Biostatistics week 11

Tree based models for regression and classification

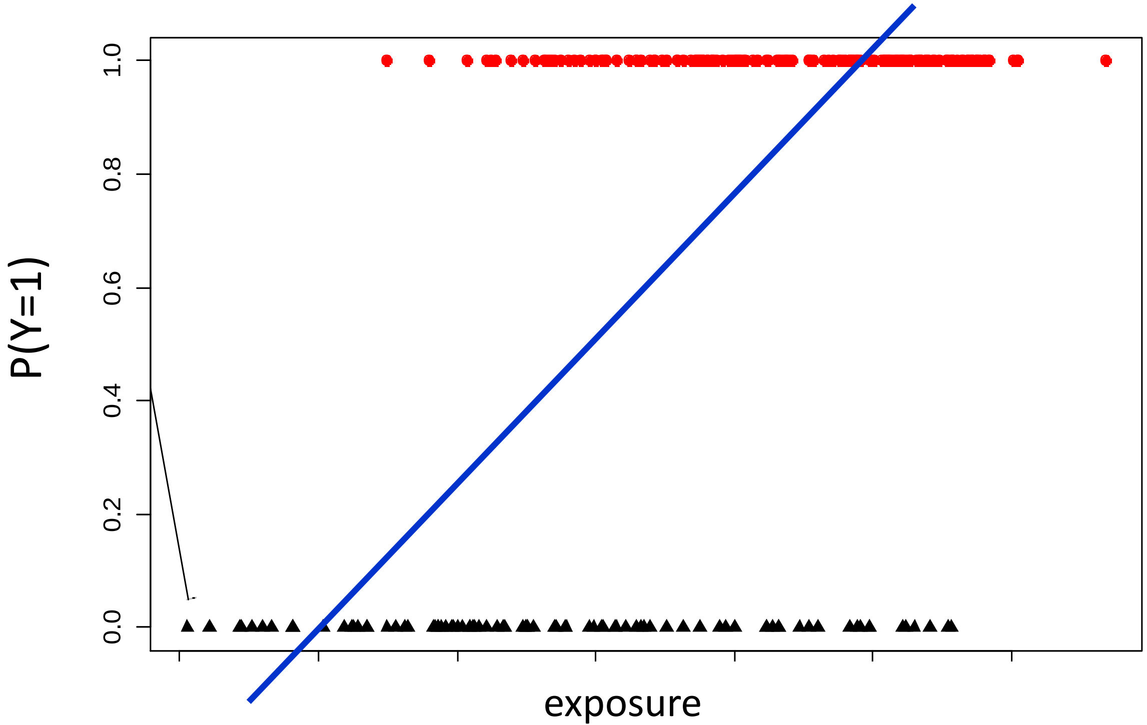
- Why going from linear regression to tree models?
- Tree models for binary or categorical outcomes
- Tree models for continuous outcomes
- Idea of Ensemble models: bagging (boosting – not covered)
- Random Forest for regression and classification
  - tree bagging
  - evaluation -> out of bag (oob) error
  - variable importance measures -> variable selection
  - stratified sampling for unbalanced training data

Lecturer: Beate Sick

Linear Regression is e.g. not suited for binary outcomes

Visualize the fitted model together with data.

```
fit = lm( y ~ exposure, data=my.dat)
```

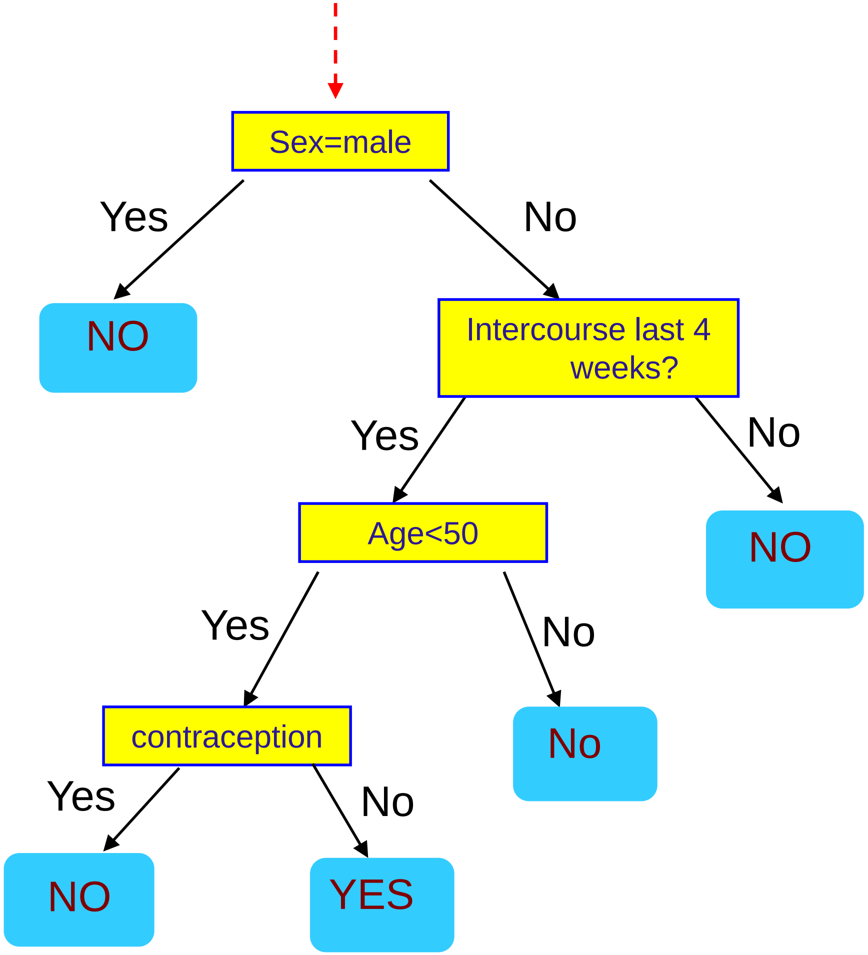


Problems:

1) linear model can **yield impossible values for p outside [0,1]**
2) model assumptions are violated, since residuals are <u>not</u> ~N(0,σ^2)

Example of a classification tree aka as decision tree

Patient suffers from nausea
Is patient pregnant?



Test Data

Sex	Inter-course	Contra-ception	age	Preg-nant
female	No	No	**32**	**?**

The response is categorical

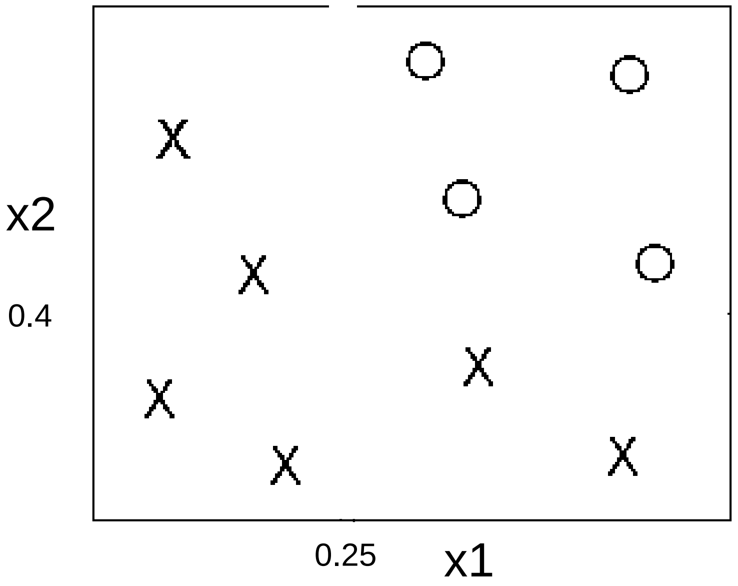
Example:
Model "pregnant" by some properties of the patient (sex, …)

How to train a classification tree: your turn

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

```
for each region $R$
    for each attribute $x_j$ in $R$
        for each possible split $s_j$ of $x_j$
            record change in score when we partition $R$ into $R^l$ and $R^r$
Choose $(x_j , s_j )$ giving maximum improvement to fit
Replace $R$ with $R^l$; add $R^r$
```

- Draw 2 splits to separate the data.
- Draw the corresponding tree

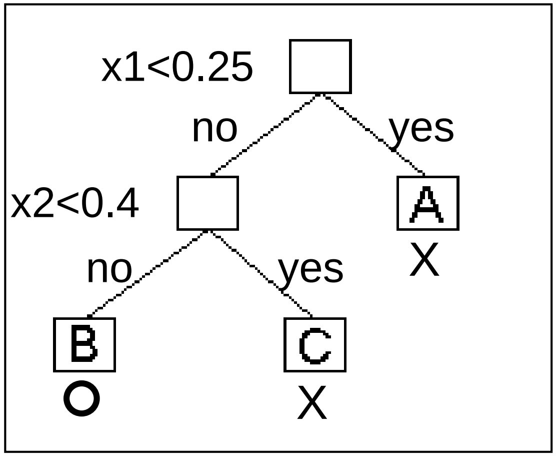


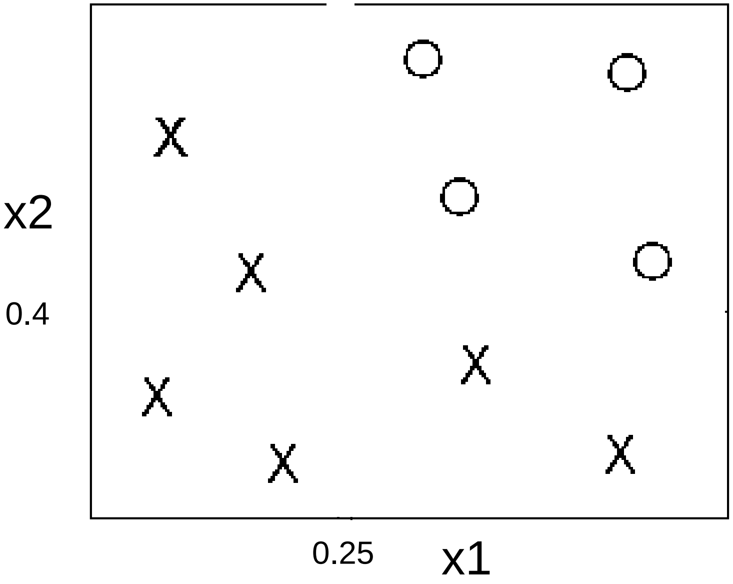
How to train a classification tree: [solution]

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

```
for each region $R$
    for each attribute $x_j$ in $R$
        for each possible split $s_j$ of $x_j$
            record change in score when we partition $R$ into $R^l$ and $R^r$
Choose $(x_j, s_j)$ giving maximum improvement to fit
Replace $R$ with $R^l$; add $R^r$
```

Score - Node impurity (in a sec)



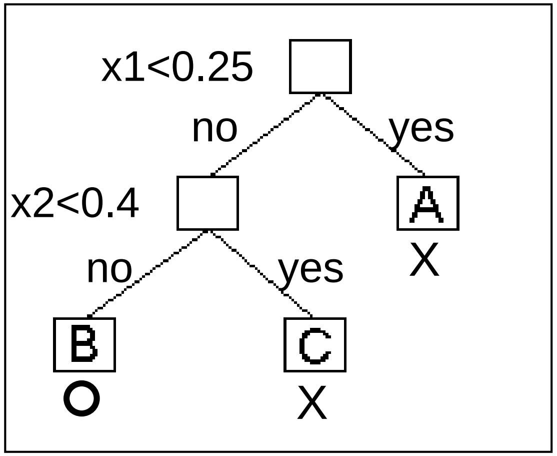


Putting it all together

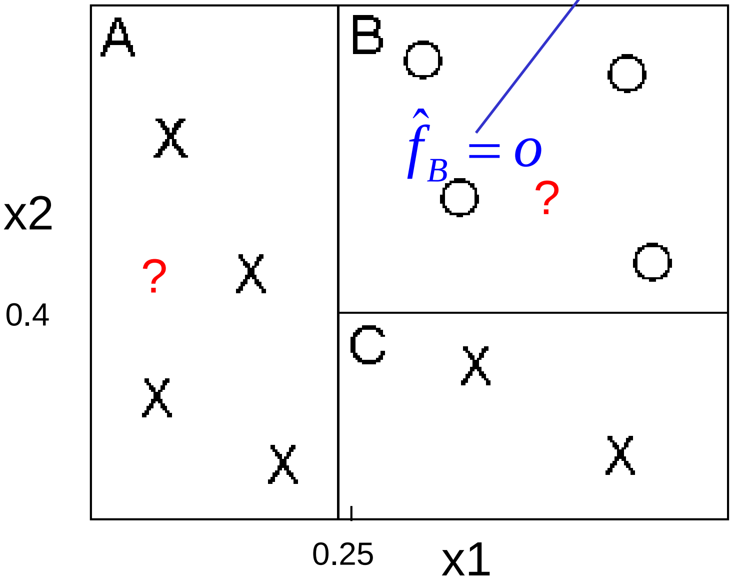
- Starting with a single region -- i.e., all given data
- At the m-th iteration:

```
for each region $R$
    for each attribute $x_j$ in $R$
        for each possible split $s_j$ of $x_j$
            record change in score when we partition $R$ into $R^l$ and $R^r$
Choose $(x_j , s_j)$ giving maximum improvement to fit
Replace $R$ with $R^l$; add $R^r$
```

Score - Node impurity (in a sec)



Model class label is given by the majority of all observation labels in the region



The most common impurity measures

$p_i$is the relative frequency of class $i = 1 \dots C$

Gini Index (default rpart)

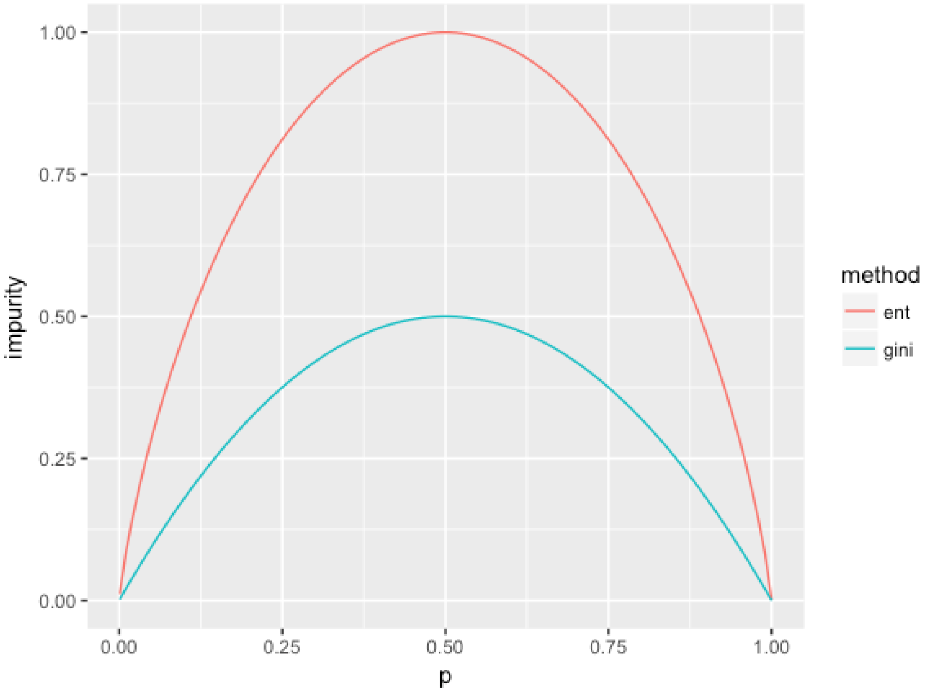
$$\text{Gini} = 1 - \sum_i^C p_i^2$$

Entropy:

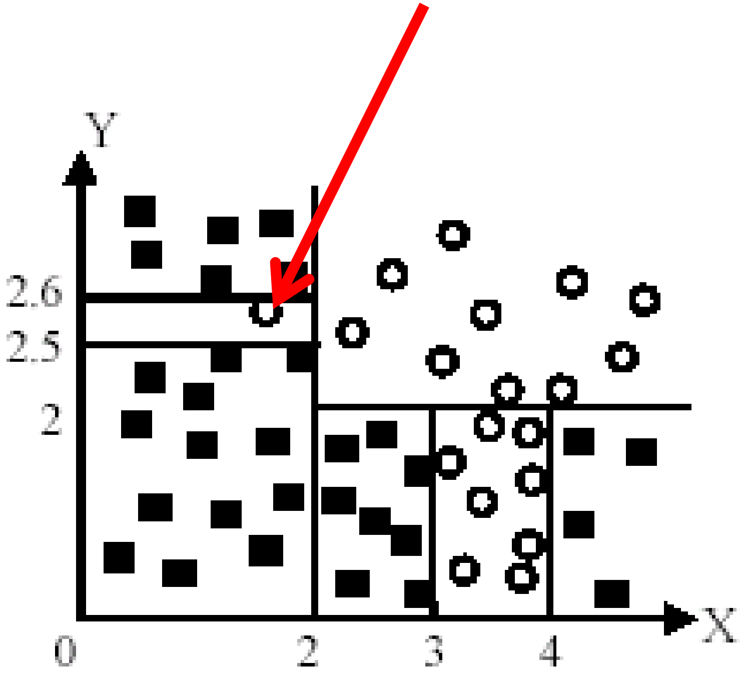
$$\text{Entropy} - \sum_i^C p_i \log_2(p_i)$$

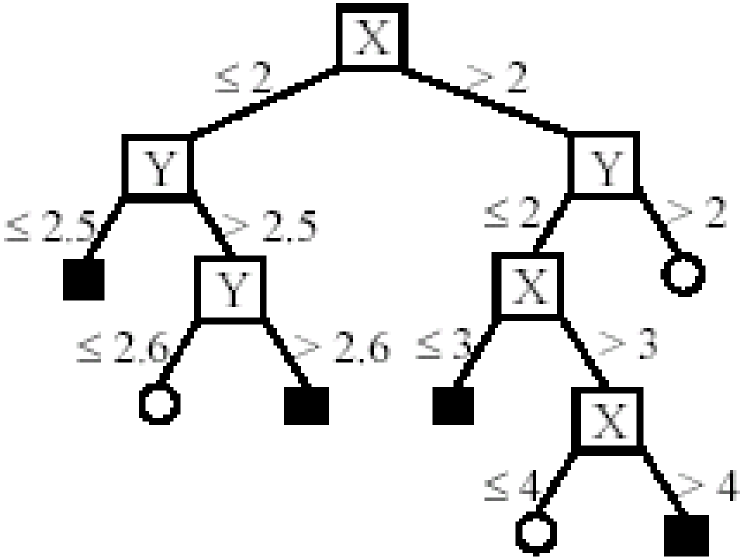
For C=2 classes
p probability for the one class
$1 - p$ for the other

Gini = $1 - p^2 - (1 - p)^2$



Notes on Overfitting

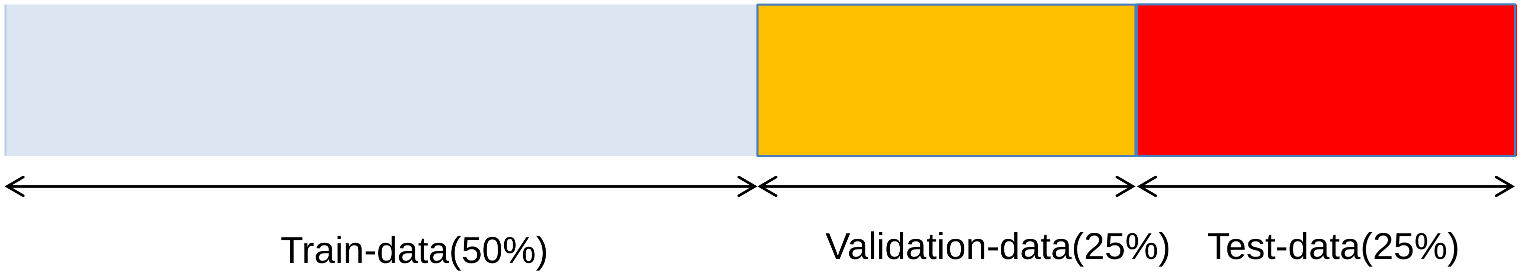




Training error does not provide a good estimate of how well the tree will perform on previously unseen records

Overfitting results in decision trees that are more complex than necessary

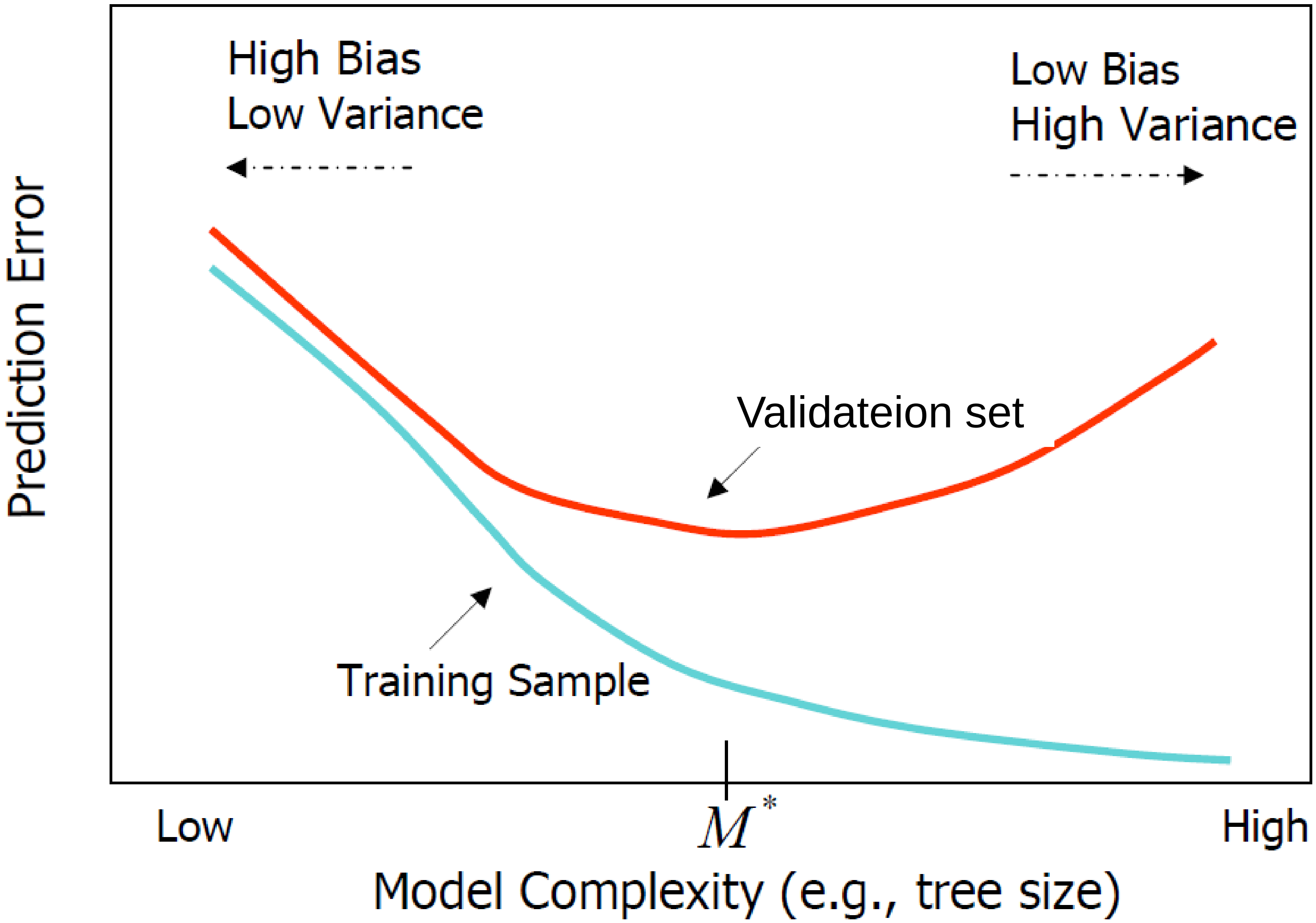
Best practice: Split in Train, Validation, and Test Set



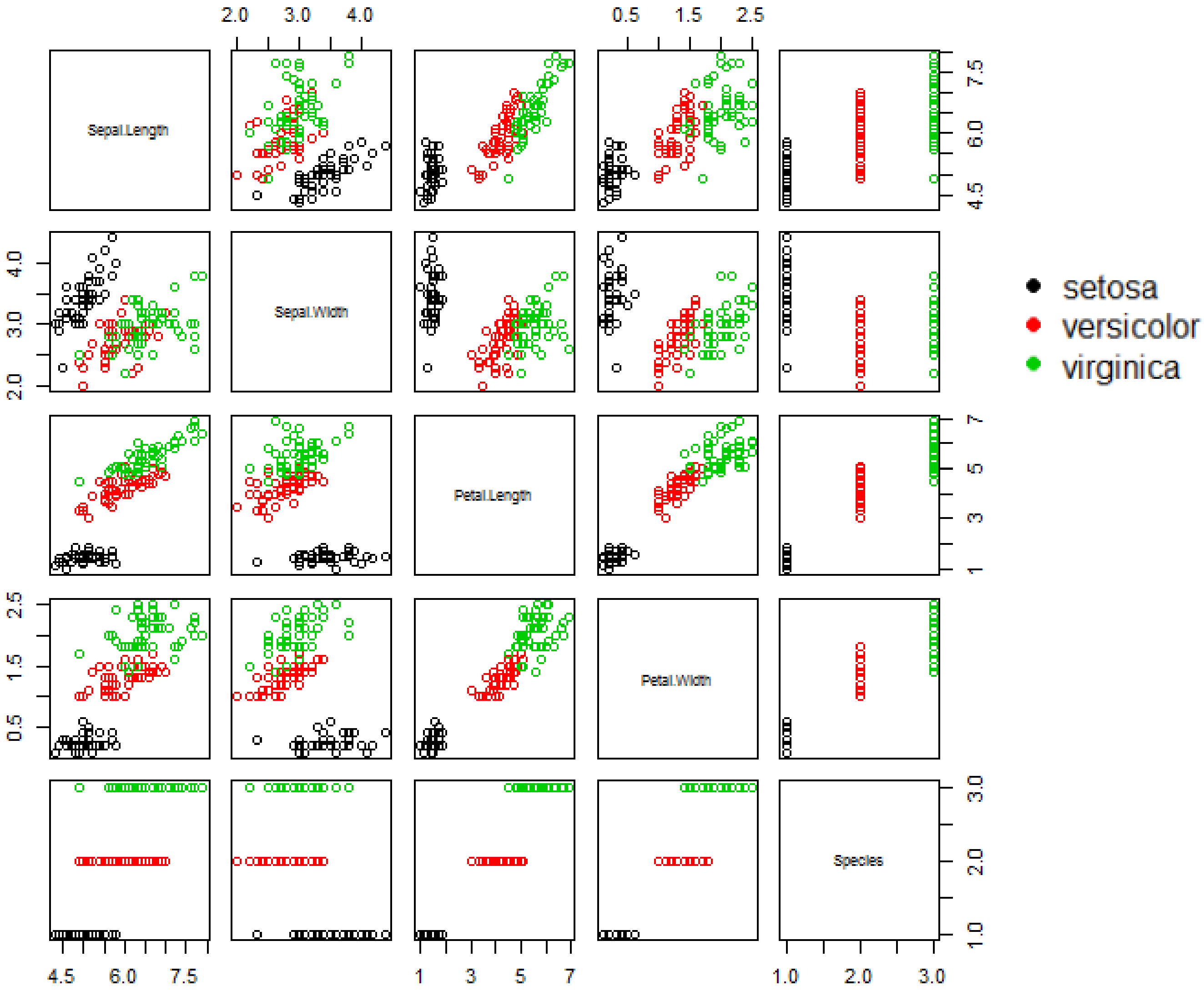
Best practice: Lock an extra test data set away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.
Reason: **When trying many models, you probably overfit on the validation set**.

Determine performance metrics, such as MSE, to evaluate the predictions **on new validation or test data**

What ist the right complexity of a model?



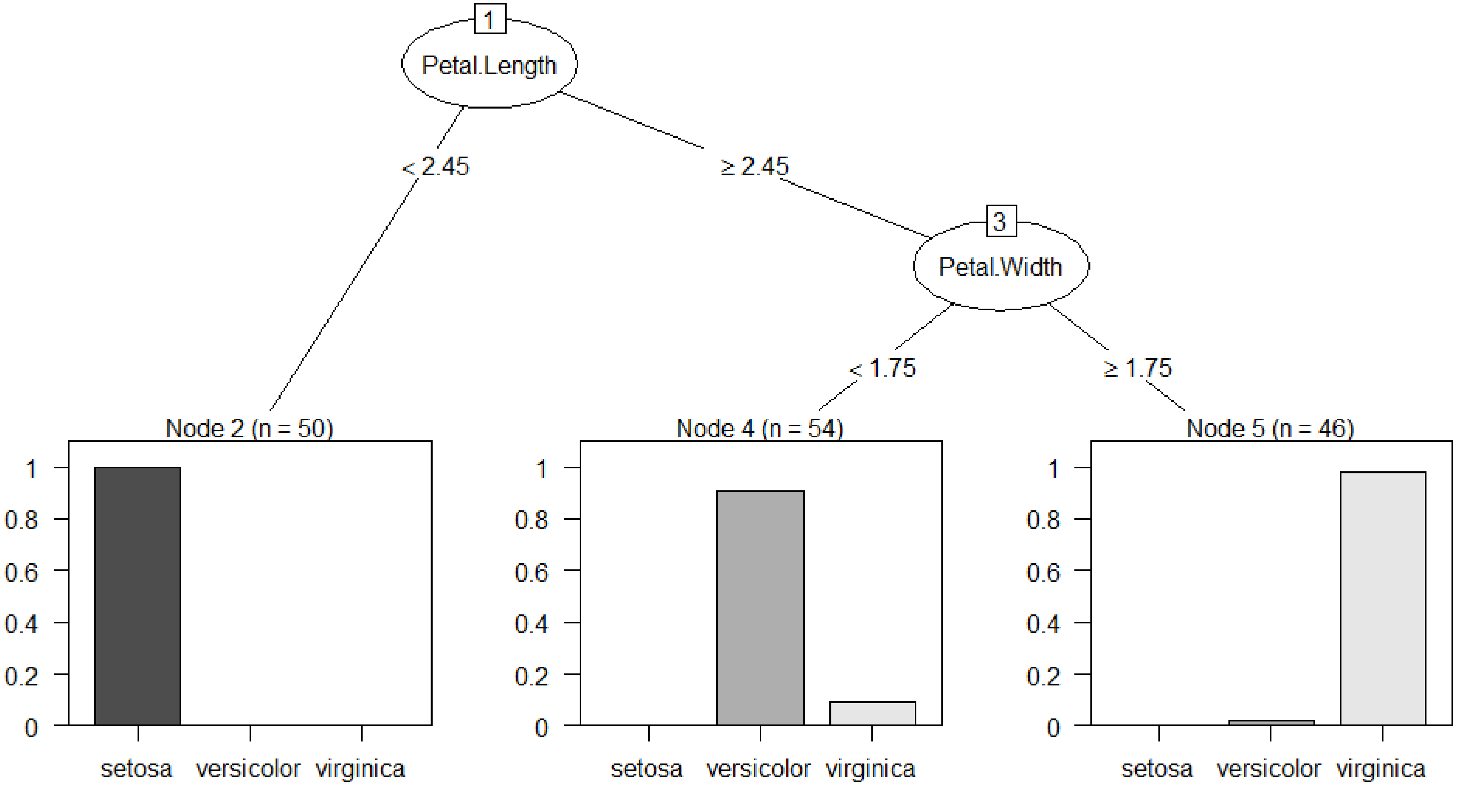
Good old iris



How good can we predict the Species based on the leaf measures?
Which measures are most important to identify the species?

Classification trees based recursive partitioning (rpart) in R

```
library(rpart)     # to fit rpart tree models
library(party)     # only required for nice plots
library(partykit) # only required for nice plots

data("iris")

clas.tree = rpart(Species ~ ., data=iris)
plot(as.party(clas.tree))
```



How to find the tree structure of a regression tree?

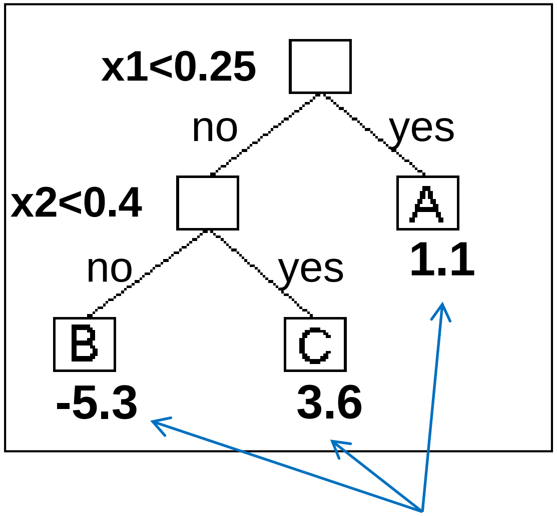
- Starting with a single region -- i.e., all given data
- At the m-th iteration:

```
for each region $R$
    for each attribute $x_j$ in $R$
        for each possible split $s_j$ of $x_j$
            record change in score when we partition $R$ into $R^l$ and $R^r$
Choose $(x_j, s_j)$ giving maximum improvement to fit
Replace $R$ with $R^l$; add $R^r$
```

Score: MSE (mean squared error)

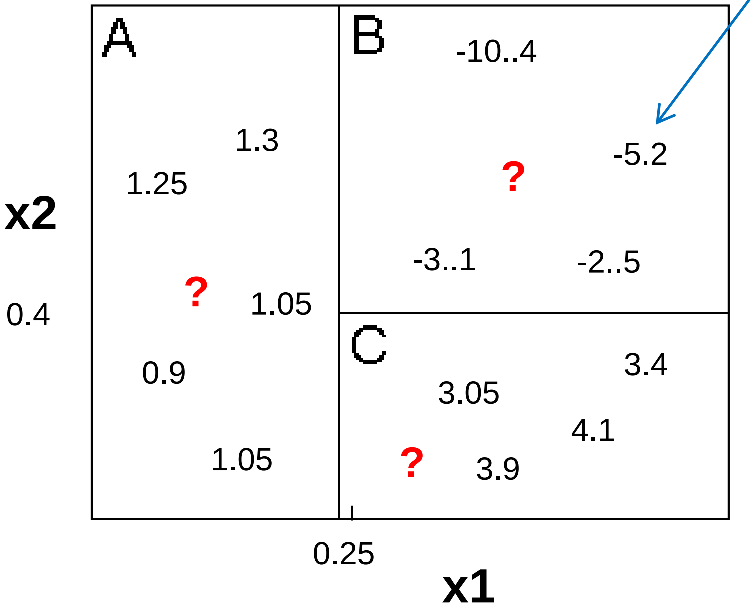
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

numbers indicate values of **continuous outcome y**



Per partition predict one outcome value

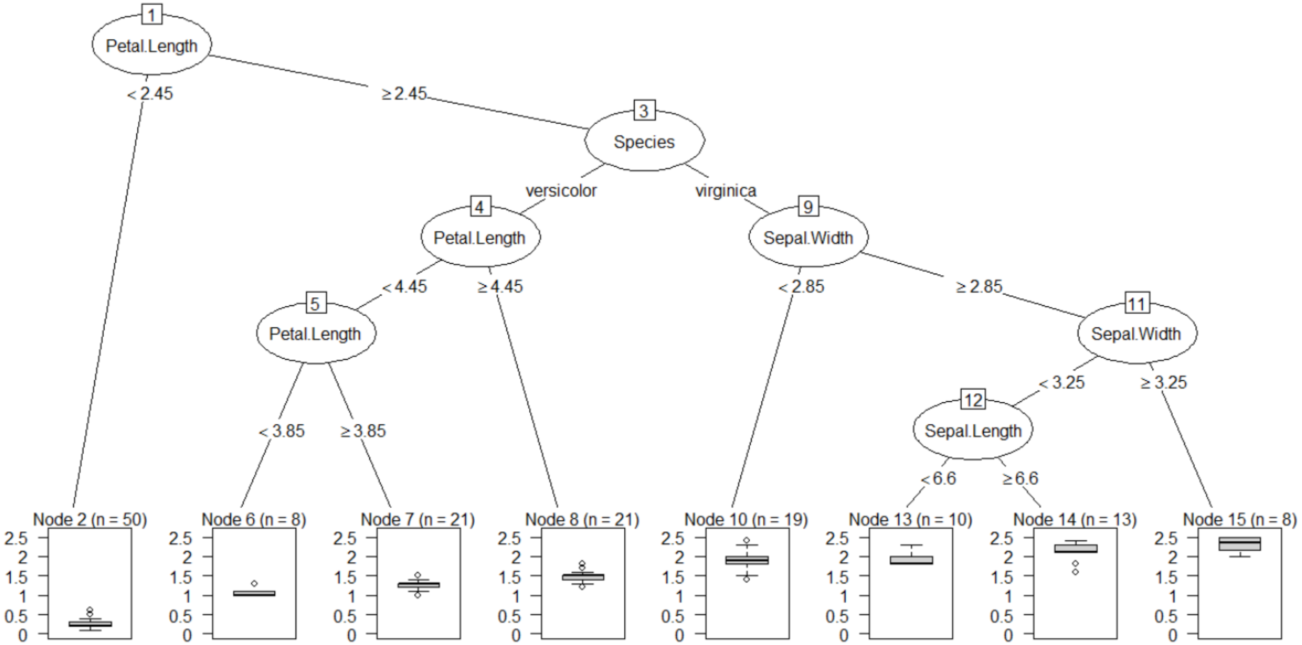
Given by the mean value of the observed data in this region



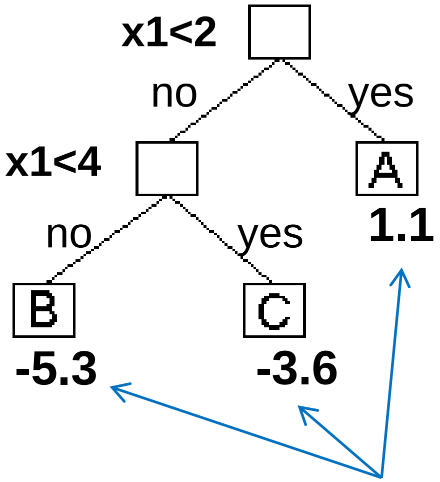
Warning: Since the “fitted/predicted” values are given by the average in a leaf, **tree based regression models cannot extrapolate** and show “shrinkage to the mean”.

Regression trees based recursive partitioning (rpart) in R

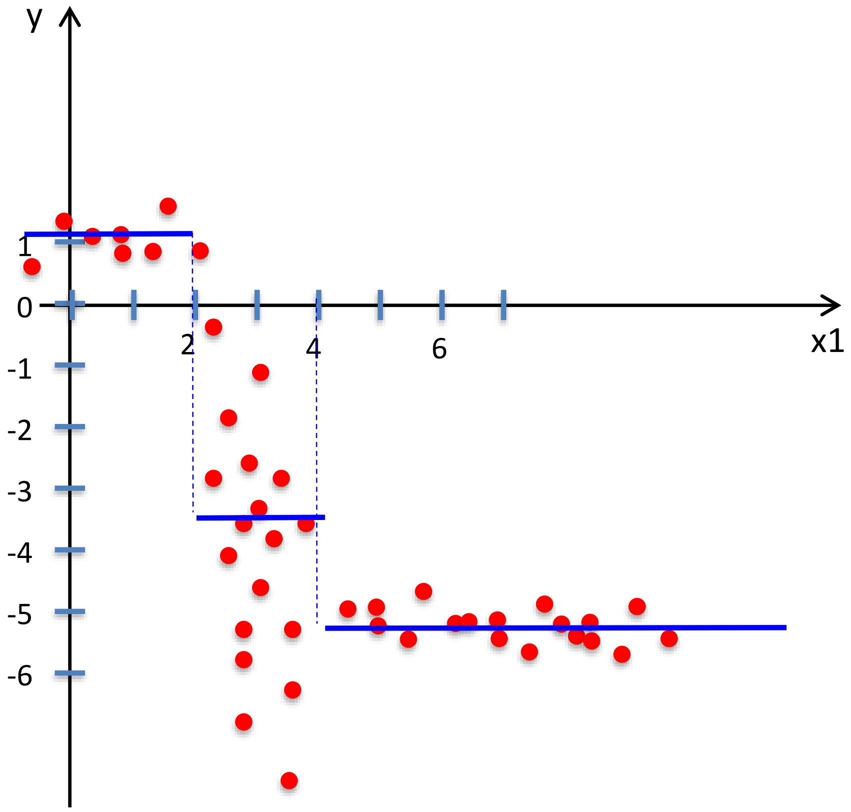
```
library(rpart)     # to fit rpart tree models
library(party)     # only required for nice plots
library(partykit) # only required for nice plots

data("iris")

reg.tree = rpart(Petal.Width ~ ., data=iris)

plot(as.party(reg.tree))
```



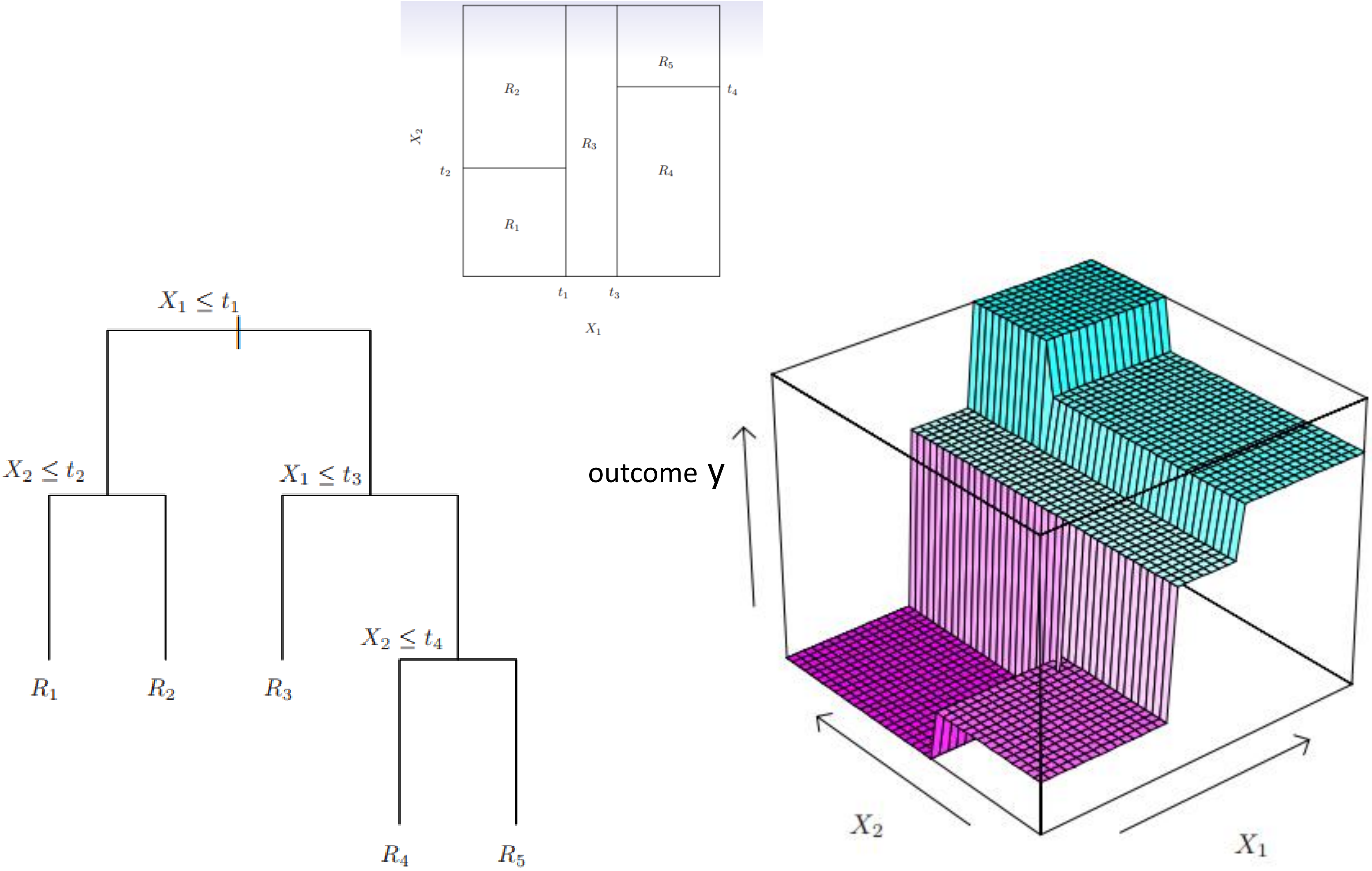
Regression tree with 1 predictor



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

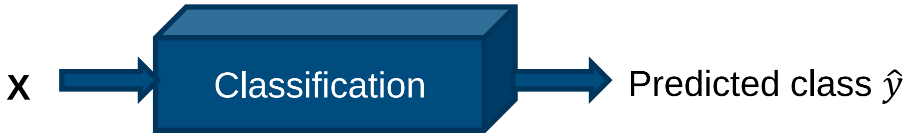


Trees do a recursive partitioning of the predictor space



Confusion matrix: Evalutation of a classifier

Evaluation is done on test set with new instances, that were not used for training, we need to known the true class y and the predicted class $\hat{y}$.

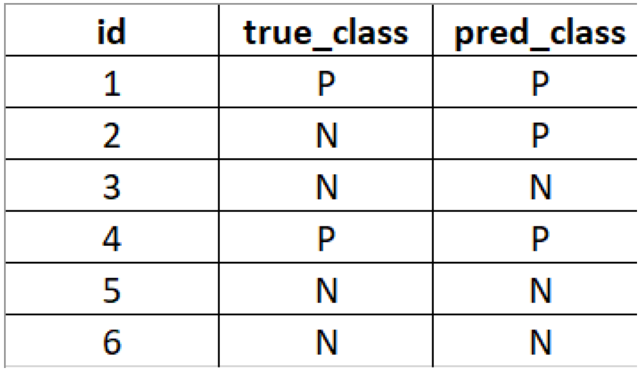


Binary Target Variable Y

Postive or Negative

1 or 0

Yes or No



id	true_class	pred_class
1	P	P
2	N	P
3	N	N
4	P	P
5	N	N
6	N	N

True class

Predicted class		**Positive**	**Negative**
	Positive	**TP=2**	**FP=1**
	Negative	**FN=0**	**TN=3**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

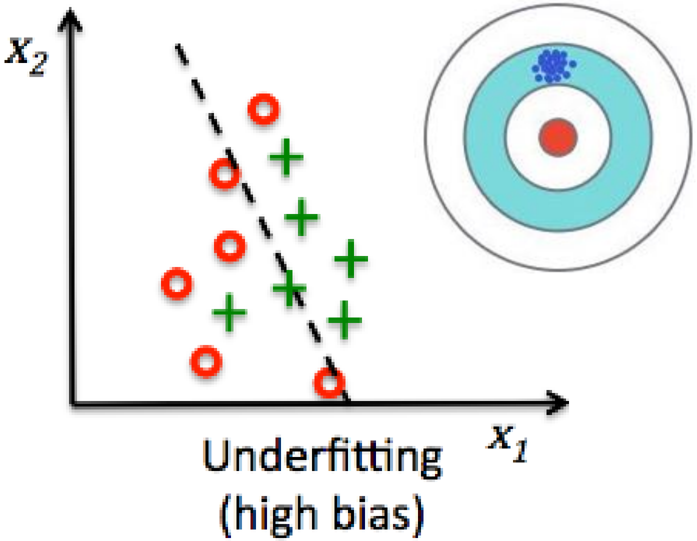
Remark: The concept of a confusion matrix and accuracy is not limited to two classes.

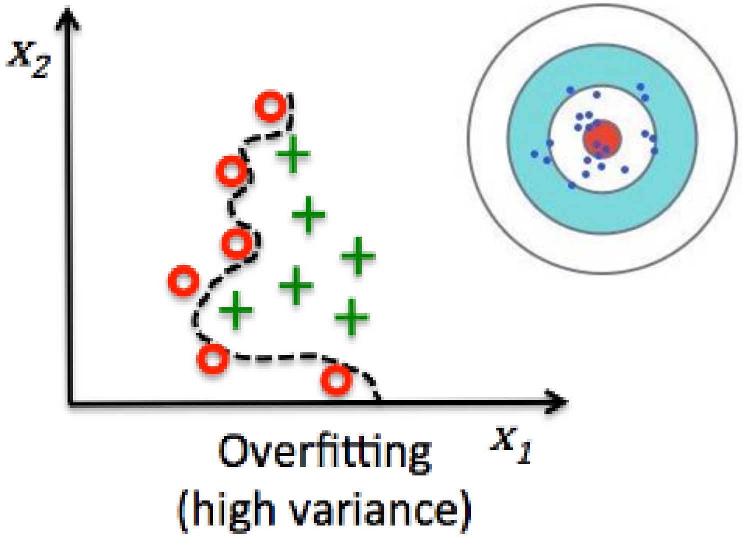
An ideal classifier has the Accuracy=1

Pros and Cons of tree models

- Pros
  - Interpretability
  - No distribution assumptions, robust to outliers
  - No variable transformation necessary (invariant to monotone trafos on predictors)
  - Can capture non-linear structures & local interactions
  - Low bias if appropriate input variables are available and tree has sufficient depth.
- Cons
  - High variation -> for prediction not used so much
  - Tend to overfit (in case of rpart-trees, if not pruned)
  - instability of trees -> interpretation can change easily
  - response surface is not smooth
  - Needs big datasets to capture additive structures or linear structurs

The concept of Bias and Variance of a classification model





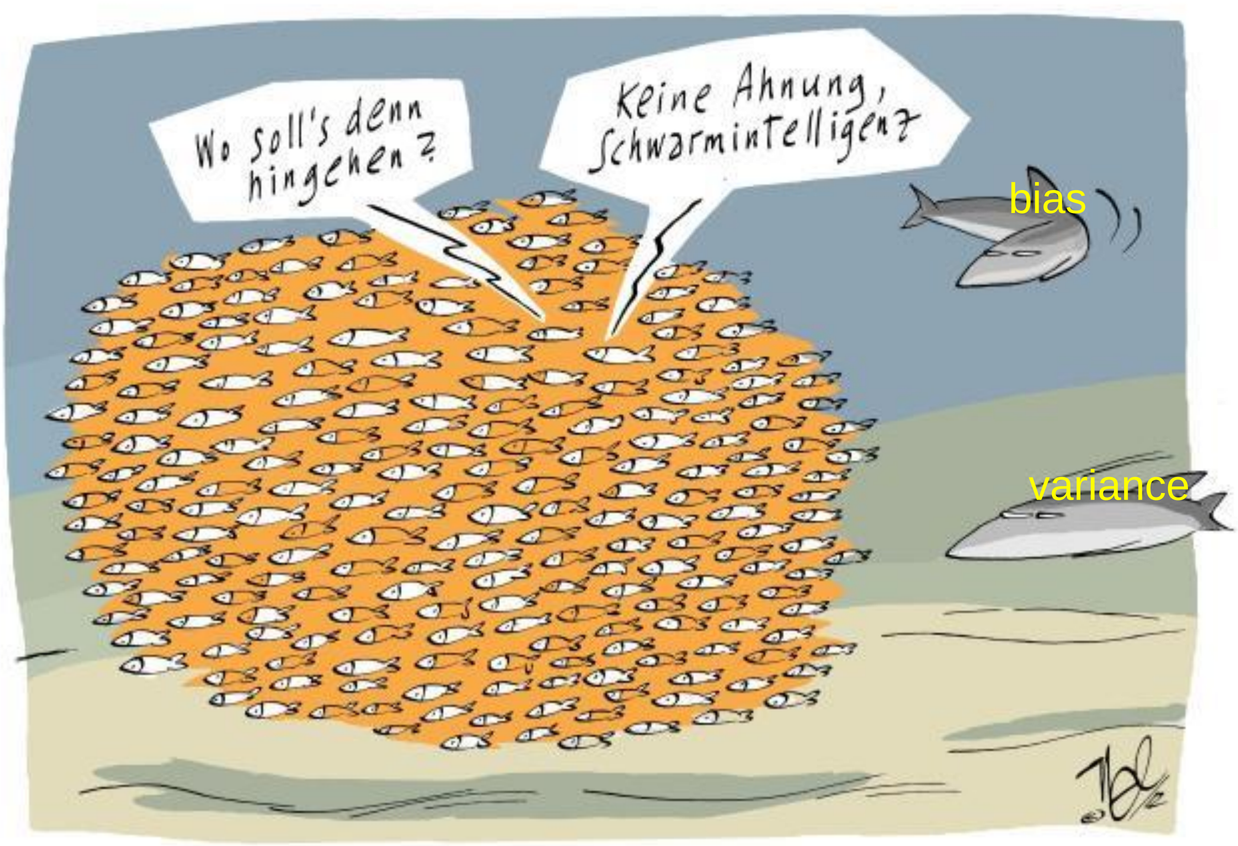
A underfitting classification model

- 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

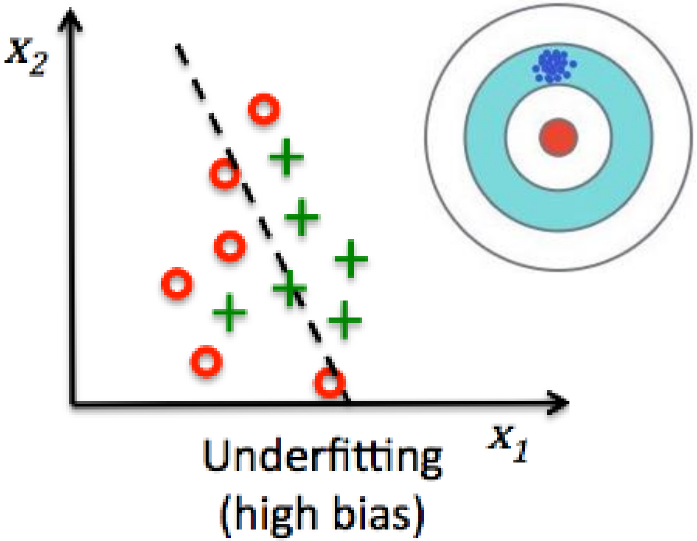
- is too flexible for data structure
- few errors on train set and non-systematic test errors (low bias)
- will vary a lot if fitted to new train data (high variance)

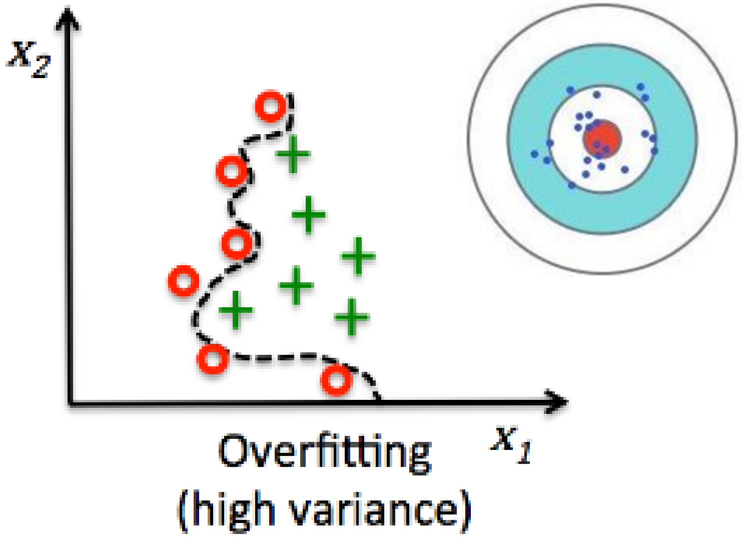
Ensemble methods are the cure!



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

Use ensemble methods to fight under and overfitting





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

Bagging
&
Random Forest

Bagging as ensemble of parallel fitted models

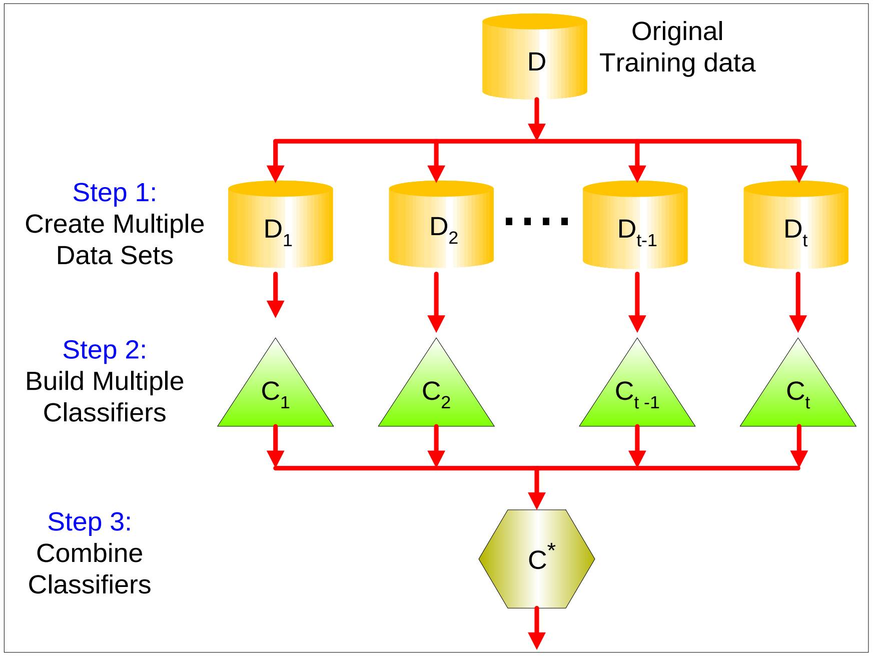
Each classifier tends to overfit its version of training data.

Bagging: bootstrapping and averaging

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

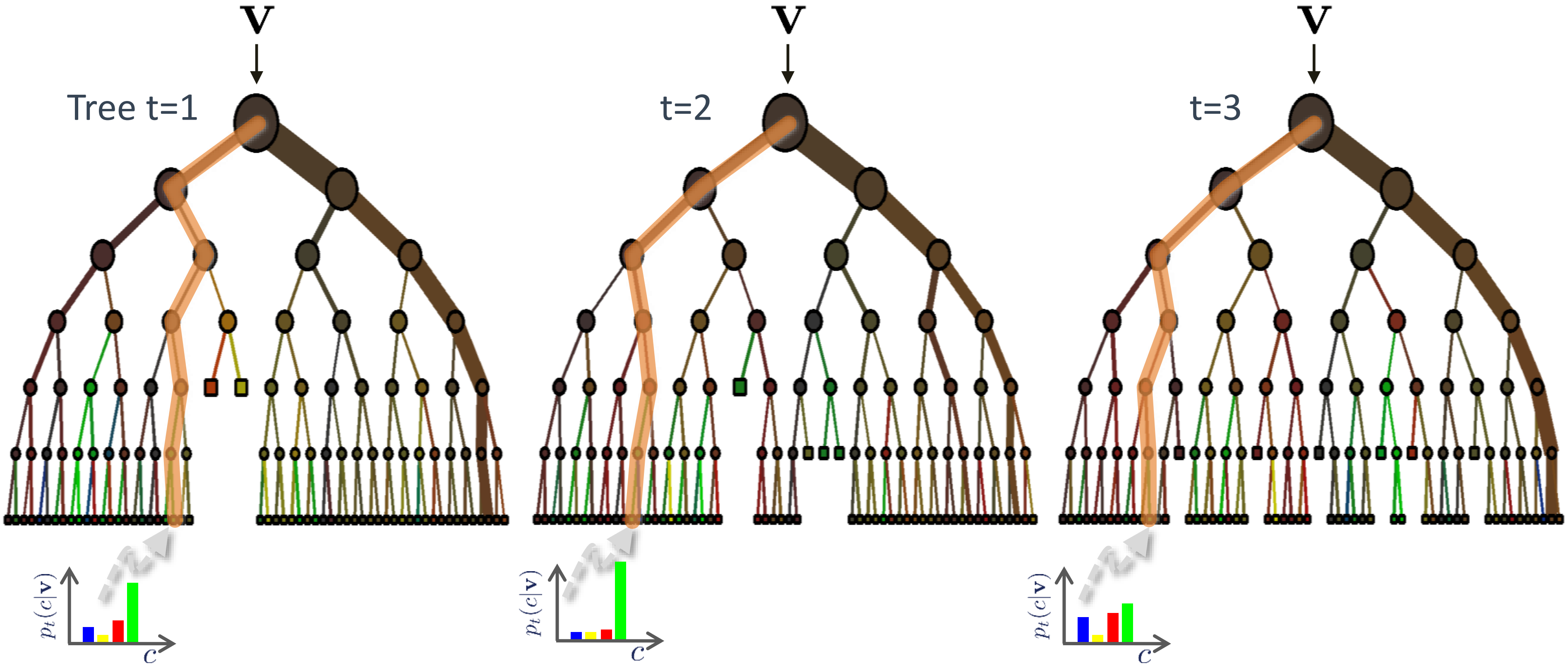
2) Minimize Bias by using flexible models and allow for overfitting

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 nore hurt.

How to classify a new observation v with a random forest?

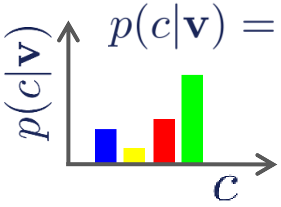


The to derive the ensemble result:

a) Each tree has a “winner-class”
Take the class which was most often the winner

b) average probabilities:

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$

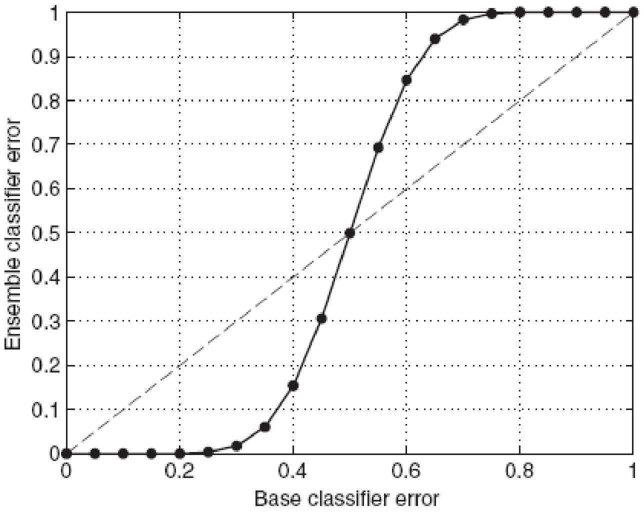


Animation taken from microsoft

Why does it help to aggregate over many classification trees?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, ε = 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)

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



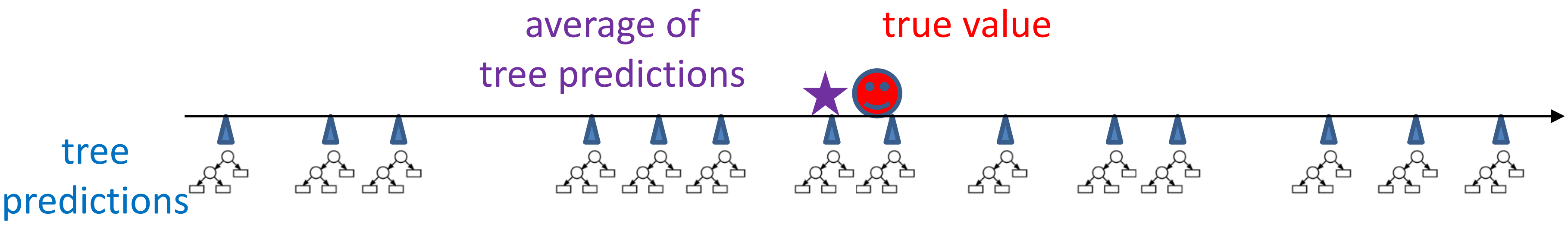
=> Ensembles are only better than one classifier, if each classifier is better than random guessing!

Source: Tan, Steinbach, Kumar

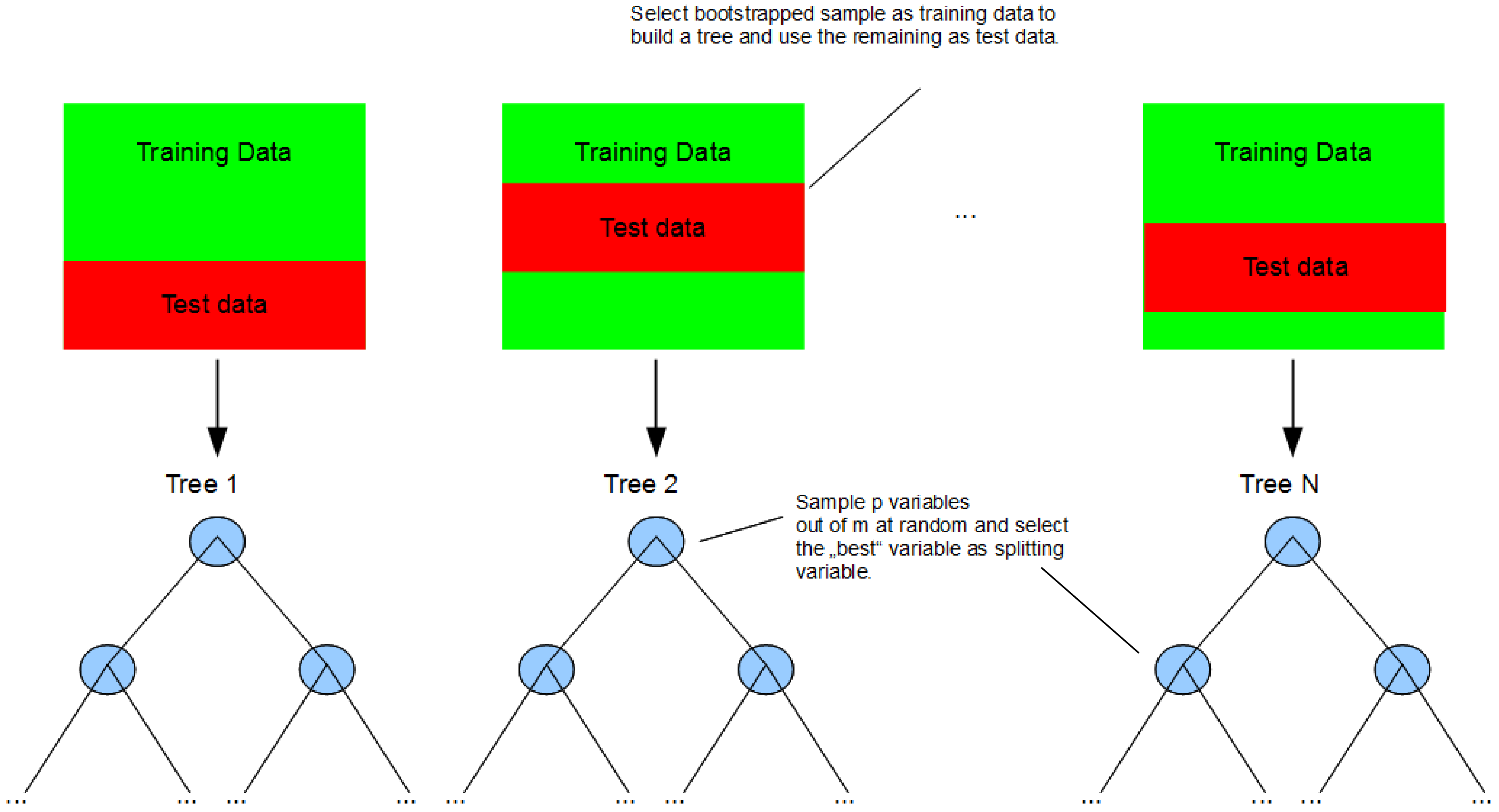
Why does it help to aggregate over many regression trees?

- Suppose there are n=25 deep regression trees
  - All trees are deep -> the trees have no or only small bias

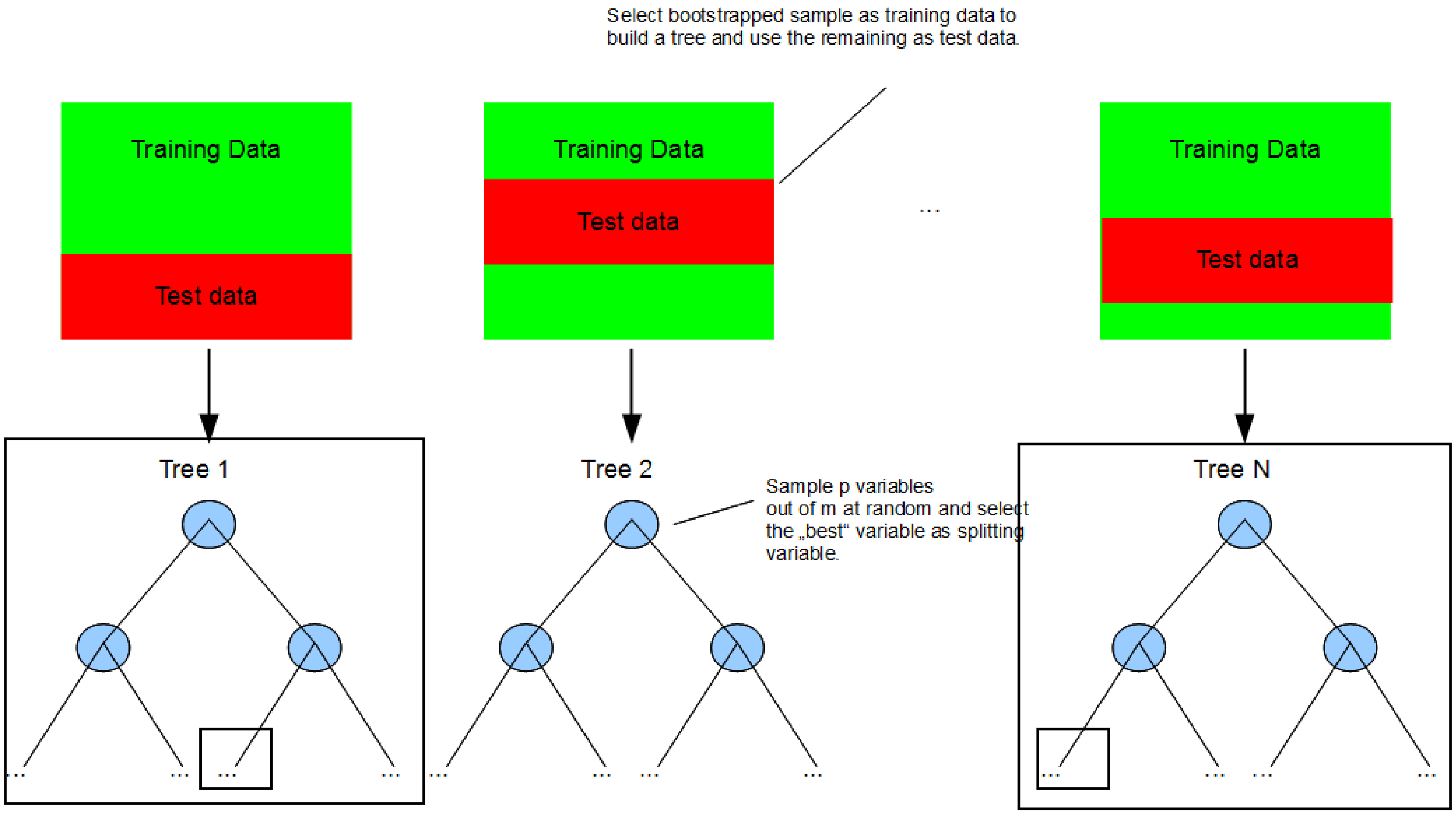
    -> the trees have high variance
  - According to the Central Limit Theorem the average of the predictions of n regression trees have the same expected value as the single trees but a standard deviation which is reduced by a factor of $\sqrt{n} = \sqrt{25} = 5$.



Random Forest: Learning algorithm


Select bootstrapped sample as training data to build a tree and use the remaining as test data.
Training Data
Test data
Training Data
Test data
...
Training Data
Test data
Tree 1
Tree 2
Sample p variables out of m at random and select the „best“ variable as splitting variable.
Tree N
...
...
...
...
...
...
...
...
...
...
...
...

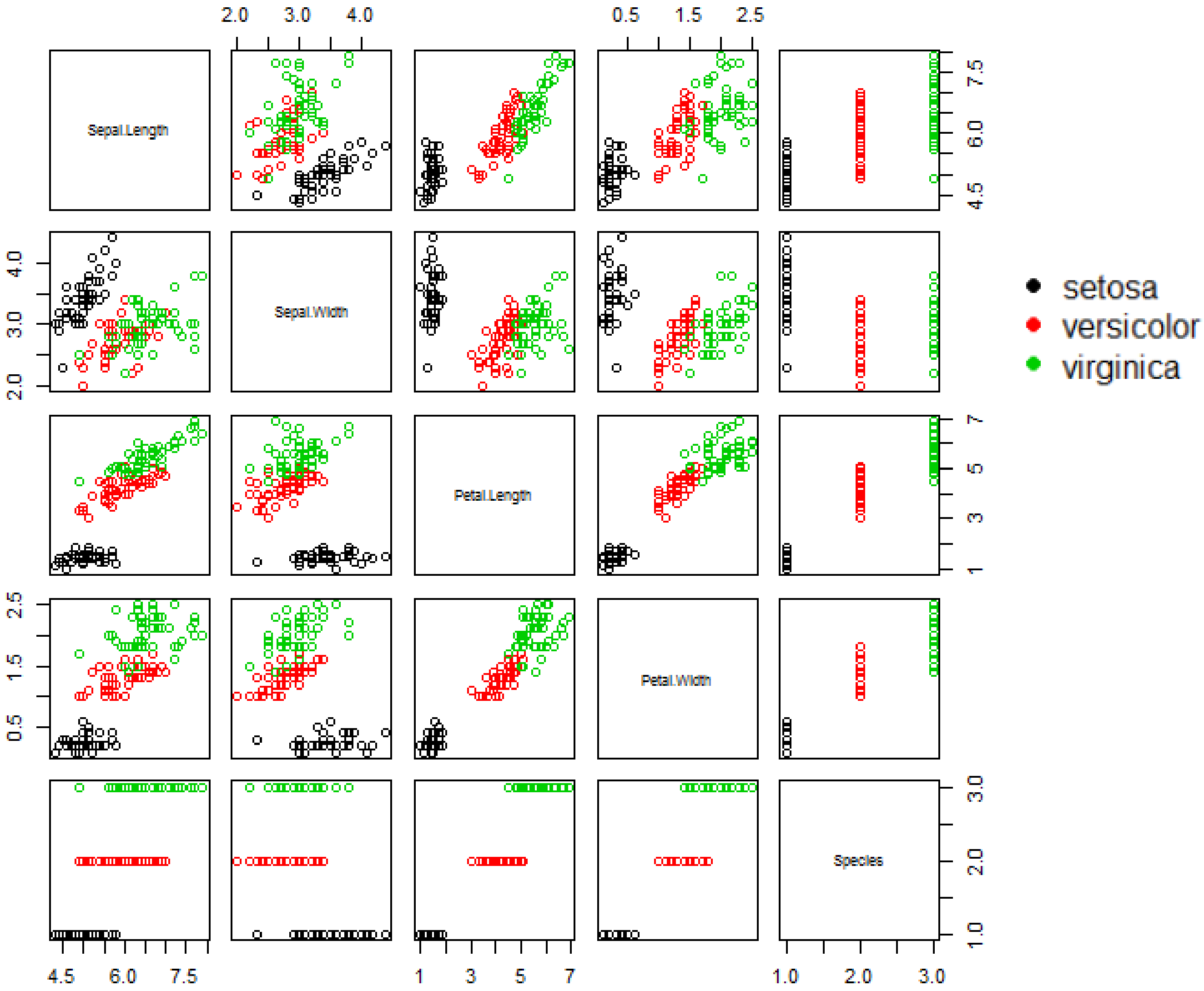
Oob-error: Out of Bag Evaluation of the Random Forest


Select bootstrapped sample as training data to build a tree and use the remaining as test data.
Training Data
Test data
Training Data
Test data
...
Training Data
Test data
Tree 1
Tree 2
Sample p variables out of m at random and select the „best" variable as splitting variable.
Tree N
...

For each observation, construct its random forest oob-predictor by averaging only the results of those trees corresponding to bootstrap samples in which the observation was not contained.

When using the predict –function on a rf-fit-object, by default oob predictions are given if the newdata-argument is not used.

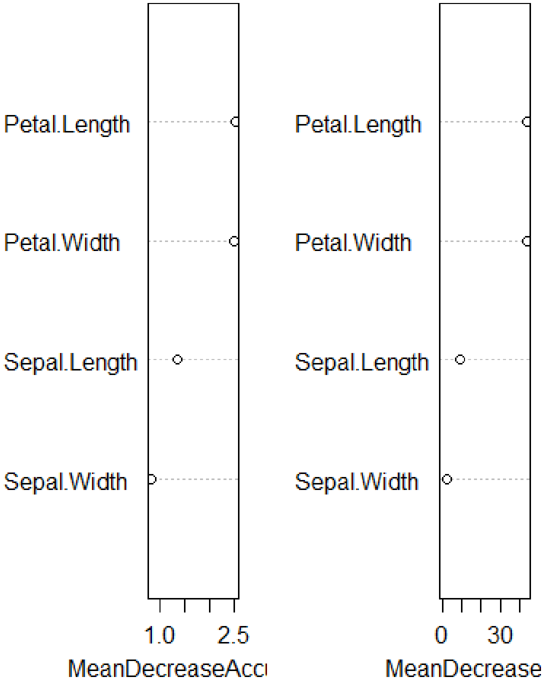
Good old iris



How good can we predict the Species based on the leaf measures?
Which measures are most important to identify the species?

Random Forest for classifying iris Species

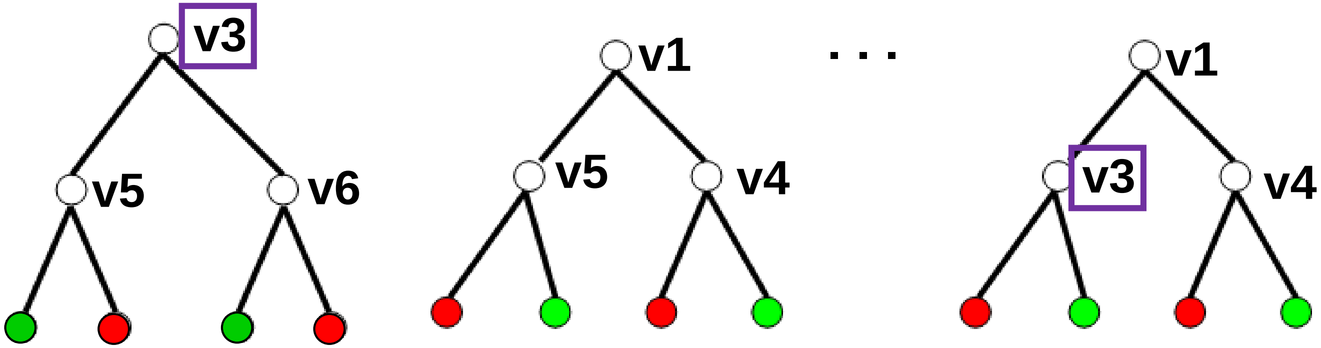
```
> library(randomForest)
> iris.rf = randomForest(Species ~ .,
           data = iris, importance = TRUE)
> print(iris.rf)
Call:
 randomForest(formula = Species ~ .,
           data = iris, importance = TRUE)
       Type of random forest: classification
             Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 6%
OOB Confusion matrix:
           setosa versicolor virginica class.error
setosa         50          0         0        0.00
versicolor      0         47         3        0.06
virginica       0          6        44        0.12
> varImpPlot(iris.rf)
```



Measures on petal leafs are more important for classification than sepal measures

Assess importance of variables

Variable Importance 1: Score improvement at each Split



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

Variable Importance 2: performance-loss by permutation

Determine importance-2 for v4:

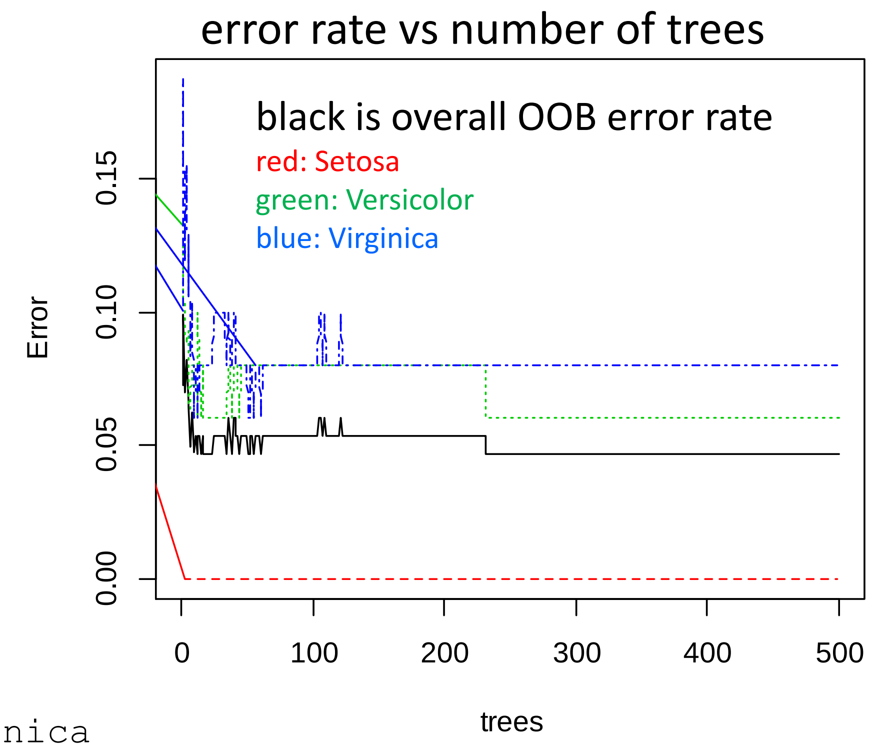
1) obtain standard oob-performance
2) values of v4 are randomly permuted in oob samples and oob-performance is again computed
3) Use decrease of performance as important measure of v4

v1	v2	v3	v4	v5	v6	v7
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

RandomForest: How to check if we have enough trees?

```
➢ names(iris.rf)
 [1] "call"             "type"
 [3] "predicted"        "err.rate"
 [5] "confusion"        "votes"
 [7] "oob.times"        "classes"
 [9] "importance"       "importanceSD"
[11] "localImportance" "proximity"
[13] "ntree"            "mtry"
[15] "forest"           "y"
[17] "test"             "inbag"
[19] "terms"
```

```
# get for each #trees in rf
# corresponing error rate
> head(rf.clas$err.rate)
            OOB     setosa versicolor   virginica
[1,] 0.07272727   0    0.13333333   0.1000000
[2,] 0.09890110   0    0.11538462   0.1875000
[3,] 0.07017544   0    0.09375000   0.1190476
[4,] 0.08196721   0    0.08823529   0.1555556
[5,] 0.06617647   0    0.09302326   0.1063830
[6,] 0.06428571   0    0.06521739   0.1276596

> plot(iris.rf)
```



black is overall OOB error rate

red: Setosa

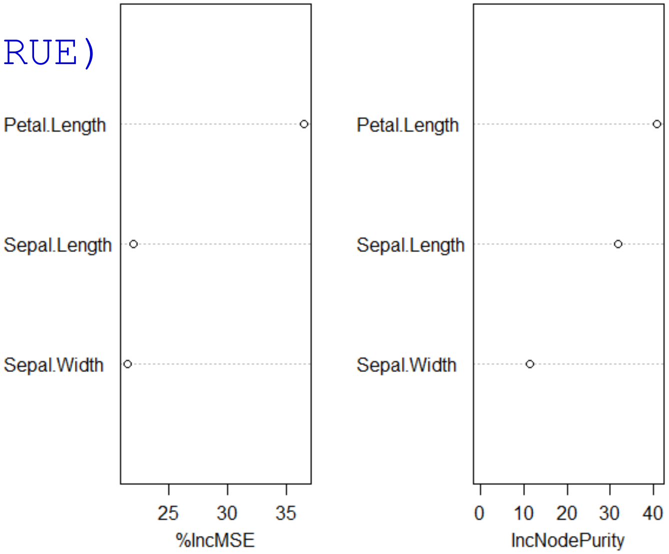
green: Versicolor

blue: Virginica

for details: http://statweb.stanford.edu/~jtaylo/courses/stats202/ensemble.html

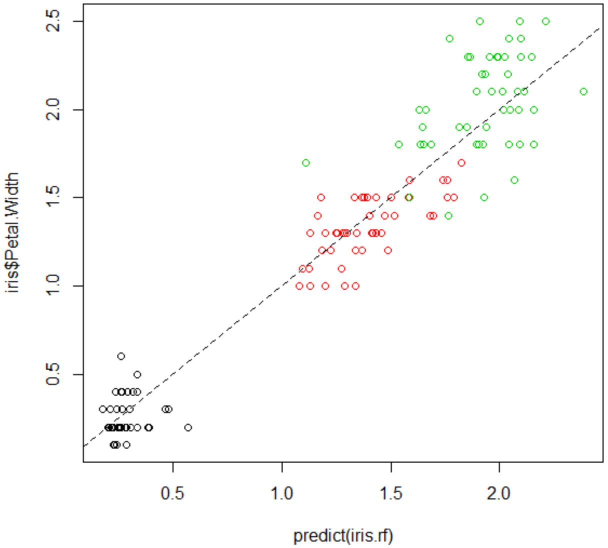
Random Forest for predicting Petal.Width via Regression

```
> iris.rf <- randomForest(Petal.Width ~ .,
                  data=iris[,1:4], importance=TRUE)
> print(iris.rf)
Call:
 randomForest(formula = Petal.Width ~ .,
data = iris[, 1:4], importance = TRUE)
     Type of random forest: regression
          Number of trees: 500
No. of variables tried at each split: 1
   Mean of squared residuals: 0.03995001
          % Var explained: 93.08
```



RF-regression allows quite well to predict the width of petal-leafs from the other leaf-measures of the same flower.

The model will probably improve, if we include the species as predictor variable.



Known Pros of Random Forest

- It is perfect benchmark learning algorithm providing oob error for free
- It is very easy to handle – no distribution assumptions -> no transformation
- Random Forest does not tend to overfit, cv incorporated
  (but the observations must be independent!)
- It can handle thousands of input variables without variable deletion
  - can handle lots of noise variable even with quite few relevant variables (6 out of 100 already works fine)
- It provides variable importance important measures and dependency plots
- It can handle strong and local interactions very well
- It is robust against outliers in the predictive variables
- It can handle imbalanced data by stratified bootstrap sampling
- It can be used unsupervised providing an alternative dissimilarity measure
- It can be used for imputation of missing data [last 2 points will covered later]

Let's praise the Random Forest

RF is not in all situations the best method - however it often just works!

The Random Forest™ is my shepherd; I shall not want.
He makes me watch the mean squared error decrease rapidly.
He leads me beside classification problems.
He restores my soul.
He leads me in paths of the power of ensembles
for his name's sake.

Even though I walk through the valley of the curse of dimensionality,
I will fear no overfitting,
for you are with me;
your bootstrap and your randomness,
they comfort me.

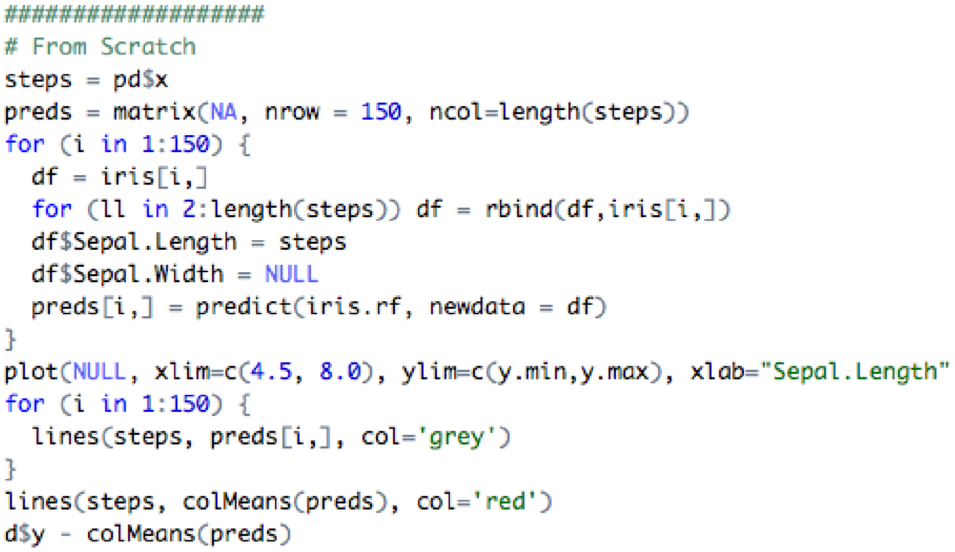
You prepare a prediction before me
in the presence of complex interactions;
you anoint me data scientist;
my wallet overflows.
Surely goodness of fit and money shall follow me
all the days of my life,
and I shall use Random Forests™
forever.

Christoph Molnarhttp://www.statsblogs.com/2014/02/06/random-forest-almighty/

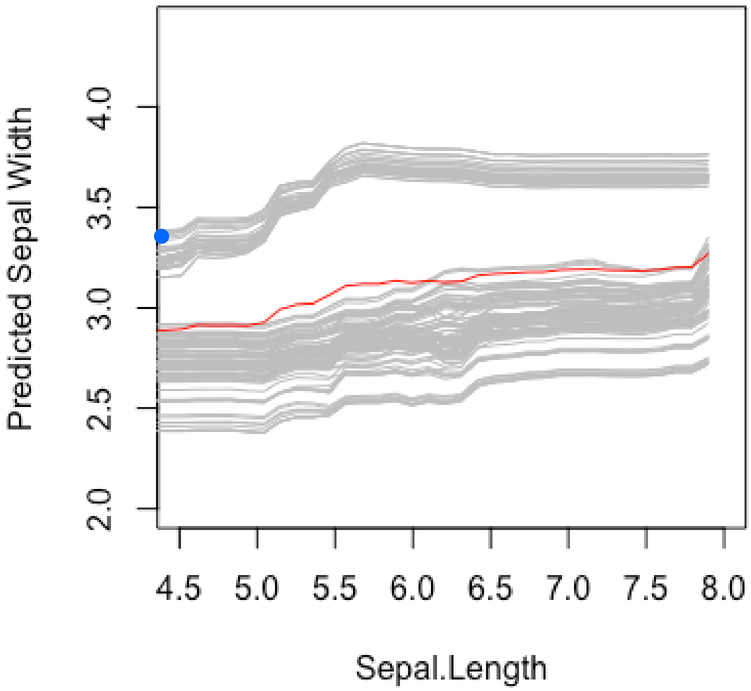
Appendix

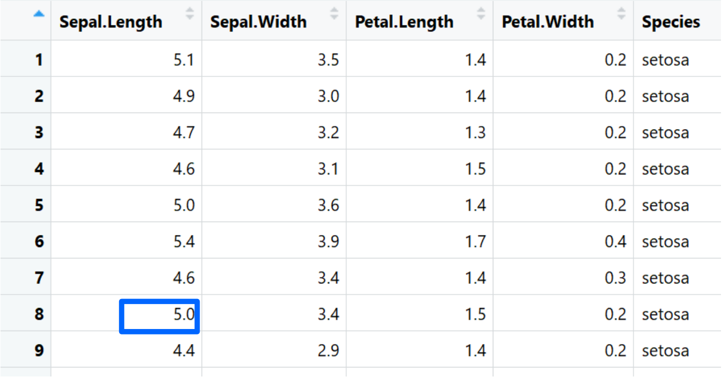
Dependency plots:
bring light into the black box

Marginal dependency plots



```
####################
# From Scratch
steps = pd$x
preds = matrix(NA, nrow = 150, ncol=length(steps))
for (i in 1:150) {
  df = iris[i,]
  for (ll in 2:length(steps)) df = rbind(df,iris[i,])
  df$Sepal.Length = steps
  df$Sepal.Width = NULL
  preds[i,] = predict(iris.rf, newdata = df)
}
plot(NULL, xlim=c(4.5, 8.0), ylim=c(y.min,y.max), xlab="Sepal.Length"
for (i in 1:150) {
  lines(steps, preds[i,], col='grey')
}
lines(steps, colMeans(preds), col='red')
d$y - colMeans(preds)
```





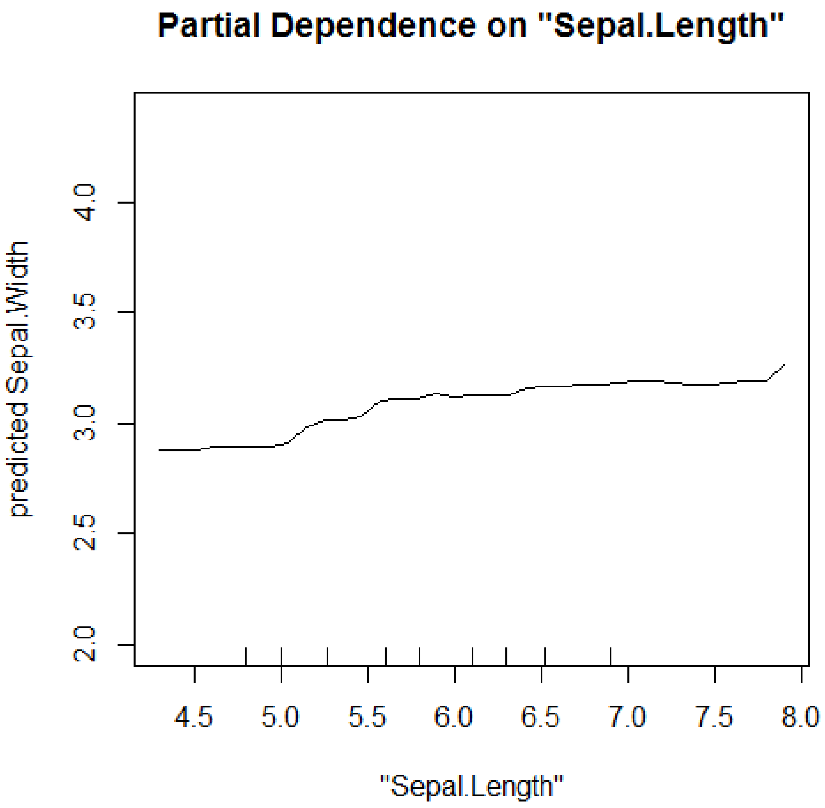
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa

One line is prediction on a training example with varying `Sepal.Length` rest constant. Red average over all training data.

Remark: Marginal dependency plots not limited to RF

RandomForest provides marginal dependency plots

```
library(randomForest)

iris.rf = randomForest(Sepal.Width ~ .,
                       data=iris )
print(iris.rf)

( y.min = min(iris$Sepal.Width) )
( y.max = max(iris$Sepal.Width) )

partialPlot(iris.rf,
            pred.data=iris,
            x.var="Sepal.Length",
            ylab="predicted Sepal.Width",
            ylim=c(y.min,y.max))
```

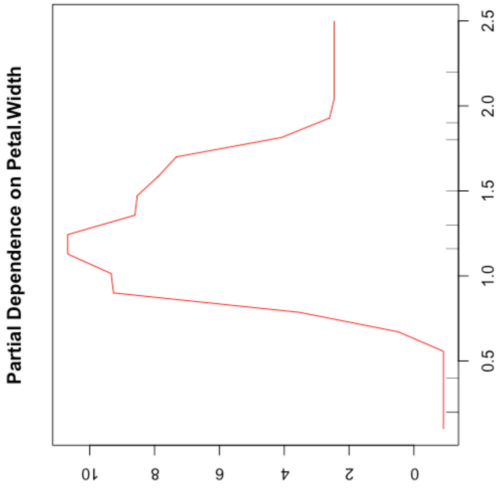


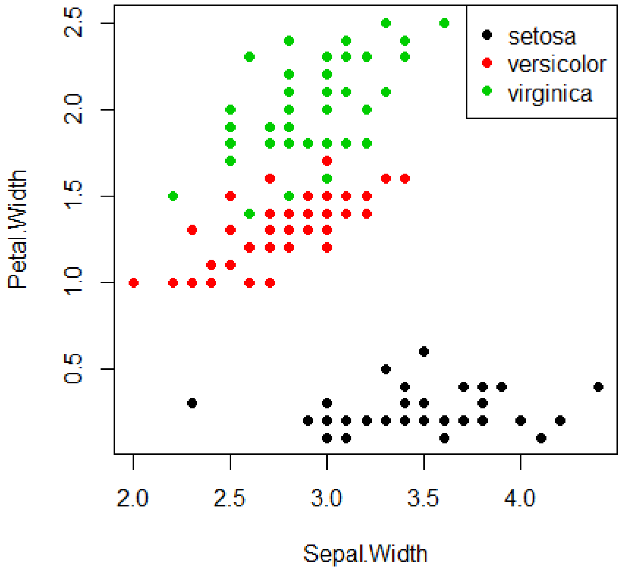
Interpretation of curve: "Mean predicted Sepal.Width" vs Sepal.Length when averaging over all observed constellation of additional co-variates.

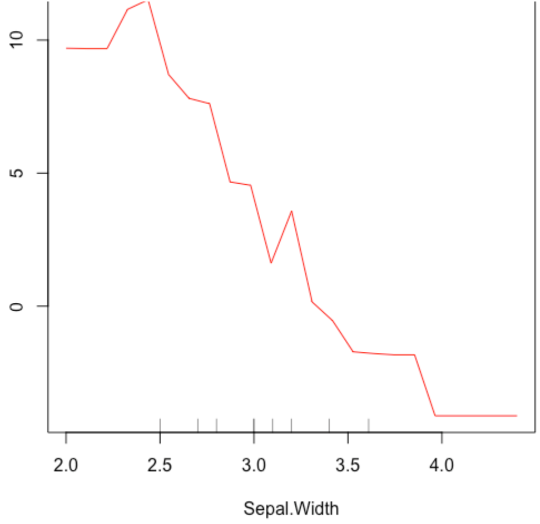
For details see next slides.

Partial dependence plots: how does a variable impact the classification?

```
partialPlot(iris.rf.c, iris, x.var=Sepal.Width,which.class="versicolor")
partialPlot(iris.rf.c, iris, x.var=Petal.Width,which.class="versicolor")
```

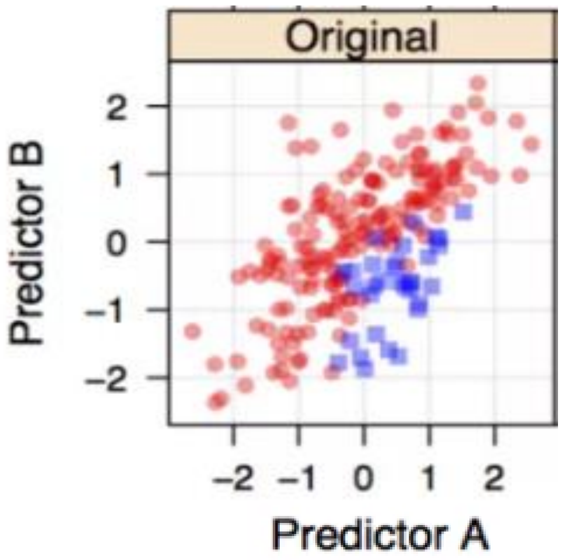






RF can handle imbalanced data

How to model imbalanced data with a RF model?



In an imbalance data set we have much more observations with class label 1 than 2. To get a high accuracy it is sufficient to classify each observation into class 1.
To improve the detection of class 2 we need to use some special approaches.

RF offers 2 approaches

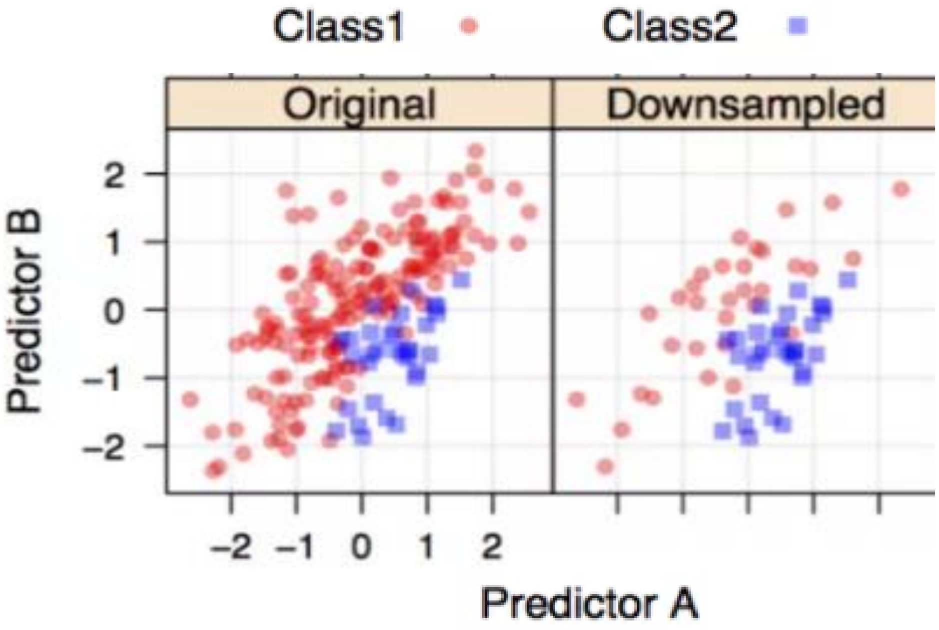
1) cost sensitive learning: weighted random forest (WRF)

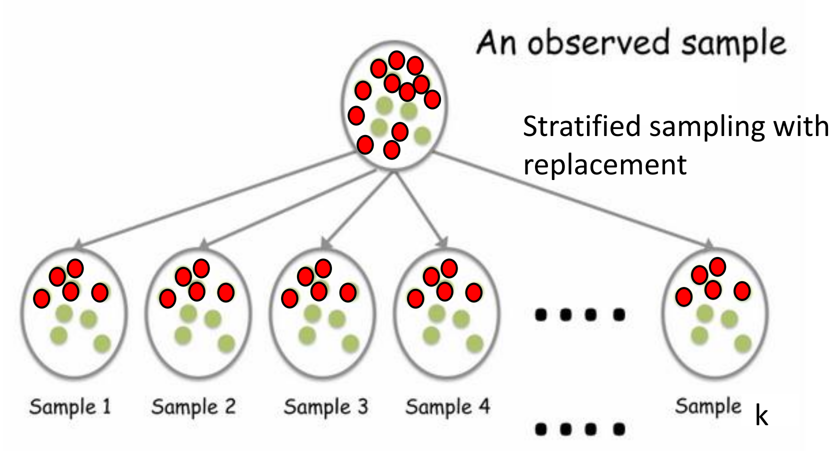
2) stratified sampling technique: balanced random forest (BRF)

Performance metrics such as weighted accuracy, sensitivity and specificity, and especially the prediction accuracy of the minority class are improved by both methods.
Chen et al. showed that BRF is computationally more efficient than WRF for imbalanced data and WRF is more vulnerable to noise compared to BRF.

Chen C, Liaw A, Breiman L: Using random forest to learn imbalanced data. University of California, Berkeley, 2004
Khalilia, M.., Chakraborty S., Popescu M., Predicting disease risks from highly imbalanced data using random forest, BMC Medical Informatics and Decision Making 2011

Stratified sampling can help in imbalanced situations





```
rf.strat=randomForest( class ~., data=my.dat  ,
                                           strata=my.dat$class,
                                           sampsize=c(35,35) )
```

If the large class has 200 observations and the small class only 35 observations, we can sample (with replacement) for each tree newly 35 observations from both classes so that each baseline classifier learns the model from a new balanced data set. In this manner still all observations from the large class can be sampled for some trees..