

# Statistisches Data Mining (StDM)

## Woche 12

*Oliver Dürr*

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

[oliver.duerr@zhaw.ch](mailto:oliver.duerr@zhaw.ch)

Winterthur, 6 Dezember 2016

# No laptops, no phones, no problems

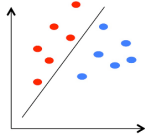


## **Multitasking senkt Lerneffizienz:**

- **Keine Laptops im Theorie-Unterricht Deckel zu oder fast zu (Sleep modus)**

# Overview of classification (until the end to the semester)

## Classifiers



**K-Nearest-Neighbors (KNN)**

**Logistic Regression**

Linear discriminant analysis

Support Vector Machine (SVM)

Classification Trees

Neural networks NN

Deep Neural Networks (e.g. CNN, RNN)

...



## Combining classifiers

Bagging

Random Forest

Boosting

## Evaluation



Cross validation

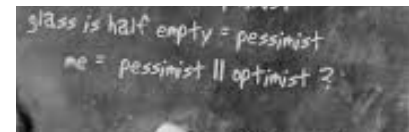
Performance measures

ROC Analysis / Lift Charts

## Theoretical Guidance / General Ideas

Bayes Classifier

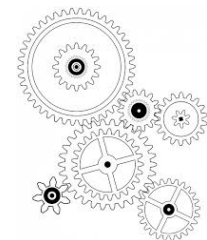
Bias Variance Trade  
off (Overfitting)



## Feature Engineering

Feature Extraction

Feature Selection



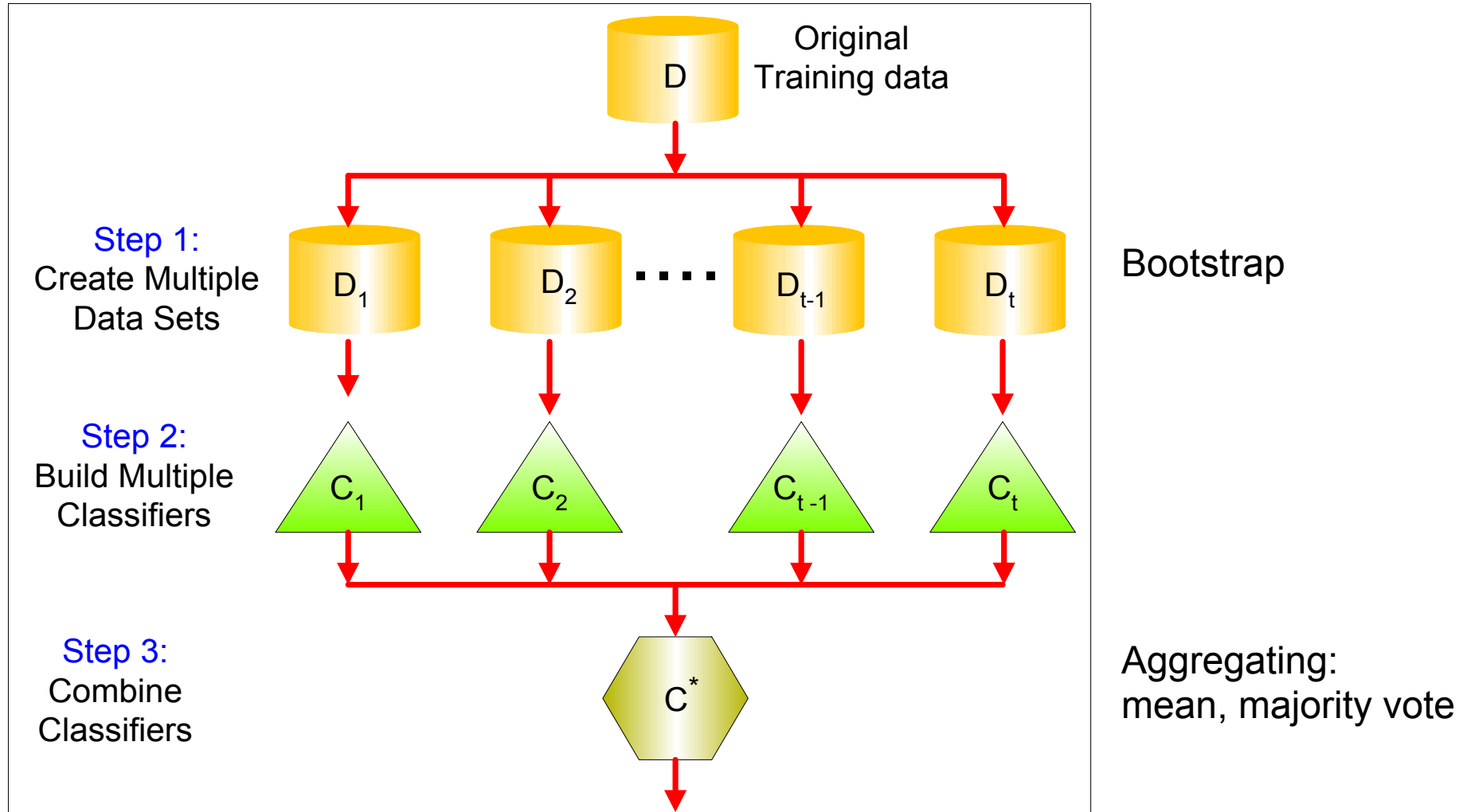
# Overview of Ensemble Methods

- Many instances of the same classifier
  - Bagging (bootstrapping & aggregating)
    - Create “new” data using bootstrap
    - Train classifiers on new data
    - Average over all predictions (e.g. take majority vote)
  - Bagged Trees
    - Use bagging with decision trees
  - Random Forest
    - Bagged trees with special trick
  - Boosting
    - An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records.
- Combining classifiers
  - Weighted averaging over predictions
  - Stacking classifiers
    - Use output of classifiers as input for a new classifier

# Bagging / Random Forest

## Chapter 8.2 in ILSR

# Bagging: Bootstrap Aggregating



# Why does it work?

- Suppose there are 25 base classifiers
- Each classifier has error rate,  $\epsilon = 0.35$
- Assume classifiers are independent (that's the hardest assumption)
- Take **majority vote**
- Majority voter is wrong if 13 or more are wrong
- Number of wrong predictors  $X \sim \text{Bin}(\text{size}=25, p_0=0.35)$

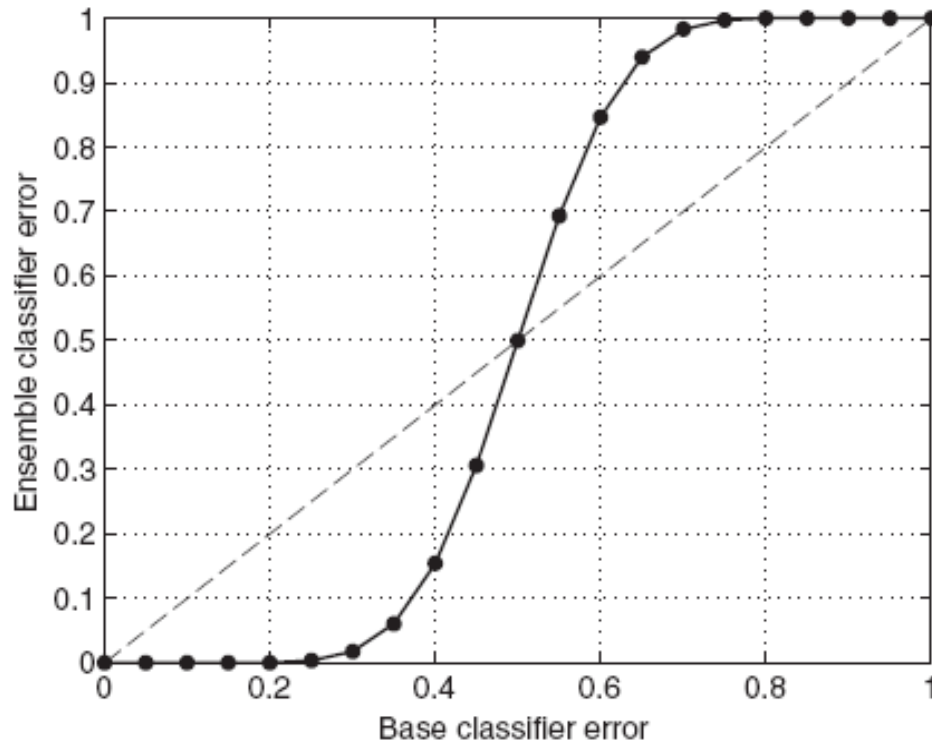
•> 1 - pbinom(12, size=25, prob = 0.35)

•[1] 0.06044491

$$\sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1 - \epsilon)^{25-i} = 0.06$$

# Why does it work?

- 25 Base Classifiers



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



# Reminder Trees

```
library(rpart)
fit <- rpart(Kyphosis ~ ., data = kyphosis)
plot(fit)
text(fit, use.n = TRUE)
```

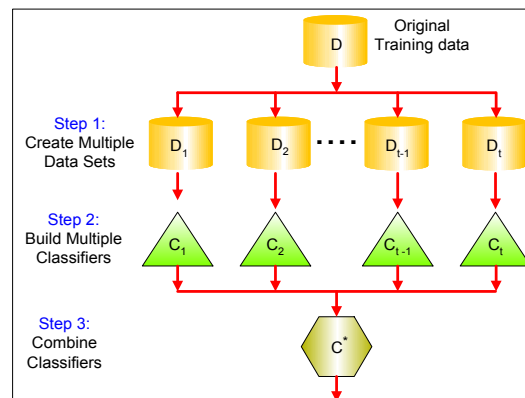
```
pred = predict(fit, data=kyphosis)
```

```
> head(pred,4) #Probabilities
```

	absent	present
1	0.4210526	0.5789474
2	0.8571429	0.1428571
3	0.4210526	0.5789474
4	0.4210526	0.5789474

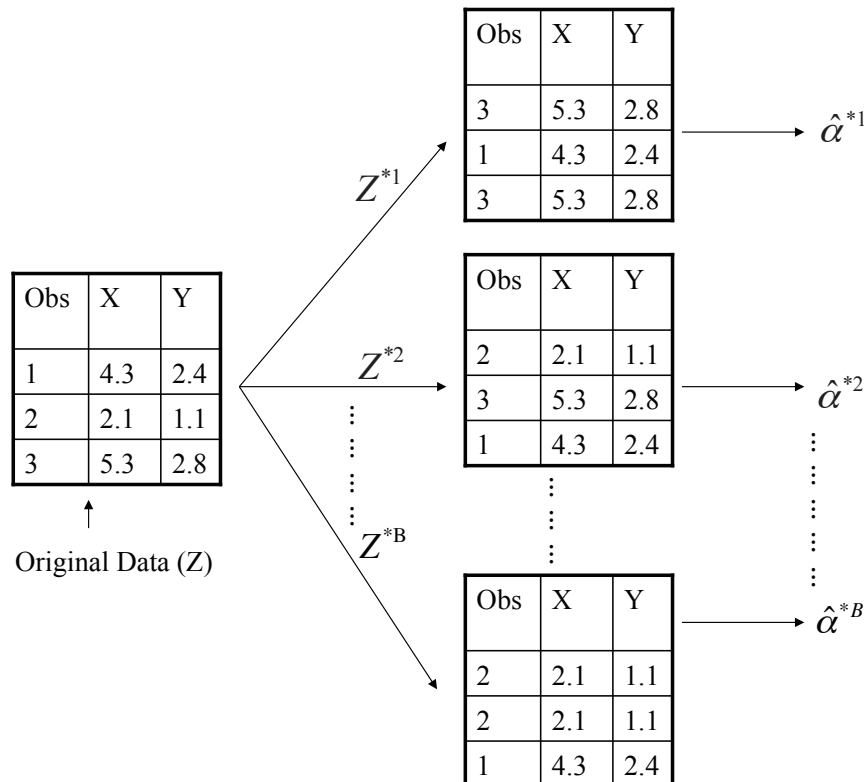
# Bagging trees

- Decision trees suffer from high variance!
  - If we randomly split the training data into 2 parts, and fit decision trees on both parts, the results could be quite different
  - Averaging reduces Variance
    - Independent  $\rightarrow \text{var} \sim \sigma^2 / n$  (not true strictly)
- Bagging for trees
  - Take bootstrap B samples from training set
  - For each bootstrap sample train a decision tree
  - For the test set take the majority vote of all trained classifiers (or average)



# Reminder: Bootstrapping

- Resampling of the observed dataset (and of equal size to the observed dataset), each of which is obtained by **random sampling with replacement** from the original dataset



# Lösung der Aufgabe: Bagging von Hand

```
library(rpart)
library(MASS) # For the data set

Boston$scrim = Boston$scrim > median(Boston$scrim)

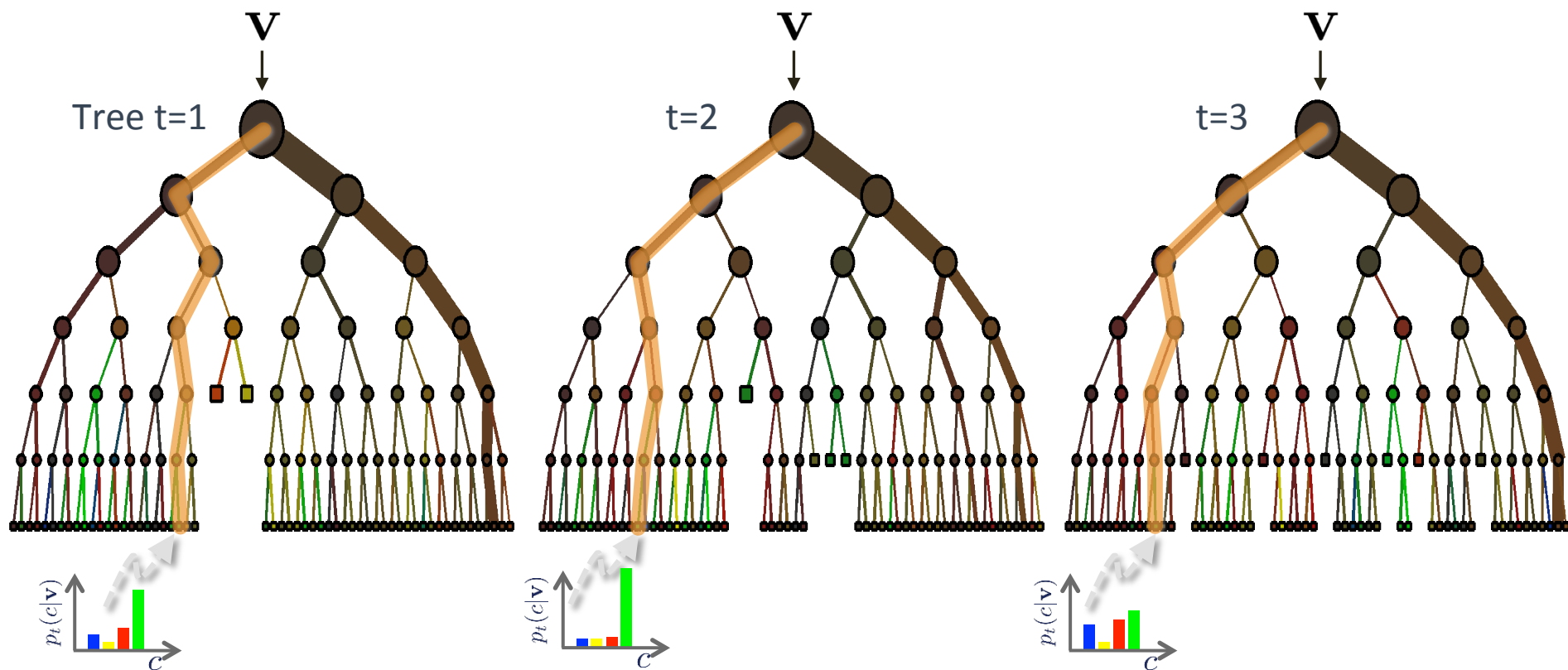
# Split in Training and Testset
idt = sample(nrow(Boston), size = floor(nrow(Boston)*3/4))
data.train = Boston[idt, ]
data.test = Boston[- idt, ]

# Single Tree
d = rpart( as.factor(crim) ~ ., data = data.train)
sum(predict(fit, data.test, type='class') == Boston$scrim[-idt]) / nrow(data.test) #0.88

n.trees = 100
preds = rep(0, nrow(data.test))
for (j in 1:n.trees ){
  # Doing the bootstrap
  index.train = sample( nrow(data.train), size = nrow(data.train), replace = TRUE )
  # Fitting to the training data
  fit = rpart( as.factor(crim) ~ ., data = data.train[index.train,])
  # Predict the test set
  tree.fit.test = predict(fit, data.test, type='class')
  preds = preds + ifelse(tree.fit.test == TRUE, 1, -1) #Trick with +-1
}

sum(Boston$scrim[-idt] == (preds > 0)) / nrow(data.test) #0.96
```

# How to classify a new observation?

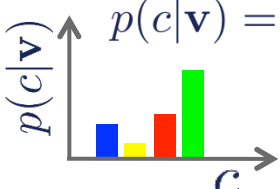


Two ways to derive the ensemble result:

1) Each tree has a “winner-class”

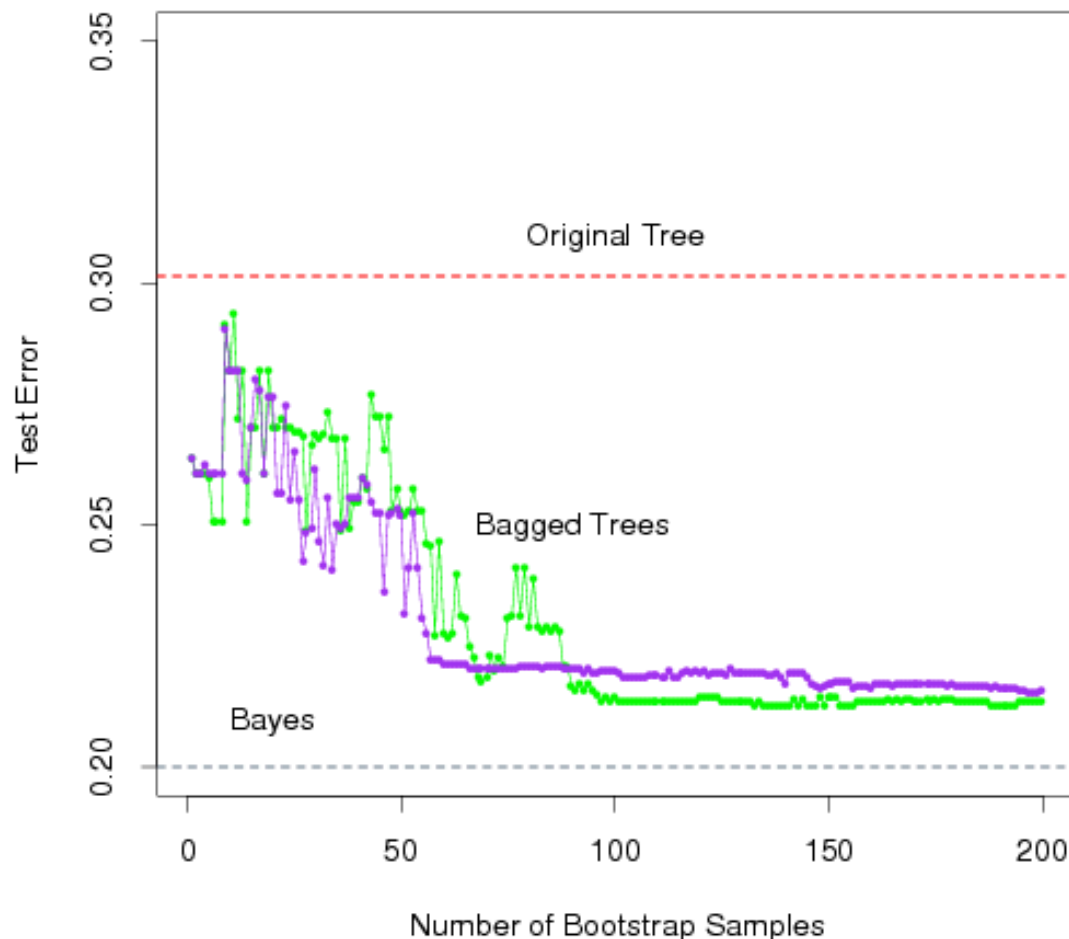
Take the class which was most often the winner

2) average probabilities:

$$p(c|V) = \frac{1}{T} \sum_t p_t(c|V)$$


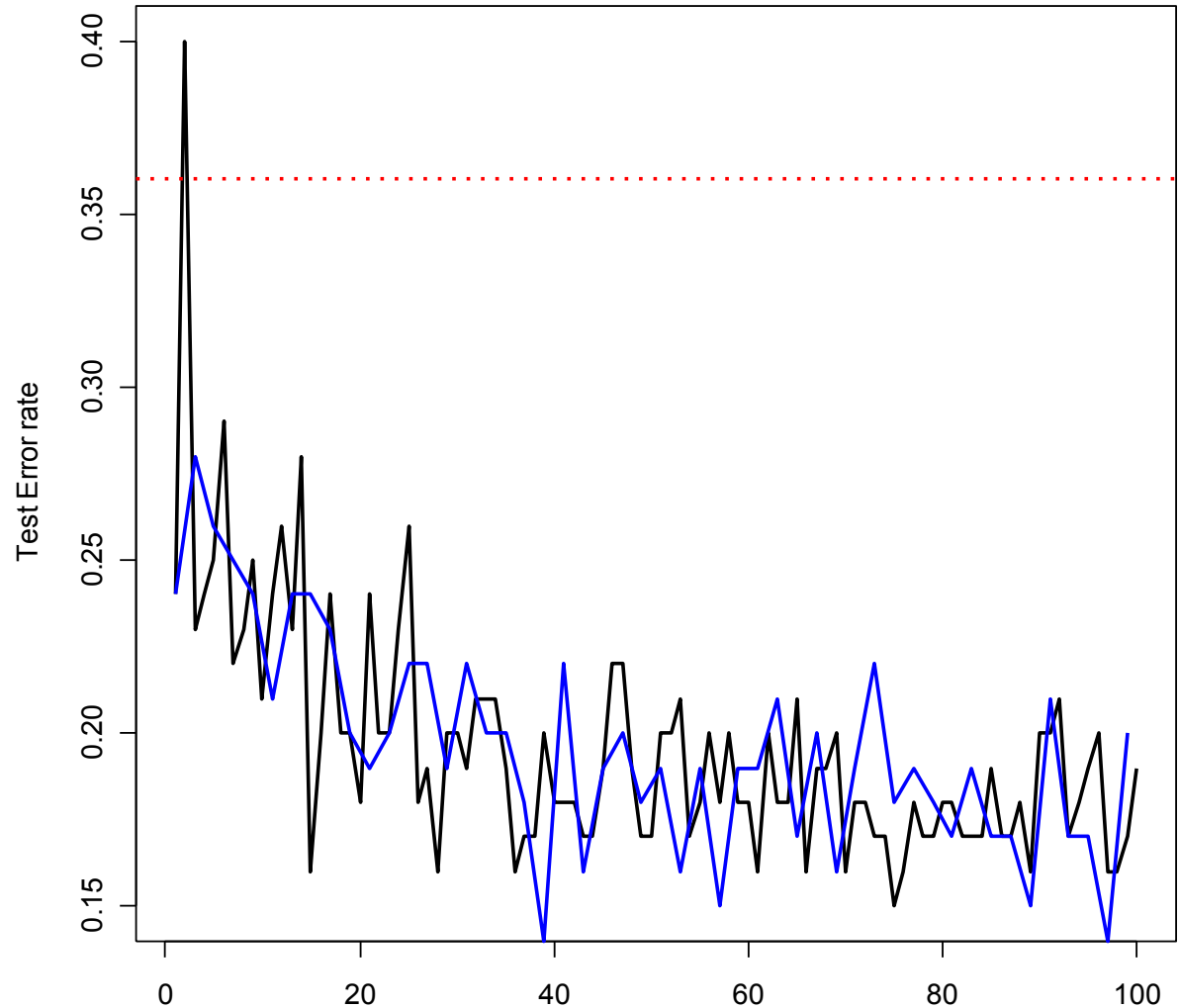
# A Comparison of Error Rates

- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates.
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey).



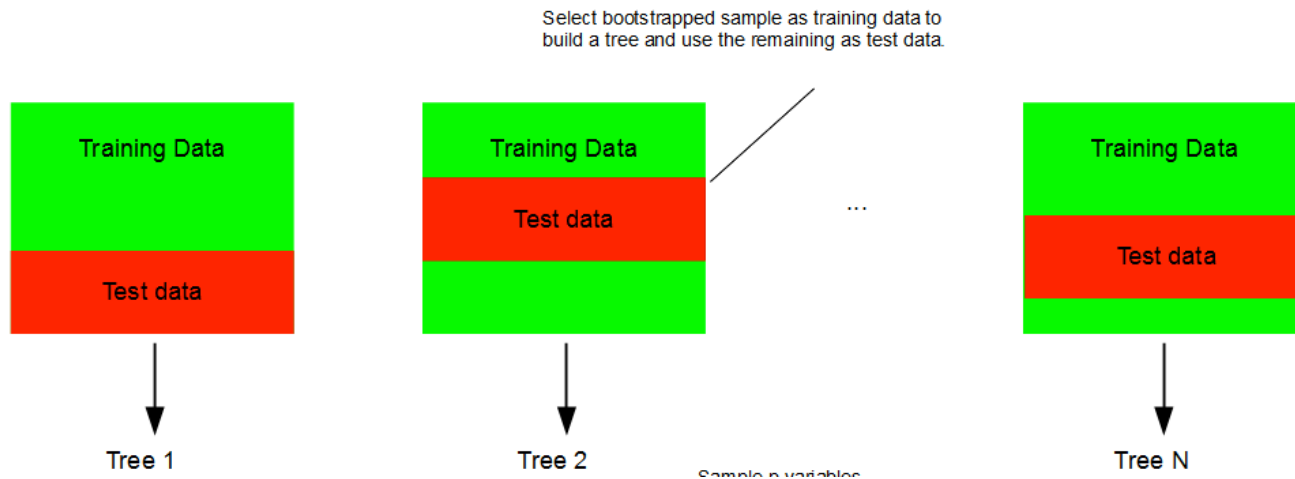
## Example 2: Car Seat Data

- The red line represents the test error rate using a single tree.
- The black line corresponds to the bagging error rate using majority vote while the blue line averages the probabilities.



# Out-of-Bag Estimation: "Cross-validation on the fly"

- Since bootstrapping involves random selection of subsets of observations to build a training data set, then the remaining non-selected part could be the testing data.
- On average, each bagged tree makes use of around  $2/3$  of the observations, so we end up having  $1/3$  of the observations used for testing





# The importance of independence

- It's a quite strong assumption, that all classifiers are independent.
- Since the same features are used in each bag, the resulting classifiers are quite dependent.
  - **We don't really get new classifiers or trees**
  - The bagging has high bias but low variance.
- **De-correlate by any means! → Random Forest**

# Random Forest

# Random Forests

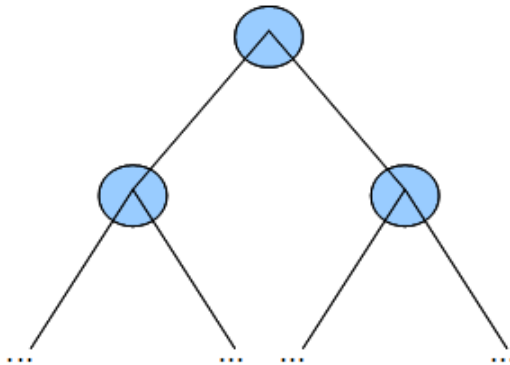
- It is a very efficient statistical learning method
- It builds on the idea of bagging, but it provides an improvement because it de-correlates the trees
- How does it work?
  - Build a number of decision trees on bootstrapped training sample, but when building these trees, each time a split in a tree is considered, **a random sample of  $m$  predictors** is chosen as split candidates from the full set of  $p$  predictors (Usually  $m \approx \sqrt{p}$ )

# Random Forest: Learning algorithm

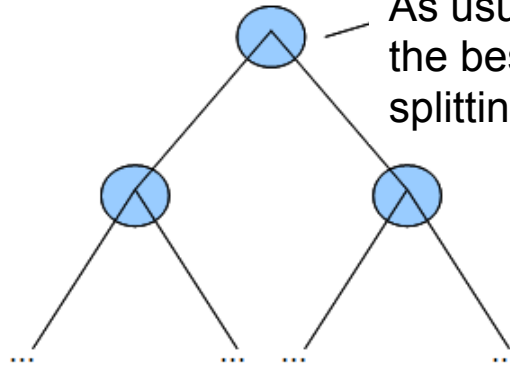
Select bootstrapped sample as training data to build a tree and use the remaining as test data.



Tree 1



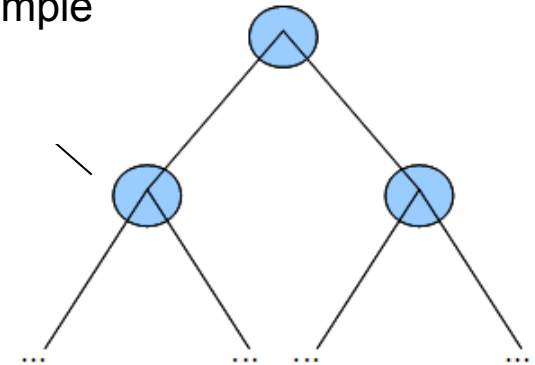
Tree 2



Sample  $m$  variables out  $p$ .  
As usual for trees sample  
the best variable for  
splitting



Tree N

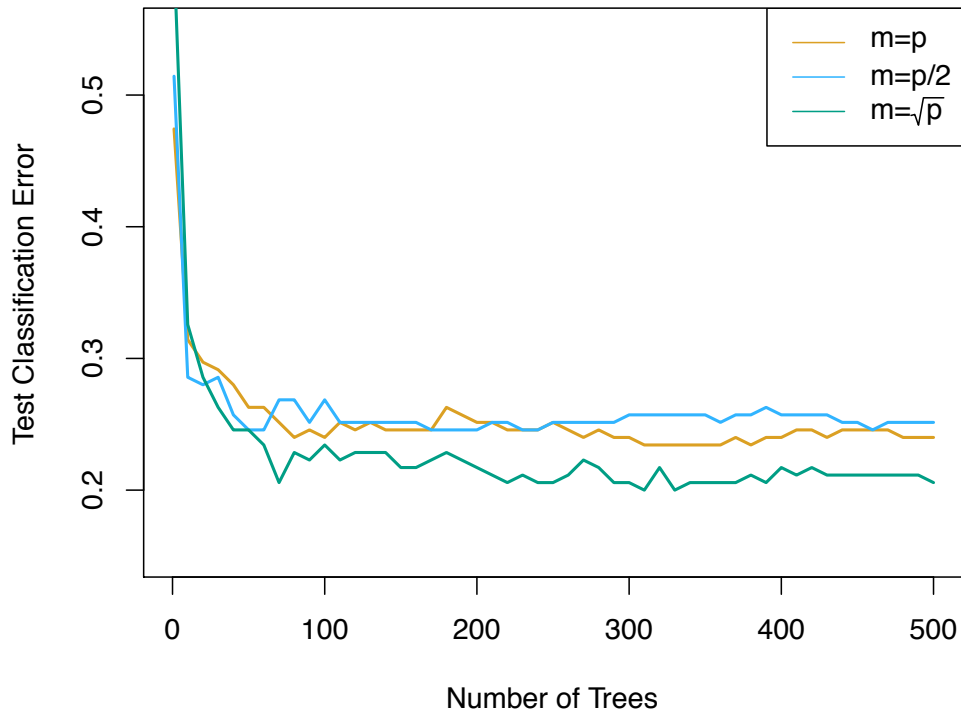


## Why are we considering a random sample of $m$ predictors instead of all $p$ predictors for splitting?

- Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictor, then in the collection of bagged trees, most or all of them will use the very strong predictor for the first split!
- All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction, and thus random forests “de-correlates” the bagged trees leading to more reduction in variance
- This makes the individual trees weaker but enhances the overall performance

# Random Forest with different values of “m”

- Notice when random forests are built using  $m = p$ , then this amounts simply to bagging.



# Random Forest in R (basic)

```
fit = randomForest(as.factor(crim) ~ ., data=data.train)
predict(fit, data.test)
```

```
####
# Parameters
randomForest(as.factor(crim) ~ ., data=data.train,
ntree=100, mtry = 42)

#ntree number of trees grown
#mtry number of features used in each split. If set to p
➔ Bagged Trees
```

# Variable Importance Measure

- Bagging typically improves the accuracy over prediction using a single tree, but it is now hard to interpret the model!
- We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
- Thus bagging improves prediction accuracy at the expense of interpretability
- But, we can still get an overall summary of the importance of each predictor using Relative Influence Plots



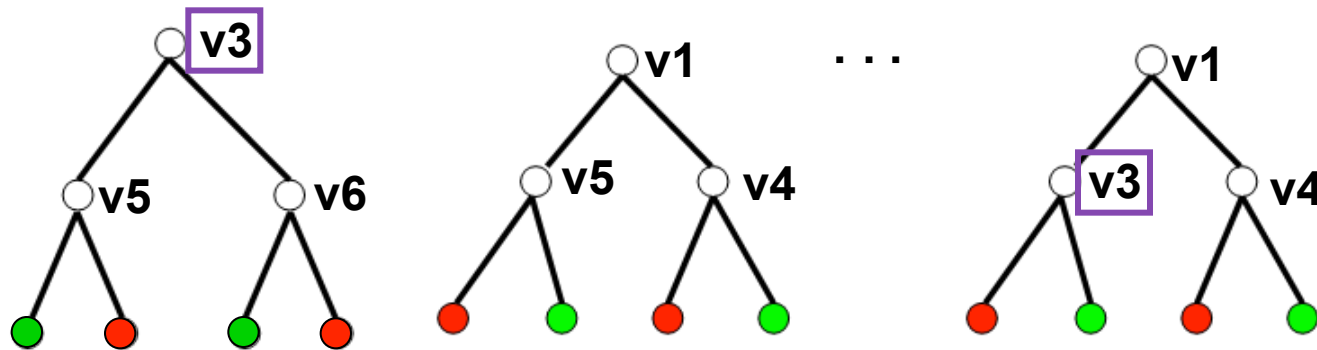
# Variable Importance 1: performance-loss by permutation

Determine importance-2 for v4:

- 1) obtain standard oob-performance
- 2) values of v4 are randomly permuted in oob samples and oob-performance is again computed. If variable is not important nothing happens. Hence:
- 3) Use decrease of performance as important measure of v4

<b>v1</b>	<b>v2</b>	<b>v3</b>	<b>v4</b>	<b>v5</b>	<b>v6</b>	<b>v7</b>
0	1	1	2	0	1	0
0	2	2	1	2	0	1
1	0	0	1	1	2	0
1	0	0	1	1	0	2
0	2	1	0	2	0	1

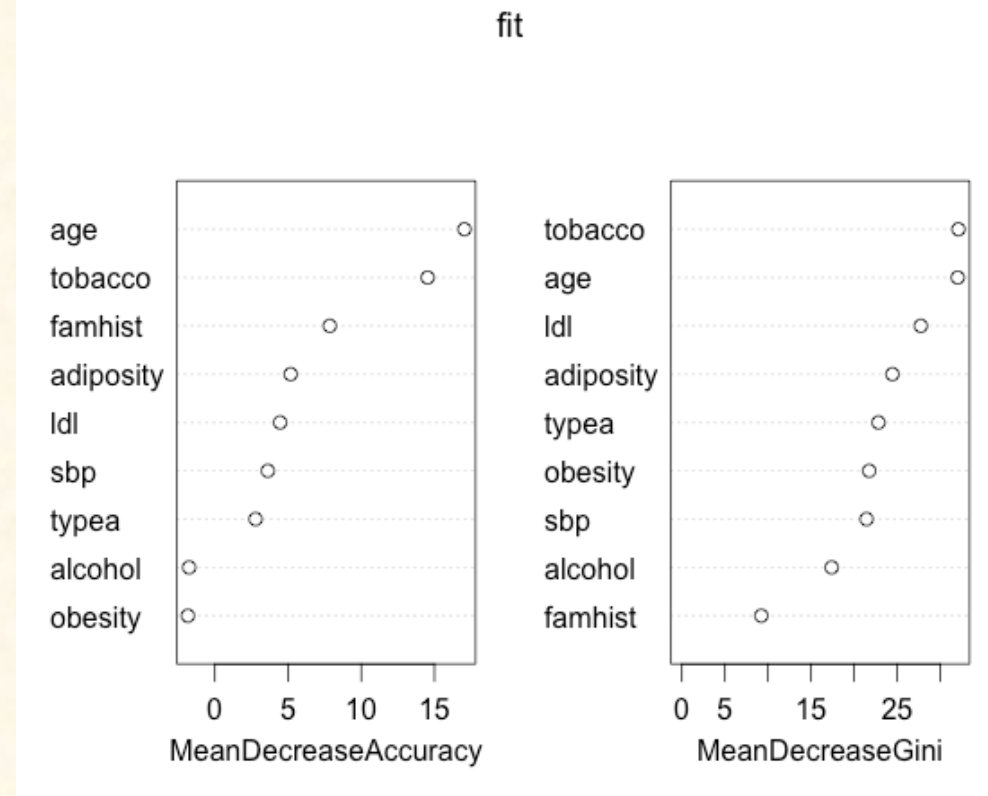
## Variable Importance 2: Score improvement at each Split



At each split where the variable, e.g. **v3**, is used the improvement of the score (Decrease of Giniindex) is measured. The average over all these **v3**-involving splits in all trees is the importance-1 measure of **v3**.

## varImp

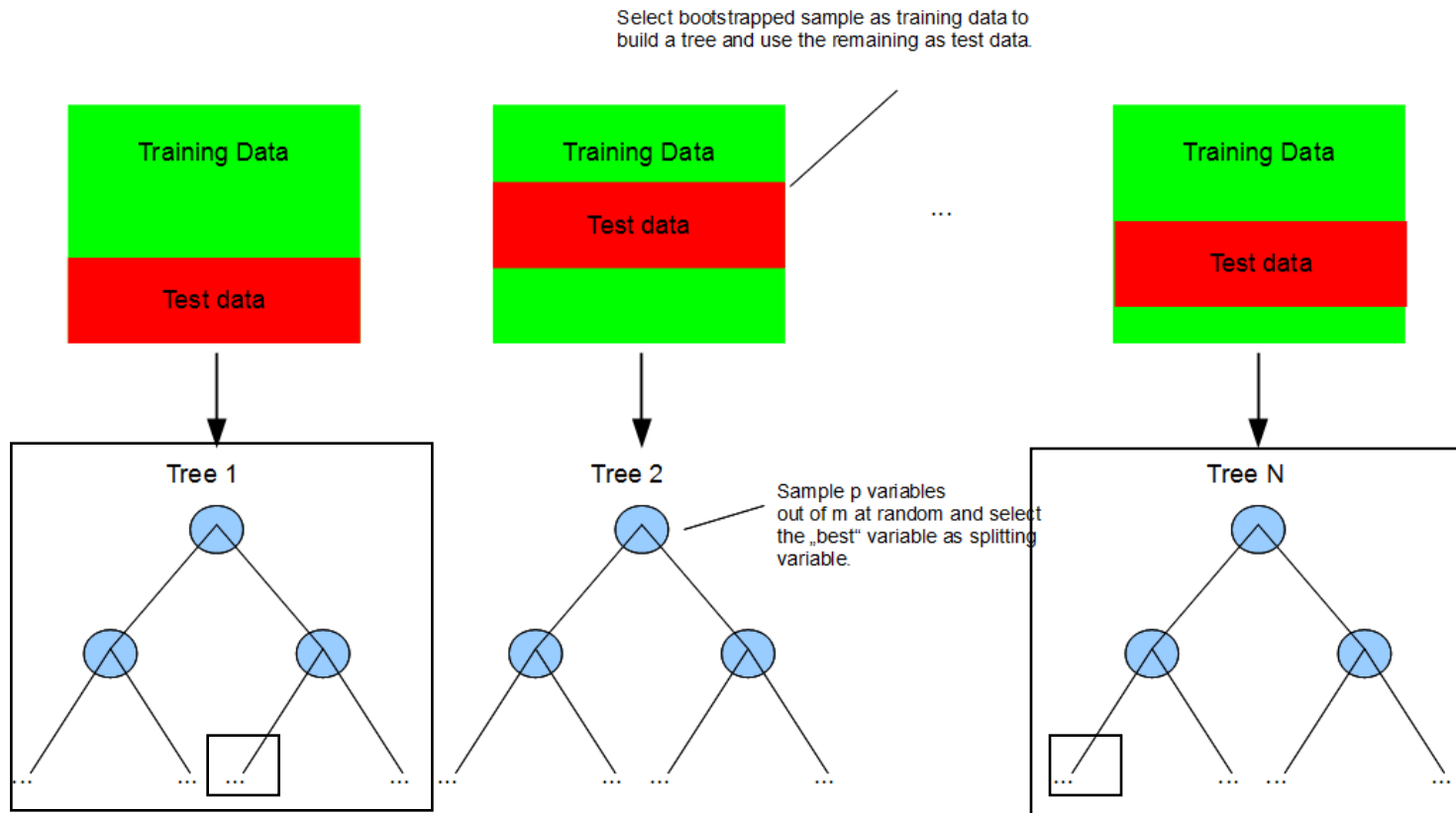
```
library(randomForest)
library(ElemStatLearn)
heart = SAheart
#importance = TRUE sonst keine Accuracy
fit = randomForest(as.factor(chd) ~ ., data =
heart, importance=TRUE)
predict(fit, data=kyphosis, type='prob')
varImpPlot(fit)
```



# Interpreting Variable Importance

- RF in standard implementation is biased towards continuous variables
- Problems with correlations:
  - If two variables are highly correlated removing one yield no degeneration in performance
  - Same with Gini index measure
  - Not specific for random forest: correlation often poses problems

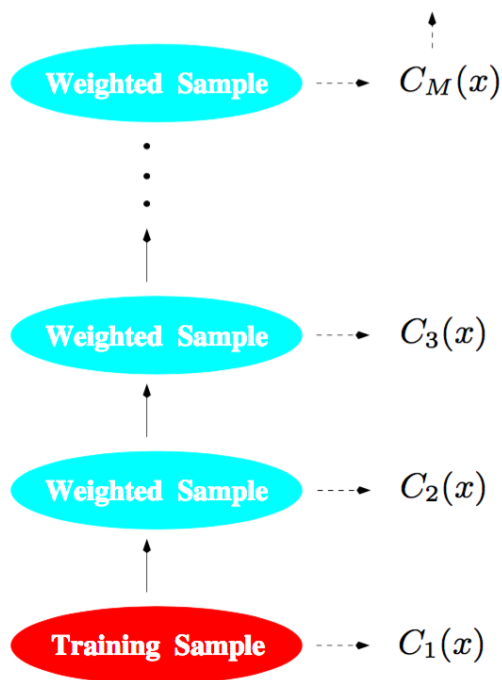
# Proximity: Similarity between two observation according to supervised RF



The test data contains now a pair of observation **random forest determines proximity by counting in how many trees both observation end up in the same leaf**. Since the RF was built in a **supervised modus**, the proximity is also **influenced by the class label** of the observations.

# Boosting

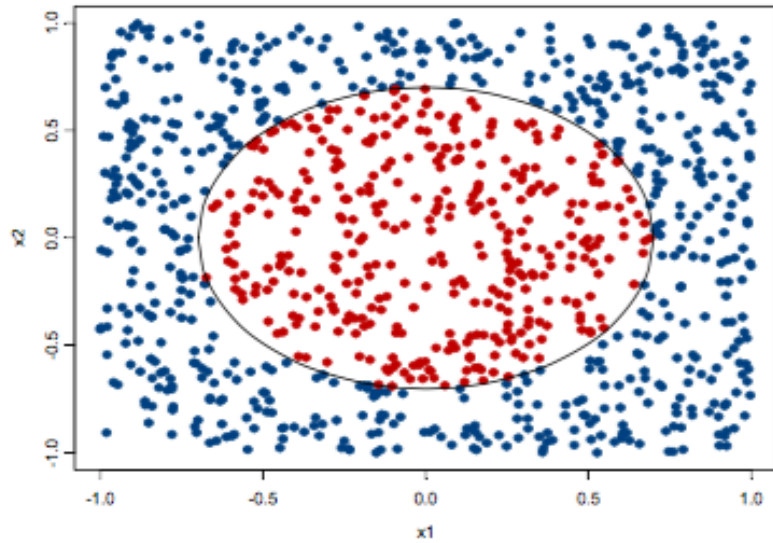
- Consider binary problem with classes coded as +1,-1
- A sequence of classifiers  $C_m$  is learnt
- The training data is reweighted (depending on misclassification of example)



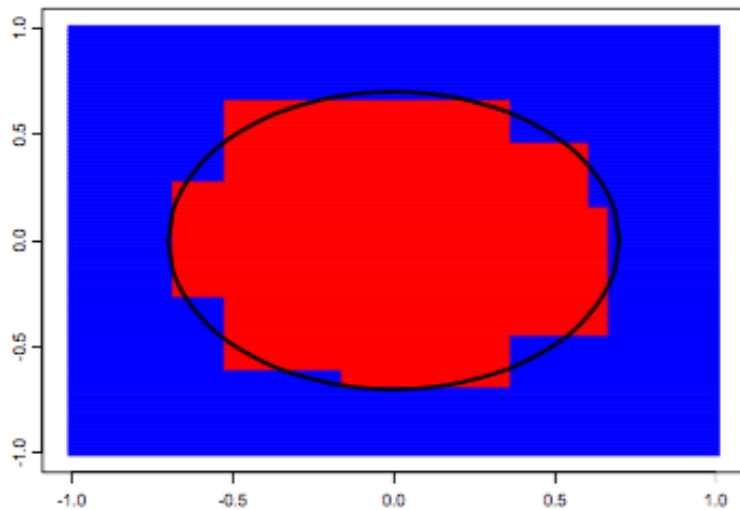
- Average many trees, each grown to re-weighted versions of the training data.
- Final Classifier is weighted average of classifiers:

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

# An experiment

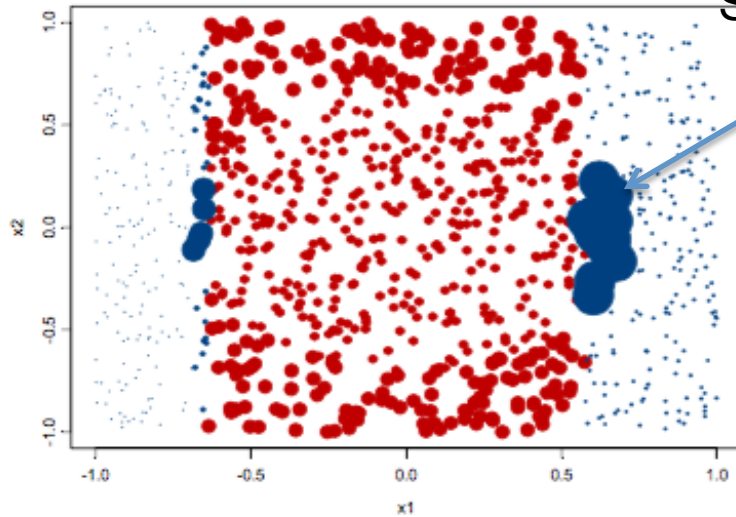


Decision Boundary of a single tree. E.g.

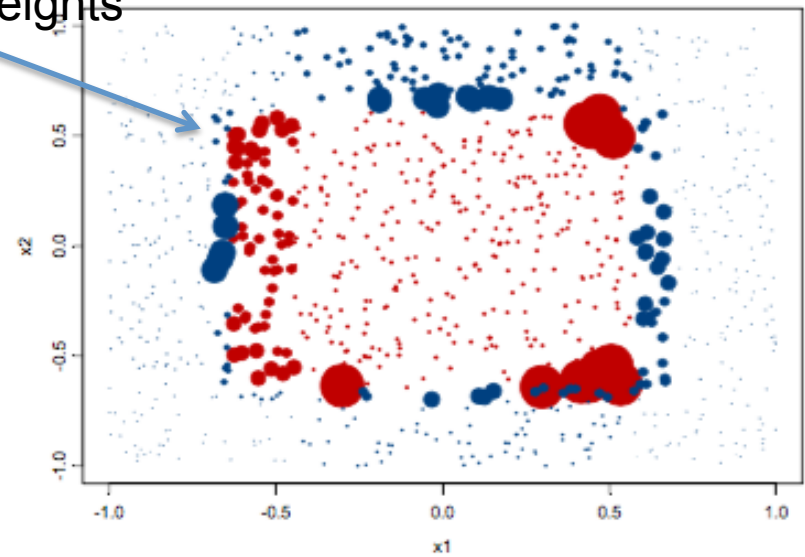


# An experiment

Decision Boundary after 1 iteration

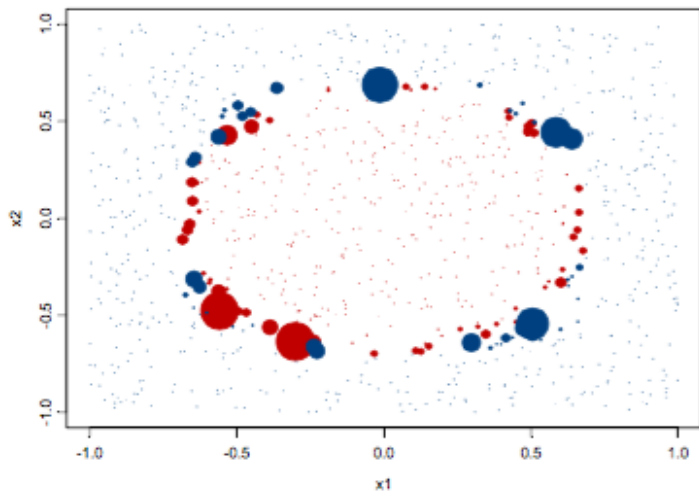


Decision Boundary after 3 iterations

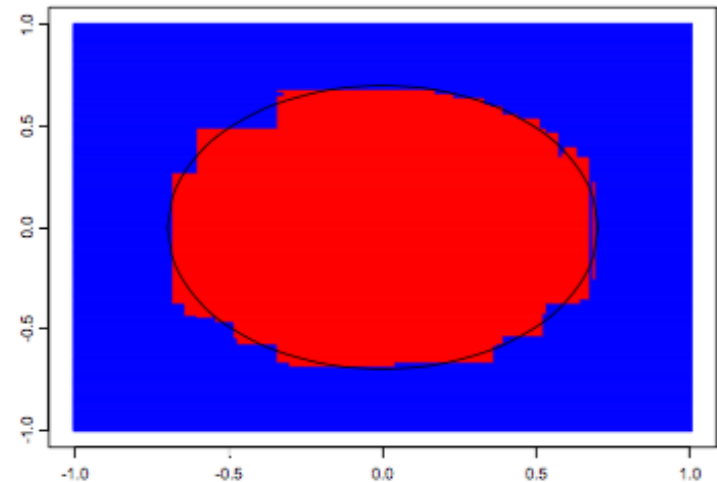


Strong Weights

Decision Boundary after 20 iterations



Decision Boundary after 100 iterations





# Idea of Ada Boost Algorithm

## Basic Idea:

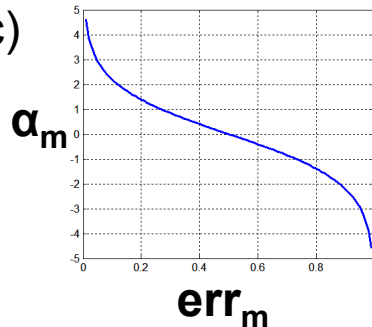
- start with identical weights  $w_i = 1/n$ , fit a learner  $f_1(\cdot)$  and evaluate its insample prediction performance for  $y_i$
  - depending on whether or how heavily an observation  $i$  was misclassified, increase its weight  $w_i$ . Hence the learner is forced to focus on the difficult-to-classify instances.
  - the contribution of the learner  $f_1(\cdot)$  to the final classifier  $F(\cdot)$  is gauged by the averaging weight  $\alpha_1$ . It is large if  $f_1(\cdot)$  performed well, and small otherwise.
- Repeat this process  $M$  times to obtain the solution  $F_M(\cdot)$

# 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  $m = 1, 2, \dots, M$ :
  - 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  $F_M(x) = \text{sign} \sum_{m=1}^M \alpha_m f_m(x)$

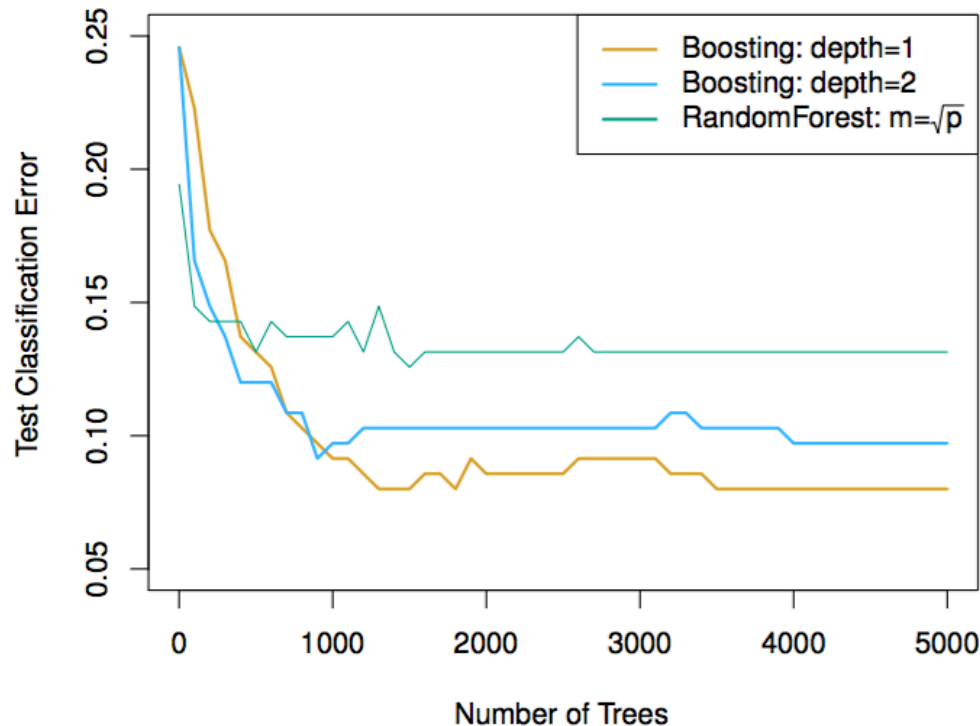
Zu c)



Error small  $\rightarrow$  large weight

Error  $> 0.5$  opposite is taken (quite uncommon only for small  $m$ )

# Performance of Boosting



Boosting most frequently used with trees (not necessary)  
Trees are typically only grown to a certain depth (1,2).  
If trees of depth 2 would be better than trees of depth 1 (stubs)  
interactions between features would be important