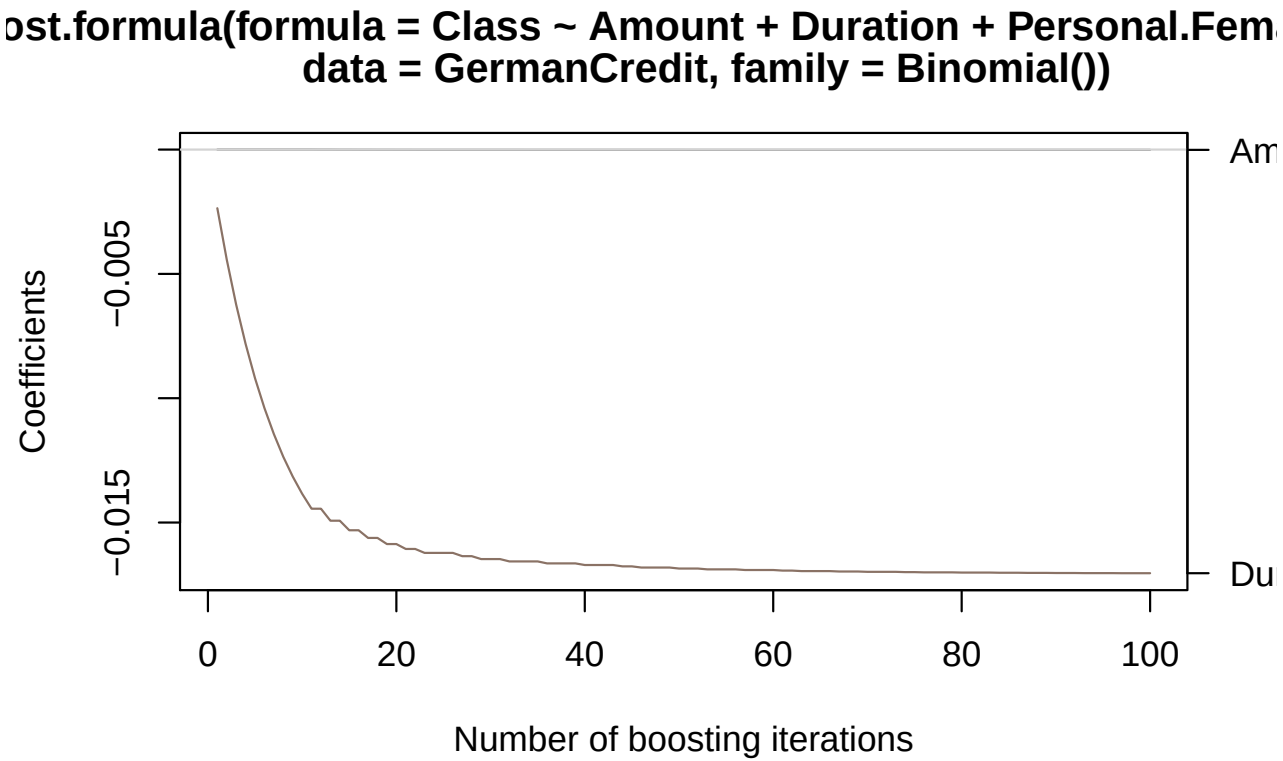
```
m.boost <- glmboost(Class ~ Amount + Duration
                          + Personal.Female.Single,
                    family=Binomial(), # needed for classification
                    data=GermanCredit)
coef(m.boost)

##   (Intercept)        Amount      Duration
##  4.104949e-01 -1.144369e-05 -1.703911e-02
## attr(,"offset")
## [1] 0.4236489
```

- Different from the normal `glm(...)` routine, the boosted version inherently performs variable selection

# Boosting in R: Convergence Plot

- Partial effects show how estimated coefficients evolve across iterations
- Plot convergence of selected coefficients

```
plot(m.boost, ylim=range(coef(m.boost,
                  which=c("Amount", "Duration"))))
```



# Boosting in R

- Main parameter for tuning is number of iterations `mstop`
- Use cross-validated estimates of empirical risk to find optimal number
  $\rightarrow$ Default is 25-fold bootstrapp cross-validation

```
cv.boost <- cvrisk(m.boost)
mstop(cv.boost) # optimal no. of iterations to prevent overfitting

## [1] 16

plot(cv.boost, main="Cross-validated estimates of empirical risk")
```

# Boosting in R

Alternative: fit generalized additive model via component-wise boosting

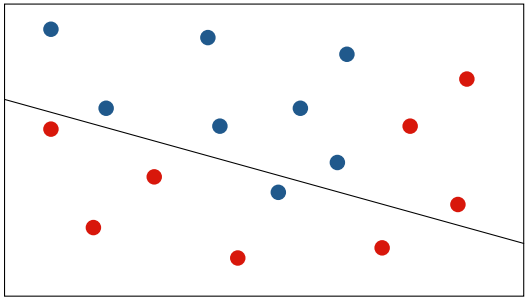
```
m.boost <- gamboost(Class ~ Amount + Duration,
                    family=Binomial(), # needed for classification
                    data=GermanCredit)
m.boost


##
##   Model-based Boosting
##
## Call:
## gamboost(formula = Class ~ Amount + Duration, data = GermanCredit,
##
##
##   Negative Binomial Likelihood
##
## Loss function: {
##      f <- pmin(abs(f), 36) * sign(f)
##      p <- exp(f)/(exp(f) + exp(-f))
##      y <- (y + 1)/2
##      -y * log(p) - (1 - y) * log(1 - p)
##  }
##
##
## Number of boosting iterations: mstop = 100
## Step size:  0.1
```

# Outline

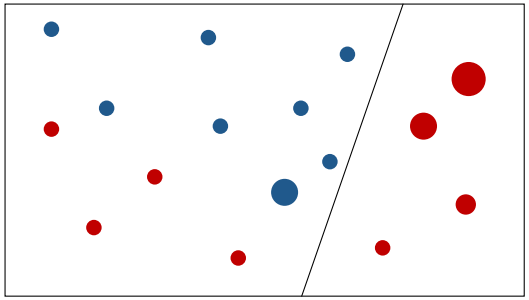
# AdaBoosting

Instead of resampling, reweight misclassified training examples

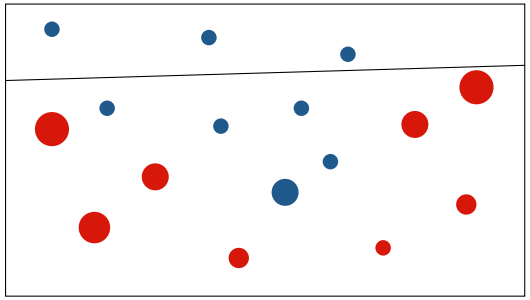
**Illustration**



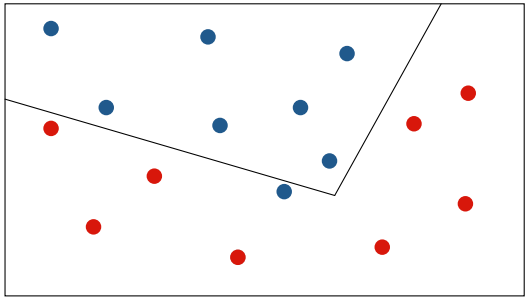
Weak classifier $C_1$



Weak classifier $C_2$



Weak classifier $C_3$



$\Rightarrow$ Combine weak classifiers $C_1$, $C_2$, $C_3$ into final classifier by majority vote

# AdaBoost

**Benefits**

- Simple combination of multiple classifiers
- Easy implementation
- Different types of classifiers can be used
- Commonly in used across many domains

**Limitations**

- Sensitive to misclassified points in training data

# AdaBoost in R

- Load required package `ada`

```
library(ada)
```

- Fit AdaBoost model on training data with `ada(..., iter)` given a fixed number `iter` of iterations

```
m.ada <- ada(Class ~ .,
             data=GermanCredit[inTrain,],
             iter=50)
```

- Evaluate on test data `test.x` with response `test.y`

```
m.ada.test <- addtest(m.ada,
                      test.x=GermanCredit[-inTrain,],
                      test.y=GermanCredit$Class[-inTrain])
```

# AdaBoost in R

```
m.ada.test

## Call:
## ada(Class ~ ., data = GermanCredit[inTrain, ], iter = 50)
##
## Loss: exponential Method: discrete   Iteration: 50
##
## Final Confusion Matrix for Data:
##          Final Prediction
## True value Bad Good
##       Bad   33   25
##       Good   2  140
##
## Train Error: 0.135
##
## Out-Of-Bag Error:  0.18  iteration= 50
##
## Additional Estimates of number of iterations:
##
## train.err1 train.kap1 test.errs2 test.kaps2
##         36         36         43         37
```

# AdaBoost in R

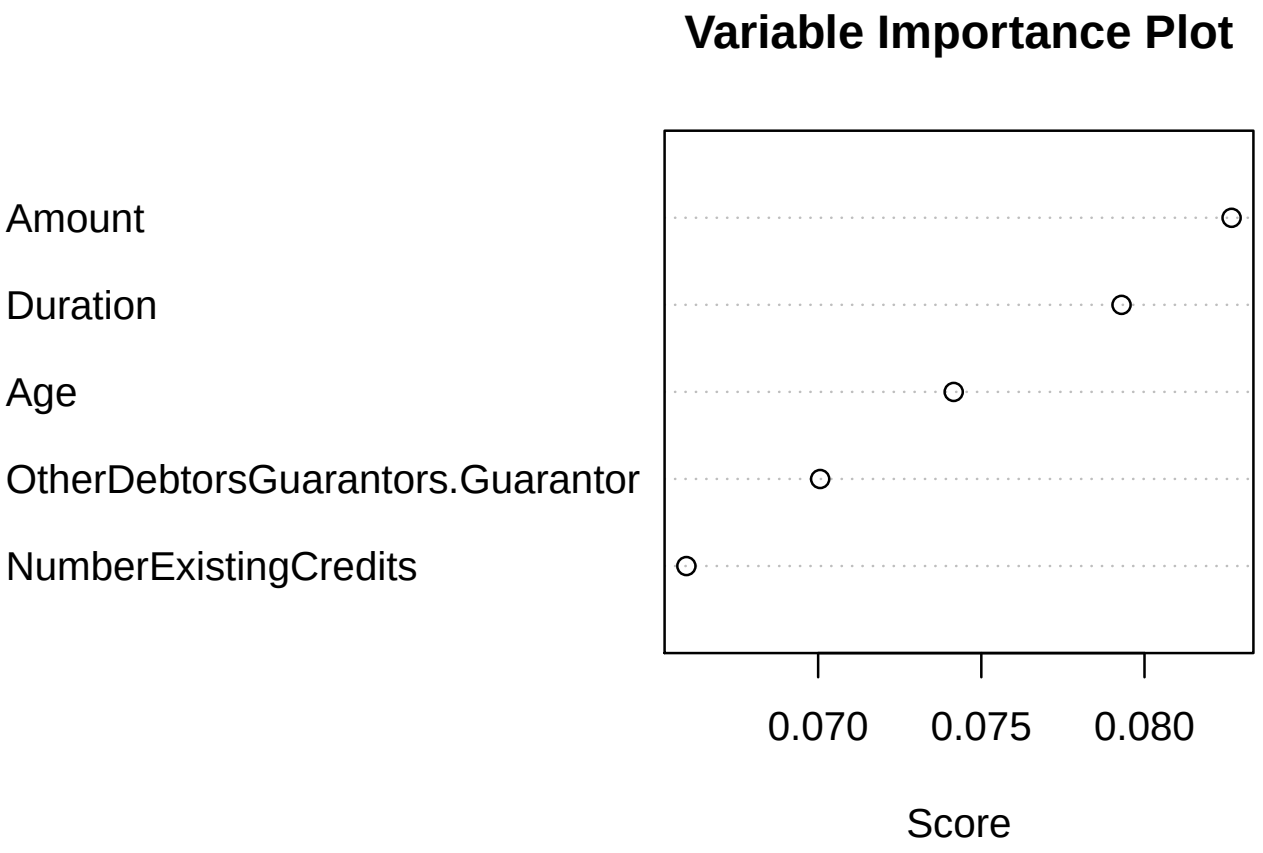
Plot error on training and testing data via `plot(m, test=TRUE)` for model *m*

```
plot(m.ada.test, test=TRUE)
```

# AdaBoost in R

Similarly as with random forest, `varplot(...)` plots the importance for the first variables

```
varplot(m.ada.test, max.var.show=5) # first 5 variables
```



# Summary

**Decision Trees**

- Highly visual tool for decision support, though the risk of overfitting

**Ensemble Learning: Random Forest, Boosting and AdaBoost**

- Idea: combine an ensemble of learners to improve performance
- Random forest combines independent decision trees
- Boosting resamples the training data, whereas AdaBoost reweights training data
  $\rightarrow$ focus subsequent learn from misclassifications
- Combined weak learners usually vote by majority on new data