Data Mining

Anwendung geeigneter Methoden zur Entdeckung von Strukturen & Zusammenhängen in Datenmengen
- Für grosse Datenbestände
- Anz. Merkmale kann Anz. Beobachtungseinh. übersteigen
- Charakteristisch, Daten wurden nicht zu einem bestimmten Zweck erhoben auch anwendbar, wenn traditionelle Techniken versagen
- Garbage in, garbage out (GIGO)

Data Preparation: Umgang mit fehlenden Werten / Ausr., Standardisierung, Transformation, aggregieren und selektieren von Var.

Modeling: Auswahl Data Mining Verfahren, Prognosen, Auswertungen

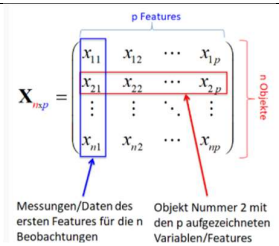
Evaluation: Statistische Evaluierung des Modells, qualitative Beurteilung

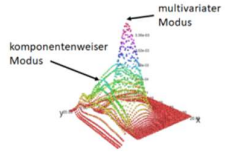
Allgemein: Multivariate Daten – p-Dimensionen

Auch bei überwachten Verfahren häufig hilfreich
- Pre-processing Schritte vor Reduktion
- Kann helfen, irrelevante Variablen eliminieren oder Noise in den Daten reduzieren
- Zeit und Rechenpower reduzieren
- Gewisse Klassifikationsmodelle können nicht mit hochdimensionalen Daten umgehen (Regression)

Modus: Multivariater Modus kann nicht über Moduswerte der Randverteilung abgeleitet werden

Median: Keine allgem. Definition für multiv. Median, am weitesten verbreitet ist L1-Median (Punkt mit minim. Summe der absoluten Abstände zu allen anderen Punkten), komponentenweiser Median ergibt nicht dasselbe
R-Code: L1median (package: robustX)

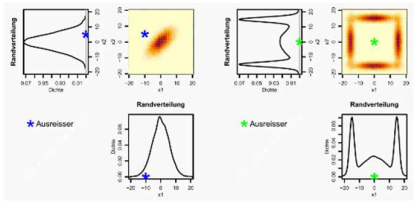




Mittelwert: Komponentenweiser Mittelwert entspricht dem multivariaten Schwerpunkt

Ausreisser: Multivariater Ausreisser (grün) ist nicht zwingend auch ein Ausreisser in der Randverteilungen (blau)

➔ Hochdimensionale Räume haben häufig extreme Merkmalswerte
→ effektive Ausreisser analysieren
➔ Alle Punkte haben fast denselben Abstand zueinander (mit dist(dat) Distanzmatrix berechnen)



Benötigte Beobachtungen:

Dimension	Anzahl Beobachtungen
1	10
2	100
p	10^p

Datenübersicht
str(dat)
dim(dat)
table(dat$zielvar) → Übersicht Verteilung
table(dat$zielvar)/sum(dat$zielvar) → Anteil

Visualisierung:
boxplot(dat) / boxplot(var ~ zielvar, data = dat)
pairs(dat[,2:4], col = c("red", "blue"))[dat$zielvar]
→ univariate Beziehung

moaicplot(~ var + zielvar, data = train_data)
barplot(table(dat$zielvar), las = 2)

Skalieren, wenn Einheiten nicht gleich sind, wenn alle Variablen gleiches Gewicht haben sollen
Zentrierung, wenn bereits alle Einheiten & Varianzen gleich / ähnlich sind, um Ausreisser eher hervorzuheben

Ähnlichkeits- und Distanzmasse

Ähnlichkeitsmass s (similarity)
→ quantifiziert die Übereinstimmung
→ je grosser der Wert desto grösser die Ähnlichkeit

Distanzmasse für quantitatve Variablen
Axiome Distanzmass
Eigenschaften (reellwertige Distanzfunktion):
1. $d(O_1, O_2) \geq 0$ → nicht negativ
2. $d(O_1, O_1) = 0$ $|d(O_2, O_2) = 0$ → Dist. zu sich selbst = 0
3. $d(O_1, O_2) = d(O_2, O_1)$ → symmetrisch

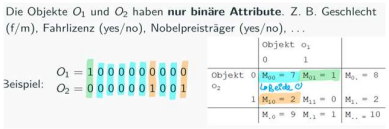
Metrische Distanzen erfüllen zusätzlich folgende Eigenschaften:
4. $d(O_1, O_2) = 0$, genau dann wenn $O_1 = O_2$
5. $d(O_1, O_2) \leq d(O_1, O_3) + d(O_3, O_2)$ → Dreiecksungleichung

Minkowski Distanz
→ $d_{ij}(o_i, o_j) = \left(\sum_{k=1}^{p} |o_{ik} - o_{jk}|^r\right)^{\frac{1}{r}}$

Euklidische Distanz (L2-Metrik)
dist() auf Distanzmatrix anwenden → nur bei num Variablen sonst as.matrix(daisy(data, type=list(ordratio))[c(«A», «C»), «B»]
#Parameter: ordratio behandelt Variablen mit Verhältnisskala ordinal, type=list(ord..) zwingend davor
→ Distanz mit Pythagoras berechnen, direkte Distanz Mass vom Range der Variablen abhängig
→ bei Bedarf für Rob. Skal. im hochdimensionalen Raum: Entspricht PCA

Manhattan Distanz (L1-Metrik)
→ $d_{ij}(o_i, o_j) = \sum_{k=1}^{n} |o_{ik} - o_{jk}|$
Für hochdimensionale Daten besser geeignet als euklidische Distanz da auf grosse Abstände nicht quadratisch eingegangen wird, nicht auf direktem Weg «um Blocks»

Distanzmasse für binäre Daten



Symmetrische Variablen: Beide Levels ähnlich häufig (z.B. Geschlecht):
- Simple-Matching: $s(o_1, o_2) = \frac{M_{11}+M_{00}}{(M_{01}+M_{10}+M_{11}+M_{00})}$
- Canberra-Metrik: $d(o_1, o_2) = 1 - s(o_1, o_2)$

Asymmetrische Variablen: Levels sehr unterschiedlich
- Jaccard Koeffizient $s(o_1, o_2) = \frac{M_{11}}{(M_{01}+M_{10}+M_{11})}$
  → Ohne Variabel mit der höchsten Anzahl
- Jaccard-Metrik $d(o_1, o_2) = 1 - s(o_1, o_2)$

Mehr als 2 Levels (nominale und ordinale Daten)
Objekte O1 und O2 haben p nominale Merkmale:
Sneath-Übereinstimmungskoeffizient
$s(o_1, o_2) = \frac{\#\text{Übereinstimmungen}}{p}$
$d(o_1, o_2) = 1 - s(o_1, o_2) = \frac{\#\text{nicht Übereinstimmungen}}{p}$

Falls geordnete Kategorien mit K Stufen (ordinale Daten)
Kategorien die Werte 1 bis K zu ordnen → x_{O_i}
Transformieren: $z_{O_i} = \frac{x_{O_i}-1}{K-1} \in [0,1]$ → Distanzen für quantitative Daten

Mischung von Variablentypen - Distanzmass von Gower
Distanzmasse zwischen 0 und 1 werden für alle Variablen verwendet.
- K-tes Merkmal ist numerisch:
  Rk: Range der Variable k für alle Objekte
  $d(o_i, o_j) = \frac{|o_{ik} - o_{jk}|}{Rk}$
- K-tes Merkmal ist binär, nominal:
  Canberra-Metrik bzw. Jaccard-Metrik für asymmetrische Variablen
- K-tes Merkmal ist ordinal:
  Transformation in Ränge und diese anschliessend wie numerische Grösse behandeln

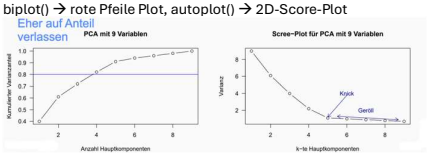
Aggregation über alle Variablen
$$d_{ij} = \frac{1}{p}\sum_{k=1}^{p} d_{ij}^{k} \rightarrow \text{Grösse zwischen } 0 \text{ und } 1$$

Distanzmatrix
Distanzmasse paarweise für alle Kombinationen. Distanzmatrix → Diagonalelemente 0, Matrix symmetrisch

```
library(cluster)
daisy(dat, type=list(asymm=3)) # type nicht zwingend
daisy(dat, type=list(asymm=3, ordratio=c(7,8)), stand =T)
# ordratio behandelt Variablen mit Verhältnisskala ordinal,
type=list(ord..) zwingend davor
round(as.matrix(daisy(dat, stand = TRUE))[c("D", "F", "G"), "A"], 1)
#Standardabweichung skal. eukl. Distanzen zwischen A und D, F und G

!! stand=TRUE, Skalierung basierend auf dem MW der absoluten Abweichung vom Zentrum → besser

Standardisierung daisy → Standardabw. skal. eukl. Distanzen
x <- scale(dat, center = TRUE, scale = FALSE) # zuerst Mittelwert von jeder Variable abziehen
sx <- colMeans(abs(x), na.rm = TRUE) x <- scale(x, center = FALSE, scale = sx) # mittlere abs. Abweichung vom Mittelwert
 round(as.matrix(daisy(x))[c("D", "F", "G"), "A"],1) # Distanzen ohne stand = TRUE für den Vergleich

Falls verschiedene Variablentypen:
daisy() wählt automatisch metric="gower"
```

Unüberwachtes Lernen

Dimensionsreduktion → Zweck: Visualisierung hochdimensionaler Daten
- Strukturen erkennen
- Ausreisser ermitteln

Methode	Ziel	Voraussetzungen	Vorgehen	Plot	R-Code
PCA (Hauptkomponenten-analyse)	Achsen finden, welche multidim. Daten erklären - Variablen sollen unkorreliert sein - Grossen Teil der Totalvarianz abdecken	• **Datenmatrix** • Originalvariablen stark korreliert • kat. Variablen als factor() • Numerische Variablen • Annahme: Daten normalverteilt und - frei von Ausreissern	1. Koordinatensystem in Schwerpunkt der Daten verschieben 2. Rotation bis Varianz entlang der ersten Hauptkomponente am grössten ist → Grösste Teil der restlichen Varianz entlang der zweiten Hauptkomponente (steht orthogonal zur Ersten) Rotation: Ergibt sich durch Multiplikation der zentrierten Datenmatrix X mit Rotationsmatrix A, → Matrizen	biplot() → rote Pfeile Plot, autoplot() → 2D-Score-Plot  Score-Plot: Ellbogenmethode	prcompt(data, scale = TRUE) library(ggfortify), autoplot(...) → Scoreplot, biplot(..) Überprüfung Streuung: boxplot(cc[, 2:x]) *# x: (alle Var - Zielvar)* Falls unterschiedliche Streuung der Variablen: pca_klassisch <- prcomp(creditcard[,2:30],scale=TRUE) plot(pca_klassisch$x[,1],pca_klassisch$x[,2], *# Plot Anteil (siehe links)* col=c("red","blue")[as.factor(creditcard$Class)], pch=20,las=1,xlab="PC1",ylab="PC2") var <- pca_klassisch$sdev^2 *# 80% der Var ermitteln*

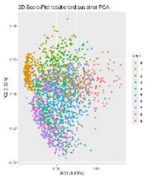
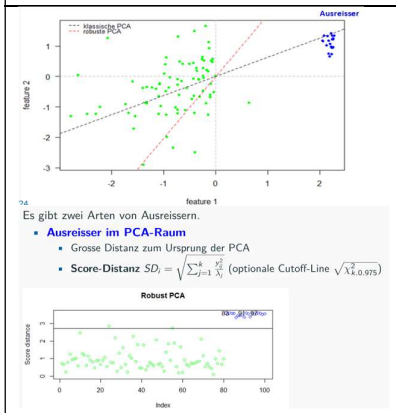
<table>
<tr>
<td></td>
<td></td>
<td>PCA Vorteile:
- Nutzen besonders hoch, wenn Originalvariablen stark korreliert sind
- Wichtigsten Hauptkomponenten können für weiterführende Analysen verwendet werden

PCA Nachteile:
- Abhängig von Skalierung
- Hauptkomponenten schwierig zu identifizieren
- Anzahl relevante Hauptkomponenten nicht eindeutig
NUR FÜR NUMERISCHE VARIABLEN</td>
<td>können nur multipliziert werden, wenn ihre Dimensionen kompatibel sind (siehe LA Regeln)

Varianz der j-Variabel: In Diagonalelementen enthalten, die anderen Werte entsprechen der Kovarianz der j-ten und k-ten Variabel

Kovarianzmatrix: Quadratisch & symmetrisch

Falls Variablen korrelieren: Werte ausserhalb der Diagonale haben Wert ≠ 0, falls keine Korrelation Werte = 0

Autoplot:
library(ggfortify)
x$label <- as.factor(x$label)
res_pca <- prcomp(x[,-1])
autoplot(res_pca, data = x,
colour = 'label',
label.size = 3) +
ggtitle("2D Score-Plot resultierend aus einer PCA")
 
res.pca <- prcomp(diabetes[,-1])
plot(res.pca$x, pch=20, main="Klassen",
col = diabetes$class)
legend("topright", legend = levels(diabetes$class),
col = 1:3, pch=20)</td>
<td>Dimensionsreduktion
Verwenden nur die ersten paar Hauptkomponenten → Daten sollen durch weniger Komponenten möglichst gut beschrieben werden
Gütekriterium: ~ 80 % der totalen Varianz</td>
<td>var_cum <- cumsum(var)/sum(var)
plot(1:length(var),var_cum, ylim=c(0,1)) # Ergebnis 80 % Varianz plotten
pc_number <- min(which(var_cum> 0.8))
abline(h=0.8,v=pc_number)
grid()# Gitternetzlinien
min(which(var_cum > 0.8)) # Dim.Reduktion: Num-Ausgabe</td>
</tr>
<tr>
<td>Robuste PCA</td>
<td>PCA Methode direkt mit robusten Schätzern durchführen</td>
<td></td>
<td></td>
<td> 

➔ Oberhalb Linie = potentielle Ausreisser</td>
<td>library(“MASS”) & library(“rrcov”), library(robustbase)
pca.rob <- PcaHubert(creditcard[, 2:30],scale = TRUE, k = 2)
Argument k: Entspricht der Anzahl berechneten Hauptkomponenten. u dieser Dimension wird auch die orthognonale Distanz berechnet.
plot(pca.rob@scores[,1], pca.rob@scores[,2], col = c("red",
"blue")[as.factor(creditcard$Class)],
pch=20, las=1, xlab="PC1", ylab="PC2")

Orthog. Ausreisser (Punkte mit grossem Rekostruktions-Fehler) in Plot mit orth. Distanzen prüfen & Cutofflinie einzeichnen:
n <- nrow(creditcard)
plot(x=1:n,y=pca.rob@od,pch=16,col=c(rgb(1,0,0,0.6),"blue")
[as.factor(creditcard$Class)])
points(x= which(c_card$Class==1),
y=p_rob@od[c_card$Class ==1],col= 'blue',pch=16)
points stellt sicher, dass alle Betrugsfälle erkennbar sind (nicht überdeck)
abline(h=pca.rob@cutoff.od)
Zeilen, die nach diesem Kriterium Ausreisser sind:
outlier.rows <- which(pca.rob@od >= pca.rob@cutoff.od)
table(ifelse((pca.rob@od >= pca.rob@cutoff.od) == TRUE,
"Ausreisser", "Kein Ausreisser"), ifelse(cc$Class == 1,
"Betrug", "Kein Betrug")) # Konf.matrix
Abdeckung in % der Hauptkomponenten: summary(pca)</td>
</tr>
<tr>
<td>MDS</td>
<td>Beziehung zwischen Obj. basierend Ähn-lichkeiten oder Distanzen im niedr.dim. Raum visualisieren. (Gleiches wie PCA, aber mit Distanzen)
► metrische MDS
► nicht metrische oder ordinale MDS</td>
<td>• Distanzmatrix
• Wenn kat. Variablen, dann dasiy() anwenden</td>
<td>PCA und metrische MDS sind äquivalent, wenn euklidische Distanzen verwendet werden (metrische Distanzen, mittels Eigenwertzerlegung werden Koordinaten für Visualisierung ermittelt).
MDS ist (meistens) auch für nicht euklidische Distanzen geeignet. Metrische MDS benötigen nur Distanzen.

Falls die Daten nicht metrisch sind: Ordinale MDS durchführen</td>
<td>normaler plot():
res.cmd <- cmdscale(eurodist, k=2) #Metrisches MDS
plot(res.cmd, pch="")
text(res.cmd, labels = rownames(res.cmd))

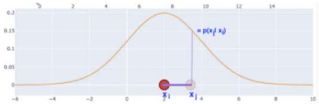
ggplot() mit Einfärbung: library(ggplot2), library(plotly)
res_mds <- cmdscale(dist_matrix23)
df <- data.frame(res_mds)
plot_data <- as.data.frame(df)
plot_data <- cbind(plot_data, Meta23)

P<-ggplot(plot_data, aes(x=X1,
y=X2,label=Name,colour=Fraktion))+
geom_text(size=3)+
geom_point() +
scale_color_manual(values=c("green","yellow","orange"))

ggplotly(P) # Möglichkeit zur Identifizierung einzelner Leute</td>
<td>Ordinale nicht-metrische MDS:
isoMDS() funktioniert nich mit Distanzen = 0
Entfernen dieser Distanzen (je nach Format der Daten):
dist_b<-as.dist(as.matrix(dist_m)[
-which(colnames(as.matrix(dist_m))=='Martin'), #Zeile
-which(colnames(as.matrix(dist_m))=='Martin')]) #Spalte
Meta_b <- Meta23[-which(Meta23$Name == 'Martin'),]

Ordinale oder nicht-metrische MDS: library(MASS)
res.iso <- isoMDS(eurodist, k=2)
plot(res.iso$points, pch="")
text(res.iso$points,
labels = rownames(res.iso$points))</td>
</tr>
<tr>
<td rowspan="2">t-SNE</td>
<td rowspan="2">Versucht Nachbarschaftsbeziehungen im hoch-dimensionalen Raum zu erhalten und in niedriger Dimension darzustellen.</td>
<td>• Datenmatrix: num
• Distanzmatrix: num./kat.

• Wenn nötig, PCA durchführen
• richtige Wahl von Parameter durch Ausprobieren

➔ Ist t-SNE / UMAP besser als PCA: Daten liegen evtl. nicht in einer Hyperebene ➔ Struktur komplexer</td>
<td>Berechnung Ähnlichkeit von einem Punkt zu allen anderen Punkten (euklidische Distanz), Punkte werden danach in eine Normalverteilung X-Achse eingetragen (sofern σ^2 gegeben ist) ➔ Bedingte W'keit
Je näher xj an xi (roter Punkt) ist, desto ähnlicher sind sich die Punkte
➔ Vorgang für alle Punkte wiederholen
  </td>
<td>Vorsicht: Resultate von t-SNE können auch irreführend sein, Clustergrössen & Abstände haben keine Bedeutung ➔ lediglich Darstellung von lokalen Nachbarschaften, mehrfaches Ausführen kann zu lokalen Optimas führen, kann nicht auf neue Punkte angewandt werden

Vergleich Plot PCA & t-SNE:
plot_data = data.frame(res_tsne$Y)
colnames(plot_data)<- c("x1", "x2")

par(mfrow=c(1,2))
plot(res_tsne$Y[,1], res_tsne$Y[,2], type="n", main = "t-SNE",
xlab = "t-SNE1", ylab = "t-SNE2")
text(res_tsne$Y[,1], res_tsne$Y[,2], labels=row.names(abst),
cex=0.8)

res_pca <- prcomp(abst, scale=FALSE)
plot(res_pca$x[,1], res_pca$x[,2], type="n", main = "PCA",
xlab = "PC1", ylab = "PC2")
text(res_pca$x[,1], res_pca$x[,2], labels=row.names(abst),
cex=0.8)</td>
<td>library(Rtsne)
Empfehlung:
1. PCA
2. t-SNE auf die ersten (50) Hauptkomponenten

perplexity: 3 · perplexity < nrow (X) – 1
➔ Mit steigender Perplexity nimmt Varianz zu
Perplexity ~ Anzahl lokale Nachbarn (Werte zwischen 5 – 30 sinnvoll)

iterationen: ausreichend gross wählen
Kategorielle Variablen: z.B. Label : as.factor(x$label)
Label für Rtsne / UMAP nicht verwenden, in Plot jedoch wieder hinzufügen.

res_tsne <- Rtsne(X=dist_matrix23[,-1], dims =2, is_distance=TRUE,
pca = FALSE, perplexity = 10, max_iter = 1000) #Zielvar. entfernen

plot_data <- as.data.frame(res_tsne$Y)
plot_data <- cbind(plot_data, Meta23)
plot_data$label <- dist_matix23$label

ggplot(plot_data, aes(x = V1, y = V2, label= Name, col=label)) +
geom_text(size=3) +
geom_point() +
scale_color_manual(values=c("green", "yellow", "orange", "blue"))</td>
</tr>
<tr>
<td>Skalieren der Ähnlichkeitsmasse:
$p_{j|i} = \frac{p(x_j|x_i)}{\sum_{k \neq i} p(x_j|x_i)}$ ➔ Summe aller $p_{j|i}$ = 1 (p(i|i) = 0)

Inkl. Berücksichtigung Varianz (σ^2) der Normalverteilung:
$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}$</td>
<td>Mass für erw. Überraschung: $H(P_i) = \sum_j p_{j|i} \log_2\left(\frac{1}{p_{j|i}}\right)$

Crowding-Problem: Dimensionsreduktion führt zu Struktur Verfälschung
-> Durch t-Verteilung gelöst
Wird die bed. W'keit zwischen 2 Punkten berechnet (xi zu xj & xj zu xi) werden Werte unterschiedlich sein: Sie stammen aus zwei Verteilungen</td>
<td></td>
<td></td>
</tr>
</table>

UMAP	• Dimensionsreduktion für mehr als 2-3 Dimensionen • Schneller als t-SNE • Oftmals globale Strukturen besser erhalten als t-SNE • Neue Daten können zur Projektion hinzugefügt werden (wenn Datenmatrix vorhanden) • Kann auf Rohdaten angewandt werden (keine vorgängige PCA)	• **Datenmatrix: num** • **Distanzmatrix: num./kat.** • numerische Werte kann auf Rohdaten ohne PCA ausgeführt werden. • richtige Wahl von Parameter durch Ausprobieren	• Abstand wird über Exponentialverteilung bestimmt, Einzeichnung auf der exp. Abfallenden Kurve -> grössere Abst. der Cluster • Mittelung der Ähnlichkeiten in beide Richtungen • Summe der unskal. Ähnlichkeiten (Dichte): $\log_2$(Anz. Nachbarn) • Minimierung der Kostenfunktion (binäre Kreuzentropie)	Clustergrössen haben keine Bedeutung (entscheidend: lokale Abstände), Abstände zwischen Clustern sind nicht immer sinnvoll (können müssen aber keine Bedeutung haben), zufälliges Rauschen sieht nicht immer zufällig aus, Cluster oft besser getrennt als in t-SNE	library(umap) res_umap <- umap(d=as.matrix(dist_matrix23), input = "dist", n_neighbors=25, min_dist=0.2) *#Weitere Parameter: random_state* umap.defaults plot_data <- as.data.frame(res_umap$layout) colnames(plot_data)<- c("x1", "x2") plot_data <- cbind(plot_data, Meta23) ggplot(plot_data, aes(x=x1, y=x2, label=Name, colour=Fraktion))+ geom_text(size=3) + geom_point() + scale_color_manual(values=c("green", "yellow", "orange")) + ggtitle("2D UMAP Visualisierung)

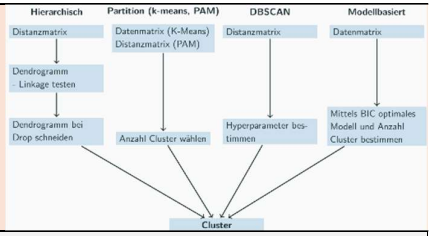
Clustering

- Einteilung von Objekten in homogene Gruppen / Cluster
- Innerhalb der Cluster: Objekte sind untereinander ähnlicher als zu Objekten aus anderen Clustern

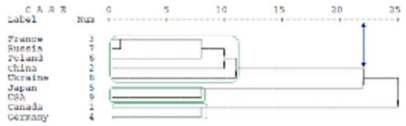
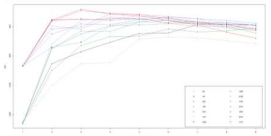
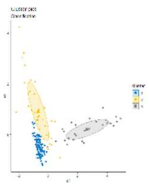
Vorgehen:
1. Explorative Datenanalyse / Datenbereinigung
2. Skalierung der Daten
3. Wahl eines Cluster-Algorithmus
4. Evaluierung Anzahl Cluster

Analyse resultierende Cluster

Visualisierung: Versch. Farben in 2D (PCA, MDS, t-SNE, UMAP)



Methode	Ziel	Voraussetzung	Vorgehen / Anwendung	Plot	R-Code
K-Means	Unterteilung der Beobachtungen in k homogene Cluster – totale Variation (WCV) innerhalb der Cluster über alle k Cluster so klein wie möglich ist. $WCV(C_k) = \frac{1}{\#C_k} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$ WCV=. eukl. Dist.^2 zw. Objekten $\#C_k$=Anz. Beob. im Cluster k p=Anz. Features (Dimensionen) → Liefert lokale Lösung, Abhängig von Startkonfiguration → mehrmals durchführen bis kleinstes WCV gefunden Nicht anwenden bei: - Verrauschten Daten / Ausreissern - Schwierig bei hochdimensionalen Daten (nicht aussagekräftig)	• **Datenmatrix** • kat. Variable als factor • Anz. Cluster muss vorgegeben werden • Festlegung der Anzahl Cluster durch: - Ellbogen-Methode - Silhouetten-Methode - Domain-Knowledge	1. Beobachtungen zufällig einer Gruppe zuordnen 2. Clusterzentrum bestimmen 3. Alle Beobachtungen werden dem Clusterzentrum zugeordnet zudem die kl. Distanz Wiederholen 2. & 3. bis fertig **Qualitätskontrolle: Silhouetten** $Silhouetten - Koeffezient,\ sil_i = \frac{b_i - a_i}{\max(a_i, b_i)}$, a_i = mittl. Distanz zwischen Punkt i und allen anderen Punkten aus dem gleichen Cluster b_i = mittl. Distanz zwischen i und allen Punkten des nächsten Nachbarclusters . bi >> ai → ben. Cluster weit weg $sil_i \to 1$ bi ≈ ai → ben. Cluster nahe $sil_i \to 0$ Beobachtungen im Mittel näher beim benachbarten Cluster → < 0 Faustregel für mittlere Breite in einem Cluster: 0.70 < sil < 1.00: Gute Struktur gefunden 0.50 < sil < 0.70: vernünftige Strukturen 0.25 < sil < 0.50: Schw. Strukt. → w. Unters. −1 < sil < 0.25: Forget it!	R-Code Silhouetten Plot: library(cluster) cl <- kmeans(x, centers=3) plot(silhouette(x=cl$cluster, dist=dist(x))) *# Bei PAM: x = res_pam$clustering* **Anzahl Cluster mit mittl. Silhouettenbreite finden:** auch für pam sil <- rep(0, 15) for (i in 2:15){sil[i] <- summary(silhouette(x=kmeans(x, k=i)$cluster, dist=dist(x)))$avg.width} *# Bei PAM: x = pam(dat, k=i)$cluster* plot(x=1:15, sil, type = "b", las=1, xlab = "Anzahl Cluster", ylab = "mittlere Silhouetten-Breite") **Opt. Clusteranz. mit Within-cluster Variation (Ellbogen-Methode)** wcv <- rep(0, 8) # Initializierung for (i in 1:8) wcv[i] <- sum(kmeans(abst, centers = i)$withinss) plot(1:8, wcv, type = "b", xlab = "Anzahl Cluster", ylab = "within-cluster Variation", las=1) Alternative: library(factoextra) anstelle von kmeans kann auch pam eingesetzt werden -> braucht aber Distanzmatrix fviz_nbclust(x, kmeans, method="silhouette") *# method = "wss"* nur für pam: library(fpc) → Benötigt Distanzmatrix cl_pamk <- pamk(x, krange=2:5) cl_pamk$nc *# Output: Anz. Cluster*	x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2) *# Samples generieren* matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2) matrix(rnorm(100, mean = 1.6, sd = 0.3), ncol = 2)) colnames(x) <- c("x", "y") cl <- kmeans(x, centers=3) *# K-Means anwenden* plot(x, col = cl$cluster, las=1) *# Plot* points(cl$centers, bg = 1:3, col = "purple", ch = 22, cex = 2) cl$withinss *# Abstände zwischen den Clustern* cl$tot.withinss *# Total Abstand* Bei > 2 Dimensionen : Vis. mit 2D-Plot mit PCA, MDS, t-SNE oder UMAP **Anzahl Cluster (k, param: center in kmeans) finden:** wcv <- rep(0, 6) *# Range für wcv generieren* for (i in 1:6) wcv[i] <- sum(kmeans(x, centers = i)$withinss) plot(1:6, wcv, type = "b", xlab = "Anzahl Cluster", ylab = "within-cluster Variation", las=1) *# Plot, Anzahl Cluster ablesen: k nach dem letzten grossen Abfall im Plot* Alternative: library(factoextra) fviz_nbclust(x, kmeans, method="wss", k.max = 10)
K-Medoids / PAM	Repräsentative Objekte finden (Medoids) Ausgehend von einer Distanzmatrix wird über alle Objekte die Gesamtdistanz der Objekte zu den Objekten im gleichen Cluster minimiert - Langsam für grosse Daten → clara()) - Anzahl Cluster muss vorgegeben sein	• **Distanzmatrix** • Festlegung der Anzahl Cluster durch: - Silhouetten-Methode - Domain-Knowledge + Robuster gegenüber Ausreissern + Kann mit allen Distanz-massen umgehen	1. k Objekte zuällig als Clusterzentrum definieren 2. Jedes Objekt dem nächsten Clusterzentrum zuweisen 3. Kostenfunktion wird evaluiert 4. Iterativ wird für jeden Cluster untersucht, ob das Vertauschen des Medoids mit einem nicht Medoid-Objekt zu einem verbesserten Kostenwert führt. Falls zumindest ein Medoid vertauscht wurde, beginnt der Algorithmus wieder bei Schritt 2. Ansonsten endet der Algorithmus.	Silhouetten Plot mit Balken: res.km2 <- kmeans(abst, 2) sil_cl2 <- silhouette(x=res.km2$cluster, dist=dist(abst)) rownames(sil_cl2) <- rownames(abst) plot(sil_cl2, cex.names=0.6) 	R-Code: library(cluster) cl_pam <- pam(x, k=3) plot(x, col = cl_pam$clustering) *#col auch so bei UMAP, tSNE MDS* points(cl_pam$medoids, col = 1:3, pch = 8, cex = 2) cl_pam$objective Bestimmung Anzahl Cluster über mittl. Silhouetten-Breite Repräsentative Beobachtungen: res_pam$medoids
Hierarch. Clustern	Ein Set von verschachtelten Clustern, welche in einem hierarchischen Baum organisiert werden können.	• **Distanzmatrix**	Anzahl Dendrogramme mit n Blättern: $\frac{(2n-3)!}{[(2^{(n-2)})(n-2)!]}$ **Top-Down (divisiv):**	**Dendrogramm:** Abstand zwischen Clustern: Punkt wo die Cluster verb. werden → z.B. 22 !!: Keine Interpretation der Abstände zwischen den Clustern	d <- dist(x) *#Distanzmatrix berechnen* hc <- hclust(d, method="ward.D2") *# "single" / "complete" / "average"* plot(hc) *#Dendrogramm erstellen*

	- Rechenintensiv - Empfindlich gegenüber Ausreissern - Grosse Datensätze schwierig interpretierbar	+ Anz. Cluster muss nicht vorgegeben werden + Flexibel in Wahl der Dist.- und Verkn.metriken	Start mit allen Objekten in einem Cluster, unterteilen in 2 Cluster usw. **Bottom-Up (agglomerativ, übliche Form):** Start mit allen Objekten als eigenes Cluster, kleinstes Element in Distanzmatrix verbinden, Distanzen in der neuen Gruppe berechnen & wiederholen bis fertig **→ Verallgemeinerung für Distanzen zwischen den Clustern: Linkage** - Single-Linkage: Kleinste Distanz, häufig einz. Abspaltungen - Complete-Linkage: Grösste Distanz, Ausreisser können zu gr. Einfluss - Average-Linkage: Mittlere Distanz, Tendenz zu rundereren Clustern - Ward: Varianz zwischen den verbunden Cluster minimieren, Tendenz zu gleich grossen Clustern	Maverik: Nur 1 Objekt in einem Cluster Lücken ansehen: hc$height  Anz. Cluster: hc$width **Heatmap:** Visualisieren einer Datenmatrix (blau = Kleine Werte, rot = grosse Werte) • Zeilenw. Vergl. um mögliche Cluster zu identifiz. • Farbbalken erlauben das Überprüfen zwischen Clustern und externen kategorialen Variablen	rect.hclust(hc, k=3, border="red") *#Cluster unterteilen, k vorgeben* grhc <- cutree(hc, k=3, h=5) *#k=Anzahl Cluster, h=Höhe des Schnittes* plot(x, col = grhc, las =1) *#Plotten der Cluster* table(Cluster=cutree(res.hc, k=3), diabetes$class) **Heatmap:** str(dat) *#Nur numerische Daten* dat_numerisch <- dat[,3:8] *#Nur die numerischen Werte auswählen* x <- t(as.matrix(dat_numerisch)) *#Als Matrix speichern* colnames(x) <- dat$Car *#Namen der Spalten* heatmap(x) *#Heatmap erstellen, weitere Parameter (distfun, hclustfun, scale)* heatmap(as.matrix(x), scale="row") Alternative: library(pheatmap) Pheatmap(x, scale="row") *#Ergänzen mit Meta-Information, Hinzufügen von Parameter: annotation_col = data.frame(Country=dat$Country)*
DBSCAN Dichtebasiert. Clustering	Erkennen von Gebieten mit hoher Datenpunktdichte als Cluster - Schwierigkeiten mit Erkennung von Clustern mit untersch. Dichten - Probleme mit hochdimensionalen Daten (Dichte kaum definierbar)	• **Distanzmatrix** • **Datenmatrix** + Für komplexere Daten mit Rauschen und Ausreissern + Anz. Cluster muss nicht vorgegeben werden	Kernpunkt: Anzahl der Datenpunkte in der ϵ-Umgebung beträgt mindestens MinPts (inklusive Kernpunkt). Randpunkt: Kein Kernpunkt, liegt aber in der ϵ-Umgebung des Kernpunkt Rauschpunkt: Weder Kern- noch Randpunkt → Cluster: Kernpunkte plus dazugehörige Randpunkte	kNNdistplot(dist_weapons, k = 19) abline(h=0.15) res.dbscan <- dbscan(dist_weapons, eps=2200, minPts = 20) weapons_london$cluster <- as.factor(res.dbscan$cluster) plot(x=weapons_london$Longitude, y=weapons_london$Latitude, col = res.dbscan$cluster+1, pch=16)	library(dbscan) db <- dbscan(df, eps = 0.15, MinPts = 5) *#eps = Max. Abstand zw. Pkt.* par(mfrow=c(1,2), mar=c(3,3,2,1), mgp=c(2, 0.5, 0)) plot(df[, 1:2], col=multishapes$shape, las=1, main="multishapes aus factoextra") plot(df[, 1:2], col=db$cluster+1, las=1, main="DBSCAN") Parameter: minPts: mind. 3, eps: kann mittels k-Nächste-Nachbarn-Graph abgeschätzt warden (teilw. jedoch auch Domain knowledge)
Modellbasiertes Clustering	Maximierung der Log-Likelihood mit Expectation-Maximization-Verfahren (EM-Algorithmus) Der Algorithmus berechnet W'keiten der Zugehörigkeit von Datenpunkten zu Dichten & gruppiert sie basierend in Cluster. - Modell aufgrund von Annahmen über Verteilung der Daten könnte falsch spezifiziert sein	• **Datenmatrix** Daten aus Mischverteilung (multivariat) + Anz. Cluster muss nicht vorgegeben werden + Identifikation von Ausreissern	- Zufällige Startparameter wählen: Mittelwert μc , Kovarianz Σc , "Clustergrösse" πc - Optimales Modell und optimale Anz. Cluster wird durch max. BIC bestimmt, BIC gewichtet Daten mit vielen Parametern starker	  Linien Plot: Modell und Anzahl Cluster wählen,bei welchem die Linie am höchsten ist.	library(mclust) mc <- Mclust(Nclus, modelNames="EII") *# spezifisches Modell* mc <- Mclust(Nclus) *# Test aller Modelle* mc$modelName *# bestes Modell* plot(mc, what="BIC") *#Plottet BIC aller Modelle (sofern mc > 1 Modell), kann auch Unsicherheit plotten: what="uncertainty" (je grosser das Symbol desto unsicherere Beobachtung)* map(mc$z) *# z = Clusterzugehörigkeit* Alternativ: library(factoextra) fviz_mclust(mc, "BIC", palette = "jco") plot(res.mc, what="classification") fviz_mclust(mc, "classification", geom = "point", palette = "jco", xlab="x1", ylab="x2") table(Cluster=res.mc$classification, diabetes$class)

Überwachtes Lernen: Vorbereitung / Allgemein

1. Zufälliges Aufteilen in Trainings- und Test-Daten
2. Anpassen des Modells auf Trainingsdaten (inkl. Preprocessing und Modellselektion)
3. Modellevalution

	Actuel value ⊕	Actuel value ⊖
Predicted values ⊕	TP	FP
Predicted values ⊖	FN	TN

Aufteilung Trainings- / Testdaten (80 % / 20 %)
!! Wichtig: Testdaten müssen strikt von Trainsverfahren getrennt warden.
Grund: Gefahr, zu optimistisch zu sein. Rückschlüsse der Modellper-formance auf neue Daten können nicht gezogen werden.
Besser wenn mehr Daten vorhanden, variabler Schätzungsfehler durch zufällige Unterteilung

R-Code: library(caret)
tindex <- createDataPartition(Studenten$x, p = 0.8, list = FALSE) *# list=TRUE: Liste, list=FALSE: Matrix*
train_set <- Studenten[tindex,]
test_set <- Studenten[-tindex,]

Fehlerarten
Fehler Trainingsdaten: Trainingsfehler / naiver Fehler / In-Sample-Fehler
Fehler neue Daten: Verallgem.fehler / Testfehler / Out-of-Sample-Fehler:

Modellkomplexität mitberücksichtigen (Strafe für Komplexität) → bei 2 Modellen mit ähnlichen Verallgemeinerungsfehlern sollte das einfachere Modell dem komplexeren vorgezogen werden.

Overfitting
Modell passt zu gut zu den Trainingsdaten, hat aber einen grossen Testfehler / Kleine Genauigkeit.
Leave-One-Out Kreuzvalidierung (LOOKV)

Modellentwicklung
Raten:
n <- dim(train_set)[1]
y_guessing <- sample(c("Male", "Female"), size=n, replace = TRUE)

Anteil aus Trainingsdaten schätzen:
n <- dim(train_set)[1]
(p <- sum(train_set$sex == "Male") / n)
y_proportion <- sample(c("Male", "Female"), size=n,prob=c(p, 1-p), replace = TRUE)

Grenzwert:
tapply(train_set$height, INDEX=train_set$sex, FUN=function(x) return(c(mean(x), sd(x))))
→ untere Grenze: MW – 2*Std.
fun_cutoff <- function(x, cutoff) ifelse(x > cutoff, "Male","Female")
y_2sd <- fun_cutoff(train_set$height, cutoff=157)

oder Prüfung verschiedener Werte:
cutoff <- seq(150, 200)
accuracy <- rep(NA, length(cutoff))
for (i in 1:length(cutoff)){
y_hat <- fun_cutoff(train_set$height, cutoff=cutoff[i])

Evaluation anhand Testdaten
y_acc <- fun_cutoff(test_set$height, cutoff=164) → Meist tiefer als bei Trainingsdaten
mean(y_acc == test_set$sex)

Metrik	Calculation	Was wird gemessen
Accuracy	$\frac{TP+TN}{ntest}$	Standard Metrik, unterschiedliche Kosten für Fehler werden nicht berücksichtigt.
Precision	$\frac{TP}{TP+FP}$	Wie viele der vorhergesagten ⊕-Objekte sind richtig?
Recall/ Sensitivität	$\frac{TP}{TP+FN}$	Wie viele der ⊕-Objekte wurden richtig vorhergesagt?
Spezifität	$\frac{TN}{TN+FP}$	Wie viele der ⊖-Objekte wurden richtig vorhergesagt?
F1-Score	$\frac{2 \cdot recall \cdot precision}{recall+precision}$	Harmonisches Mittel von Recall und Precision
Kappa	$\frac{accuracy - E[accuracy]}{1 - E[accuracy]}$	Relative Verbesserung im Vergleich zu einem zufälligen Prädiktor

1. Daten in n Datensätze aufteilen, ein Datensatz weglassen 2. Modell mit allen n Datensätzen trainieren und mit ausgeschlossenem Punkt validieren 3. Testfehler aus der Anzahl Fehlklassifizierungen schätzen → Zufälligkeit nicht mehr gegeben (+) → Kleinster Bias (+) → Numerisch, somit komplex/langsam (-)	accuracy[i] <- mean(y_hat == train_set$sex)} max(accuracy) cutoff[which.max(accuracy)] Erste Bewertung: $Accuracy\ (Richtigkeit) = \frac{Anz.richtig\ klassifiziert}{Anz.richtig+An\ .fal}$ $Errorrate = 1 - Accuracy$ R-Code: mean(y_proportion == train_set$x)	Konfusionsmatrix (tab <- table(predicted = y_acc, actual = test_set$sex)) mean(y_acc [test_set$sex =="Male"] == test_set $x[test_set$x=="Male"]) *# Acc. nach Geschlecht* TP + TN / SUM: conf_m[1,1] + conf_m[2,2] / sum(conf_m)
Bootstrap-Methode - B neue Stichproben aus den Daten erzeugt - Gleiche Grösse n wie die urspr. Stichproben, Ziehen mit Zurücklegen ➔ Idee: Datensatz als Ersatz für unbekannte zugrundeliegende Verteilung und ziehe daraus neue Stichproben - Bootstrap-Stichprobe aus den Trainingsdaten Validierung auf den Daten, die nicht in der Bootstrap-Stichprobe sind	**No-Free-Lunch-Theorem** - Vereinfachung um überflüssige Details zu eliminieren Gibt kein Modell, dass für jedes Problem am besten funktioniert	**K-fache Kreuzvalidierung** 1. Daten in k sich nicht überschneidende Gruppen einteilen 2. Modell k-mal trainieren (wieder für alle ausser einer Gruppe, jedes Mal diese Gruppe validieren) 3. Testfehler aus der Anz. falsch klassi. Beobachtungen schätzen → Schneller als LOOKV → Ergebnisse weisen gewisse Zufälligkeit auf → Etwas grösserer Bias als LOOKV, in der Regel jedoch unproblematisch & besser da geringere Varianz als bei LOOKV 10-fache Kreuzvalidierung oft Standard

Überwachtes Lernen: Klassifikation

Methode	Anwendung / Bemerkungen	+ / -	R-Code / Parameter	
kNN **Nächste-Nachbar-Klassifizierer**	• lokale Struktu: ähnliche Datenpunkte gehören tendenziell zu selben Cluster • nicht linear separierbare Daten Klasse der Mehrheit der nächsten Nachbarn (euklidische Distanz) wird übernommen **Bestimmung von k:** - Kleines k : Zu komplexes Modell - Grosses k : einfaches Modell - **Pattsituation verhindern:** **2 Klassen: ungerades k wählen** **> 2 Klassen: k > 1 wählen**	+ Einfach + Flexible Erkennung von Klassen + Gut bei Daten mit lokalen Clustern / Variabilität + Keine Annahme von Verteilung - Empfindlich gegenüber Ausreissern - Rechenintensiv - Wahl k Wert → nicht linear separierbare Daten	library(caret) res.knn <- train(y = trainDat$class, x = trainDat[, -class], method="knn", tuneGrid=data.frame(k=seq(1,20,2)) *# Preprocessing: Parameter preProcess = c("center", "scale", "pca") anfügen* res.knn <- train(form = class ~. , data =trainDat, method= **"knn"** , ...) pred <- predict(res.knn, newdata=testDat) *#Vorhersage* **ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 5)** method : - "repeatedcv": wiederholte k-fache Kreuzvalidierung - "cv": normale k-fache Kreuzvalidierung - "LOOCV": leave-one-out Kreuzvalidierung - "boot": !!!Bootstrap (Standard)!!! - "none": nichts number: k für k-fach (bei boot: 25, bei KV: 10) repeats: Anz. Wiederholungen der wiederholten Kreuzvalidierung model <- train(x, y, method = "knn", trControl = ctrl,tuneGrid=data.frame(k=1:10)) plot(model) *#k bei grösster Accuracy wählen*	Ohne caret: library(class) knn(train=trainDat[,-"label"], cl=trainDat$label, test=newDat, k=k) confusionMatrix(dat=pred, reference = testDat$Species) **Accuracy bei unbalancierten Daten nicht geeignet** Fehlerrate: mean(pred_test != x_train$Y) Bei k=1: Fehllerrate 0, da nur eigenes Label Klasse definiert
Naive-Bayes-Klassifizierer	**Bayes Klassifizierer** Beobachtung wird zur wahrscheinlichsten Klasse, basierend auf den Features, zugewiesen. Entscheidungsgrenze: Wo die W'keit für beide Klassen gleichgross ist $P(Y = C_l \mid X) = \frac{Anfangsw'keit\ Klasse\ l * W'keit\ nach\ Beobachtung\ von\ x}{Randw'keit\ von\ X, W'keit für\ Prädiktorvariablen}$ Likelihood bei multivariaten Daten komplex → Naive Bayes-Klassifikator **Naiver Bayes Klassifikator** Annahmen: Alle Features unabhängig & gleiches Gewicht	+ robust gegen fehlende Daten + Unabh. von Anz. Var. - Annahme, dass alle Var. unabh. voneinander sind - Bei starker korrelation der Variablen, keine guten Ergebnisse → Gut geeignet für Textdaten, fehlende Daten, grosse Merkm.räume & geringe Stichprobengrösse	library(naivebayes) ctrl <- trainControl(method = " ", number=x, repeats=y) r.nb <- train(x=einkaufDat[,1:3], y=einkDat$eink, method= **"naive_bayes"** , trControl = trainControl(method="none"), tuneGrid = data.frame(laplace=0, usekernel=FALSE, adjust=0)) *# Bei FALSE ist adjust zwingend nötig* predict(r.nb, newdata=data.frame(alter="21-30", einkommen="mittel", geschlecht="Frau")) Parameter: Laplace: Faktor für Laplace-Korrektur Usekernel: Empirische Dichteschätzung für numerische Daten Adjust: Bandbreite für Dichteschätzung Alternative:library(e1071) naiveBayes(Einkauf ~ . , data=dat)	Problem wenn Häufigkeiten = 0: Laplace Korrektur / Smooting → Zähler: +konstanter Faktor (meistens zwischen 1 und 2) → Nenner: Faktor mal die Anzahl Features dazuaddieren Umgang mit stetigen Variablen: Können diskretisiert werden, je nach Implementierung aber auch direkt verwendet werden (sofern normalverteilt, nichtparametr. Dichteschätzung)

LDA / QDA Lineare & quadratische Diskriminanzanalyse

1. Modellierung Verteilung der Features X der verschiedenen Gruppen in den Trainingsdaten individuell.
2. Verwende das Theorem von Bayes um die bedingte Wahrscheinlichkeit → P(Y = C_l |X) zu bestimmen
3. Ordne die Beobachtung der Klasse mit der höchsten A-posteriori Wahrscheinlichkeit zu

Normalverteilung, Unabhängigkeit vorausgesetzt
- ➔ Transformationen können helfen um Daten «normalverteilter» erscheinen zu lassen (log & Wurzel für Exp.verteilungen / Box-Cox für schiefe Verteilungen)
- ➔ Ausreisser entfernen für unverzerrtere Schätzungen
- ➔ Gleiche Varianz: LDA geht davon aus, dass Eigenvariable die gleiche Varianz hat. → Standardisieren, sodass MW von 0 und eine Std.abw. von 1 haben

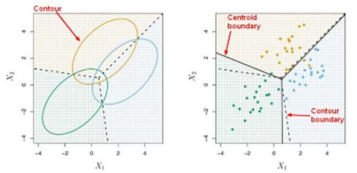
+ LDA: red. Dimensionalität
+ Effizient bei kl. Datensätz.
+ QDA: Berücksichtigung Untersch. Kovarianzstrukt

LDA:
- Annahme norm.Verteilung – Empfindlich gegenüber Ausreissern
- nur für separierbare lineare Daten

QDA:
- Achtung: Overfitting, Anzahl Var > Anzahl Datpkt

```
library(caret)
ctrl <- trainControl(method = " ", number=x, repeats=y)
# Modelfit LDA
lda_fit <- train(y= trainDat$Species, x = trainDat[, 1:4], method = "lda")
res_lda = predict(lda_fit, testDat) # Vorhersage
confusionMatrix(data=res_lda, reference = testDat$Species)
# Modellfit QDA
qda_fit <- train(y= trainDat$Species, x = trainDat[, 1:4], method = "qda", ...)
res_qda = predict(qda_fit, testDat) confusionMatrix(data=res_qda, reference =
testDat$Species) # Vorhersage
```

➔ Gerade gestrichelte Linie deutet auf lineare Diskr.analyse hin, bei quadratischer -> Gebogene Linie



Logistische Regression

W'keit:	5/(5+3)	zwischen 0 und 1
Gegenw'keit:	1 – W'keit	''
Odds:	5/3	zwischen 0 und ∞
Odds aus W'keiten:	W'keit / Gegenw'keit → p / (1 - p)	
log(Odds):		zischen -∞ und ∞

$\log\left(\frac{p(Y|X)}{1-p(Y|X)}\right) = \beta_0 + \beta_1 * x$ → Schätzung β0/1 mittels Maximierung Likelihood

Log Reg:
+ W'keitsvorhersagen für Klassenzugehörigkeit
+ Gut für grosse Dat.sätze
- Anz. Beobachtungen > Anz. Variablen
- bei nicht-linearen Daten schlechte Vorhersagen
- Ann. unabh. Beob.

Multi Reg:
+ Merhrklassen-Klassifikation
+ Modellierung komplexe Zusammenh. (Kl. & Var.)
- Ann. lineare Beziehung
- Ann. unabh. Beob.
- Ausreichend Daten, um robuste Schätzung der Koeffizienten zu erhalten

Logistische Regression

```
ctrl <- trainControl(method = " ", number=x, repeats=y)

library(caret)
Bankrott$Aktiv <- as.factor(Bankrott$Aktiv)
train_index <- createDataPartition(Bankrott$Aktiv, p = 0.75, list = FALSE)
res.glm <- train(y = Bankrott[train_index, 'Aktiv'],x =Banktt[train_index, -
                 which(names(Bankrott) == 'Aktiv')],
                 method = "glm" , family =  "binomial") #method = "polr" für ord. Var.
summary(res.glm)$coefficients
confusionMatrix(predict(res.glm, newdata=Bankrott[-train_index, ]),
                 Bankrott[-train_index, 'Aktiv'])$table
Für W'keiten: predict(res_model, newdata, type = 'prob')
```

```
             Estimate Std. Error   z value     Pr(>|z|)
(Intercept)  2.3822733  1.0093421  2.370131 0.017781770
distance    -0.6500704  0.2296738 -2.830407 0.004648986
```

Koeffizienten sind immer log-Odds →
exp(x$coefficients[2])

Multinom. Regression

```
ctrl <- trainControl(method = "cv", number = 10)
multinom_fit <- train(y = iris[train_index,  'Species'], x =
                 iris[train_index,
                 -which(names(iris) == 'Species')],
                 method = "multinom",
                 trControl = ctrl, tuneLength=7,
                 trace = FALSE)
confusionMatrix(predict(multinom_fit,  newdata=iris[-
train_index, ]),
                 iris[-train_index, 'Species'])
```

Multinomiale logistische Regression

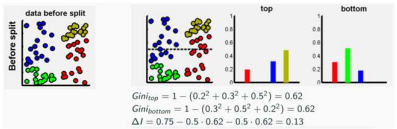
Multinomiale logistische Regression → > 2 Klassen
Eine der k Klassen wird zur Referenzkategorie k_0 (optimal grösste Klasse)
Für alle anderen Klassen wird eine logistische Regression gegenüber der Referenzkat. durchgeführt:

$\log\left(\frac{p(Y_i = k|X_i)}{p(Y_i = 0|X_i)}\right) = logit(p_k)$, Total pk =1

Klässifikationsbäume

Intuitives Klassifikationsverfahren um Objekte mittels (binären) «ja/nein»-Fragen einer Klasse zuzuordnen. Abhängig von der zuvor gestellten Frage. Werden in Baumstruktur zusammengetragen.

Baum verfügt über folgende Elemente: Wurzel, Knoten für Aufteilung, Äste, Endknoten/Blätter



Trainingsfehler liefert keine gute Schätzung, wie gut der Baum auf ungesehene Daten performen wird
Geeignete Komplexität: Fehler = Bias + Varianz + unreduzierbarer Fehler

Stoppregeln: (Problem, wenn Wachstum zu früh gestoppt wird)
- ΔI(N) soll Schwellenwert nicht unterschreiten
- Min. Anz. Objekte, bei welcher ein Knoten aufgeteilt wird
- Mind. Objekte, welche im Endknoten noch vorhanden sein müssen

Besserer Ansatz: Baum vollständig wachsen lassen & zurückschneiden
Cost-complexity Pruning:
$R(\alpha) = Fehlerrate\ in\ den\ Endknoten + \alpha * |T|$
R(α): Mass, welches Modellgüte & Komplexität verbindet
|T| : Anzahl der Endknoten
A: Komplexitätsparameter
→ Ziel R(α) so klein wie möglich, finden durch Kreuzvalidierung

+ robust gegenüber Ausreisser
+ keine Verteilungsann.
+ keine Variablentransfor. notwendig.
+ Umgang mit kat. & num. Werte
+ erwartungstr. Schätzer

- führt schnell zu Overfitting -
Instabilität der Bäume, Interpretation leicht änderbar
- benötigt grosse Datenmengen

```
ctrl <- trainControl(method = " ", number=x, repeats=y)

library(caret), modelLookup("rpart")
ctrl <- trainControl(method = "none")
r.tree <- train(y = iris[train_index, 'Species'], x = iris[train_index,
                 -which(names(iris) == 'Species')], method="rpart" ,
                 trControl = ctrl, tuneGrid = data.frame(cp = 0.01))
#Paramenter für minimale Anz. Objekte für Trennung: control =
rpart.control(minsplit=20), Mindestanzahl Objekte im Endknoten: control =
rpart.control(minbucket= round(minsplit/3)
plot(r.tree$finalModel, margin=0.2, uniform=TRUE) # Visualisierung
text(r.tree$finalModel)

Alternative: library(rpart)
tree <- rpart(Species ~., data =my.train)

Kreuzvalidierung cost-complexity Pruning: library (caret)
ctrl <- trainControl(method = "repeatedcv", number =10, repeats = 3) tree <- train(y = y,
x = X, method="rpart", trControl =ctrl,  tuneGrid=data.frame(cp=...))
plot(tree)
```

Bestimmung der Verunreinigung (impurity measures):
Fehlerrate: 1- max(p_i), i = i-te Klasse
Gini-Index: Erwartet Fehlerrate beim Knoten N, falls Label zufällig von den am Knoten vorhanden Klassen selektiert werden: $1 - \sum_{i=1}^{g} p_i^2$
Entropie: $-\sum_{i=1}^{g} p_i * \log_2(p_i)$
$\Delta I(N) = I(N) - \alpha_L \cdot I(N_L) - (1 - \alpha_L) \cdot I(N_R)$ → α_L = Ant. Beob. Untergr. N_L
→Aufteilung so, dass ΔI(N) maximal

Bias-Varianz Tradeoff
Bias: Differenz zwischen der mittl. Vorhersage unseres Modells und dem richtigen Wert.
→ hoher Bias: Zu stark vereinfachtes Modell mit hoher Fehlerrate bei Trainings- und Testdaten

Varianz: Variabilität der Modellvorhersage für einen bestimmten Datenpunkt
→ Hohe Varianz: Schneiden bei Trainingsdaten sehr gut ab, haben aber hohe Fehlerquoten bei ungesehenen Daten, Andeutung auf zu komplexes Modell

Gleichgewicht: Gesamtfehler = $Bias^2$ + Varianz + Irreduzierbarer Fehler
- ➔ Testen mehrerer Modelle (suche Gleichgew.)
- ➔ Kreuzval. / Anpassung Hyperparameter
- ➔ Regularisierung z.B. mittels Strafterm (Vermeidung Overfitting)
- ➔ Feature engineering (Bearb. Eingabefeatures)
- ➔ Verwendung Ensemble-Methoden

Überwachtes Lernen: Ensemble Methoden

Viele Klassifikatoren zusammen verwenden, um leistungsfähiger zu sein
→ Gesamtklassifizierer (Erfolgreich wenn einz. Klassifizierer im Schnitt unverzerrt)
Voraussetzung Einzelkl.: stochastisch unabhängig, Variabilität reduzieren

Ensembles sind nur dann besser, wenn jeder Klassifizierer besser ist als zufälliges Raten!

Vorteile: Verbesserung der Genauigkeit, Machen Modell robuster
Nachteile: Reduziert Interpretierbarkeit eines Modells, zeitaufwändig

Stacking: Kombination heterogener schwachen Klassifikatoren unter Verwendung von Metamodellen → Unterteilung in mehrere Trainsätze

Bagging: Kombination von homogenen, schwachen Klassifikatoren, die parallel anhand unabhängiger Bootstrap-Stichproben trainiert werden (z.B. Random Forest)

Boosting: Iteratives Verfahren (adaptive Modifikation der Trainingsdaten durch stärkere Fokussierung auf zuvor falsch klassifizierte Datensätze, z.B. Gradient boosting, ADABOOST, XGBoost)

Random Forest

Anwendung / Vorgehen

1. Trainingsset erstellen: N mal mit Zurücklegen aus allen N verfügbaren Trainingsfällen ziehen (Bootstrapsamples) → restliche Daten für Validierung (vorher abtrennen)
2. Ungestutzten Klassifikationsbaum mit Modifikation erstellen: Bei jedem Knoten wird die beste Aufteilung nur unter einer zufälligen Auswahl von mtry Features gesucht, Entkorrelation mit allen Mitteln → Maximierung Varianzreduktion indem Korrelation zwischen Bäumen klein gehalten wird
   → Verhindert Overfitting
3. Vorhersage aus Mehrheitsabstimmung aller Bäume

Kategoriale Variablen mit Random Forest
Möglich, kann jedoch komplex werden bei vielen Faktorstufen
One-hot-encoding bei Random Forest nicht anwenden!

+ / -

+ Robust gegenüber Ausr.
+ Benchmark Methode
+ Out-of-Bag Fehlerrate
+ Umgang mit tausend. Eingangsvar.
+ neigt nicht zu Overfitting
+ kann mit unbalancierten Daten umgehen
+ Auf Reg.probleme anwendbar

R-Code / Parameter

```
library(caret)
fitC <- trainControl(method = "oob")
r.rf <- train(Species ~., data=iris, method= "rf" , trControl = fitC,
              tuneLength=3, ntree=1000) #ntree Anzahl Bäume
r.rf$results
r.rf$finalModel$confusion #Überprüfung Out of Bag Fehler

Mit Ranger: library(caret), library(ranger)
rangerOut <- ranger(formula = survived ~ . , data=trainDat,
num.trees =  1000, mtry = 2, respect.unordered.factors = "order")
              #respect.unordered.factors = "order" = target encoding
survPred <- predict(rangerOut, data = testDat)$predictions
confusionMatrix(survPred, testDat$survived)

Vergleich verschiedene Modelle:
results <-resamples(list(LDA=model_lda, QDA=model_qda,
              knn=model_knn, RandomForest = model_rf))
summary(results)

Visualisierung:
bwplot(results) #boxplots
dotplot(results) #dot plots
```

One-hot-encoding: Erhöht Dimensionalität des Merkmalsraums (kategoriale Variable wird in mehrere Variablen aufgeteilt), kann z.B. bei k-NN nützlich sein

Target encoding: Jede Stufe eines kategorialen Merkmals wird durch durchschnittliche Antwort in Zielvariable ersetzt, nützlich in Datenreichem Kontext, Achtung: Mittelwerte können verzerrt werden

Umgang mit unbalancierten Daten
Alternativer Cutoff wählen & wenn W'keit grösser als dieser Cutoff als Churner klassifizieren.
- Betrachtung ROC-Kurve (siehe unten)
- Anpassung Prior-Wahrscheinlichkeit (siehe unten)

Auch möglich durch Sampling Methoden

AdaBoost

Binärer Klassifikator
- Kombination von schw. Lerner (Bäume häufig nur 1 Stufe)
- Nicht alle Modelle gleich gut. Bessere Modelle stärker gewichtet
- Jedes Modell berücksichtigt Fehler vom vorgängigen stärker

Totaler Fehler des Modell 1
$err_1 = \sum_{i=1}^{N} w_i \cdot I(y_i \neq M_1(x_i)) = \frac{1}{8}$

Gewicht im Ensemble von Modell 1
$\alpha_1 = \frac{1}{2}\log(\frac{1-err_1}{err_1}) = 0.97$

Probleme wenn Fehler 0 oder 1: Addieren eines kleinen Fehlerterms

Beispiele:
$w_{2_3} = \frac{1}{8} \cdot \exp(0.97) = 0.33$
$w_{2_1} = \frac{1}{8} \cdot \exp(-0.97) = 0.05$

Gewichtung für nächsten Durchlauf, noch unskaliert (muss noch normalisiert werden)

```
library(caret), library(adabag)
r.Ada <- train(Churn ~., data =churn_train, method="AdaBoost.M1")
confusionMatrix(predict(r.Ada, newdata=churn_test),
              reference=churn_train$Churn)
```
→ Funktioniert nur bei 2 Klassen, Hyperparameter-Tuning kann sehr lange dauern

Boosting Allgmein:
Anderst als bei Bagging: Die verschiedenen homogenen Klassifizierer werden nicht mehr unabhängig voneinander sondern sequenziell trainiert.
Grösstes Gewicht auf Beobachtungen, welche im vorherigen Modell falsch vorhergesagt wurden.
→ Unterschiedliche Gewichtung der Modelle im Ensemble
Hauptaugenmerk auf Reduzierung des Bias → weniger rechenaufwendig

Stochastic Gradient Boosting

GBM

Schätzen der Zielvariable mit schwachem Algorithmus (Regression, Klassifikation)
Vorhersage schrittweise anhand der Residuen verbessern

Vorgehen **Gradient Boosting:**
1. Initialisierung Vorhersagemodells: $\hat{y} = F_0(x)$
2. Von m = 0 bis M
   a) Berechnung Residuen: $r = y - F_m(x)$
   b) Modellieren der Residuen: $\hat{r} = f_m(x)$
   c) Neues Modell: $F_{m+1}(x) = F_m(x) + \lambda f_m(x)$

M: Anzahl Bäume (1000 – 10000)
λ: Lernrate (kleine Rate: reduziert Effekt der einzelnen Bäume, verbessert aber Vorhersage auf lange Sicht. Kleinere Werte liefern bessere Performance – sind jedoch rechenintensiver, Empfehlung 0.01 – 0.001)
Allgemeine Gefahr von Overfitting bei Gradient Boosting

Vorgehen **Stochastic Gradient Boosting:**
1. Initialisierung Vorhersagemodells: $\hat{y} = F_0(x)$
2. Von m = 0 bis M
   a) Berechnung Residuen: $r = y - F_m(x)$
   zufälliges Ziehen der Residuen z.B. 50%
   b) Modellieren der Residuen: $\hat{r} = f_m(x)$
   c) Neues Modell: $F_{m+1}(x) = F_m(x) + \lambda f_m(x)$

➔ Stabilere Optimierung und geringere Gefahr für lokale Optima

Hinweis: Stochastic Gradient Boosting ähnelt dem Gradient Boosting, jedoch wird hier bei jedem Schritt nur eine zufällige Teilmenge der Trainingsdaten verwendet. Dies führt zu einer erhöhten Vielfalt und kann das Modell robuster gegenüber Rauschen und Ausreißern machen.

```
library(caret), library(gbm), modelLookup("gbm")
ctrl <- trainControl(classProbs = TRUE, method="cv", number=5,
              summaryFunction = mnL) #mnLogLoss
gbm_fit <- train(Species ~ . , data = my.train, method = "gbm",
              trControl = ctrl, distribution = "multinomial",
              metric = "logLoss", verbose = FALSE, tuneGrid =
              data.frame(expand.grid(n.trees = c(200, 500, 1000,
              2000),
              interaction.depth = seq(2, 4), shrinkage = 0.01,
              n.minobsinnode = 10))) # Alternative Verteilungen z.B.
              "bernoulli" oder "gaussian"
confusionMatrix(predict(gbm_fit, newdata = my.test),
my.test$Species)
```

Verlustfunktion definiert, was als Fehler zwischen den vorhergesagen Werten $\hat{y}$ und den tatsächlichen Werten y angesehen wird. -> Quantifiziert, wie gut das Modell F(x) sich der Zielvariable nähert.

$$\hat{y} = F_0(x) + \sum_{m=1}^{M} \lambda \cdot f_m(x) = F(x)$$

- Vorhersage falsch: Verlustfunktion hat hohen Wert
- Variabel anwendbar: z.B. Regression oder Klassifikation
- Ziel: Minimierung Verlustfunktion

Regression: MSE oder MAE (Mean absolute error)

Klassifikation: 0-1 Verlust oder Negative Log Likelihood (Cross-Entropie für W'keitsvoraussagen)

Regression MSE

$L(\hat{y}_i) = (y_i - \hat{y}_i)^2$

$\frac{dL}{d\hat{y}_i} = -2(y_i - \hat{y}_i)$

$\frac{dL}{d\hat{y}_i} \propto -r_i$

Klassifikation Log Loss (ohne Herleitung)

$\mathcal{L}(\hat{y}_i) = y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i))$

$\mathcal{L}(\hat{y}_i) = -y_i \log(\widehat{odds}_i) + log(1 + e^{\log(\widehat{odds}_i)})$

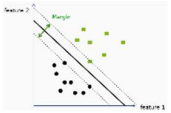
$\frac{d\mathcal{L}}{d\log(\widehat{odds}_i)} = -y_i - \frac{e^{\log(\widehat{odds}_i)}}{1+e^{\log(\widehat{odds}_i)}}$

$\frac{d\mathcal{L}}{d\log(\widehat{odds}_i)} = -y_i - \hat{p}_i = -r_i$

<table>
<tr>
<td>XGBBoost</td>
<td>- Bietet mehr Regularisierungsoptionen, um Modellkomplexität zu reduzieren und verringert so Overfitting
- Berücksichtigt bei Baumentwicklung nur eine reduz. Anzahl Features
- Betrachtet Gradient 2. Ordung -> findet bessere Richtung zur Reduktion Lossfunktion</td>
<td>Vorteile
• sehr leistungsfähig
• flexibel

Nachteil/Gefahren
• viele Hyperparameter
• anfällig auf Overfitting
• Interpretierbarkeit
XGBoost ist eine Optimierung des Gradient Boosting-Algorithmus</td>
<td>library(caret), library(xgboost), modelLookup("xgbTree")
ctrl <- trainControl(classProbs = TRUE, method = "none",
summaryFunction = mnLogLoss)
xgb_fit <- train(Species ~ . , data =my.train, method = "xgbTree",
trControl = ctrl, metric="logLoss", verbose=FALSE,
tuneGrid = data.frame(nrounds = 50, max_depth = 3,
eta = 0.3, gamma = 0, colsample_bytree = 0.6,
min_child_weight = 1, subsample = 0.5)) # one-hot encoding, nrounds: Anz. Bäume, max_depth: Baumtiefe, eta: Lernrate, gamma: Regularisierung Baumkomplexität, colsample_bytree: Wie viel % der Daten pro Baum, min_child_weight: Mind. Beob., subsample: % der zufälligen Beobachtungen
confusionMatrix(predict(xgb_fit, newdata=my.test), my.test$Species)</td>
<td></td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>Flexibles Klassifikationsverfarhen mit oft sehr guter Performance – gut geeignet bei vielen Features oder limitierter Anzahl Beobachtungen

- Jede Beobachtung wird als p-dimensonaler Vektor dargestellt
- SVM konstruiert Hyperebene, um 2 Klassen zu trennen

 
Maximale Margin Klassifikator wählt Ebene so, dass der Abstand von der Hyperebene zum nächsten Trainingspunkt maximal ist.

Support Vectors: Trainingsdaten die am nächsten bei der Hyperebene liegen
(Abstand zu Hyperebene: ≤ M), alle anderen Daten tragen nicht zur Festlegung der Grenze bei

Optimierungsproblem: $y_i \cdot (\beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_p \cdot x_{ip}) \geq M$</td>
<td>Oftmals nicht eindeutig trennbare Daten:
Fehlklassifizierung erlauben:
$y_i \cdot (\beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_p \cdot x_{ip}) \geq M \cdot (1 - \epsilon_i)$
$\epsilon_i \leq 1/C$ → Tuningparameter C darf nicht negativ sein.
Grössere Margin erlaubt mehr Fehlklassifikationen, da geringere Kosten
Bei grossen Kosten, kleinere Margin

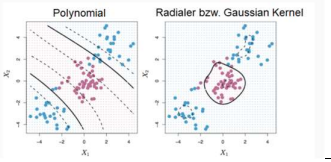
Bei kleinen Margin: Grenzen anfällig falls leichte Veränderungen in Trainingsdaten, Klassifikator mit grosser Variabilität und kleinem Bias, Gefahr von Overfitting</td>
<td>library(e1071), modelLookup("svmLinear2")
Daten als factor definieren
ctrl <- trainControl(method="repeatedcv", number=10, repeats=3)
model_sva <- train(y~., data=dat, method= "svmLinear2" ,
trControl=ctrl,
tuneGrid=data.frame(cost=c(0.01, 0.1, 1)))
model_sva$bestTune

pred <- predict(model_sva, testdat)
confusionMatrix(data =pred , reference= testdat$y)

SVM: Mehr als 2 Klassen
→ Binärer Klassifikator: Kann nur 2 Klassen trennen

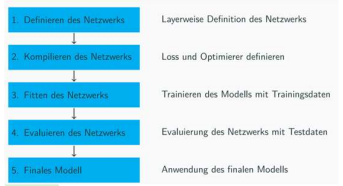
Mit K Klassen:
OVA (One versus All): Fit K verschiedene SVM Modelle: Jeweils Klasse k gegen den Rest. Neue Beobachtung mit der grössten Distanz zur Hyperebene klassifizieren.
OVO (One versus One): Fit alle paarweise SVM-Modelle. Neue Beobachtung in die Klasse klassifizieren, welche die meisten paarweisen Vergeleiche gewinnt.</td>
<td>Vorteile
• Effektive Trennung von Datenpunkte in Klassen
• Robust gegenüber Overfitting
• Mit OVA oder OVO Ansätze könnte geclustert werden.

Nachteile:
• Sensibel gegenüber Skalierung
• Schwierigkeit bei grossen Datensätzen</td>
</tr>
<tr>
<td>Support Vector Machine nicht linear (Gaussian Kernel)</td>
<td>Kernel Trick → SVM
Polynome manuell hinzufügen, schnell schwierig – daher ist die bessere Lösung Kernels. (Funktion, welche Ähnlichkeit zwischen 2 Beobachtungen quantifiziert.)</td>
<td>Vorteile:
• Effektiv bei Trennen von nichtlinearen Entscheidungsgrenzen
• Robust gegenüber Overfitting
• Effektive Verarbeitung von hochdim. Daten

Nachteile:
• Empfindlichkeit gegenüber Skalierung und Hyperparameterwahl
• Schwierigkeit bei grossen Datensätzen</td>
<td>ctrl <- trainControl(method="repeatedcv", number=10, repeats=3)
svm_rad <- train(y~., data=dat, method="svmRadial", trControl=ctrl
tuneGrid=expand.grid(sigma=seq(0.5,5,0.5),
C=c(0.01, 0.1, 0.5, 1, 1.5)))
svm_rad$bestTune
confusionMatrix(svm_rad, "none")</td>
<td>Ansatz über Skalarprodukt:
Allgemeine schreibweise mit Kernel K: $f(x) = \beta_0 + \sum_{i \in S}^{n} \hat{\alpha}_i K(x, x_i)$
▪ Linearer Kernel: $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$
▪ Polynomieller Kernel d-ten Grades $K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$
▪ Gaussian bzw. Radialer Kernel $K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$
Beispiele:
  </td>
</tr>
<tr>
<td>Deep-Learning-Ansätze

CNN Keras</td>
<td>Deep-Neuronale Netze starten mit Rohdaten und lernen selbstständig während des Trainings, geeignete hierarchische Merkmale zu extrahieren, um diese für die Klassifizierung zu verwenden (End-to-End-Ansatz, bisher Extrahieren von handgefertigten Merkmalen und trainieren eines Modells)
Ziel: mathematische Funktionen zu finden die die Attribute auf die Ausgabe abbilden kann

Neuronale Netze - Trainieren des Netzes:
- Gewichte des Netzwerkes sind die Parameter des Modells
- Anpassung der Gewichte durch Optimierung der Loss Funktion mit Gradientenverfahren → Iteratives Verfahren
- Zufällige Auswahl der Gewichte für die erste Runde
- Forward Pass: Input-Werte werden ins Netz gespiesen und durch das Netz propagiert
- Backward Pass: Jedes Gewicht und jeder Bias Term wir ein kleines Stückchen in Richtung angepasst, die den Fehler kleiner macht (Gradientverfahren)
- Grösste Anpassung über Lernrate (Hyperparameter), hat einen sehr grossen Einfluss
Um Overfitting zu vermeiden, ein Teil der Neuronen während dem Training abspalten</td>
<td></td>
<td>Netzwerk Architekturen
R-Code: Bilden eines CNNs in keras
model_cnn <- keras_model_sequential()
model_cnn %>%
layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = 'relu',
input_shape = c(28, 28, 1)) %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_dropout(rate = 0.25) %>%
layer_conv_2d(filters = 64, kernel_size = c(3, 3),
activation = 'relu') %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_dropout(rate = 0.25) %>% layer_flatten() %>%
layer_dense(units = 64, activation = 'relu') %>% layer_dropout(
rate = 0.5) %>%
layer_dense(units = 10, activation = 'softmax')
summary(model_cnn)

Deep Learning braucht nicht viel Preprocessing, aber Standardisierung der Daten hilft oft. → !! Aktivierungsfunktion muss zum Wertebereich der Daten passen.</td>
<td>Keras - Workflow:
 
R-Code: library(keras3), keras3::install_keras()
1. Definieren des Netzwerkes
model <- keras_model_sequential() layer_dense(model, units = 16,
input_shape = c(4), activation = 'relu') # input_shape: nur im 1. Layer notwendig, activation: Aktiverungsfunk., rate: Anteil auszulassende Gewichte
layer_dropout(model, rate = 0.2)
layer_dense(model, units = 16, activation = 'relu')
layer_dropout(model, rate = 0.2)
layer_dense(model, units = 3, activation = 'softmax')
summary(model)</td>
</tr>
</table>

Netzwerk Architekturen
Fully Connectet Neurale Netzwerke (FCNN): für tabellarische Daten, Eingabereihenfolge spielt keine Rolle, geordnete Information geht verloren

Convolutional Neural Networks (CNNs): Für Bilder, geteilte gewichte über das ganze Bild (nicht für jedes Pixel einzelne Gewichte)

Data Augmentation kann helfen um Datenmenge zu erhöhen → leicht modifizierte Kopien von bereits vorhandenen Daten erstellt.

Limitierte Daten / Computerressourcen / Zeit:
- Vortrainierte Modelle, nur letzter Layer wird neutrainiert (Transfer Learning, oder in parallelen Modell Side Learning)
- Klassischer Klassifikator (SVM) zur Klassifikation anwenden

Bilderkennung: Vortrainierte Modelle anwenden

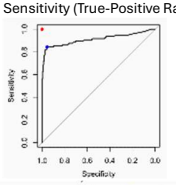
```
# 2. Kompilieren des Netzwerkes
model <- compile(model, optimizer = optimizer_adam(learning_
            rate = 0.001), loss = 'categorical_crossentropy',
            metrics = c('accuracy')) Preprocessing der Daten

# 3. Fitten des Netzwerkes
history <- fit(model, x=train_scale, y=train_labels, epochs = 100,
            batch_size = 32, validation_split = 0.2, verbose = 0)
plot(history)

# 4. Evaluieren des Netzwerkes
valuate(model, x=test, y=test_labels, verbose=0)

# 5. Finales Modell
predict(model, test[1:10, ])
```

ROC / AUC

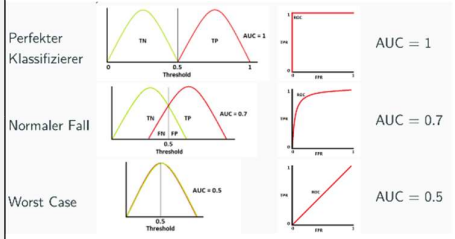
Güte

LogLoss

Sensitivity (True-Positive Rate (tpr)) gegen 1-Specificity (False-Positive-Rate (fpr)) für verschiedene Cut-Off-Werte



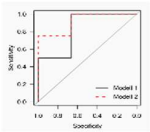
Modell mit perfekter Trennung geht durch Punkt(1,1)
Möglicher Cutoffwert nächster Punkt auf Kurve
→ ROC-Kurve zeigt, dass Modellvergleiche auf Basis der Accuracy irreführend sein können, da bessere Cutoff-Werte möglich sind.



Modellevaluierung mit AUC
Umwandlung von W'keiten zu Klassen → Informationsverlust
Modellbewertung anhand der Fläche unter der Kurve (AUC)

Effektiv bei unbalancierten Klassen, berücksichtigt Rangfolge, nicht empfindlich gegenüber Absoluten Werten → kann irreführend sein

Veränderung folgender Parameter in predict:




```
pred_prob <- predict(rf_fit,
newdata=churn_test, type="prob")
pred_alt <-
as.factor(pred_prob[,1]>0.2)
```

R-Code: **ROC & AUC** library(pROC)
```
r_roc1 <- roc(response = Modell1$obs, predictor= Modell1$A) # ROC
r_roc1$auc # AUC
r_roc2 <- roc(Modell2$obs, Modell2$A, , levels=c("B", "A")) # Modell 2
r_roc2$auc
plot(r_roc1, las=1) # Plot → Variable 1
lines(y=r_roc2$sensitivities, x=r_roc2$specificities, # → Variable 2
            col="red", lwd=2, lty=2)
legend("bottomright", legend=c("M1", "M2"), lwd=2,
            lty=c(1,2), col=c("black", "red"), bty="n")
```

Oder direkt in Train:
```
ctrl <- trainControl(classProbs = TRUE, method = "cv",
            number = 10, summaryFunction =
            multiClassSummary)
iris.knn <- train(y = iris[train_index, 'Species'],x =
            iris[train_index, -which(names(iris) ==
            'Species')], method="knn",
            trControl = ctrl, tuneLength = 3,
            metric = "AUC")
```

Moellevaluierung mit Log-Loss
Metrik, welche die Güte der Kalibrierung misst (auch Cross-Entropie)
$LogLoss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$,
N = Anz. Beob., M = Anz. Klassen, y_{ij}: 1 wenn Beobachtung zu Kl. j gehört sonst 0
→ Gibt an wie nahe die Vorhersagewahrscheinlichkeit an der tatsächlichen Klasse liegt. Je grösser die Abweichung, desto höher der Log-Loss-Wert → Datenspezifisch
➔ Sinnvoll für W'keitsschätzungen, nützlich für Optimierung, Anfällig für Ausreisser

R-Code: library(caret)
```
-sum(log(Mod1$A)*(Mod1$obs =="A")+log(M1$B)*(M1$obs=="B"))/7
caret::mnLogLoss(Modell1, lev = levels(Modell1$obs))
```

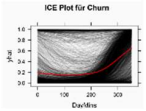
oder direkt in Train:
Siehe oben AUC: Parameter metric = "logLoss"

Güte einer Wahrscheinlichkeit
Kalibrierungskurve: Grafische Beurteilung, ob eine Modell gut kalibriert ist → Wenn vorhergesagte W'keit mit der tatsächlichen Häufigkeit der Ereignisse übereinstimmt, es wird mit den MW gearbeitet

R-Code: library(predtools)
```
calibration_plot(data, obs = 'y', pred = 'prob') # Standard 10 Gruppen
```
➔ Kalibrierung ist perfekt, wenn alle Punkte auf der Winkelhalbierenden liegen, oberhalb = Modell überschätzt W'keiten, unterhalb = Modell unterschätzt W'keiten

VIP / ICE

Einfluss einer Variable auf Klassifizierung
Ind. Conditional Expectation (ICE) Plot / Partial Dependence Plot (PDP)



Für jede Beobachtung werden Vorhersagen entlang des Wertebereichs der Variable erstellt.
x-Achse: Wertebereich
y-Achse: Veränderung W'keit für die Zielklasse (Churn) rote Linie: Mittelwert

→ Vorsicht Wechselwirkungen werden nicht berücksichtigt

Wichtigkeit einer Variable
- Modellspezifische Ansätze:
  Direkt mit dem Modell verknüpft
  Nicht für alle Algorithmen verfügbar
  Nicht über verschiedene Algorithmen vergleichbar
- Modellunabhängige Ansätze:
  Flexible & für alle Klassifikationsalgorithmen verfügbar z.B. Permutationsansatz von RF (eine Variable weglassen, je grösser die Abnahme der Vorhersagegenauigkeit, desto grösser der Einfluss der Variable)

→ Wenn Modell schlecht, Variable Importance nicht aussagekr.

Modellspezifischer Ansatz mit RF
```
iris_rf <- train(Species ~., data=iris, method="rf", ...,
importance = TRUE)
varImp(iris_rf, scale = FALSE) # für Wichtigkeit einer Variable
```

Modellunabhängiger Ansatz: library(vip)
```
r_vi <- vi(q, train = churn_test, target = "Churn",  #q = train(...)
            method = "permute", metric="accuracy",
            pred_wrapper = predict, nsim = 3) # nsim
Anzahl Wiederholungen (Zufallsprozess)
vip(r_vi, geom = "boxplot")
```

ICE-Plots, library(pdp)
```
r_par <- partial(q_fit, pred.var = "Variable", type="classification",
            which.class="TRUE", prob=TRUE, ice = TRUE) # prob: bei
FALSE: logit-Werte, ice: ICE-Kurven, ansonsten nur PDP Kurve
plotPartial(r_par, main="ICE Plot für Churn")
```

ICE-Plot kann auch zur Best. der Wichtigkeit angewandt werden:
Flache Kurve = Variable nicht wichtig, library(vip)
```
res_vi <- vi(q_fit, train=churn_train, method = "firm", target = "Churn",
            metric = "accuracy", pred_wrapper = predict , ice = TRUE)
vip(res_vi)
```

Sampling Methoden

Prior-W'keit

Feature Engin.

Fehl. Werte

Sampling Methoden
Prior Sampling: Falls Dysbalance bereits zu beginn bekannt, kann die Datenaufnahme angepasst werden

Up-Sampling: Ziehen aus der kleineren Klasse bis Klassen ausgeglichen sind (SMOTE)

Down-Sampling: Beobachtungen aus Mehrheitsklasse löschen / auf Anzahl der Minderheitsklasse anpassen (!Verlust von Daten)
→Erst nach dem Aufteilen in Trainings- und Testdaten durchführen

R-Code: Down-Sampling bei Random Forest
```
ctrl <- trainControl(method = "oob")
rf_fit <- train(x = x,y = y, data = dat, method = "rf", ..., strata=dat$y,
            sampsize=c(rep(min(table(dat$y)), nlevels(dat$y))))
data_down <- downSample(x=diabetes[train, ],
            y=diabetes$diabetes[train], list=FALSE)
```

Anpassung Prior-Wahrscheinlichkeit
Bei einigen Klassifizierer kann eine Prior-Wahrscheinlichkeit eingestellt werden: Naive Bayes-Klassifikator / Diskriminanzanalyse
Standardmässig wird diese aus der Verteilung in den Daten bestimmt, kann aber auch angepasst werden: Dafür muss dem train Modell folgender Parameter mitgegeben werden:
```
lda_fit <- train(Churn ~., method = "lda", prior=c(0.5,0.5), ...)
```

PCA & Eigenwerte / Eigenwertvektoren:
Rotationsmatrix A kann durch Eigenvektoren und Kovarianz-Matrix bestimmt werden. Zugehörige Eigenwerte entsprechen den Varianzen der einzelnen Hauptkomponenten

Feature Engineering
Umwandlung von Rohdaten in Features (Datenverständnis notwendig)
→ Oftmals grösseren Einfluss als Algorithmus

Textdaten
- Tokenisierung
- Stoppwörter (uninformative Wörter) entfernen
- Stemming / Lemmatisierung (Wörter in Stammform bringen)
- Document-Term Matrix (Tabelle mit Häufigkeiten der Wörter)
- Bag-of-Words (Vektor mit Anz. Beob. für jedes Wort in Wortliste)

Bilddaten
- Deep Learning Methoden
- Features aus Bildern extrahieren (Features sind Vektoren)

Umgang mit fehlenden Werten
- Missing completely at Random (MCAR):
  Kein Zusammenhang zwischen den fehlenden Daten und den restlichen Werten (zufällige Teilmenge)
- Missing at Random(MAR):
  Fehlende Werte haben einen Zusammenhang mit den beobachteten Werten, aber nicht mit den fehlenden Werten selbst
- Missing not at Random (MNAR):
  Beziehung zwischen fehlendem Wert und dessen Wert

Ansätze:
Complete Case Analysis (MCAR, MAR – bei MNAR !! → Bias):
- Entfernen von Beobachtungen (Zeilen)
  train(...., na.action = na.omit)
- Wenn z.B. > 60 % eines Features fehlt evtl. die ganze Spalte entfernen

		Einschub: Eigenwerte und Eigenvektoren $M \cdot x = \lambda \cdot x \quad \begin{bmatrix} m_{11} & \cdots & m_{1p} \\ \vdots & \ddots & \vdots \\ m_{p1} & \cdots & m_{pp} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$ Eigenvektor x einer quadratischen Matrix ist ein vom Nullvektor verschiedener Vektor, dessen Richtung durch Multiplikation mit der Matrix nicht verändert wird. Eigenvektor wird nur gestreckt. Streckungsfaktor λ ist Eigenwert der Matrix.		Imputation von fehlenden Daten (vervollständigen): - Einsetzen eines konstanten Wertes der sich von anderen Werten unterscheidet (z.B. neue Faktorstufe) - Einsetzen von Mittelwert, Median oder Moduswert (oftmals nicht empfehlenswert) - Schätzen des Wertes mit prädiktivem Modell - Mehrfache Imputation R-Code: library(caret) knn_imp <- preProcess(iris_miss, method= "knnImpute" / "bagImpute" / "medianImpute") *# danach predict(knn_imp, data_missing)* Multiple imputation mit library(mice)

Falsch	**In Spezialfällen richtig**	**Richtig**
Um das 90% Quantil von hochdimensionalen Daten zu berechnen, kann man die Beobachtungen der Grösse nach ordnen und dann bestimmt die Beobachtung das 90% Quantil, für die 90% aller anderen Beobachtungen kleiner-gleich sind	Wenn alle Variablen numerisch sind, dann macht es keinen Sinn, für eine PCA die Variablen zu skalieren	Der Jaccard Koeffizient der beiden Punkten A=(0, 1, 0, 0) und B=(0, 1, 0, 0) ist 1
Bei einer 10-fold cross-validation, sind 10% der Daten im Testdatensatz.	Bei einem Lasso-Modell werden die Koeffizienten erwartungstreu geschätzt.	Wenn zwei Beobachtungseinheiten bei einem Klassifikationsbaum in derselben Partition des Feature Raums landen, dann haben sie die gleichen Feature Werte
Die Support Vector Machine ist ein probabilistisches Klassifikationsmodell.		Die PCA besteht aus einer Verschiebung und einer Rotation des Koordinatensystems.
Um einen Clustering-Algorithmus anwenden, braucht man eine Zielvariable.		Eine Dimensionsreduktion mit linearer MDS und euklidischen Distanzen führt zum selben Ergebnis wie eine klassische PCA
Bei einer PCA kann die zweite Dimension/Hauptkomponente mehr Varianz erklären als die erste Dimension.		Ist bei t-SNE der perplexity Parameter zu hoch, können Strukturen sichtbar werden, die in den Daten gar nicht vorhanden sind.
Die einzelnen Hauptkomponenten einer PCA können korreliert sein		PAM hat im Gegensatz zu K-Means immer eine Beobachtung als Center für jedes Cluster.
Eine MDS kann mit einer Korrelationsmatrix als Input umgehen		Hierarchische Clustering-Algorithmen benötigen eine Distanzmatrix, um verschachtelte Cluster zu bilden.
Clustering mit K-Means findet immer eine globale Lösung (d.h. globales Minimum).		Bei einer Kreuzvalidierung würden wir, wenn die Rechenleistung keine Rolle spielen würde, immer eine Leave-One-Out Kreuzvalidierung machen.
Wenn ein Punkt nach dem Clustering im Silhouetten-Plot einen negativen Silhouettenkoeffizienten aufweist, heisst das, dass dieser Punkt ein Ausreisser ist und keinem Cluster zugewiesen werden kann.		Ein Vorteil von Ensemble-Verfahren aus schwächeren Lernern ist, dass die Varianz im Ensemble deutlich kleiner wird, als sie es von den einzelnen Lernern war.
Ein knn-Klassifizierer wird dasselbe Ergebnis liefern, egal ob die Daten skaliert oder unskaliert verwendet werden.		
Je komplexer ein Klassifikationsalgorithmus ist, umso kleiner wird sowohl der Trainingsfehler als auch der Testfehler.		