

# Data-Mining

Oberseminar Sommersemester 2020

## Vortrag 2: Bewertung und Klassifikation von Daten

Benedikt Grothues

# Inhalt

- 1. Klassifikation**
  - a. Kurzübersicht**
  - b. Vorgehensweise**
- 2. Variablenklassen**
- 3. Datenpartitionierung**
- 4. Beurteilung von Klassifikationsergebnissen**
  - a. Konfusionsmatrizen**
  - b. Grenzwertoptimierungskurve**
- 5. Vorstellung Klassifikation**
  - a. Naive Bayes**
  - b. k-nächste-Nachbarn**
  - c. Lineare-Diskriminanz-Analyse**
  - d. Support-Vektormaschinen**
  - e. Entscheidungsbäume**

# Klassifikation

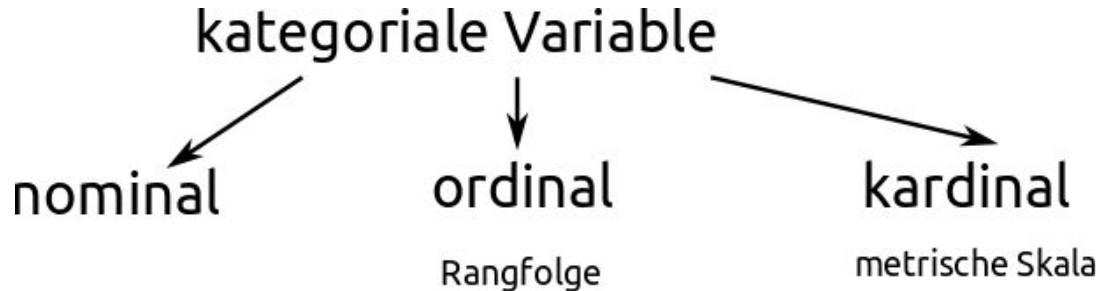
- überwachtes Lernverfahren: label Datensatz
- Objekte zu bekannten Klassen zuzuordnen
- Ziel ist eine hohe Klassifikationsgüte

# Klassifikation - allgemeine Vorgehensweise

1. Datenerhebung- und aufbereitung
2. Bestimmung relevanter Einflussgrößen
3. Auswahl der Klassifikationsmethode
4. Daten-Partitionierung
5. Trainieren des Klassifikators/Modells
6. Validieren des Klassifikators - Anpassung der Modell-Parameter
7. Ermitteln der Güte mit Testdaten
8. Praktischer Einsatz

# Variablen, Merkmale

- Zielgröße (response, dependent, target, predicted variable; outcome, lable attribute)
- Einflussgröße (explanatory, input, regular, independent, prediction variable; outcome)
- Kategorisierung (Skalenniveau)



# Erzeugen von zufälligen Datenpartitionen

Partitionen unabhängig → belastbare Abschätzung

daher: Aufteilung des Datensatzes in

- Trainingsdaten: *Lernen* des Klassifikators (build model)
- Validationsdaten: *Optimieren* der Modellparameter
- Testdaten: *Bewertung* der Klassifikationsgüte

Weiterführend: Resampling

1. *Bootstrapping*
2. *Kreuzvalidierung*
3. Sonderfall *Jackknife method*

# Erzeugen von zufälligen Datenpartitionen in R

- Verwendung caret-package (**C**lassification **A**nd **R**egression **T**raining)
- Funktion createDataPartition()
- zwei Partitionen mit 80% Training und 20% Validation

```
> training.idx <- createDataPartition(data$variable, p = 0.8, list = FALSE)
> training.part <- data[training.idx, ]
> validation.part <- data[-training.idx, ]
```

- drei Partitionen mit 70 % Training je 15% Validation & Test

```
> trg.idx <- createDataPartition(data$variable, p = 0.7, list = FALSE)
> trg.part <- data[trg.idx, ]
> temp <- data[-trg.idx, ]
> val.idx <- createDataPartition(temp$MEDV, p = 0.5, list = FALSE)
> val.part <- temp[val.idx, ]
> test.part <- temp[-val.idx, ]
```

# Klassifikationsbeurteilung - Konfusionsmatrix

## Kenngroßen

- Gesamtklassifikation  
 $n = tp + tn + fp + fn$

- **Korrektklassifikation**  
 $t = tp + tn$

- **Falschklassifikation**  
 $f = fp + fn$

## Triviale Beurteilung

- Korrektklassifikationsrate  $t/n$
- Falschklassifikationsrate  $f/n$

		Klassifikation	
		positive	negative
Wahrer Wert	true	true positive	false negative
	false	false positive	true negative



# Klassifikationsbeurteilung - Konfusionsmatrix

## Kenngroßen

- relevante Klassifikation

$$r = tp + fn$$

- irrelevante Klassifikation

$$i = fp + tn$$

		Klassifikation	
		positive	negative
wahrer Wert	true	true positive	false negative
	false	false positive	true negative

## Abgeleitete Größen

- Richtig-Positiv-Rate (true-positive-rate  $tpr$ ):  $tp/r$
- Richtig-Negativ-Rate (false-positive-rate  $fpr$ ):  $fp/i$

# Klassifikationsbeurteilung - Konfusionsmatrix in R

- Einlesen der Daten:

```
cp <- read.csv("college-perf.csv")
```

- Erzeugen von geordneten factor-Variablen

```
cp$Perf <- ordered(cp$Perf, levels = c("Low", "Medium", "High"))
```

```
cp$Pred <- ordered(cp$Pred, levels = c("Low", "Medium", "High"))
```

- Ausgeben der Konfusionsmatrix

```
tab <- table(cp$Perf, cp$Pred, dnn = c("Actual", "Predicted"))
```

Datensatz: college-perf.csv

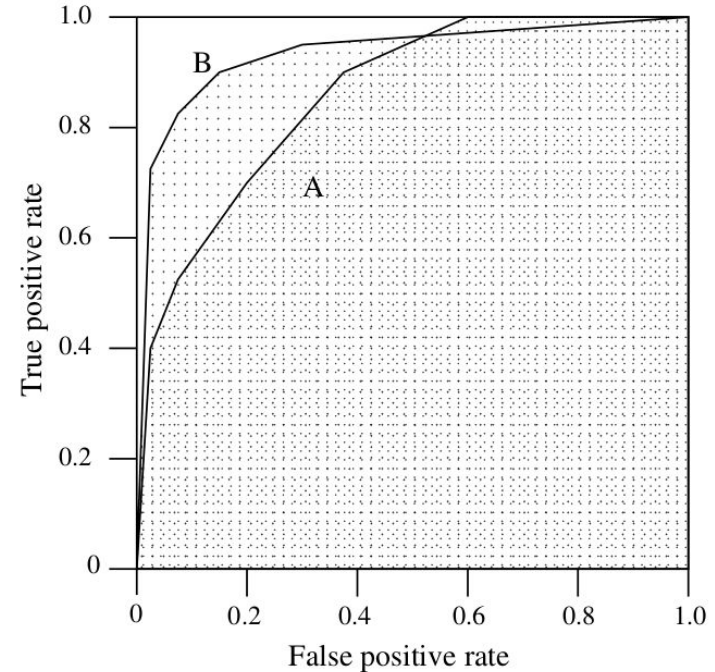
	SAT	GPA	Projects	Community	Income	Perf	Pred
1	1380	2.53	1	0	41800	Low	Low
2	1100	3.18	1	5	37600	Low	Low
3	1110	2.73	2	10	34800	Medium	Medium
4	1180	2.49	3	0	24100	Low	High
5	1240	2.89	3	5	56000	Medium	Medium
6	1140	2.85	2	0	50800	Low	Low
7	970	2.37	1	0	47000	Medium	Medium
8	1100	2.67	2	0	50900	Medium	Medium

GPA: grade point average (US Notendurchschnitt)

SAT: scholastic assesement test (Studierfähigkeit)

# Grenzwertoptimierungskurve - ROC-Graph

- engl. receiver operating characteristic
- *tpr* über *fpr* aufgetragen
- Graph zeigt Güte
  - eines bestimmten Klassifikators
  - auf einem festen Datensatz
  - mit verschiedenen Parameterwerten
- Variieren der Parameter erzeugt Kurve
- Ermöglicht grafische Analyse zur Parameterwahl



Quelle: An introduction to ROC analysis,  
Tom Fawcett; Pattern Recognition Letters  
27 (2006) 861–874

# Grenzwertoptimierungskurve - ROC-Graph

- Rohdaten mit Wahrscheinlichkeiten und Klassenzugehörigkeit

```
dat <- read.csv("roc-example-1.csv")
```

- prediction-Objekt erzeugen

```
numerisch: pred <- prediction(dat$prob, dat$class)
```

```
kategorisch: pred <- prediction(dat$prob, dat$class, label.ordering = c("non-buyer", "buyer"))
```

- performance-Objekt: richtig-positiv-rate über falsch-positiv-rate

```
perf <- performance(pred, "tpr", "fpr")
```

- Grafische Ausgabe

```
plot(perf)
```

Quelle: An introduction to ROC analysis,  
Tom Fawcett; Pattern Recognition Letters  
27 (2006) 861–874

# Naiver Bayes-Klassifikator

- probabilistisches Verfahren
- basiert auf dem Satz von Bayes
- Prognose der wahrscheinlichsten Klassenzugehörigkeit
- Wahrscheinlichkeiten aus Trainingsdaten berechnet

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Vorteile	Nachteile
hohe Effizienz	erfordert stoch. unabhängige Einflussgrößen
Robust gegen fehlende Daten	Einflussgrößen müssen diskret sein

# Naiver Bayes-Klassifikator in R

#0. Arbeitsverzeichnis aufräumen und Pfad setzen

```
rm(list = ls())  
setwd(dr = "Dokumente/4._Semester/Oberseminar_Data_Mining")
```

#1. nötige Packages laden

```
library(e1071)  
library(caret)
```

#2. Datensatz einlesen

```
ep <- read.csv("electronics-purchase.csv")
```

#4. Erzeugen von zwei Partitionen (Indexe)

```
set.seed(1000)  
train.idx <- createDataPartition(ep$Purchase, p = 0.67, list = FALSE)
```

#4. Naive-Bayes-Klassifikationsmodell erzeugen auf Basis Trainingsdaten

```
nb.model <- naiveBayes(Purchase ~ ., data = ep[train.idx,])
```

#5. Modell ausgeben lassen

```
nb.model
```

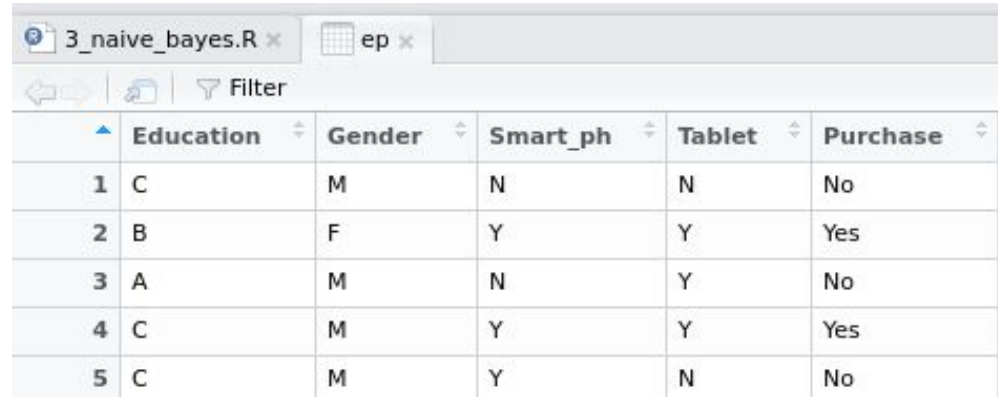
#6. Prognose der Testdaten auf Basis des Modells

```
nb.pred <- predict(nb.model, ep[-train.idx,])
```

#7. Ausgabe der Konfusionsmatrix für die Testperformance

```
(nb.tab <- table(ep[-train.idx,]$Purchase, nb.pred, dnn = c("Actual", "Predicted")))
```

Datensatz: **electronics-purchase.csv**



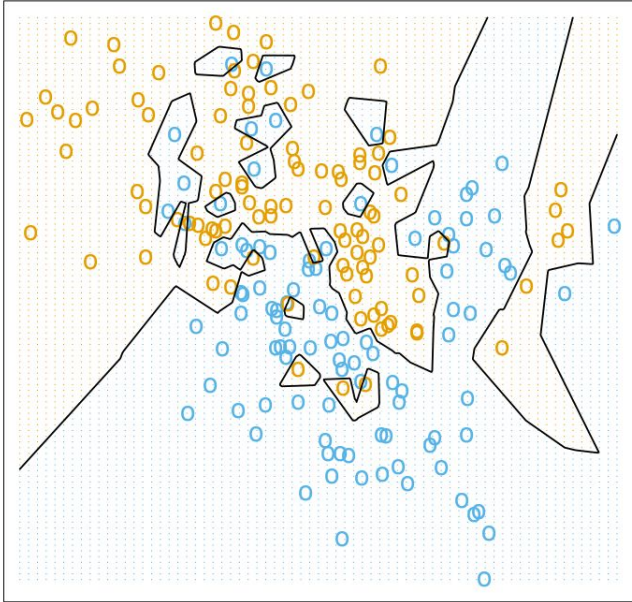
	Education	Gender	Smart_ph	Tablet	Purchase
1	C	M	N	N	No
2	B	F	Y	Y	Yes
3	A	M	N	Y	No
4	C	M	Y	Y	Yes
5	C	M	Y	N	No

## k-Nächste-Nachbarn

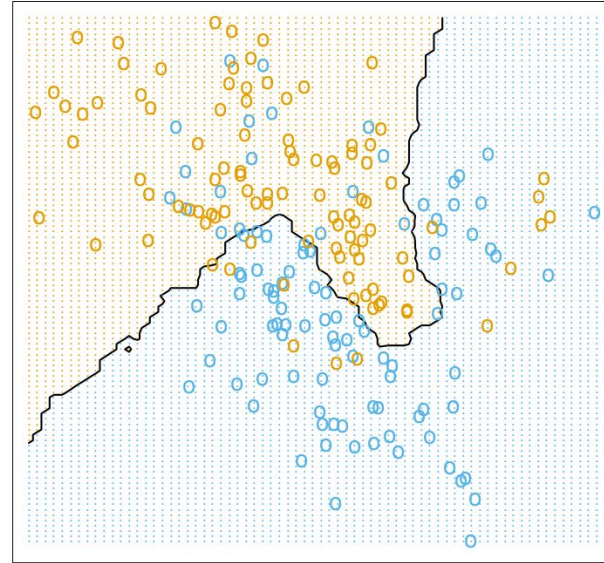
- Prognose ist Mittelwert der  $k$  nächsten Nachbarn
- Einflussgrößen zwingend numerisch
- Zielgröße kategorial
- Bestimmung des  $k$  ist problemabhängig
- benötigt drei Partitionen
  - Validierung des optimalen  $k$  basierend auf Trainingsdaten
  - Test-Prognose auf Basis von  $k$  und Trainingsdaten
- kein Modell, das trainiert werden muss
- Gesamter Trainingsdatensatz wird berücksichtigt

# k-Nächste-Nachbarn

1-Nearest Neighbor Classifier



15-Nearest Neighbor Classifier



quelle: Hastie et al: The Elements of Statistical Learning; Springer



# k-Nächste-Nachbarn in R

#0. Arbeitsverzeichnis aufräumen und Pfad setzen

```
rm(list = ls())  
setwd(dir = "~/Dokumente/4._Semester/Oberseminar_Data_Mining/Datasets/")
```

#1. nötige Packages laden

```
library(class)  
library(caret)
```

#2. Datensatz einlesen

```
vac <- read.csv("vacation-trip-classification.csv")
```

#3. Standardisieren der Einflussgrößen 'Income' and 'Family\_size'

```
vac$Income.z <- scale(vac$Income)  
vac$Family_size.z <- scale(vac$Family_size)
```

#4. Erzeugen von drei Partitionen für das knn-Modell

```
set.seed(1000)  
train.idx <- createDataPartition(vac$Result, p = 0.5, list = FALSE)  
train <- vac[train.idx, ]  
temp <- vac[-train.idx, ]  
val.idx <- createDataPartition(temp$Result, p = 0.5, list = FALSE)  
val <- temp[val.idx, ]  
test <- temp[-val.idx, ]
```

#5. Erzeugen der Prognosen für Validierungsdaten mit k=1

```
pred.val <- knn(train[,4:5], val[,4:5], train[,3], 1)
```

#6. Erzeugen der Konfusionsmatrix für k=1 für Validationsdaten

```
(eremat.valk1 <- table(val$Result, pred.val, dnn = c("Actual", "Predicted")))  
plot(eremat.valk1, col = rainbow(2))
```

#7. Erzeugen der Prognosen für Testdaten mit k=1

```
pred.test <- knn(train[,4:5], test[,4:5], train[,3], 1)
```

#8. Erzeugen der Konfusionsmatrix für k=1 für Testdaten

```
(eremat.test <- table(test$Result, pred.test, dnn = c("Actual", "Predicted")))  
plot(eremat.test, col = rainbow(2))
```

## Datensatz: vacation-trip-classification.csv

	Income	Family_size	Result	Income.z	Family_size.z
1	120000	3	Buyer	1.113089602	0.1110935
2	88000	4	Non-buyer	-0.322142281	1.2220288
3	90000	2	Non-buyer	-0.232440289	-0.9998418
4	85000	2	Non-buyer	-0.456695270	-0.9998418
5	110000	2	Buyer	0.664579639	-0.9998418
6	102000	2	Buyer	0.305771668	-0.9998418

# Datensatz

- banknote-authentication.csv
- 1372 Einträge
- Wavelet-Transformation
- Fotos von Banknoten
- Originale und Fälschungen

## Einflussgrößen

- Varianz
- Schiefe
- Krümmung
- Entropie

	variance	skew	curtosis	entropy	class
1	3.6216000	8.66610	-2.80730000	-0.446990	0
2	4.5459000	8.16740	-2.45860000	-1.462100	0
3	3.8660000	-2.63830	1.92420000	0.106450	0
4	3.4566000	9.52280	-4.01120000	-3.594400	0
5	0.3292400	-4.45520	4.57180000	-0.988800	0
6	4.3684000	9.67180	-3.96060000	-3.162500	0
7	3.5912000	3.01290	0.72888000	0.564210	0
8	2.0922000	-6.81000	8.46360000	-0.602160	0

## Vorverarbeitung in R

*#2. Datensatz einlesen*

```
bn <- read.csv("banknote-authentication.csv")
```

*#3. Zielvariable in factor-Variable wandeln*

```
bn$class <- factor(bn$class)
```

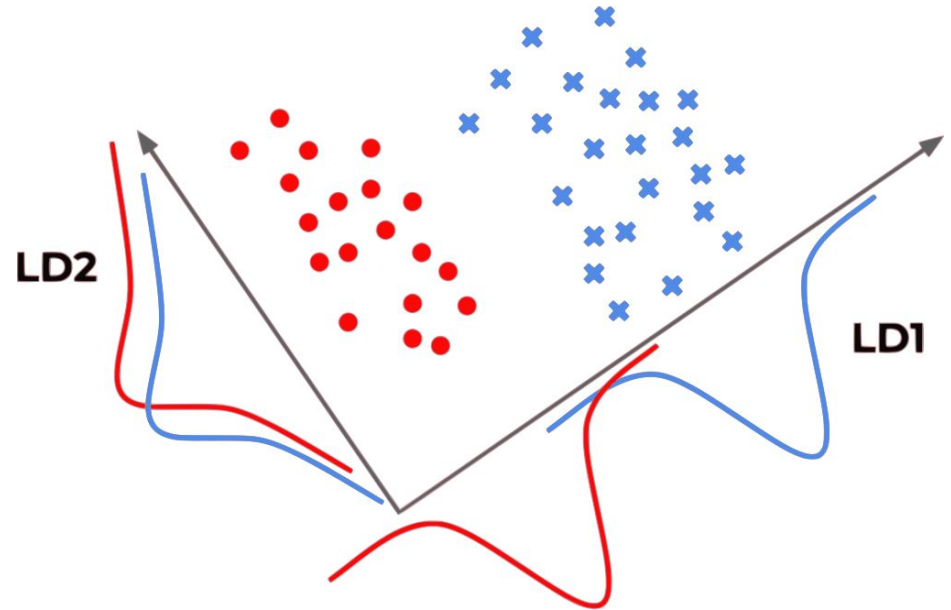
*#4. Erzeugen von zwei Partitionen*

```
set.seed(1000)
```

```
train.idx <- createDataPartition(bn$class, p = 0.7, list=FALSE)
```

# Lineare Diskriminanzanalyse (LDA)

- Einflussgrößen normalverteilt
- lineare Trennung der Gruppen
- Maximierung von Abstand der Klassenzentren (mean-value)
- Minimierung der Varianz innerhalb der Gruppen



# Lineare Diskriminanzanalyse (LDA) in R

*0. Arbeitsverzeichnis aufräumen und Pfad setzen*

```
rm(list = ls())  
setwd(dr =  
"Dokumente/4._Semester/Oberseminar_Data_Mining/")
```

*#1. nötige Packages laden*

```
library(MASS)  
library(caret)
```

*#2. Datensatz einlesen*

```
bn <- read.csv("banknote-authentication.csv")
```

*#3. Zielvariable in factor-Variable wandeln*

```
bn$class <- factor(bn$class)
```

*#4. Erzeugen von zwei Partitionen*

```
set.seed(1000)  
train.idx <- createDataPartition(bn$class, p = 0.7, list=FALSE)
```

*#5. Linear-Diskriminanzanalyse-Klassifikationsmodell erzeugen auf Basis Trainingsdaten*

```
lda.model <- lda(bn[train.idx, 1:4], bn[train.idx, 5])  
#lda.model <- lda(class ~ ., data = bn[train.idx,])
```

*#6. Erzeugen der Modell-Prognosen auf den Trainingsdaten*

```
bn[train.idx, "Pred"] <- predict(lda.model, bn[train.idx,  
1:4])$class
```

*#7. Ausgabe der Konfusionsmatrix der Trainingsperformance*

```
table(bn[train.idx, "class"], bn[train.idx, "Pred"], dnn =  
c("Actual", "Predicted"))
```

*#8. Erzeugen der Prognosen auf den Testdaten*

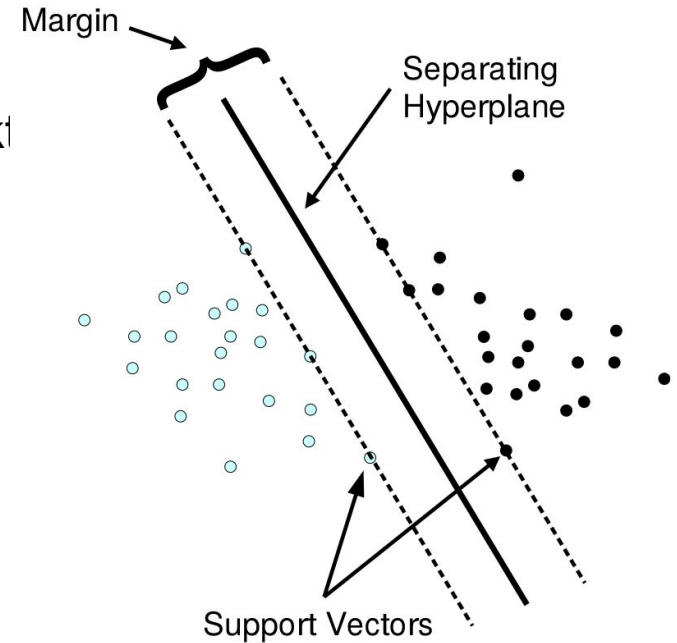
```
bn[-train.idx, "Pred"] <- predict(lda.model, bn[-train.idx,  
1:4])$class
```

*#9. Ausgabe der Konfusionsmatrix der Testperformance*

```
table(bn[-train.idx, "class"], bn[-train.idx, "Pred"], dnn =  
c("Actual", "Predicted"))
```

# SVM - Support Vector Machine

- Klassentrennung durch optimale Hyperebene
- sogenannter large-margin-classifier
- betrachtet nur relevante Punkte (Support Vektoren)
- bei nichtlinearen Trennlinien: Kerneltrick
- findet auch nichtlineare Klassengrenzen
- hoher Rechenaufwand



# SVM - Support Vector Machine in R

*#0. Arbeitsverzeichnis aufräumen und pfad setzen*

```
rm(list = ls())  
setwd(dr =  
"Dokumente/4._Semester/Oberseminar_Data_Mining/")
```

*#1. nötige Packages laden*

```
library(e1071)  
library(caret)
```

*#2. Datensatz einlesen*

```
bn <- read.csv("banknote-authentication.csv")
```

*#3. Zielgröße 'class' in factor-variable wandeln*

```
bn$class <- factor(bn$class)
```

*#4. Erzeugen von zwei Partitionen*

```
set.seed(1000)  
train.idx <- createDataPartition(bn$class, p=0.7, list=FALSE)
```

*#5. SVM-Klassifikationsmodell erzeugen auf Basis Trainingsdaten*

```
svm.model <- svm(class ~ ., data = bn[train.idx,])
```

*#6/7. Prüfen der Modellperformance mit Trainingsdaten und Ausgabe der Konfusionsmatrix*

```
(tab.train <- table(bn[train.idx, "class"], fitted(svm.model), dnn =  
c("Actual", "Predicted")))  
round(prop.table(tab.train)*100,1) #Pozentual
```

*#8. Erzeugen der Prognosen auf den Testdaten*

```
pred <- predict(svm.model, bn[-train.idx,])
```

*#9. Ausgabe der Konfusionsmatrix der Testperformance*

```
(tab.val <- table(bn[-train.idx, "class"], pred, dnn = c("Actual",  
"Predicted")))  
round(prop.table(tab.val)*100,1) #Pozentual
```

*#8. Grafische Darstellung des Modells für Trainingsdaten*

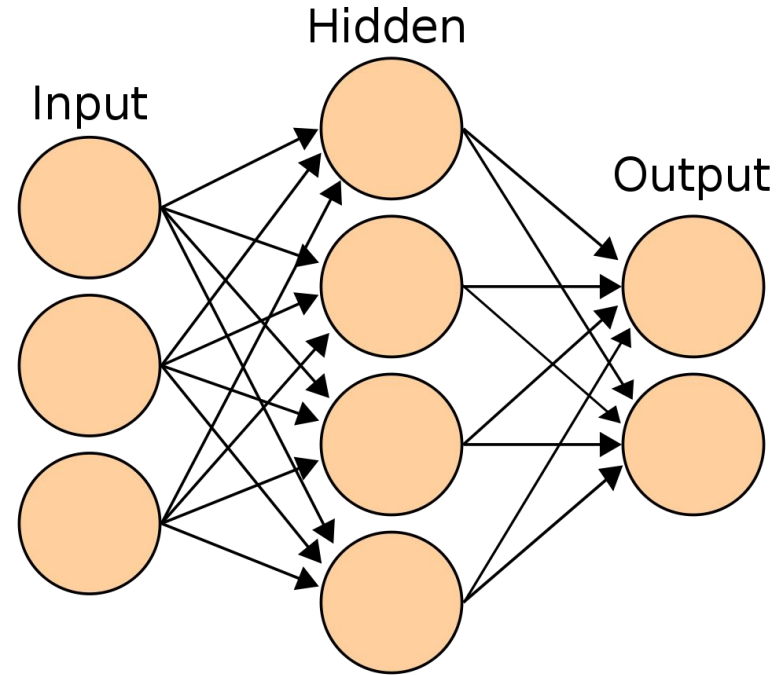
```
plot(svm.model, data=bn[train.idx,], skew ~ variance)
```

*#9. Grafische Darstellung des Modells für Testdaten*

```
plot(svm.model, data=bn[-train.idx,], skew ~ variance)
```

# Künstliches Neuronales Netz

- je Einflussgröße ein Eingabeneuron
- Ausgabeschicht für Zielgröße
- Anpassung der Schwellwerte durch Training
- Neuronales Netz bildet Klassifizierungsaufgabe ab



# Künstliches Neuronales Netz in R

```
#0. Arbeitsverzeichnis aufräumen und Pfad setzen
rm(list = ls())
setwd(dir = "~/Dokumente/4._Semester/Oberseminar_Data_Mining/Datasets/")

#1. nötige Packages laden
library(nnet)
library(caret)

#2. Datensatz einlesen
bn <- read.csv("banknote-authentication.csv")

#3. Zielvariable in factor-Variable wandeln
bn$class <- factor(bn$class)

#4. Erzeugen von zwei Partitionen
train.idx <- createDataPartition(bn$class, p=0.7, list = FALSE)

#5. Neuronales-Netz-Klassifikationsmodell erzeugen
nnet.model <- nnet(class ~., data=bn[train.idx,], size=3, maxit=10000, decay=.001, rang =
0.05)

#6. Erzeugen der Prognosen auf den Testdaten
nnet.pred <- predict(nnet.model, newdata=bn[-train.idx,], type="class")

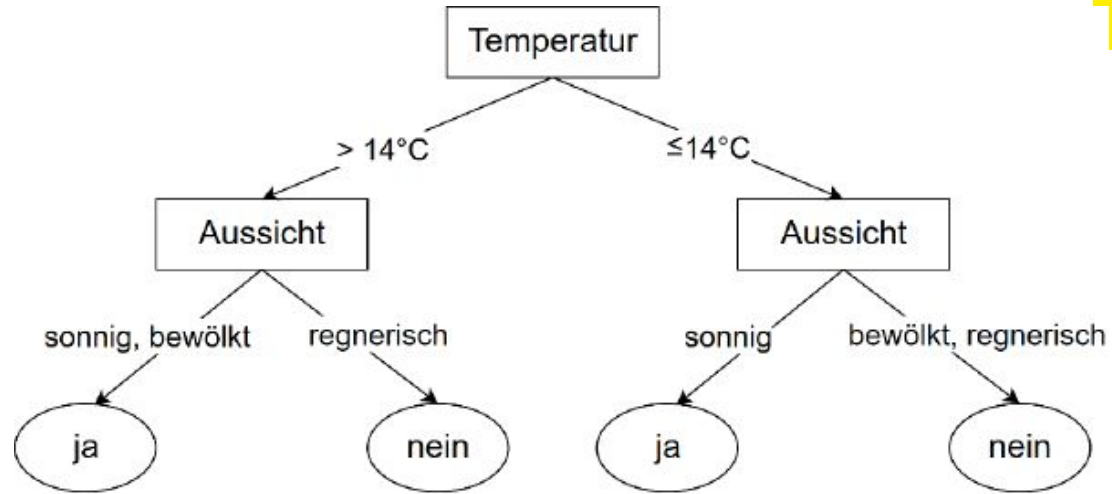
#7. Ausgabe der Konfusionsmatrix der Testperformance
(table(bn[-train.idx,]$class, nnet.pred))
```

- `nnet()` erzeugt single-hidden-layer neural network
- `size`: Anzahl der Neuronen in der versteckten Schicht
- `maxit`: max. Anzahl Iterationen  
Default 100
- `decay`: weight decay. Default 0
- `rang`: Intervall Initialgewichte



# Entscheidungsbaum

- selektiert Einflussgrößen
- top down Wichtigkeit
- verwendet diskrete Einflussgrößen
- leicht zu interpretieren
- Konstruktion berücksichtigt schrittweise Informationszugewinn



Quelle: <https://blog.liwde.de/posts/2018/von-baeumen-netzen-und-maschinen/>

# Entscheidungsbaum in R

#0. Arbeitsverzeichnis aufräumen und pfad setzen

```
rm(list = ls())  
setwd(dr = "Dokumente/4._Semester/Oberseminar_Data_Mining")
```

#1. load required packages

```
library(rpart) #Recursive Partitioning and Regression Trees  
library(rpart.plot)  
library(caret) #Classification And REgression Training
```

#2.read data

```
titanic_data <- read.csv("titanic.csv")
```

#Initialisieren des Pseudozufallszahlengenerators

```
set.seed(1000)
```

#Partition mit zufälligen 70% der Einträge

```
train.index <- createDataPartition(titanic_data$Survived, p = 0.7, list = FALSE)
```

#build classification model

```
mod <- rpart(Survived ~ Pclass + Sex + Age + Fare, data = titanic_data[train.index, ],  
method = "class", control = rpart.control(minsplit = 20, cp = 0.01))
```

#plot the model

```
rpart.plot(mod, box.palette="RdBu", shadow.col="gray", nn=TRUE)
```

#Prognostizierte Klassenzugehörigkeit der Testdaten

```
pred.class <- predict(mod, titanic_data[-train.index,], type = "class") #factors  
pred.prob <- predict(mod, titanic_data[-train.index,], type = "prob") #probabilities
```

#Konfusionsmatrix der Klassifizierung

```
(tab <- table(titanic_data[-train.index,]$Survived, pred.class, dnn = c("Actual",  
"Predicted")))) #total  
round(prop.table(tab)*100,1) #Pozenatural
```

#Erstelle Prediction-Objekt

```
pred.rocr <- prediction(pred.prob[,2], titanic_data[-train.index,"Survived"])
```

#Generiere Performance-Objekt

```
perf.rocr <- performance(pred.rocr, "tpr", "fpr")  
plot(perf.rocr)
```

## • Datensatz: titanic.csv

	Survived	Pclass	Name	Sex	Age	Siblings.	Parents/	Fare
742	0	1	Capt. Edward Gi...	male	70.00	1	1	71.0000
692	0	1	Col. John Weir	male	60.00	0	0	26.5500
645	1	1	Col. Oberst Alfo...	male	56.00	0	0	35.5000
31	0	1	Don. Manuel E ...	male	40.00	0	0	27.7208
397	0	2	Dr. Alfred Pain	male	23.00	0	0	10.5000
793	1	1	Dr. Alice (Famh...	female	49.00	0	0	25.9292
763	0	1	Dr. Arthur Jacks...	male	46.00	0	0	39.6000

## Welche Methode klassifiziert die Banknoten am Besten?

- LDA
- SVM
- NN
- DT

## Quellen und Literatur

- I. Thomas A. Runkler: ***Data Mining Modelle und Algorithmen intelligenter Datenanalyse*** ISBN 978-3-8348-1694-8
- II. Viswanathan, Gohil, Yu-Wei: *R: Reciepes for Analysis, Visualisatzon and ML* Packt Publishing
- III. Joel Grus: *Data Science from Scratch: First Principles with Python*
- IV. Trevor Hastie, Robert Tibshirani, Jerome Friedman: *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, Springer 2017