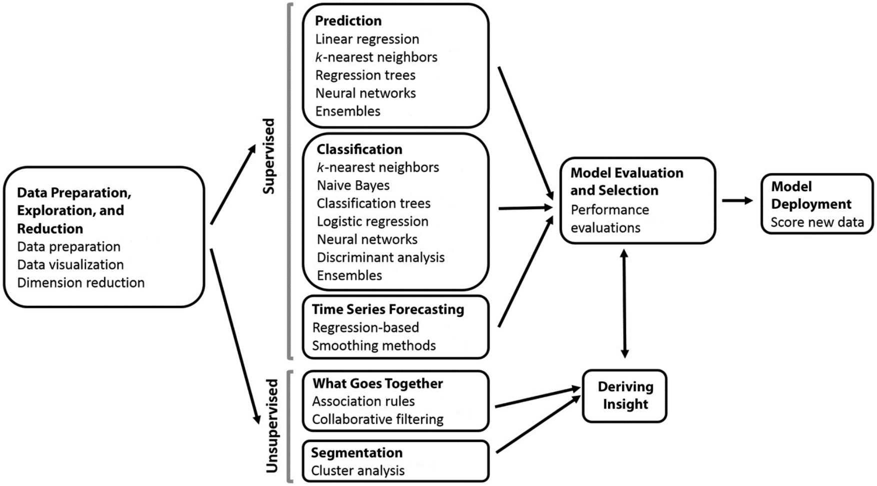
Outlook of the unit

3 **Prediction and classification approaches**

- Classifiers
    - Logistic regression
    - Discriminant analysis
- Classifier performance
- Tree-based methods: Decision trees
    - Classification trees
    - Regression trees

Classifiers

Prediction and classification approaches - A short overview



Examples of classification

- Banking: determining whether a transaction is fraudulent or an applicant is a good or bad credit risk.
- E-commerce: forecasting whether a particular order will be paid for or which customer will buy a specific product.
- National security: identifying whether a certain behavior indicates a possible thread.
- Tourism: determining the rating a hotel should be awarded.
- Medicine: diagnosing whether a particular disease is present.

Assumptions

- Extracting knowledge from massive data sets assumes that they contain non-random, useful information.
- Classification algorithms, especially predictions, do not have further assumptions about how the data was generated.
- Most classical statistical methods are not valid if their **assumptions** are not fulfilled.

This does not mean that big data is always good data.

Definition: Basics

n number of observations in a sample
p number of predictor variables (features)

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$ matrix of predictor variables

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$ vector of response (dependent / target) variable

$\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$, where x_i is a vector of length p, represents the observed data from which a statistical model is built.

Definition: Classification

Classification employs a set of p inputs to explain a value of an outcome variable y.

- x is an object in an object space $\mathbb{X}$
- $y = y(x)$ is the response in a response space $\mathbb{Y}$
- Algorithm $a(x)$ is a function from $\mathbb{X}$ to $\mathbb{Y}$

In classification, $\mathbb{Y}$ is a set of discrete categories, $\{C_1, C_2, ..., C_K\}$, exclusively and exhaustively defining possible states of an element.
$\mathbb{Y} = \{C_1, C_2\}$ binary classification

$\mathbb{Y} = \{C_1, C_2, ..., C_K\}$ multiclass classification

Place of classification among other methods

- Classification is a method of supervised learning for a **categorical** output
- If y is continuous: it is a regression problem (also supervised learning)
- If categories C_k are not defined: unsupervised learning (e.g. clustering)

Process of generating and using a model: remember

Step 1: training a model Historical data is used to generate a statistical model that predicts a response variable from a set of predictor variables

Step 2: deploying a model Predictions for the response variables for a new data set with the same predictor variables are made

Training model: Major decisions

Given a real-world problem and data you need to decide on:

1. which classifier to use
2. which features to take from the data
3. which evaluation metric to optimize

Large 'Zoo' of Classification Methods

Classifiers differ by:

- ways to construct decision boundary (linear vs. non-linear).
- types of data they can use as inputs.
- probabilities to be a member of a particular class or only class membership is predicted.
- number of times a model is built (individual vs. ensemble).

Constraints

Decision of which classifier to use is subject to constrains:

- analyst's understanding of algorithms
- interpretability
- runtime
- scalability

Logistic regression

What is meant by binary logistic regression analysis?

In the previous lectures, you learned the basic concepts of the **linear regression model and their estimation**.

Why do we need additional regression methods?

- Previously worked with the predictions continuous target variables.
- What can we use to explain categorical variables?

$\rightarrow$ **target**: Model a dependent **binary variable** to make an appropriate prediction of probability (using covariates).

Examples of binary variables

Let's imagine independent observations $y_1, \ldots, y_n$ of a variable of interest Y, which can take **only two values** (e.g. 0 and 1), e.g.:

- Workpiece is defective(1)/not defective(0)
- Customer is considered creditworthy: yes(1)/no(0)
- Therapy is successful: yes(1)/no(0).

$\rightarrow$ Is linear regression **optimal** for modeling binary dependent variables?

Illustration: UAE funding award to high schools

- This award is given for the best high school graduation performance in AL Ain.
- What is the probability of receiving this price, depending on the hours of literature the high school graduates have read in the last year?
- The variable *Award* is coded 0 if the person did not win the prize and 1 if the person won the prize.

Illustration: UAE funding award

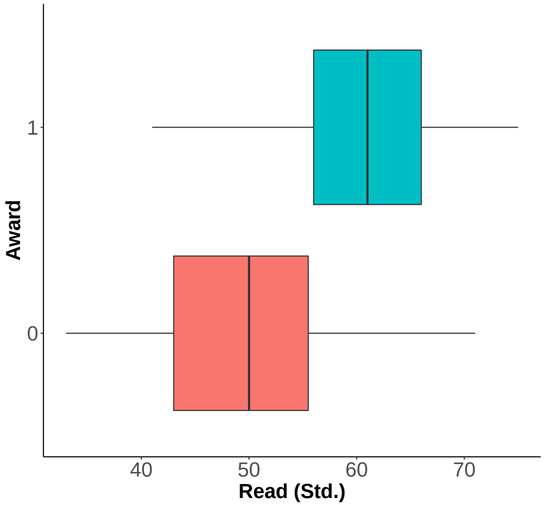


Illustration: Prediction of probabilities?

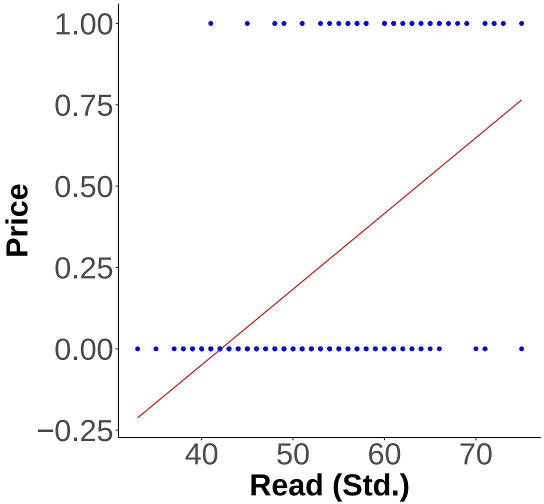
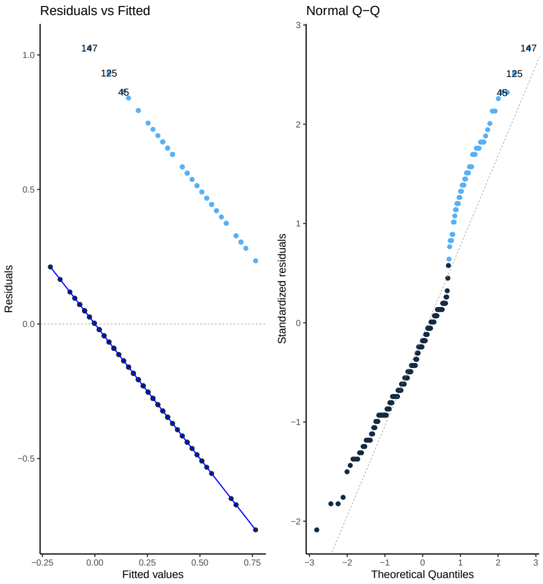


Illustration: Model assumptions



Problems with linear regression

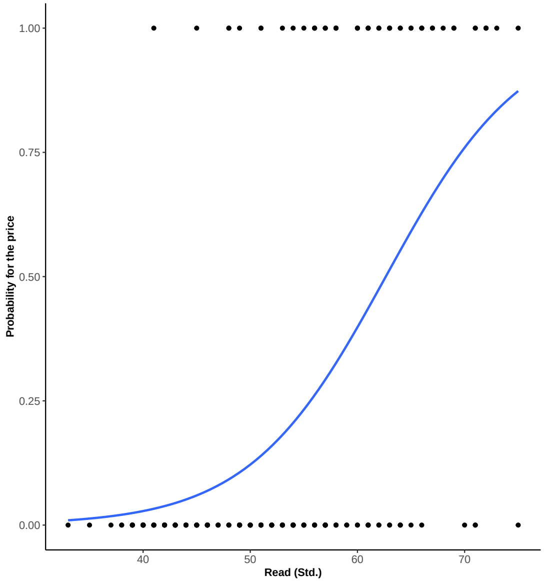
- Insufficient spread: residuals do not spread uniformly; clear systematic spread
- Violation of normal distribution assumption: residuals must be normally distributed; this is clearly not the case here
- Variance homogeneity not fulfilled
- Implausible values for modeling a probability in the estimation results: Values < 0 and > 1

$\rightarrow$ **Binary logistic regression analysis**

Modeling the influence of explanatory variables

- A distribution model for binary target variables Y_i is the binomial distribution, $Y_i \sim B(1, \pi_i)$ with $\mathrm{Var}(Y_i) = \pi_i(1 - \pi_i)$ and $\pi_i = P(Y_i = 1) = E(Y_i)$.
- In what follows, we will call one of the two possible values of Y a success and the other as a failure.
- We call π_i as the probability of success.
- **Target**: Given p explanatory variables: $x_1, \dots, x_p$, we model the influence of the explanatory variables on π_i.
  - How does the probability of a customer repaying their loan repaid as a function of the customer's age?

Illustration: Prediction of probabilities!



Logistic regression definition

This model is used to fit a relationship between a categorical outcome variable **y** (for instance: binary 0-1) and a set of predictors (covariates) **X**:

Response/ Mean function:

$$\pi_i = P(Y_i = 1 \mid \boldsymbol{x}_i'\boldsymbol{\beta}) = F(\eta_i) = \frac{\exp(\eta_i)}{1 + \exp(\eta_i)}$$

Link function:

$$g(\pi_i) = \log\left(\frac{\pi_i}{1 - \pi_i}\right) = \eta_i = \boldsymbol{x}_i'\boldsymbol{\beta},$$

where $\eta_i = \boldsymbol{x}_i'\boldsymbol{\beta}$ denotes the linear predictor.

Odds ratio

$\rightarrow$ The probability π_i is transformed to $odds_i$:

Closely related to the probability of an event occurring is the notion of **chance (odds-ratio)**.

- It denotes the ratio between the probability of success and the failure probability.

$$Odds_i := \frac{P(Y_i = 1)}{P(Y_i = 0)} = \frac{P(Y_i = 1)}{1 - P(Y_i = 1)} = \frac{\pi_i}{1 - \pi_i}$$

- Example: If the odds in favor of success (winning) of a racer are 20:1, then we *expect* him to win 20 and lose one out of 21 20 games and lose one.

Global interpretation of coefficients

- Since the marginal effects are not constant, the coefficients $\hat{\beta}$ can be poorly interpreted directly.
- Instead, one usually interprets $\exp(\hat{\beta})$.

coefficient $\hat{\beta}$	chance $\frac{P(Y_i=1)}{P(Y_i=0)}$	probability $P(Y_i = 1)$
$\hat{\beta} > 0$	increases by $\exp(\hat{\beta})$	increases
$\hat{\beta} < 0$	decreases by $\exp(\hat{\beta})$	decreases
$\hat{\beta} = 0$	remains the same	remains the same

Illustration: interpretation of results

- Model to be estimated: $\frac{\exp(\beta_0+\beta_1 * Read)}{1+\exp(\beta_0+\beta_1 * Read)}$
- Estimated parameters: $\hat{\beta}_0 = -9.8$ and $\hat{\beta}_1 = 0.16$

$\rightarrow$ $\exp(0.16) = 1.169$: The chance that a student will win the prize given an increase of of the lessentime by one hour is valid, averaged by 1.169 times higher.

$\rightarrow$ $\frac{\exp(-9.8+0.16*70)}{1+\exp(-9.8+0.16*70)} = 0.8$: The probability with which a a student, who reads 70 hours in the final year of school will win the prize.

$\rightarrow$ $Odds_{1(70)} := \frac{\pi_{1(70)}}{1-\pi_{1(70)}} = 4$: The student (Observation i=1) who reads 70 hours has a 4 times higher chances of winning the prize than not winning it.

- The constant parameter cannot be interpreted meaningfully.

Linear discriminant analysis

Discriminant analysis

- In case the response variable contains **more than two categories**, discriminant analysis is more suitable than the logistic regression.
- It predicts the probability of belonging to a given category based on explanatory variables.

Why do we need another method, when we have logistic regression?

- When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly **unstable**. Linear discriminant analysis does not suffer from this problem.
- linear discriminant analysis is popular when we have more than two response classes.
- If n **is small** and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.

Linear Discriminant analysis: ideas behind

- Logistic regression involves directly modeling $Pr(Y = k|X = x)$ using the logistic function, for the case of two response classes.
- Commonly, we model the conditional distribution of the response Y , given the predictor(s) X.
- We now consider an **alternative** and less direct approach to estimating these probabilities.
- In this alternative approach, we model the **distribution** of the predictors X separately in each of the response classes (i.e. given Y), and then use **Bayes' theorem** to flip these around into estimates for $Pr(Y = k|X = x)$.
- When these distributions are assumed to be normal, it turns out that the model is very similar in form to logistic regression.

Linear discriminant analysis: ideas behind

- **L**inear **D**iscriminant **A**nalysis (**LDA**) is an alternative to the logistic regression.
- It helps to find the **linear combination** of predictors that provide the best possible separation between the groups.
- It this data to divide the space of covariates into regions. The regions are labeled by categories and have **linear boundaries** (L in LDA).
- Those boundaries are a consequence of assuming that the covariates for each category have the same **multivariate normal distribution**.

Bayes classifier

- It assigns each observation to the most likely class, given its predictor values.
- In a two-class problem where there are only two possible response values, say class 1 or class 2, the Bayes classifier corresponds to predicting class one if $Pr(Y = 1|X = x0) > 0.5$, and class two otherwise.
- Bayes theorem for classification:

$$Pr(Y = k|X = x) = \frac{Pr(Y = k|X = x)Pr(Y = k)}{Pr(Y = k)}$$

$$= \frac{\pi_k f_k(x)}{\sum_{l=1}^{k}(\pi_l f_l(x))}$$

Example 1: Iris data

The following data were recorded for three species of irises. The objective is to develop a rule for classifying *flowers* from the species *versicolor* and *verginica* based on the four variables SL, SW, PL, PW.

```
# Read data
data("iris")
str(iris)
# Check the correlation and classification between
    variables
pairs.panels(iris[1:4],
gap = 0,
bg = c("red", "green", "blue")[iris$Species],pch =
    21)
# Create partition
set.seed(123)
ind = sample(2, nrow(iris), replace = TRUE, prob = c
    (0.6, 0.4))
training = iris[ind==1,]
testing = iris[ind==2,]
```

Example 1: Iris data

```
# Linear discriminant analysis
iris_lda = lda(Species~., training)
iris_lda
attributes(iris_lda)
# Prediction: Fitted values
p = predict(iris_lda, training)
# Stacked histogram for discriminant function
ldahist(data = p$x[,1], g = training$Species) # it
    discriminates almost perfectly!
ldahist(data = p$x[,2], g = training$Species) # The
    second function doesn't discriminate at all
# Biplot for LD1 and LD2:
ggord(iris_lda, training$Species, ylim = c(-10, 10))
# Partition plot: It provides the classification of
    each and every combination in the training
    dataset.
partimat(Species~., data = training, method = "lda")
partimat(Species~., data = training, method = "qda")
```

Example 1: Iris data

```
# Confusion matrix and accuracy: training data
    p1  = predict(iris_lda, training)$class
    tab = table(Predicted = p1, Actual = training$
        Species)
    tab
    sum(diag(tab))/sum(tab)
# Confusion matrix and accuracy: testing data
    p2   =  predict(linear, testing)$class
    tab1 =  table(Predicted = p2, Actual =
          testing$Species)
    tab1
  sum(diag(tab1))/sum(tab1)
```

Example 1: Iris data

```
# Everything is not linear-quadratic discriminant
    analysis
iris_qda=qda(Species~.,data=iris)
iris_qda
#Check the accuracy of our analysis of qda
Predictions_qda=predict(iris_qda,iris)
table(Predictions_qda$class, iris$Species)
# Discriminant analysis density plots
p <- predict(iris_lda, training)
p.df <- data.frame(LD1 = p$x, class = p$class)
print(p.df)
LD1 = p$x
ggplot(p.df) + geom_density(aes(LD1[,"LD1"], fill =
    class), alpha = 0.2)
ggplot(p.df) + geom_density(aes(LD1[,"LD2"], fill =
    class), alpha = 0.2)
```

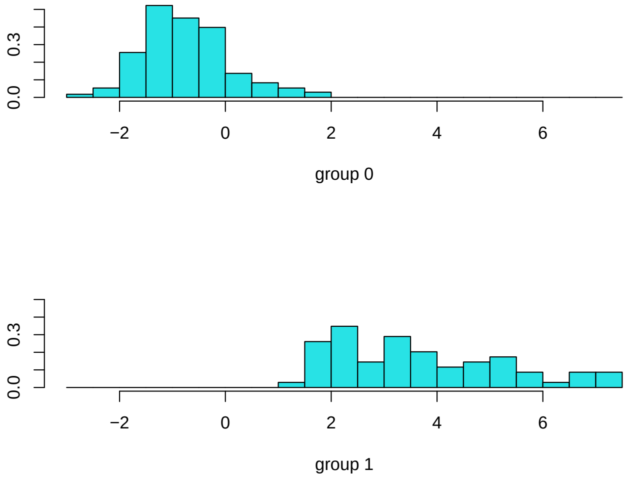
Example 2: Boston housing data

With the lda()-command we can perform a lda

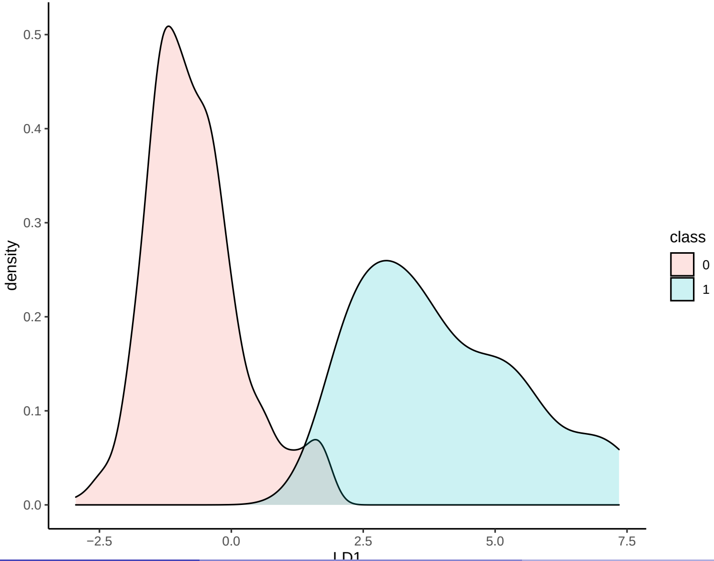
```
# Discriminant analysis
linear <- lda(CAT..MEDV~., train)
linear
p <- predict(linear, train)
ldahist(data = p$x[,1], g = train$CAT..MEDV)

# Discriminant analysis density plots
sm.lda <- lda(CAT..MEDV~., train)
print(sm.lda)
p<- predict(sm.lda, train)
p.df <- data.frame(LD1 = p$x, class = p$class)
print(p.df)
lggplot(p.df) + geom_density(aes(LD1, fill = class),
    alpha = 0.2)
```

Stacked histogram for discriminant function values: Boston housing data



Density plots from discriminant analysis: Boston housing data



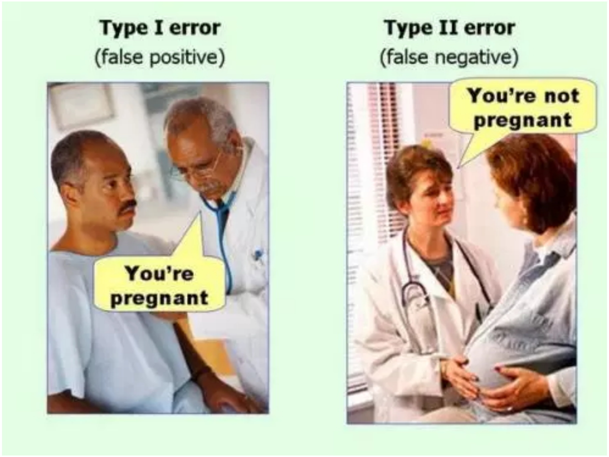
Classifier performance

Judging classifier performance

Some first comments:

- A natural criterion for judging the performance of a classifier is the probability of making a **misclassification** error.
- Misclassification means that the record belongs to one class but the model classifies it as a member of a different class.
- A classifier that makes no errors would be perfect - unrealistic.
- Is there a minimal probability of misclassification that we should require of a classifier?

Misclassification error



Benchmark: The naive rule

- A very simple rule for classifying a record into one of m classes, ignoring all predictor information is to classify the record as a member of the **majority class**.
- The **naive rule** is used mainly as a baseline or benchmark for evaluating the performance of more complicated classifiers.
- Similar to using to $\overline{y}$ as the naive benchmark in the numerical outcome case, the naive rule for classification relies solely on the y information and excludes any additional predictor information.
- A good classifier that uses external predictor information should outperform the naive rule.

A more complex approach: Logistic regression

A popular model for making classifications is the logistic regression model. To predict which class a new record $\mathbf{x}_i$ will belong to we compute its probability to belong to each class by

$$\hat{\pi}_i = \frac{\exp\left(\hat{\eta}_i\right)}{1 + \exp\left(\hat{\eta}_i\right)} = \frac{\exp\left(\boldsymbol{x}_i'\hat{\boldsymbol{\beta}}\right)}{1 + \exp\left(\boldsymbol{x}_i'\hat{\boldsymbol{\beta}}\right)}$$

and then possibly rank a set of new records from highest to lowest probability in order to act on those with the highest probability.

Accuracy measures - the classification matrix

- Classification matrix summarizes the correct and incorrect classifications that a classifier produced.
- Rows and columns of the confusion matrix correspond to the predicted and true (actual) classes.
- Example:

		Actual class	
		0	1
Predicted class	0	2600	100
	1	100	200

- Diagonal cells give the number of correct classifications.
- Off-diagonal cells give counts of misclassification.
- Classification matrix gives estimates of the true classification and misclassification rates.

Accuracy measures - the classification matrix

- We summarize the classification for the validation data as follows.
- **Classification matrix**:

		Actual class	
		C_1	C_2
Predicted	C_1	$n_{1,1}$	$n_{2,1}$
class	C_2	$n_{1,2}$	$n_{2,2}$

- Estimated **misclassification rate**:

$$err = \frac{n_{1,2} + n_{2,1}}{n_v},$$

where n_v is the total number of units in the validation data.

- **Estimated accuracy**:

$$accuarcy = 1 - err = \frac{n_{1,1} + n_{1,1}}{n_v}.$$

Propensities and cut-off for classification

- First step in most classification algorithms is to estimate the probability π (propensity) that a unit belongs to each of the classes.
- If overall classification accuracy is of interest, the unit can be assigned to the class with the highest probability.
- In many records, a single class is of special interest, so we will focus on that particular class.
- It may make sense in such cases to consolidate classes so that you end up with two: the class of interest and all other classes.
- The default **cutoff** value in two-class classifiers is 0.5.
- It is possible, however, to use a cutoff that is either higher or lower than 0.5. Two examples:
  - unequal misclassification costs
  - unequal importance of classes.

Evaluation Metrics (1)

General form of a 2×2 confusion matrix

		Actual value		
		C_1	C_2	Row total
Predicted value	C_1'	True Positive	False Positive	P'
	C_2'	False Negative	True Negative	N'
Column total		P	N	

Note: C_1 is assumed to correspond to a positive class

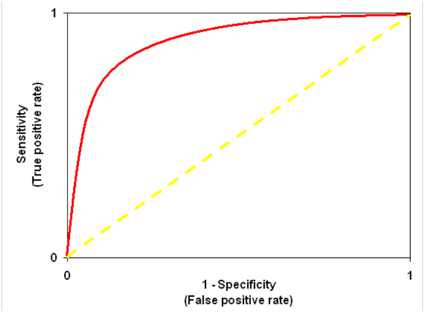
Evaluation Metrics (2)

A variety of predictive measures can be derived from a confusion matrix:

Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Error rate	$1 - Accuracy$
Sensitivity (TP rate)	$\frac{TP}{TP+FN}$
Specificity (TN rate)	$\frac{TN}{TN+FP}$

ROC Curve

The **R**eceiver **O**perating **C**haracteristic (ROC) curve is a way to visualize interrelationship between sensitivity and specificity



AUC (area under curve) indicates model goodness, 1 being a perfect model and below 0.5 (yellow line) a useless model (worse then a coin flip).

Classification: Boston housing data

We investigate classification accuracy from a model for the median value of a home above $ 30.000: Training and validation.

```
> # Run logit model: Median value 0/1 ~ LowerPopul. +
   Number of rooms + Teacher/Pupil ratio
> fit<-glm(CAT..MEDV~LSTAT+RM+PTRATIO,data=train,family =
    "binomial")
> summary(fit)
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -12.5173     5.3884  -2.323 0.020179 *
LSTAT        -0.4006     0.1186  -3.378 0.000730 ***
RM            3.5765     0.6994   5.113 3.16e-07 ***
PTRATIO      -0.5783     0.1531  -3.778 0.000158 ***
(Dispersion parameter for binomial family taken to be 1)

Null deviance: 347.03  on 405  degrees of freedom
Residual deviance: 101.51  on 402  degrees of freedom
AIC: 109.51
```

Classification: Boston housing data

The `confusionMatrix()`-command returns the classification matrix.

```
> # Create predictions - training vs. validation set
pred_t<-predict(fit,newdata = train,type = "response")
pred_v<-predict(fit,newdata = vali,type = "response")
> # Evaluate performance - training vs. validation set
>  # Training - logit model
confusionMatrix(ifelse(pred_t>0.5,1,0), train$CAT..MEDV
   )
Reference
Prediction   0   1
0 338   8
1   6  54          Accuracy : 0.9655
>   # Validation - logit model
confusionMatrix(ifelse(pred_v>0.5,1,0), vali$CAT..MEDV)
Reference
Prediction  0  1
0 76  6
1  2 16          Accuracy : 0.92
```

Classification: Boston housing data

We focus on the the validation set under the model and the naive methods.

```
> # Naive benchmark: the average
> y_fit_naive<-median(train$CAT..MEDV)

> # Create predictions
> pred_v_reg<-predict(fit,newdata = vali,type = "response
  ")
> pred_v_naiv<-rep(y_fit_naive,length(vali$MEDV))
```

Classification: Boston housing data

We compare the confusion matrices.

```
> # Evaluate performance - validation set

> # Validation - logit model vs naive benchmark
> confusionMatrix(as.factor(ifelse(pred_v_reg > 0.5, 1,
   0)), as.factor(vali$CAT..MEDV))
          Reference
Prediction  0  1
0 83  3
1  2 12

> confusionMatrix(as.factor(ifelse(pred_v_naiv > 0.5, 1,
   0)), as.factor(vali$CAT..MEDV))
          Reference
Prediction  0  1
0 85 15
1  0  0
```

Classification: Boston housing data

We check the overall classification accuracy.

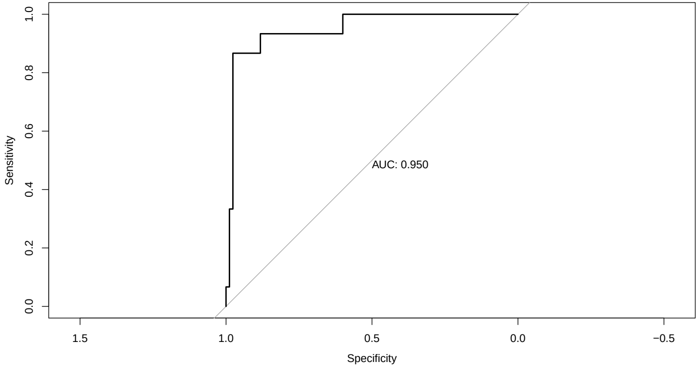
```
> predicted.classes <- as.factor(ifelse(pred_v_reg > 0.5,
    1, 0))
> observed.classes <- as.factor(vali$CAT..MEDV)
> # Estimated accuracy - logit model
> accuracy <- mean(observed.classes == predicted.classes)
0.95
> # Estimated missclassification rate - logit model
> error <- mean(observed.classes != predicted.classes)
0.05

> # Confusion matrix, proportion of cases - logit model
> table(observed.classes, predicted.classes) %>%
prop.table() %>% round(digits = 3)
                predicted.classes
observed.classes    0    1
0 0.83 0.02
1 0.03 0.12
```

Classification: Boston housing data

We check the graphical representation of logits' accuracy.

```
  > # Compute the eceiver operating characteristics curve
      (roc) - logit model
> res.roc <- roc(observed.classes, pred_v_reg)
> plot.roc(res.roc, print.auc = TRUE)
```

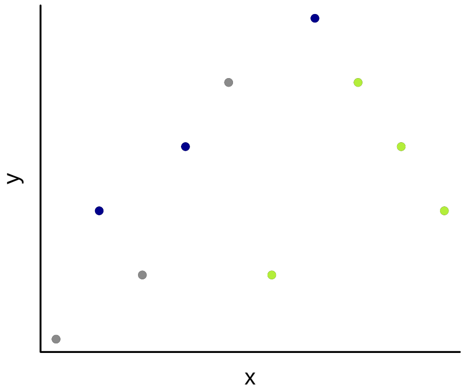


Cross-validation

Cross-validation (CV) as an evaluation method

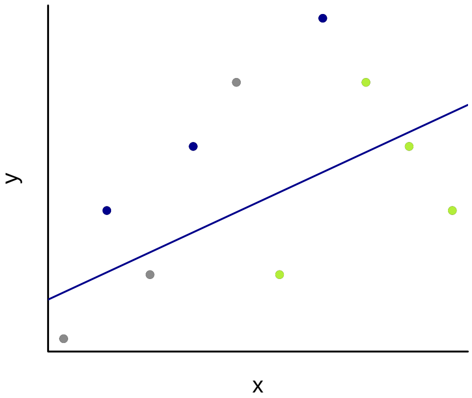
- **Problem**: parameters of a prediction are learned and tested on the same data $\rightarrow$ **Overfitting** - model memorizes training data but fails to predict unseen data
- **General idea**:
  1. split data set into training set and testing set
  2. fit model using training set only
  3. use model to predict values of the testing set
  4. use errors between prediction and actual value for evaluation
- **Improvement (k-fold CV)**: divide data set into k subsets and repeat the steps k times $\rightarrow$ $k-1$ subsets are used as training sets and one subset as testing set, error is the average error across all k trials

k-fold Cross-validation



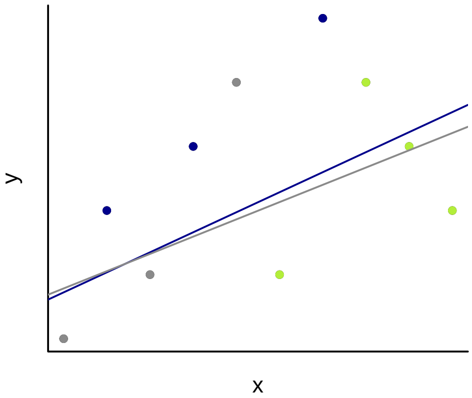
1. Divide data set randomly into k subsets, here $k = 3$

k-fold Cross-validation



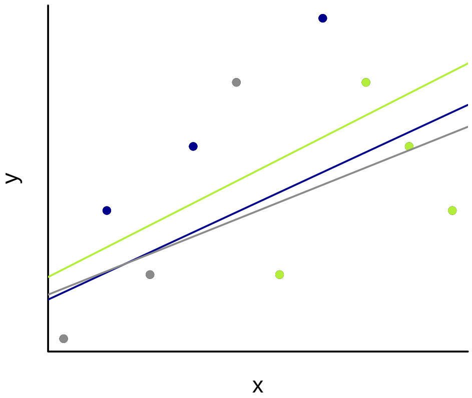
1. Divide data set randomly into k subsets, here $k = 3$
2. Train using the gray and green points and test for the blue

k-fold Cross-validation



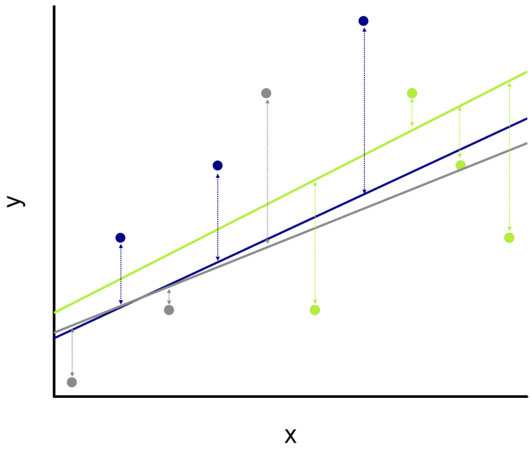
1. Divide data set randomly into k subsets, here $k = 3$
2. Train using the gray and green points and test for the blue
3. Train using the blue and green points and test for the grey

k-fold Cross-validation



1. Divide data set randomly into k subsets, here $k = 3$
2. Train using the gray and green points and test for the blue
3. Train using the blue and green points and test for the grey
4. Train using the blue and gray points and test for the green

k-fold Cross-validation



1. Divide data set randomly into k subsets, here $k = 3$
2. Train using the gray and green points and test for the blue
3. Train using the blue and green points and test for the grey
4. Train using the blue and gray points and test for the green
5. Use errors between prediction and actual value for evaluation

CV for misclassified observations

- So far, we used CV in the regression setting where the outcome is quantitative, and so have used *MSE* to quantify test error.
- When the outcome is categorical, CV works as described earlier, except that rather than using *MSE* to quantify test error, we instead use the number of misclassified observations.
- LOOCV error rate

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i).$$

- The k-fold CV error rate and validation set error rates are defined analogously.

Cross-validation: Boston housing data

Cross-validation can be automatically computed for any generalized linear model using the `cv.glm()`-command.

```
> # Run linear regression model: Median value ~ Crime
    rate + Number of rooms + Teacher/Pupil ratio
> fit<-glm(MEDV~CRIM+RM+PTRATIO,data=Daten)

> # Leave-one-out cross-validation
> cv_one_err<-cv.glm(Daten,fit)
> cv_one_err$delta
[1] 35.00064 34.99989

> # 5-fold cross-validation
> cv_5_err<-cv.glm(Daten,fit,K=5)
> cv_5_err$delta
[1] 34.98018 34.89865
```

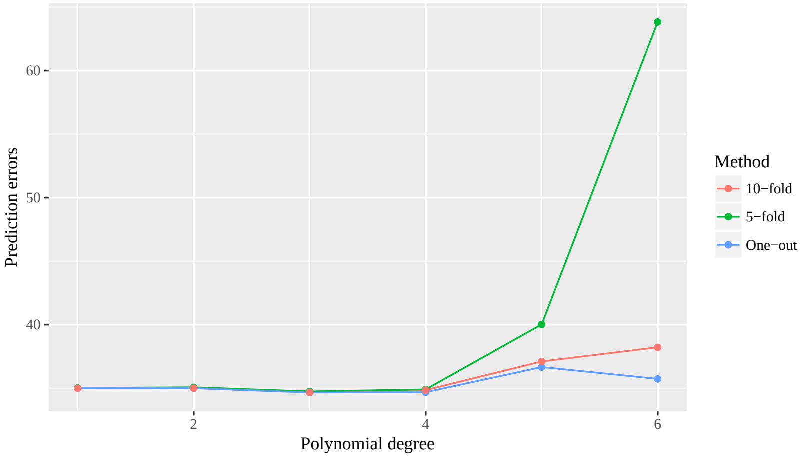
Cross-validation: Boston housing data

We investigate prediction error metrics from a model for the median value of a home: Linear vs. polynomial model for variable *Crime rate*.

```
> cv_error<-NULL
>
> for(i in 1:6){
+    fit_poly<-glm(MEDV~poly(CRIM,degree=i)+RM+
     PTRATIO,data=Daten)
+    cv_error[i]<-cv.glm(Daten,fit_poly,K=10)$delta
     [1]
+ }
> cv_error
[1] 34.708 34.813 35.076 34.423 41.874 51.530
```

The results don't show a clear improvement from using higher-order (> 4) polynomials. However, results may depend on the random split.

Cross-validation: Boston housing data



Each CV approach was run 100 separate times, each with a different random split of the data into K parts. The figure shows the median prediction error over replications.

Tree-based methods: Decision trees

Prediction and classification approaches - A short overview

- Linear regression, logistic regression and multinomial logistic regression are common methods for prediction and classification.
- These methods relate a response variable to a set of explanatory variables - parametric nature.
- Linear relationships are being considered, which means that the effect on the response of a change in the explanatory variable from x to $x + 1$ is the same for any value x.
- With regression/classification trees, the models become truly nonparametric and they allow for very flexible representations.

Tree-based methods: Regression/classification trees

- Here we describe tree-based methods for regression and classification.
- These involve stratifying or segmenting the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.
- Decision trees can be applied to both regression and classification problems.

Pros and Cons: Regression/classification trees

+ Tree-based methods are simple and useful for interpretation.

- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.

+ Hence we also discuss bagging, random forests (and boosting). These methods grow multiple trees which are then combined to yield a single consensus prediction.

+/- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

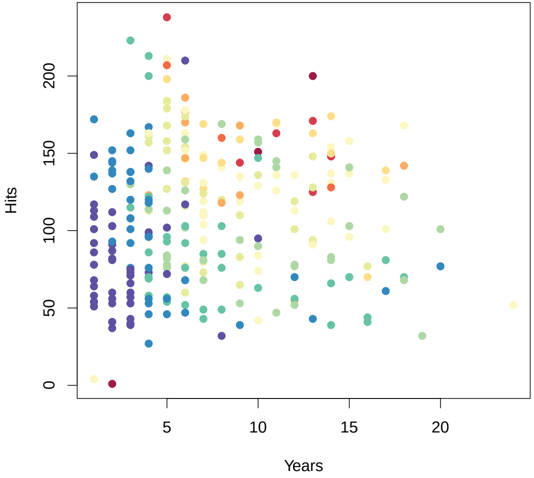
Predicting baseball players' salaries using regression trees

In order to motivate regression trees, we begin with a simple example.
Example:

- We use the *Hitters* data set (R-package: ISLR) to predict a baseball player's salary based on the variable *years* and *hits*.
- Years is the number of years that he has played in the major leagues.
- Hits is the number of hits that he made in the previous year.
- We log-transform salary so that its distribution has a more symmetric shape (salary is measured in thousands of dollars).

Predicting baseball players' salaries using regression trees

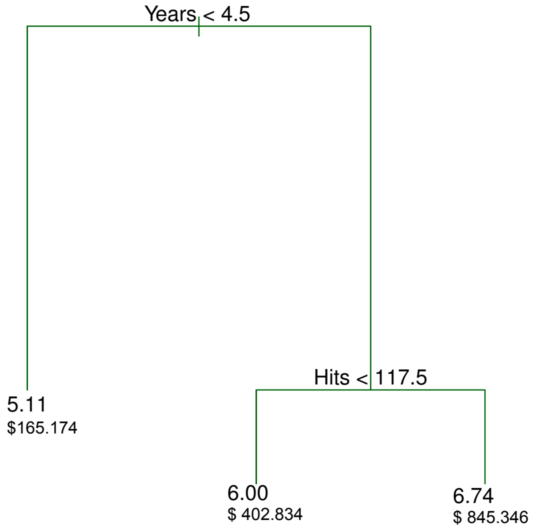
How would you stratify the data?



Salary is color-coded from low (blue, green) to high (yellow, red)

Predicting baseball players' salaries using regression trees

How would you stratify the data?

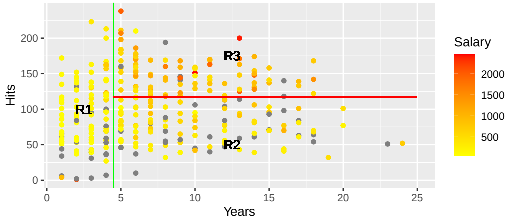


Predicting baseball players' salaries using regression trees

- A regression tree for predicting the log salary based on the number of years (played in the major leagues) and the number of hits (made in the previous year).
- At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.
- The split at the top of the tree results in two large branches. The left-hand branch corresponds to $Years < 4.5$, and the right-hand branch corresponds to $Years \geq 4.5$.
- The tree has two internal nodes (each decision defines a node) and three leaves/ terminal nodes. The number in each leaf is the mean of the response for the observations that fall there.

Predicting baseball players' salaries using regression trees

Overall, the tree stratifies or segments the players into three regions (terminal nodes or leaves) of predictor space:



$R_1 = \{X | Years < 4.5\}$, $R_2 = \{X | Years \geq 4.5, Hits < 117.5\}$,
$R_3 = \{X | Years \geq 4.5, Hits \geq 117.5\}$

Terminology for trees

- In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes*.
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*.
- In the hitters tree, the two internal nodes are indicated by the text $\textit{Years} < 4.5$ and $\textit{Hits} < 117.5$.

Interpretation of results

- *Years* is the most important factor in determining *Salary*, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of *Hits* that he made in the previous year seems to play little role in his *Salary*.
- Among players who have been in the major leagues for five or more years, the number of *Hits* made in the previous year does affect *Salary*, and players who made more *Hits* last year tend to have higher salaries.

- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain.

Regression trees

Building regression trees: Segmentation of the space

The process of building a regression tree consists of mainly two steps:

1. We divide the predictor space - that is, the set of possible values for $X_1, X_2, ..., X_p$ - into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Comment to step 1): The regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.

Building regression trees: Segmentation of the space

How should we divide the predictor space?
The goal is to find boxes $R_1, R_2, ..., R_J$ that minimize residual sum of squares

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j} \right)^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within R_j.

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a top-down approach that is known as recursive binary splitting.
- The best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Building regression trees: Segmentation of the space

- We first select (both) the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $R_1(j,s) = \{X | X_j < s\}$ and $R_2(j,s) = \{X | X_j \geq s\}$ leads to the greatest possible reduction in *RSS*.
- We minimize the equation:

$$\min_{j,s} \left[\sum_{i: x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i: x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2 \right]$$

- Next, we repeat the process, looking for the best X_j and cutpoint s to split the data further to minimize the *RSS* within each of the resulting regions. We split one of the two previously identified regions. We now have three regions.

Building regression trees: Segmentation of the space

- We split one of these three regions further to minimize the *RSS*.
- The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Prediction

We predict the response for a given test (validation) observation using the mean of the training observations in the region to which that test observation belongs.

Tree Pruning

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test (validation) set performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \dots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree - *Cost complexity pruning*.

Tree Pruning - Cost complexity pruning

For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} \left(y_i - \hat{y}_{R_m}\right)^2 + \alpha|T|$$

is as small as possible.

- $|T|$ indicates the number of leaves/ terminal nodes.
- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- When $\alpha = 0$, then the subtree T will simply equal T_0, the formula just measures the training error.
- As α increases, there is a price to pay for having a tree with many terminal nodes.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: Constructing a regression tree

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, ..., K$:
   - Build a regression tree (for a given α) on all $K - 1$th folds of the training data.
   - Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Prediction and classification methods: Example in R

Regression trees in R: Boston housing data

The `rpart()`-command constructs a regression tree and returns the results as an object.

```r
# Loading libraries and the data
library(rpart)
library(partykit)

# Run a regression tree using the rpart command
fit<-rpart(formula, data, method, minsplit, cp, subset)
```

- `minsplit` specifies min. number of units that must exist in a node in order for a split to be attempted.
- `cp` defines the complexity parameter α in the pruning of the tree.
- `subset` specifies the units in the training data set from the available data.

Regression trees in R: Boston housing data

Grow a general regression tree with
Crime rate + River 1/0 + Number of rooms + Teacher/Pupil ratio

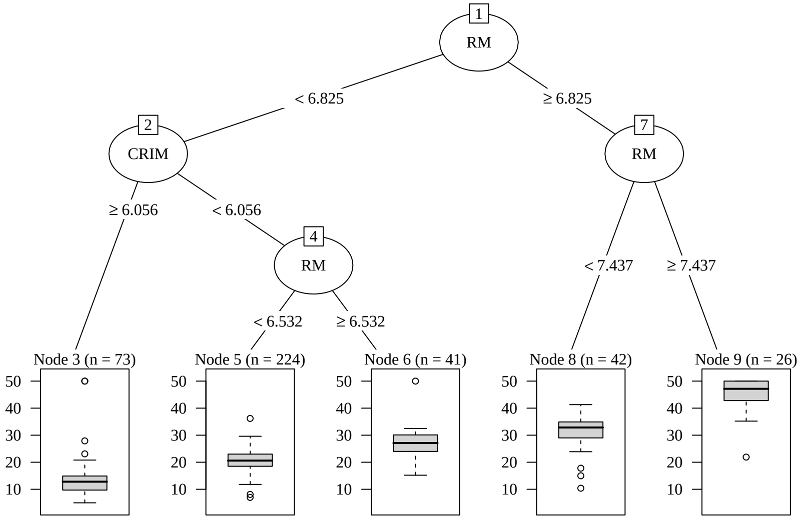
```
# Change complexity parameter alpha to 0 - full tree
> fit<-rpart(MEDV~CRIM+CHAS+RM+PTRATIO, data=Daten,
   minsplit=10, cp=0, subset=inTrain)
# Display the results of cross-validation
> printcp(fit)
Variables actually used in tree construction:
[1] CHAS    CRIM    PTRATIO RM
Root node error: 34771/406 = 85.643
           CP nsplit rel error  xerror     xstd
1  0.45424798      0   1.00000 1.00491 0.092344
2  0.10154523      1   0.54575 0.59452 0.063519
3  0.09118098      2   0.44421 0.50018 0.070210
.  ..........      .   ....... ....... ........
57 0.00014848     74   0.13146 0.43197 0.075392
58 0.00013117     75   0.13131 0.43179 0.075389
59 0.00000000     76   0.13118 0.43207 0.075387
```

Regression trees in R: Boston housing data

The `prune()`-command prunes the regression tree. The tree can be visualized by `plot()`-command.

```
> # Visualize cross-validation results
> plotcp(fit)
>
> # Prune the tree
> fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"]
[1] 0.02605373
> pfit<- prune(fit, cp=0.02605373) # from cptable
>
> # Plot the pruned regression tree
> plot(as.party(pfit))
```

Regression trees in R: Boston housing data



Regression trees in R: Boston housing data

Compare the performance of the pruned tree with the full tree on the validation data.

```
# Predictions for the validation data
> pred_v_tree<-predict(fit,newdata=vali)
> pred_v_ptree<-predict(pfit,newdata=vali)

# Evaluate performance
> accuracy(pred_v_tree,vali$MEDV)
              ME   RMSE   MAE    MPE    MAPE
Test set -0.6629 5.8775 4.001 -9.270 21.3263
> accuracy(pred_v_ptree,vali$MEDV)
              ME   RMSE   MAE    MPE    MAPE
Test set -0.6485 5.6786 3.881 -9.676 20.3017
```

Classification trees

Classification trees - Some first comments

- A **classification tree** is very similar to a regression tree, except that it is classification used to predict a **qualitative response** rather than a quantitative one.
- We predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
- We are often interested not only in the **class prediction** corresponding to a particular terminal node region, but also in the **class proportions** among the training observations that fall into that region.

Building classification trees

- We use recursive binary **splitting** to grow a classification tree.
- In the classification setting, *RSS* cannot be used as a criterion for making the binary splits.
- A natural alternative to *RSS* is the classification error rate:

$$E = 1 - \max_k(\hat{p}_{mk}),$$

  where $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class.

- The **classification error rate** is simply the fraction of the training observations in that region that do not belong to the most common class.

Building classification trees - Two other measures

- The classification error is not sufficiently sensitive for tree-growing.
- The Gini index is defined by:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

  and measures the variation across the K classes (Gini is small when all $\hat{p}_{mk}$ are close to zero or one).
- Gini index is a measure of *node purity* - a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is cross-entropy

$$D = -\sum_{k=1}^{K} \underbrace{\hat{p}_{mk} \log(\hat{p}_{mk})}_{\leq 0}.$$

- The cross-entropy takes a value near zero if the $\hat{p}_{mk}$ are all near zero or near one (small values indicate that nodes are pure).

When to Stop

If tree growing is never stopped, it can result in perfectly pure nodes of one observation each. This is likely to overfit the training data and give poor prediction on the test set.

Strategy to overcome too complex models:

- stop if decrease in impurity is smaller than a threshold
- grow a very large tree and prune it back ('cut branches')

Classification trees in R: Boston housing data

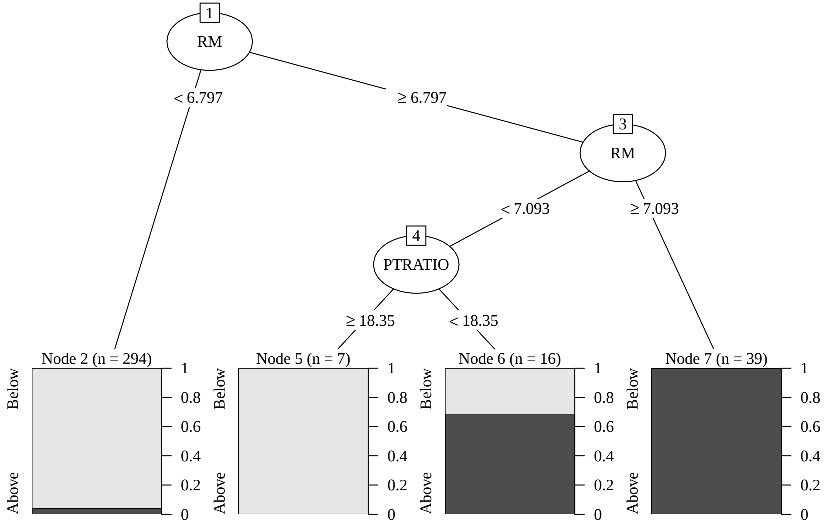
Classification trees can be fitted by using the same commands.

```
# Run a classification tree using the rpart command
> fit<-rpart(MEDV_Fac~CRIM+CHAS+RM+PTRATIO, data=Daten,
   method="class", minsplit=5, cp=0, subset=inTrain)

> # Prune the tree
> fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"]
[1] 0.03225806
> pfit<- prune(fit, cp=0.03225806) # from cptable

> # Plot the pruned classification tree
> plot(as.party(pfit))
```

Classification trees in R: Boston housing data



Classification trees in R: Boston housing data

Compare the performance of the pruned tree with the full tree on the validation data.

```
# Predictions for the validation data
> pred_v_tree<-predict(fit,newdata=vali,type="class")
> pred_v_ptree<-predict(pfit,newdata=vali,type="class")

# Evaluate performance
> confusionMatrix(pred_v_tree, vali$MEDV_Fac)
Prediction Below Above
     Below   120     6
     Above     8    16
               Accuracy : 0.9067
> confusionMatrix(pred_v_ptree, vali$MEDV_Fac)
Prediction Below Above
     Below   122     3
     Above     6    19
               Accuracy : 0.94
```

Advantages and disadvantages of trees

+ Trees are very easy to explain to people.
+ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches.
+ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).

- Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.
- The decision trees suffer from high variance.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved.