Unüberwachtes Lernen: Dimensionsreduktion

Zweck Dim.reduktion

- Visualisierung wichtig. Bei hochdim. Daten Visualisierung nicht möglich → Verfahren, um Zusammenhänge/Strukturen zu sehen.
- Preprocessing für weitere Analysen (bspw. für überwachtes Lernen)
- Ausreiser detektieren
- Hilft irrelevante Variablen zu eliminieren oder Noise zu reduzieren

Quantifizierung

- **Quantifizierung** Ähnlichkeit zwischen zwei Objekten mit Ähnlichkeits-/Distanzmass.
- **Ähnlichkeitsmass s:** quantifiziert Übereinstimmung. Je grösser Wert, desto grösser Ähnlichkeit.
- **Distanzmass d:** (z.B. euklid. Dist zwischen Punkten). Je kleiner Wert, desto grösser Ähnlichkeit. → Grundlagen für MDS, T-SNE, UMAP und Clustering

Distanzmass d d <- dist(dat)

- O_1, O_2 sind 2 Objekte. Distanz O_1 zu O_2 ist $\mathbb{R}$ Zahl $d(O_1, O_2)$
- Distanzmass $d(O_1, O_2)$ → reellwertige Distanzfunktion
- **Eigenschaften**
  - $d(O_1, O_2) \geq 0$ nicht-negative Funktion
  - $d(O_1, O_1) = 0,\ d(O_2, O_2) = 0$ Distanz zu sich selbst ist 0
  - $d(O_1, O_2) = d(O_2, O_1)$ symmetrisch
- Metrische **Distanzen (Metrik)** erfüllen zusätzlich:
  - $d(O_1, O_2) = 0$ genau dann, wenn $O_1 = O_2$
  - $d(O_1, O_2) \leq d(O_1, O_3) + d(O_3, O_2)$ Delta-Ungleichung

Distanzmasse für binäre Daten

- Balancierte Variable: Simple-Matching/Canberra-Metrik symm
- Unbalancierte Variable: Jaccard Koeffizient/Metrik → asymm

Euklidische Dist.

- L2-Metrik → euclidean
- Bei Beobachtungen im Raum gehen wir intuitiv euklid. Dist. aus.
- Euklid. Dist. 2 Beobachtungen o_i, o_j, mit p numerisch Features: $d_{ij}(o_i, o_j) = \sqrt{\sum_{k=1}^{p}(o_{ik} - o_{jk})^2}$

Distanzmatrix D

- Paarweise Distanz in symmetrischer Matrix.
- Diagonale sind alle 0.

Manhattan Distanz

- L1-Metrik → manhattan
- Ein Block ist eine Einheit.
- Wie viele Blocks muss man von O_1 nach O_2 zurücklegen.
- Manhattan Metrik: $d_{ij}(o_i, o_j) = \sum_{k=1}^{n} |o_{ik} - o_{jk}|$
- Manhattan für hochdimensionale Daten besser, da grosse Abstände nicht quadrierten werden.

Minkowski Distanz (Allgm)

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

Kategoriell

- Objekte haben p nominale Merkmale: Sneath-Übereinstimmungskoeffizient $s = \#Übereinstimmung/p$ → factor
- Ordinale Kategorien mit K Stufen: Transformieren $z_{o_i} = \frac{x_{o_i} - 1}{K - 1} \in [0,1]$ ordratio

Mix Variablentyp

- Distanzmass gower

library(cluster); dist <- daisy(**dat**, metric = "gower", stand = F, type = list(asymm = 3, ordratio = c(7,8))); str(d) #metric = Distanzsmass, stand = Standardisierung, type = Behandlung Variable Nr. X als

Manifold-Hypothese: Reale hochdimensionale Daten liegen auf einer niedrigdimensionalen Mannigfaltigkeiten, die in den hochdimensionalen Raum eingebettet sind.

M	Idee/Ansatz	Voraussetzungen/Vorgehen	Vorteil (+) / Nachteil (-)	R-Code [Default]

Hauptkomponenten Analyse PCA	• Dimensionsreduktion und Visualisierung hochdim. Daten
	• Methode zur Detektion von Ausreissern
	• **Idee:** Finde diejenigen Achsen, welche die multidimensionalen Daten am besten erklären.
	• Drehung zentrierten Originalvariablen in Hauptkomponentensystem, so dass diese unkorreliert sind und absteigender Weise möglichst grossen Teil Totalvarianz abdecken.
	• **Ziel:** Dimensionsreduktion: Verwenden nur die ersten paar Hauptkomponenten und hoffen, dass die Daten dadurch möglichst gut beschrieben werden
	• **Faustregel:** ~ 80 % der Var_{total} soll durch ersten PC's beschrieben werden (sonst zu wenig aussagekräftig)
		• **Daten**matrix
		• Nummerische Variablen
		Alternative:
		• Standardisierung meistens mit mean() und sd()
		• Häufig wäre es besser/sicherer Standardisierung robust durchzuführen. Statt Mittelwert/Standardabweichung robuste Masse Median/MAD nehmen. → Ausreisser in einzelnen Variablen werden besser sichtbar
		• Standardisierung muss für prcomp() manuell durchgeführt werden und PCA anschliessend mit manuell standardisierten Daten durchgeführt werden prcomp(**dat_robust**, scale = F).
			+ Reduziert Dimensionen
			+ keine Information gehen mit PCA verloren!
			– Nur für nummerische Variablen
			– Varianzen bzw. Korrelationen sind sensitiv bezüglich Ausreisser → Ergebnis der PCA kann von Ausreissern stark verfälscht werden!
			– Oftmals genügen die ersten zwei bis drei Hauptkomponenten nicht, um genügend der Var_{total} zu beschreiben (mindestens 80 %).
				pca <- prcomp(**dat**, scale = T/F)
				summary(pca)
				plot(PC2 ~ PC1, data = pca$x, xlab = "PC1", ylab = "PC2", col = **dat$Class**)
				
				var <- pca$sdev^2 # 80 % Faustregel
				var_cum <- cumsum(var)/sum(var)
				plot(1:length(var), var_cum, las = 1)
				abline(h = 0.8, col = "red", lty = 2)
				(pc_number <- min(which(var_cum > 0.8)))
				
				#Alternative
				library(ggfortify); biplot(pca)
				autoplot(pca, colour = **1:3[cancer$Level]**)

Robuste PCA	• PCA mit robusten Schätzern (robuste Kovarianzmatrix)
	• Ziel Detektion von Ausreissern
	○ Ausreisser PCA-Raum: Grosse Distanz zum Ursprung der PCA → Score-Distanz $SD_i = \sqrt{\sum_{j=1}^{k} \frac{y_{ij}^2}{\lambda_j}}$
	○ Orthogonal Ausreisser: nicht richtig erfasst in PCA Rm. Orthogonale Distanz $OD_i = \left\|x_i - \hat{\mu} - P \cdot y_i^T\right\|$
		• **Daten**matrix
		• Nummerische Variablen
			+ Reduziert Dimensionen
			+ Nicht ausreiseranfällig
			
			– Nur für nummerische Variablen
				library("rrcov") #k = Ausgabe Dimension
				rob <- PcaHubert(**dat**, k=2, scale=T)
				plot(rob@scores[,1], rob@scores[,2], col = c("red", "blue")[as.factor(**card$Class**)], pch = 20, xlab = "PC1", ylab = "PC2")
				plot(x = rob@od); abline(h = rob@cutoff.od)

Multi-Dimensional Scaling (MDS)	• Verwendet Distanzen zwischen Beobachtungen und erzeugt Projektion der Punkte in 2D.
	• Wir haben Distanzmatrix von vielen Beobachtungen → Visualisierung in 2-(3)-Dimensionen
	Klassische metrische MDS: Minimierung Kostenfunktion $Cost = \sum_{i<j}(d_{ij} - d_{ij}^*)^2$
	• Weitere MDS-Versionen: Ordinale oder nicht-metrische MDS isoMDS(): versucht low-dimensionale Repräsentation der Ränge v. Distanzen zu finden. Cost-Funktion verlangt explizit keine euklidischen Distanzen.
		• **Distanz**matrix
		• Geht für alle Variablentypen (kategorielle → gower)
		• Gibt metrische und nicht-metrische MDS
		
		PCA und metrische MDS sind äquivalent, wenn **euklidische** Distanzen verwendet wurden.
			+ Geht für kategorielle Variablen
			+ Benötigt nur Distanzmatrix
			+ Metrische MDS ist (meistens) auch okay für nicht-euklidische Distanzen.
			
			– isoMDS() kann nicht mit Duplikaten (also Distanz 0) umgehen
				# Metrische MDS
				cmd <- cmdscale(**dist**, k = 2)
				plot(cmd, pch = "") #k=Ausgabe Dimension
				text(cmd, labels = rownames(cmd))
				# Nicht-metrische MDS
				library(MASS); library(cluster)
				d <- as.matrix(daisy(**dat**, metric = "gower"))
				iso <- isoMDS(**d**, k = 2) #k = Ausgabe Dim.
				plot(iso$points, pch = ""); text(iso$points, labels = rownames(iso$points))

Problem mit PCA / metrischem MDS: Gibt fast keinen Grund, dass Daten wirklich in Hyperebene liegen sollten. → Deshalb neue Ansätze (t-SNE/UMAP): Lokale Distanzen korrekt wiedergegeben.

M	Idee/Ansatz	Voraussetzungen/Vorgehen	Vorteil (+) / Nachteil (-)	R-Code [Default]
Stochastic Neighbour Embedding (t-SNE) von Laurens van der Maaten (2008)	• Versucht Nachbarschaftsbeziehungen im hochdimensionalen Raum zu erhalten und niedriger Dim. darzustellen. • Berechnung euklid. Abstand $x_i, x_j: \|x_i - x_j\|^2 = \sqrt{(x_i^x - x_j^x)^2 + (x_i^y - x_j^y)^2}$ • Dichtewert der Distanz $\|x_i - x_j\|^2 = p_{x_j\|x_i} = p_{j\|i}$ unskalierte Ähnlichkeit von x_j zu x_i betrachtet von x_i • Skalieren Ähnlichkeitsmasse $p_{j\|i}$ durch Dividieren der Summe aller Ähnlichkeitsmasse: $p_{j\|i} = \frac{p(x_j\|x_i)}{\sum_{k\neq i} p(x_j\|x_i)}$, somit gilt $\sum_j p_{j\|i} = 1$ (wobei $p(i\|i) = 0$) • Zwei Punkte haben zueinander unterschiedliche bedingte Wahrscheinlichkeiten $p_{i\|j}, p_{j\|i}$ (stammen aus zwei versch. Verteilungen) → $p_{ij} = \frac{p(j\|i)+p(i\|j)}{2n}$, $n = \#$ Dimensionen • Bedingte Wahrscheinlichkeit: $p_{j\|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$ • σ_i ist abhängig vom Punkt x_i und basiert auf Dichte in Nachbarschaft vom Punkt. • Varianz wird indirekt über Wahl der Perplexität (Hyperparameter, vereinfacht = Anzahl Nachbarschaftspunkte) definiert: Perplexity $= 2^{H(P_i)} = 2^{-\sum_j p_{j\|i}\log_2 p_{j\|i}}$ (mit steigender Perplexity nimmt σ zu) • Entropie $H(P_i) = -\sum_j p_{j\|i}\log_2 p_{j\|i} = \sum_j p_{j\|i}\log_2\left(\frac{1}{p_{j\|i}}\right)$ Mass für die erwartete Überraschung (Werte zwischen 0 (keine Überraschung) und $\log_2(\#\,Punkte)$ maximale Überraschung). • $\log_2\left(\frac{1}{p_{j\|i}}\right)$ = Mass für Überraschung, wenn $p_{j\|i}$ gross → wenig Überraschung σ^2 eher klein → Range für Anzahl lokaler Nachbarn eher klein → kleine Perplexity • Wird im niedrig-dimensionalen Raum Gaussverteilung als Ähnlichkeitsmass verwendet, gibt es **Crowding-Problem** (durch Dimensionsreduktion können nicht mehr alle Strukturen/Distanzen korrekt wiedergegeben werden). • t-SNE löst Problem, indem Ähnlichkeiten q_{ij} über t-Verteilung (df = 1) berechnen: $q_{j\|i} = \frac{\left(1+\|y_i - y_j\|^2\right)^{-1}}{\sum_{k\neq i}\left(1+\|y_i - y_j\|^2\right)^{-1}}$	• **Daten- oder Distanzmatrix** • Kategoriale Variablen können nur via gower() Distanzmatrix der Funktion Rtsne() übergeben werden. Datenmatrix geht nur für numerische Variablen. **Algorithmus** • Initialisierung stoch. Testen: set.seed() • Berechnung **euklid.** Abstand zwischen allen Punkten • Berechnung Dichtewert Distanz $p_{j\|i}$ • Skalieren Ähnlichkeitsmasse, $p_{ij} = \frac{p(j\|i)+p(i\|j)}{2n}$ • Berechnung bedingte Wahrscheinlichkeiten • Erstellung niedrig-dimensionaler Raum mit gleichen Punkteanzahl wie im ursprünglichen Raum, welche zufällig verteilt werden. • Berechnung Wahrscheinlichkeiten q_{ij} (Ähnlichkeiten) im niedrig-dimensionalen Raum (Bsp. 2D). Punkte mit Optimierungsverfahren so anpassen, dass sie möglichst nah an bedingte Wahr'keit p_{ij} im hoch-dimensionalen Raum kommen. Wird durch Minimieren der Kullback-Leibler divergence Verlustfunktion getätigt. • Algorithmus mit verschiedenen Seeds durchführen. Zufällige Initialisierung der Optimierung kann bei verschiedenen Durchläufen zu unterschiedlichen Lösungen (lokalen Optima) führen. → Mehrfaches Ausführen des Algorithmus (beste Visualisierung; niedrigster Wert für die Zielfunktion itercosts) • Mit Hyperparameter **perplexity** experimenteiern, verschiedene Werte nehmen: ○ **Je mehr und dichter Daten, desto höher der Wert → Faustregel $\sqrt{N}$** ○ Es gilt: $3 \cdot perplexity < nrow(X) - 1$ • **Hyperparameter Anzahl Iteration (max_int) sollte ausreichend gross gewählt werden für stabile Konfiguration** • Algorithmus passt Abstände an lokale Unterschiede in Dichte an. So werden dichte Cluster vergrössert und breite Cluster verkleinert.	+ Lokale Nachbarschaften bleiben eher erhalten – kann Clusterung liefern, welche eigentlich gar nicht existiert – Anwendung auf neue Punkte standradmässig nicht möglich – Clustergrössen bedeuten nichts – Abstände bedeuten möglicherweise nichts – Abstände zwischen Clustern können nicht interpretiert werden – Dimensionsreduktion mit t-SNE als Input für ein Clusterverfahren ist gefährlich, da das Clusterverfahren die Struktur von t-SNE stark übernimmt.	library(Rtsne) set.seed(**7**) tsne <- Rtsne(**dat**, dims = 2, **is_distance = F**, perplexity = **30**, max_iter = 1000) dims: Ausgabedimension [2] pca: Angabe, ob zuerst PCA auf Daten angewendet werden soll [TRUE] pca_center → Datenzentrierung vor PCA [TRUE] pca_scale: Skalierung Daten vor PCA [FALSE] initial_dims: Anzahl Dimensionen, die von PCA verwendet werden [50] is_distance: Indikator, ob X Distanzmatrix ist [FALSE] check_duplicates: Überprüfung, ob Duplikate vorhanden [TRUE] theta: Geschwindigkeit/Genauigkeitskompromiss [0.5] 0.5 schnelle Implementierung; 0 exakte Berechnung perplexity: Hyperparameter; entspricht ungefähr Anzahl erwarteter Nachbarn [30] → Faustregel $\sqrt{N}$: $(5{,}30), (5{,}50)$ max_iter: Hyperparameter; Anzahl der Iterationen [1000] plot_data <- as.data.frame(tsne$Y) plot_data <- cbind(plot_data, **Meta23**) ggplot(plot_data, aes(x = V1, y = V2, label = **Name**, colour = **Fraktion**)) **Dimensionsreduktion als Input für ein Clusterverfahren** library(Rtsne); set.seed(60) tsne <- Rtsne(**dat**, perplexity = **100**, max_int = 5000) d_tsne <- as.data.frame(tsne$Y) names(d_tsne) <- c("tsne1","tsne2") d_tsne$label <- as.factor(**cancer$Level**) library(dbscan) (res.dbscan <- **dbscan**(tsne$Y, eps = 3, minPts = 25)) library(ggplot2) ggplot(data = d_tsne, aes(x = tsne1, y = tsne2, col = **res.dbscan$cluster**, label = label)) + geom_text(size=3)

M	Idee/Ansatz	Voraussetzungen/Vorgehen	Vorteil (+) / Nachteil (-)	R-Code [Default]

Uniform Manifold Approximation and Projection (UMAP)	• Initialisierung fix (spektrales Embedding): Zufälligkeit durch stochastischer Gradientenabstieg (random_state)
	
	Hauptunterschiede UMAP zu t-SNE:
	• Abstand im hochdimensionalen Raum wird über Exponentialverteilung bestimmt, (ohne Normalisierung)
	• Mittelung Ähnlichkeiten in beide Richtungen: $p_{ij} = p_{i\|j} + p_{j\|i} - p_{i\|j}p_{j\|i}$
	• Der Parameter n_neighbors (Anzahl nächsten Nachbarn) steuert, wie UMAP zwischen lokaler und globaler Struktur abwägt. Bei niedrigen Werten konzentriert sich UMAP mehr auf lokale Struktur, bei hohen Werten auf Struktur Gesamtbild. Ähnlich wie bei perplexity in t-SNE.
	• Initialisierung der Punkte in niedrigen Dimension erfolgt nicht zufällig, sondern über spektrale Einbettung.
	• Für Ähnlichkeit in niedrigen Dimension wird eine Verteilung ähnlich der t-Verteilung verwendet (gesteuert über den Parameter min_dist → bei niedrigen Werten werden die Punkte dichter zusammengepackt).
	• Kostenfunktion mit binärer Kreuzentropie anstelle der Kulback-Leibler-Divergenz
	
	• Hyperparameter n_neighbors und min_dist wichtig → testen versch. Konfigurationen!
		• **Daten- oder Distanzmatrix**
		• Kategoriale Variablen können nur via gower() Distanzmatrix der Funktion umap() übergeben werden. Datenmatrix geht nur für numerische Variablen.
		
		Ähnlichkeit hochdimensionalen Raum
		• Berechnung Abstand zwischen Punkten in hoher Dim.
		• Für jeden Punkt wird Abstand unter exponentiell abfallenden Kurve aufgetragen.
		• Kurvenform hängt von Anzahl Nachbarn (einschließlich des Punktes selbst) ab. → $\log_2(\#\ neighbors)$ entspricht Summe unskalierten Ähnlichkeiten (Dichte).
		• Ähnlichkeiten werden für alle Punktkombinationen auf gleiche Weise erzeugt.
		
		UMAP Konzept
		• Nach Initialisierung (Spektral Embedding) werden Punkte so verschoben, dass Ähnlichkeiten in niedrigen Dimension mit Ähnlichkeiten in hohen Dimension übereinstimmen.
		• Die Ähnlichkeit in der niedrigen Dimension wird anhand einer Kurve ähnlich der t-Verteilung bestimmt.
		• Bei der Anpassung wird die Kostenfunktion (binäre Kreuzentropie) minimiert.
		• Hierfür wird stochastische gradient descent Algorithmus verwendet. Dieser hat Zufallskomponente, weshalb mehrere Durchläufe des Algorithmus, trotz eindeutiger Initialisierung, zu **unterschiedlichen Ergebnissen** führen können.
			Vorteile gegenüber t-SNE
			+ Anwendung auf neue Punkte möglich (wenn Datenmatrix)
			+ Dimensionsreduktion auch möglich für mehr als 2-3 Dimensionen
			+ UMAP kann auf Rohdaten ohne PCA-Vorverarbeitung ausgeführt werden. Die PCA kann aber auch hier hilfreich sein.
			+ Anwendung direkt möglich auf grosse Datensätze (performanter, schneller als t-SNE)
			+ Grösserer Fokus auf globale Struktur (Abstände zwischen Gruppen teilweise interpretierbar) als bei t-SNE
			+ Cluster oft besser getrennt
			
			– Wie t-SNE kann auch UMAP falsch interpretiert werden.
			– Clustergrößen in Diagramm haben keine Bedeutung (lokale Abstände sind entscheidend)
			– Abstände zwischen Clustern haben möglicherweise keine Bedeutung und sind nicht immer sinnvoll
				library(umap)
				res_umap <- umap(d = **iris**, n_neighbors = 25, min_dist = 0.05, **input = "dist"**, random_state = 8)
				plot(res_umap$layout, col = **iris$Species**)
				
				n_neighbors: Anzahl nächsten Nachbarn [15]
				min_dist: Abstand Punkte in Visualisierung [0.1]
				input: Datenmatrix "data" oder Distanzmatrix "dist" (**as.matrix(dist)**)
				random_state: Seed für Realisierung
				
				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) +
				scale_color_manual(values = c("green", "yellow", "orange", "blue", "red", "darkgreen"))
				
				#Dimensionsreduktion als Input
				#für ein Clusterverfahren
				library(umap)
				res.umap <- umap(d=**x[, -1]**, n_neighbors = 20, min_dist = 0.05, n_components= 3)
				
				dim(res.umap$layout)
				
				library(mclust)
				mc <- Mclust(**res.umap$layout**, G = 10:20)

Hilfsmittel/Tools für Dimensionsreduktion / Clustering

M	Idee/Ansatz	Voraussetzungen/Vorgehen	R-Code [Default]
Skalierung	**Skalierung ändert Ergebnis** • Beispiel PCA: Bei grossen Unterschieden in absoluten Varianz wird unskalierten Daten Richtung v. ersten Hauptkomponenten durch Variable mit grosser Varianz bestimmt. • Oftmals lohnt es sich eine skalierte Version gegenüber einer unskalierten Version durchzuführen (bspw. bei einer Dimensionsreduktion). Dabei ist auf die Unterschiede in den anschliessenden Resultate zu achten. Welchen Einfluss hat die Skalierung?	**Faustregeln: Skaliere,** • Variablen verschiedene Einheiten haben (z.B. kg, m) • wenn explizit gewollt ist, dass alle Variablen das gleiche Gewicht haben. **Nicht skalieren,** • wenn alle Variablen die gleiche Einheit haben und vergleichbar sind. • Wenn man nicht weiss, was tun, ist es meistens besser standardisieren.	scale(dat)

Unüberwachtes Lernen: Clustering (Beschreibende Methode)

Ziel	Typische Schritte einer Clusteranalyse	Unterscheidung Clusterverfahren
• Set von Objekten in homogene Gruppen/Cluster unterteilen. Objekte im gleichen Cluster sollen untereinander ähnlicher sein als zu Objekten in anderen Clustern. • Um Clusteranalyse durchzuführen, muss ein Ähnlichkeits-/Distanzmass, für Objektevergleich, definiert werden.	• Schritt 1: Explorative Datenanalyse und Datenbereinigung • Schritt 2: Skalierung Daten • Schritt 3: Wahl geeigneter Clusteralgorithmus (verschiedene Ansätze testen) • Schritt 4: Evaluierung geeigneten Anzahl Cluster • Schritt 5: Analyse der resultierenden Cluster	• **Partitionsverfahren** (Raum unterteilen): Unterteilung der Objekte in sich nicht überschneidende Cluster, so dass jedes Objekt in genau einem Cluster ist. • **Hierarchisches Clustern:** Cluster werden innerhalb von grösseren Clustern unterteilt und bilden Baum von Cluster

	Idee/Ansatz	Voraussetzungen/Vorgehen	Vorteil (+) / Nachteil (-)	R-Code [Default]

K-Means / WCV (within-cluster variation)	• Unterteilen der Beobachtungen in k homogene Cluster, so dass die totale Variation innerhalb der Cluster (WCV within-cluster variation) über alle K Cluster, so klein wie möglich ist.
	• $\min_{C_1,\dots,C_k}\left(\sum_{k=1}^{K} WCV(C_k)\right)$
	• WCV für einen Cluster k C_k, $WCV(C_k) = = \frac{1}{\#C_k}\cdot\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2$
	• Verw. quadrierte euklidische Distanzen
	• $\#C_k$ Anz. Beobachtungen im Cluster k
	• p Anz. Features (Dimension)
	• Minimierung WCV nicht trivial $\frac{1}{K!}\sum_{k=1}^{K}(-1)^{K-k}\binom{K}{k}k^N$
		• **Daten**matrix
		
		• Simpler guter Algorithmus (liefert lokale Lösung):
		1. Beobachtungen zufälliger Gruppe zuordnen
		2. Für jeden Cluster wird das Zentrum bestimmt.
		3. Alle Beobachtungen werden Clusterzentrum zugeordnet, zudem sie kleinste Distanz haben.
		○ Wiederholen Schritte 2-3 bis es keine Änderungen mehr gibt.
		○ **Lösung von Startkonfiguration abhängig.**
		○ Workaround: Algorithmus mehrmals durchführen und Simulation mit tiefstem WCV verwenden.
		
		• Algorithmus für div. k durchführen
		• WCV bestimmen und ihn vs. k auftragen. k nach letzten grossen Abfall bestimmen (Ellbogen-Methode) → Anzahl Cluster
		• Festlegen Anz. Clustern: Verschiedene Ansätze: Ellbogen-, Silhouetten-Methode, Domain knowledge, etc. → verschiedene Ansätze können zu versch. Anz. Cluster führen
			+ Schnell und einfach
			
			– Clusteranz. muss vorgegeben werden
			– Probleme mit **verrauschte** Daten/**Ausreisser**
			– **Schwierigkeiten** mit **hochdim. Daten**, führt zu wenig aussagekräftigen Ergebnissen
			– nicht fähig, Cluster mit **nicht-konvexen Formen** zu erkennen
			– kann in **lok**. **Minima** hängen bleiben
			– benötigt Datenmatrix
			– kann **nur** mit **euklidischen Distanzen** umgehen (andere nicht möglich).
			– Tendenz, «**kugelförmige**» Gruppen mit **gleich vielen Objekten** zu bilden.
			– Erkennt **längliche Ausdehnung/verschiedene Streuungen** in Gruppen schlecht
			– bei **Ausreissern** können komische Effekte auftreten, falls nicht für jeden Ausreisser (oder für Anhäufung von Ausreissern) ein separates Cluster gebildet wird.
			– Lösung von Startkonfiguration abhängig
				set.seed(**17**) #verschiedene seeds testen
				cl <- kmeans(**dat**, centers = 3)
				plot(**x**, col = cl$cluster, las = 1)
				points(cl$centers, ch = 2, cex = 2) #Zentrum
				cl$withinss #WCV Wert pro Klasse
				
				#Visualisierung bei mehr als 2 Dimensionen: 2D-Plot mit PCA, MDS, t-SNE/UMAP
				
				# Bestimmung k
				k <- **20**; wcv <- rep(0, k)
				for (i in 1:k)
				wcv[i] <- sum(kmeans(**dat**, centers = i)$withinss)
				
				plot(1:k, wcv, type = "b", xlab = "Anzahl Cluster", ylab = "WCV") #Ellbogen-Plot
				
				# Alternative
				library(factoextra)
				fviz_nbclust(**dat**, kmeans, method = "wss", k.max = 10)

K-medoids / PAM	• PAM (Partitioning Around Medoids)
	• Sehr ähnlich zu k-means
	• Finden repräsentative Objekte, genannt Medoids, in Clustern
	• Ausgehend von einer Distanzmatrix wird über alle Objekte die Gesamtdistanz der Objekte zu den Objekten im gleichen Cluster minimiert $\min\left(\sum_{i=1}^{n}\sum_{i=j}^{n} d(i,j)\cdot z_{ij}\right)$
	• WCV-Ansatz funktioniert nicht, da Kriterium nicht monoton abnehmend
	• Max. durchschnittliche Silhouettenbreite kann auch hier bestimmt werden
		• **Distanz**matrix
		
		Algorithmus
		1. k Objekte werden zufällig als Clusterzentren gewählt
		2. Jedes Objekt wird dem nächsten Clusterzentrum zugewiesen
		3. Kostenfunktion wird evaluiert
		4. Iterativ wird für jeden Cluster untersucht, ob Medoid Vertauschen 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.
			Vorteile gegenüber K-Means
			+ robuster gegenüber Ausreissern
			+ Distanzmatrix
			+ geht mit allen Distanzmassen
			+ Medoids als repräsentative Objekte v. Cluster (hilft zu interpretieren)
			
			– Clusteranz. muss vorgegeben werden
			– Bestimmung über Gütekriterium (Ellbogen-Methode bei K-Means) nicht möglich, da dieses nicht monoton abnimmt (dafür kann dies aber mit Silhouettenbreite bestimmt werden)
			– langsam für grosse Daten (→ clara())
				library(cluster); cl_pam <- pam(**dist**, **k = 3**)
				cmd <- cmdscale(**dist**, k = 2) #für Plot
				plot(cmd, col = cl_pam$clustering)
				points(cl_pam$medoids, pch = 8, cex = 2)
				cl_pam$medoids #Medoid
				
				k <- **20**; sil <- rep(0, k)
				for (i in 2:k){
				sil[i] <- summary(silhouette(x = pam(**dist**, k = i)$cluster, dist = **dist**))$avg.width}
				plot(x = 1:k, sil, type = "b", xlab = "Anz Cluster", ylab = "mittlere Silhouettenbreite")
				abline(v = which.max(sil))
				
				# Alternative (braucht Datenmatrix!):
				library(factoextra)
				fviz_nbclust(**dat**, pam, method = "silhouette")
				
				# Oder auch direkt mit library(fpc)
				cl_pamk <- pamk(**dat**, krange = 2:5)

Silhouetten als Qualitätskontrolle

- Silhouetten-Koeffizient einer Beobachtung i: $sil_i = \frac{b_i - a_i}{\max(a_i, b_i)} \in [-1,1]$
- a_i: mittlere Distanz zwischen Punkt i und allen anderen Punkten aus gleichen Cluster
- b_i: mittl. Distanz zwischen i und allen Punkten nächstgelegenen Nachbarcluster
- $b_i \gg a_i \rightarrow$ benachbarter Cluster weit weg $sil_i \rightarrow 1$
- $b_i \approx a_i \rightarrow$ benachbarter Cluster nahe $sil_i \rightarrow 0$

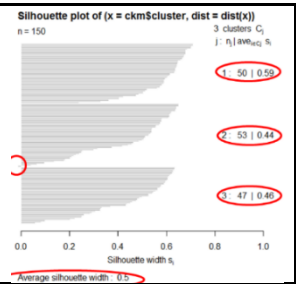
→ Mittl. Silhouettenbreite kann auch Wahl optimalen Anzahl Cluster verwenden

Faustregel für mittlere Breite im Cluster

- $0.70 < sil < 1.00$: Gute Struktur wurde gefunden
- $0.50 < sil < 0.70$: vernünftige Strukturen
- $0.25 < sil < 0.50$: Schwache Strukturen → weitere Analyse
- $-1 < sil < 0.25$: Forget it!

Was bedeutet negativer Silhouettenkoeffizient?

- Beobachtung ist im Mittel näher bei den Beobachtungen im benachbarten Cluster als zu denen im Eigenen.
- Clustering muss nicht zwingend schlecht sein.



R-Code

K-Means

```
cl <- kmeans(dat, centers = 3)
plot(silhouette(x = cl$cluster, dist = dist(dat)))
```

PAM

```
cl_pam <- pam(dist, k = 3)
plot(silhouette(x = cl_pam$clustering, dist = dist))
```

Hierarchische Clusteranalyse

Idee/Ansatz

- Ein Set von verschachtelten Clustern, welche in einem hierarchischen Baum organisiert werden können.
- Anzahl Dendrogramme mit n Blättern: $(2n-3)!/[(2(n-2))(n-2)!]$
- Können nicht alle Möglichkeiten testen → Brauchen Heuristik
- Top-Down (divisiv):
  - Start: alle Objekte in **einem** Cluster
  - Betrachte jede Möglichkeit die Daten in 2 Cluster zu unterteilen
  - Wähle die Beste aus
  - Repetieren für beide Cluster, usw. bis alle Objekte einen eigenen Cluster bilden.
- Bottom-Up (agglomerativ):
  - Start: alle Objekte als **eigene** Cluster
  - Finde das beste Paar zum verbinden
  - Repetieren bis alle Cluster zu einem vereint sind ⇐ übliche Form

Heatmap

- Visualisierung einer Datenmatrix
- blaue Werte → kleine Werte
- rote Werte → grosse Werte
- Permutation der Zeilen und Spalten, so dass ähnliche Objekte nahe sind
- Erlaubt zu verstehen, in welchen Merkmal sich Cluster unterscheiden
- Farbbalken erlauben das Überprüfen von Hypothese über Assoziationen zwischen Clustern und externen kategorialen Variablen, die beim Clustering nicht verwendet wurden.

Voraussetzungen/Vorgehen

- **Distanz**matrix

Algorithmus, Start: Distanzmatrix

1. Jedes einzelne Objekt wird als Gruppe betrachtet. → n Gruppen
2. Suche das kleinste Element in der Distanzmatrix. Entsprechende Elemente werden zu einer Gruppe verbunden und aus der Distanzmatrix entfernt.
3. Berechne die Distanzen von der neuen Gruppe zu allen anderen.
4. Wiederhole Schritte 2 und 3 bis nur noch eine Gruppe übrigbleibt.

- Problem: Brauchen Verallgemeinerung, um Distanzen zwischen Clustern bestimmen. → Linkage

Distanzen zwischen Clustern: Linkage

- **Single-Linkage:** kleinste Distanz zwischen den zwei Clustern (kleinste Distanz zwischen 2 Objekten) → single
- **Complete-Linkage:** grösste Distanz zwischen 2 Objekten → complete
- **Average-Linkage:** mittlere Distanz zwischen allen Objekten → average
- **Ward:** versucht die Varianz der verbunden Cluster zu minimieren (hierarchische Version von K-Means) → ward.D2
- Clusterergebnis hängt von der Datenstruktur, dem Distanzmass und der Linkage-Methode ab.
- **Allgemein: Clustering ist exploratives Tool!**

→ Wählen Linkage Methode, welche «beste» Resultat produziert!

Vorteil (+) / Nachteil (-)

+Verschiedene Distanzen möglich (euklidisch, Manhattan, Gower, ...)

Nachteile

–

Dendrogramm

- Position des Verbindungsknotens auf Abstandsskala gibt Abstand zwischen Clustern an (abhängig von Verknüpfungsmethode).
- Wenn zwei Cluster auf einer Höhe von bspw. 22 verbunden wurden, bedeutet das, dass der Abstand zwischen diesen Clustern 22 beträgt.
- Wollen feststellen, wo Abstand zwischen den Clustern, die kombiniert werden, gross ist, um Cluster zu unterteilen. Wir vergleichen die Abstände zwischen sequenziellen Verbindungsknoten (hier vertikale Linien).
- Achtung! Interpretation der Abstände zwischen den Clustern nicht zulässig.
- Um sinnvolle Cluster zu finden, wird das Dendrogramm bei einem Schwellwert geschnitten. Dazu betrachtet man die Differenzen der Cluster-Distanzen zwischen den Aufsplitterungen.
- Üblicherweise sind Differenzen zuunterst am kleinsten und oben oft gross.
- Dazwischen gibt es meistens eine Zone, wo sich Differenzen plötzlich stark verkleinern. Oberhalb davon ist geeigneter Ort, um den Schwellwert zu setzten.

Erhalt Schwellenwert:

```
hc$height[length(hc$height):1]
diff(hc$height)[(length(hc$height)-1):1]
```

R-Code [Default]

```
hc <- hclust(dist(dat), method = "ward.D2"); plot(hc, las = 1)

#In k = 3 Cluster unterteilen/einzeichnen
rect.hclust(hc, k = 3, border = "red") #Schwellenwert sieh +/-
grhc <- cutree(hc, k = 3) #Einteilung Beobachtung in Cluster
grhc <- cutree(hc, h = 5) #Einteilung mit Höhe des Schnittes

# Plotten der Cluster mit MDS
cmd <- cmdscale(dist); plot(cmd, xlab = "", ylab = "",
      col = c( "green", "blue", "red")[cutree(hc, h = 8)])
text(cmd, labels = rownames(cmd),
      col = c("green", "blue", "red")[cutree(hc, h = 8)])

plot(silhouette(x = grhc, dist = dist))

k <- 20; sil <- rep(0, k)
for (i in 2:k){
  sil[i] <- summary(silhouette(x = cutree(hc, k = i),
                               dist = dist))$avg.width}
plot(x = 1:k, sil, type = "b", xlab = "Anz Cluster", ylab = "mittlere Silhouettenbreite"); abline(v = which.max(sil))

#Alternative: Funktion agnes aus cluster
```

 # Heatmap

```
dat_num <- dat[,3:8]
x <- t(as.matrix(dat_num))
colnames(x) <- dat$Car
heatmap(x) # Default
# distfun = dist -> euclidean,
# hclustfun = hclust -> linkage = complete,
# scale = "row" -> Zeilen standardisiert (mean = 0, sd =1)

# Elegante Alternative
library(pheatmap)
pheatmap(x, scale = "row", angle_col = 45)

# Ergänzen mit Meta-Daten
an <- data.frame(Country = dat$Country)
rownames(an) <- colnames(x)
pheatmap(x, scale = "row", annotation_col = an)
```

Partitionierungsmethoden (K-Means, PAM-Clustering) und hierarchisches Clustering funktionieren gut bei kompakten und gut getrennten Clustern. Mit komplexeren Daten und insbesondere Rauschen und Ausreissern in den Daten können sie nur schlecht umgehen. Insbesondere Ausreisser sind für Daten typisch. → Alternative DBSCAN - Density-Based Spatial Clustering of Applications with Noise

Modellbasiertes Clustering

Idee/Ansatz

- Idee: Daten stammen aus Mischverteilung, wobei Daten aus jedem Cluster aus anderen Verteilung stammen.
- Punkte können mit gewissen Wahrscheinlichkeit einem Cluster zugeordnet werden.
- Statistisches Modell für Beschreibung Clusterform. Populärste Annahme multivariate Normalverteilung für Cluster (für stetige Daten)
- Multivariate Normalverteilung (Verallgemeinerung der Normalverteilung auf mehrere Dimensionen)
- Bestimmt durch Verteilungsparameter:
  - Erwartungswertvektor $\mu(\mu_{X1}, \mu_{X2}, \dots)$
  - Kovarianzmatrix Σ

Expectation-Maximization (EM-Algorithmus):

- Gesucht n Cluster aus multivariaten Normaldichten mit $E(X) = \mu_j$ und Kovarianzmatrix Σ_j $(j = 1, \dots, n)$
- Clustergrössen in Anteilen $\pi_1, \pi_2, \dots, \pi_n$ (wobei $\Sigma_j(\pi_j) = 1$)
- Parameterschätzung (μ_j, Σ_j, π_j) aus Daten
- Maximierung Log-Likelihood mit iterativem EM-Algorithmus

Da Clustering auf Modell basiert, kann Unsicherheit für einzelnen Beobachtungen analysiert werden.

Voraussetzungen/Vorgehen

- **Daten**matrix

Start Algorithmus

- Zufällige Startparameter (μ_j, Σ_j, π_j)
- **E-Step**
  - r_{ic}: W'keit, dass Punkt x_i zu Cluster c gehört
  - Berechnung $Likelihood \cdot Prior$ für Model c und Normalisieren der Gewichte über alle Cluster.
  - Je wahrscheinlicher x_i zum c-ten Cluster gehört, desto grösser r_{ic}
- **M-Step**
  - Für jeden Cluster c werden Parameter anhand gewichteten Datenpunkte aktualisiert
  - $m_c = \Sigma_i(r_{ic}), \quad \pi_c = \frac{m_c}{m}, \quad \mu = \frac{1}{m_c}\Sigma_i\left(r_{ic}x^{(i)}\right)$
  - $\sigma_c = \frac{1}{m_c}\Sigma_i r_{ic}\left(x^i - \mu_c\right)^T\left(x^i - \mu_c\right)$

Rechenaufwand

- Kovarianzmatrix jedes Clusters hat Dim. $p \times p$. Falls p gross (oder n) müssen sehr viele Parameter geschätzt werden. Wird instabil → Vereinfachung durch Restriktion Clustermodelle
- einfachste Restriktion $\Sigma_j = \sigma^2 \boldsymbol{I}$, für $j = 1, \dots, n$ **I** Einheitsmatrix; σ^2 Varianz → alle Cluster sind sphärisch (kugelförmig) mit gleichem Radius

Bestes Modell?

- Optimales Modell und optimale Anzahl Cluster kann über maximales BIC (Bayes'sches Informationskriterium) ermittelt werden.
- $BIC_{mclust} = 2 \cdot I_{M_c}\left(x_1, \dots, x_n \middle| \hat{\theta}_{mle}\right) - v \cdot \log(n)$
- I_{M_c}: Log-Likelihood des Modells M_c mit Max-Likeli-Schätzung (MLE) geschätzten Param $\hat{\theta}_{mle} = \left(\hat{\mu}_{mle}, \hat{\Sigma}_{mle}\right)$
- v: Anz. zu schätzender Parameter
- n: Anzahl Datenpunkte
- Mass für Effizienz des Modells
- Modellkomplexität (Parameteranz.) wird bestraft

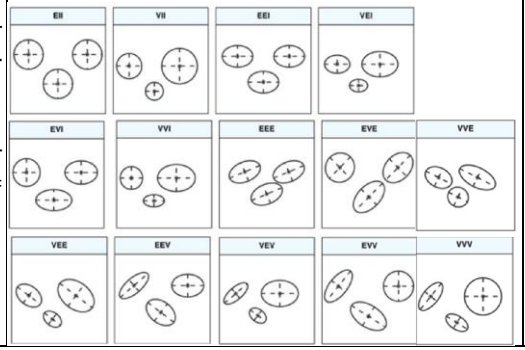
Vorteil (+) / Nachteil (-)

+ Da Clustering auf Modell basiert, kann Unsicherheit für einzelnen Beobachtungen analysiert werden.
– Benötigt Datenmatrix

Vor- und Nachteile

Verschiedene Modelle

Modell	Verteilung	Grösse	Form	Orientierung
EII	sphärisch	gleich	gleich	—
VII	sphärisch	variable	gleich	—
EEI	diagonal	gleich	gleich	Achsen
VEI	diagonal	variable	gleich	Achsen
EVI	diagonal	gleich	variable	Achsen
VVI	diagonal	variable	variable	Achsen
EEE	elliptisch	gleich	gleich	gleich
EVE	elliptisch	gleich	variable	gleich
VEE	elliptisch	variable	gleich	gleich
VVE	elliptisch	variable	variable	gleich
EEV	elliptisch	gleich	gleich	variable
VEV	elliptisch	variable	gleich	variable
EVV	elliptisch	gleich	variable	variable
VVV	elliptisch	variable	variable	variable



R-Code [Default]

```
library(mclust)
# spezifisches Modell
mc <- Mclust(dat, modelNames = "EII")
mc <- Mclust(dat) # alle Modelle
mc$modelName # bestes Modell

# Visualisierung BIC aller Modelle
plot(mc, what = "BIC")

# Cluster
plot(mc, what = "classification")

# Alternative Visualisierung
library(factoextra)
fviz_mclust(mc, "BIC", palette = "jco")

fviz_mclust(mc, "classification", geom = "point", palette = "jco")

# Unsicherheit
mc$uncertainty

plot(mc, what = "uncertainty")

fviz_mclust(mc, "uncertainty", palette = "jco", xlab = "x1", ylab = "x2")

# → grössere Symbole zeigen unsicherere Beobachtungen
```

	Idee/Ansatz	Voraussetzungen/Vorgehen	Vorteil (+) / Nachteil (-)	R-Code [Default]

Dichtebasiertes Clustering DBSCAN	• **Idee:** Erkennen von Gebieten mit hoher Datenpunktdichte als Cluster.
	• Dichte Datenpunkts wird durch Anzahl der Punkte in seiner Umgebung ϵ geschätzt. ϵ-Umgebung besteht aus allen Punkten, die höchstens den Abstand ϵ haben → Parameter ϵ
	• Mit Parameter minPts wird festgelegt, wie viele Punkte in der ϵ-Umgebung von einem Punkt p liegen müssen (Zentrumspunkt wird dazu gezählt), damit p zu einem Cluster gehört.
	
	Drei Kategorien von Punkten:
	• Kernpunkt: Anzahl Datenpunkte in ϵ-Umgebung beträgt mindestens minPts (inklusive Kernpunkt).
	• Randpunkt: Kein Kernpunkt, liegt aber in ϵ-Umgebung eines Kernpunktes.
	• Rauschpunkt: weder Kern-/Randpunkt.
	→ Cluster: Verbundene Kernpunkte plus dazugehörige Randpunkt.
	
	Bestimmung Parameter
	• **minPts**: min. 3, je grösser Datensatz, desto grösser. Grössere Werte sind bei Datensätzen mit Rauschen und vielen Duplikaten meist besser und ergeben mehr signifikante Cluster. Wie gross sollten die Cluster mindestens sein?
	• **eps**: kann mithilfe k-Nächste-Nachbarn-Graph abgeschätzt werden. Werden mittleren Abstände aller Punkte zu seinen k nächsten Nachbarn berechnet (mit $k = minPts - 1$). Abstände aufsteigender Reihenfolge auftragen. ϵ-Wert dort, wo Plot Ellbogen zeigt. Wenn ϵ zu klein, wird grosser Teil nicht geclustert. Zu hohem Wert verschmelzen Cluster. Kleine ϵ Werte sind vorzuziehen
	→ **Domainknowledge** ist für Parameterbestimmung nützlich.
		• **Distanz**matrix
		
		DBSCAN (Density-Based Spatial Clustering of Applications with Noise) Algorithmus:
		1. Distanzmatrix über alle Punkte.
		2. Zufälliger (noch nicht besuchter) Punkt auswählen.
		3. Benachbarte Punkte mit Distanz ≤ ϵ identifizieren.
		4. Weniger als minPts Punkte in der Nachbarschaft → Rauschpunkt (kann sich später wieder ändern) ⇒ 2. Mindestens minPts Punkte → Cluster wird gestartet.
		5. Punkt → Kernpunkt; bildet mit Punkten in der Nachbarschaft Cluster. Restliche Punkte im Cluster → Randpunkte.
		6. Iteration über alle Randpunkte. Falls in deren Nachbarschaft ebenfalls min. minPts vorhanden → Kernpunkt und Cluster wird erweitert. Prozess bis alle Punkte des Clusters gefunden wurden. ⇒ 2
		7. Ende Algorithmus, wenn alle Punkte besucht. Benachbarte Kernpunkte mit ihren Randpunkten bilden Cluster.
			+ Keine Vorgabe von Clusteranzahl
			+ Filtert Rauschen i.d.R. gut aus Daten
			+ Separate Ausweisung der Ausreisser
			+ Beliebige Form/Grösser der Cluster
			+ Benötigt nur Distanzmatrix
			
			– Schwierigkeiten mit Erkennung von Clustern unterschiedlicher Dichten.
			– Probleme bei hochdimensionalen Daten, da hier Dichte schwierig zu definieren ist (Curse of dimensionality).
			– Bestimmung von ϵ teilweise schwierig; erfordert Domänenwissen.
			– Parameter müssen bestimmt werden
				library(dbscan)
				db <- dbscan(**dist**, eps = 0.15, MinPts = 5)
				
				# Output, Cluster Nr. 0 = Ausreiser
				# 0 1 2 3 4 5
				# 31 410 405 104 99 51
				
				#Mit Dimreduktion 2-D Plot erstellen:
				plot(**df$x**, **df$y**, col = db$cluster + 1)
				
				# Bestimmung Parameter
				library(dbscan)
				
				kNNdistplot(x = **dist, k = 3**)

Dilemma beim Clustern	Analyse der Cluster: Qualitätscheck
• Was ist ein Cluster? • Welche Feature/Merkmale/Variablen sollen verwendet werden? • Sollen die Daten normalisiert werden? • Gibt es Ausreisser in den Daten? Was haben die für einen Effekt? • Wie definieren wir die paarweise Ähnlichkeit? • Wie viele Cluster gibt es in den Daten? • Welche Clustering-Methode sollte verwendet werden? • Sind die entdeckten Cluster valid? ➔ **Alle diese Fragen haben normalerweise keine klare Antwort!**	• Visualisierungen: ○ Dendogramme ○ Heatmap (für numerische Daten; Interpretation der Farben) ○ farbliche Clustervisualisierung in 2D Plot (PCA, MDS, t-SNE, UMAP) • Betrachte Zentrum oder repräsentatives Element (PAM) der Cluster • Silhouette-Plot • Quantifizierung der WCV within-cluster variation • Konfusionsmatrix (benötigt Labels / Ground truth Informationen; Clustern dann weniger interessant) • Analyse Featuresrelevanz → Klassifikation mit Cluster-ID als Label und betrachte Variablen Importance (vgl. Klassifikation) • Domainwissen beiziehen! → generell schwierig in hohen Dimensionen

Überwachtes Lernen: Allgemeines zur Klassifikation

Ziel	Idee	Feature extrahieren	Ausgangslage	Ablauf	Fallstricke
Für Objekte, deren Klassenzugehörigkeit unbekannt ist, aufgrund der Merkmale (Variablen, Features) die richtige Klasse vorherzusagen.	• Trainiere Klassifizierer anhand von Trainingsdaten, bei welchen Klassen bekannt sind. • Verwende Klassifizierer um neue Beobachtung ohne Label zu klassifizieren. • Haben Trainingsdaten und unbekannte Daten respektive Testdaten. • Training/Validierung: Verwenden, um Modell trainieren und Hyperparameter bestimmen. • Testdaten: Zur Evaluation der Performance des finalen Modells	• Wollen geeignete Feature/Merkmale aus Daten extrahieren, was nicht immer einfach ist. • Verschiedene Merkmale können kombiniert werden, um neue Merkmale zu generieren (z.B. PCA, UMAP Embedding). • Feature Engineering	• Haben X und Y gegeben. Y Labels sind kategoriell und X ist Datenmatrix mit diversen Variablen. • In Klassifikation versuchen wir Labels Y anhand Features X vorherzusagen. • In Regression ist Zielvariabel Y numerisch.	1. **Zufälliges Aufteilen** in Training- und Test-Daten (i.d.R. 80 % und 20 %) 2. **Anpassen Modell** auf Trainingsdaten (inkl. Preprocessing & Modellselektion) 3. **Modellevaluation**: Wie gut wird Modell bei neuen, nicht gesehenen Daten performen?	• Kein Trainingsverfahren (Datenaufbereitung gehört dazu) darf Testdaten sehen. Sperren diese bis zum letzten Test weg! Grund: Wenn verschiedene Modelle anhand Testdaten evaluiert, besteht Gefahr, dass man zu optimistisch ist. Rückschlüsse der Modellperformance auf neue Daten können nicht gezogen werden. • Unsymmetrische Daten: Unausgeglichene Datensätze können Ergebnis verzerren (Bsp. binäre Variable, 1 → 90 %, 2 → 10 %)

Aufteilung Datensatz: library(caret); index <- createDataPartition(**dat$y**, p = 0.8, list = F) #80 % Trainingsdaten, Output Index von Trainingsdaten trainDat <- **dat**[index,]; testDat <- **dat**[-index,]

Evaluationsmetrik	Berechnung	Was wird gemessen
Accuracy/Genauigkeit	$\frac{TP+TN}{ntest}$	Standardmetrik, unterschiedliche Kosten für Fehler werden nicht berücksichtigt
Precision/Präzision Pos Pred Value	$\frac{TP}{TP+FP}$	Wie viele vorhergesagten ⊕-Objekte sind richtig? → wollen hohen TP, tiefen FP
Recall/Sensitivität True-Positive-Rate	$\frac{TP}{TP+FN}$	Wie viele ⊕-Objekte wurden richtig vorhergesagt? → wollen hohen TP, tiefen FN
Spezifität True-Negative-Rate	$\frac{TN}{TN+FP}$	Wie viele der ⊖-Objekte wurden richtig vorhergesagt?
F1-Score	$\frac{2 \cdot recall \cdot precision}{recall+precision}$	Harmonisches Mittel v. Recall % Precision
Kappa	$\frac{accuracy - E[accuracy]}{1 - E[accuracy]}$	Relative Verbesserung im Vergleich zu einem zufälligen Prädiktor

$E[accuracy] = \frac{TP+FP}{ntest} \cdot \frac{TP+FN}{ntest} + \frac{TN+FN}{ntest} \cdot \frac{TN+FP}{ntest}, \quad ntest = TP + FP + FN + TN$

Evalutationsmetriken: Je nach Fragestellung kann eine andere Metrik wichtig sein.

Fehlerarten

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

- **Trainingsfehler**, naiver Fehler bzw. In-Sample-Fehler: Fehler an Trainingsdaten evaluiert
- **Verallgemeinerungsfehler**, Testfehler bzw. Out-of-Sample-Fehler: Fehler an neuen, bisher ungesehenen Daten (aus der Stichprobe) evaluiert
- Komplexeres Modell performt bei Trainingsdaten immer besser als einfaches Modell
- **Modelle** sollten anhand von **Out-of-Sample-Daten ausgewertet** werden, um Verallgemeinerungsfehler zu ermitteln.
- Beim Vergleich der Performance an Trainingsdaten sollte Modellkomplexität mitberücksichtigt werden (Strafe für Komplexität).
- Bei zwei Modellen mit ähnlichen Verallgemeinerungsfehlern sollte das einfachere Modell dem komplexeren Modell vorziehen.
- Overfitting Phänomen (Überanpassung): Modell passt gut zu Trainingsdaten (kleiner Trainingsfehler/grosse Genauigkeit), hat aber hohen Testfehler/kleine Genauigkeit.

Modellselektion

- Welches Modell sollten wir wählen?
- Vergleich unterschiedlicher Modelle.
- Oft gibt es einen «Hebel» (Hyperparameter), der die Komplexität/Performance eines Klassifikators steuert.
- Welche Merkmale sollten verwendet werden?
- Soll ein Merkmal transformiert werden (Wurzel, Logarithmus, etc.)?

Mod.validierung

- Wie abschätzen, wie gut Modell mit ungesehenen Daten ist?
  - Wichtig: Ohne Verwendung Testdaten.
  - Modellvergleiche auf Testdatenbasis führen zu überoptimistischen Fehlerschätzungen!

  →Validierung bereits in Trainingsdaten erforderlich

  → Welchen Ansatz? Verwenden (mehrer) 10-fache KV/LOOKV

Einfache Validierung

- Aufteilung Gruppen: Trainings- und Validierung
- – Schätzfehler sind variabel abhängig, welche Beobachtungen in Trainings-/Testmenge sind.
- – Nur Teilmenge Beobachtungen (Training) wird zur Modellanpassung verwendet. Statistische Methoden schneiden schlechter ab, wenn sie auf weniger Beobachtungen trainiert werden.
- – Testfehler für Modellanpassung an gesamten Datensatz wird daher oft überschätzt.

Leave-One-Out KV LOOKV

1. Teilen Daten in n Datensätze auf, wobei Trainingsdaten aus allen Punkten ausser **einem** bestehen.
2. Modell wird einmal mit allen n Datensätzen trainiert und dann am ausgeschlossenen Punkt validiert.
3. Schätzen Testfehler aus Anzahl Fehlklassifizierungen z.B. Anteil Fehlklassierung: $CV_{(n)} = \frac{1}{n}\sum_{i=n}^{n} Err_i, \; Err_i = I(y_i \neq \hat{y}_i)$

+ Nicht zufällig in Aufteilung zwischen Trainings- und Validierungsdaten.
+ Kleinster Bias bei Schätzung Testfehler (Verwendung fast aller Daten zur Schätzung)
– Numerisch komplex/langsam (Modell n-mal ausgeführen)

K-fachen KV – KKV

1. Unterteilen Daten in k sich nicht überschneidende Gruppen.
2. Modell k-mal trainiert für alle Gruppen ausser einer, und jedes Mal mit dieser Gruppe validiert.
3. Schätze Testfehler aus Anz. falsch klassifiziert Beobachtung: $CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} Err_k$

+ Schneller als LOOKV
– Ergebnis weist gewisse Zufälligkeit auf (abhängig Datenaufteilung in Gruppen)
–/+ Etwas grösserer BIAS als LOOKV. I.d.R. unproblematisch und besser, da geringere Varianz als LOOKV.

→**10-fache KV Standard**

Bootstrap-Valid.

- Erzeugt Stichproben aus vorhand. Daten.
- Gleiche Grösse n wie ursprüngliche Stichprobe, ziehen mit Zurücklegen $x = (x_1, \ldots, X_n)$
- Benutze Datensatz als Ersatz für unbekannte zugrundeliegende Verteilung und ziehe daraus Stichproben.
- Stichprobe aus Trainingsdaten.
- Validierung auf Daten, die nicht in Bootstrapstichprobe

Out of Bag

Evaluation Random Forest oob Fehlerrate:

- Bei jedem Baum werden nicht verwendeten Testdaten jeweils zur Evaluation der Fehlerrate verwendet.
- Mitteln über alle Bäume → **KV on the fly!**

R-Code Trainieren [Def]

Für Steuerung der KV trainControl()
ctrl <- trainControl(method = …)

- "LOOCV": LOOKV (~~Parameter~~)
- "cv": reg. *k*-KV (number = ***k***) [10]
- "repeatedcv": rep k-KV (repeats = Anz. Rep. KVs, number = ***k***)
- ["boot"]: (number = Anz Rep) [25]
- "oob": out of Bag RandForest
- "none": nichts

train(x, y, method = "knn", trControl = ctrl, tuneGrid = data.frame(k = 1:10), preProcess = c("center", "scale"))
tuneGrid: Testparameter (alternativ tuneLength)
center: Daten zentrieren
scale: Skalierung mit sd()
pca: PCA mit thresh: Anteil Varianz, pcaComp Anz. PCs

Überwachtes Lernen: Klassifizierer

No-Free-Lunch-Theorem

- Modell ist vereinfachte Version der Realität: Vereinfachung, um überflüssige Details eliminieren und sich auf Aspekte zu konzentrieren, die man verstehen will.
- Vereinfachungen beruhen auf Annahmen; Annahmen können in einigen Situationen zutreffen, in anderen nicht. Modell, das bestimmte Situation gut erklärt, kann in anderen Situation versagen.
- Theorem besagt, dass es kein Modell gibt, das für jedes Problem am besten funktioniert. Annahmen eines Modells für ein Problem gelten möglicherweise nicht für andere Probleme. Beim maschinellen Lernen ist es üblich, mehrere Modelle auszuprobieren, um das zu finden, das am besten funktioniert. ⇒ Darum verschieden Ansätze (Klassifizierer)

	Idee/Ansatz	Voraussetzungen/Vorgehen	Pro (+) / Contra (-)	R-Code [Default]

k-Nächste-Nachbarn KNN	• Ausgangslage: Objekt x_0 mit Klassenlabel unbekannt.
	• Finde die k Trainingsobjekte mit der kleinsten Distanz zu x_0 (eukld. Dist. → Daten ggf. skalieren!)
	• Verwende die Klasse der Mehrheit der k Nachbaren als Klassenlabel für x_0
	Was ist die richtige Modellkomplexität?
	• Wollen Überanpassung vermeiden, aber dennoch ausreichend komplex sein.
	• Ein kleines k führt zu einem komplexen Modell, ein grosses k bedeutet ein einfaches Modell.
	• Bei **zwei Klassen** sollte **immer ungerade Anzahl nächste Nachbarn** wählen, damit eindeutige Entscheidung getroffen wird. Sonst wird in Pattsituation Klassenzugehörigkeit ausgelost. Bei > 2 Klassen ist es nicht immer möglich, Pattsituation mit $k > 1$ zu verhindern
		•
		
		Skalierung der Daten
		• Beim knn-Klassifizierer, so wie bei vielen anderen ML-Ansätzen, spielt es eine Rolle, welche Grössenordnung die einzelnen Features habe.
		• Feature mit grösseren Skala haben mehr Einfluss.
		• Damit alle Feature den gleichen Einfluss haben, müssen die Daten skaliert werden.
		
		Achtung: Bei k = 1 muss die Fehlerrate bei den Trainingsdaten immer 0 sein, da nur das eigene Label die Klasse definiert.
			+
			
			– Ist nur für euklidische Distanzen ausgelegt
			– Ist für Datensätze mit vielen Merkmalen weniger geeignet
				res.knn <- train(y = trainDat$**class**, x = trainDat[, **1:4**], method = "knn", trControl = ctrl, tuneGrid = data.frame(k = **seq(1,15,2)**)) #tuneGrid = Welche k ausprobieren?
				plot(res.knn) #Darstellung Accuracy nach # Nachbarn
				
				confusionMatrix(predict(res.knn, testDat), reference = testDat$**class**)
				
				#Alternative mit Formelschreibweise (FSW)
				res.knn <- train(form = **class** ~., data = trainDat, method = "knn") #FSW führt One-Hot-Codierung durch!
				
				#Alternative ohne caret mit knn Funktion:
				library(class)
				knn(train = trainDat[,-"**label**"], cl = trainDat$**label**, test = newDat, k = k)

Naive Bayes-Klassifikator	Bayes Klassfizierer:
	• Verwendung insbesondere bei Textdaten (NLP)
	• Optimaler Klassifizierer: Beobachtung wird zur wahrscheinlichsten Klasse, gegeben Features, zugewiesen. ⇒ $P(Y = C_I\|X)$ → bedingt Wahr'keit
	• Optimal falls Verteilung bekannt.
	• Bei welchen x-Wert liegt Entscheidungsgrenze? → Dort wo Wahrscheinlichkeit für beide Klassen gleich gross ist. ⇒ Perfekte Klassifikation nicht möglich
	
	Wie sieht Entscheidungsgrenze aus? (p = # Var.)
	• p = 1 → Entscheidungsgrenze: ein Punkt
	• p = 2 → Entscheidungsgrenze: eine Linie
	• p = 3 → Entscheidungsgrenze: eine 2D Ebene
	• p = L → Entscheidungsgrenze: L-1 Hyperebene
	• Likelihood kann im multivariaten Fall komplex sein. Braucht Vereinfachungen. → Naive Bayes
	
	• Problem wenn Häufigkeiten 0 (d.h. eine Kategorie einer Klasse kommt in Trainingsdaten nicht vor). → a posteriori wird 0 → Laplace Korrektur/Smoothing: konstanter Faktor (i.d.R. zwischen 1-2) wird bei Zählern dazu addiert, im Nenner wird Faktor mal Anz. Features dazu addiert.
		• Hauptsächlich für kategorielle Features
		Annahmen:
		• Alle Features sind voneinander unabhängig
		• Alle Features haben gleiches Gewicht.
		• Schätzung Likelihoods für Features über Häufigkeitsverteilung der Klassen in Trainingsdaten
		Theorem von Bayes für Klassifikation
		• $P(Y = C_I\|X) = \frac{P(C_I)\cdot P(X\|Y=C_I)}{P(X)}$
		• $P(C_I)$; a priori-Wahrscheinlichkeit (Anfangswahrscheinlichkeit) für Klasse I. → Bevor Daten sehen, klassifizieren gemäss dieser Wahrscheinlichkeit.
		• $P(Y = C_I\|X)$; beobachten Merkmalswert x und bestimmen damit a posteriori-Wahr'keit.
		• $P(X\|Y = C_I)$ Likelihood; Wahrscheinlichkeit für Beobachtung der Daten gegeben die Klasse.
		• $P(X)$; Randwahrscheinlichkeit für X, Wahrscheinlichkeit für die Prädiktorvariablen. ⇒ Klassifizierung gem. höchster a posteriori-Wahr'keit
		Umgang mit stetigen Variablen
		• Können diskretisiert werden.
		• Je nach Implementierung direkt verwenden:
		○ Annahme normalverteilt (Schätzung Parameter mit mean und var) → Gaussian Naive Bayes
		○ Nichtparametrische Dichteschätzung
			+ Hauptsächlich für kategorielle Features $(x_1, \ldots, x_j)$ ausgelegt
			
			– Annahme Unabhängigkeit sehr stark, aber Ansatz funktioniert häufig überraschend gut.
				ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
				nb <- train(y = trainDat$**Species**, x = trainDat[, **1:4**], method = "naive_bayes", trControl = ctrl, tuneGrid = data.frame(laplace = 0, usekernel = c(FALSE, rep(TRUE, 3)), adjust = c(0, seq(1, 2, 0.5))))
				
				• tuneGrid, data.frame() ermöglicht es unterschiedlichen Features mit unterschiedlichen Schätzern laufen lassen
				• laplace: Faktor für Laplace-Korrektur, 0 = keine Korrektur, ansonsten Wert zwischen 1-2
				• usekernel: TRUE = empirische Dichteschätzung für numerische Daten, FALSE = Gaussian Naive Bayes
				• adjust: Bandbreite für Dichteschätzung, wenn usekernel = T, dann Bandbreite angeben, bei usekernel = F, dann adjust = 0
				
				confusionMatrix(predict(nb, testDat), reference = testDat$**Species**)
				
				predict(nb, newdata = data.frame(**alter = "21-30", einkommen = "mittel", geschlecht = "Frau"**)) #Voraussage aufgrund bestimmter Features
				
				# Alternative mit library(e1071):
				naiveBayes(**einkauf** ~ . , data = **einkauf[,1:3]**)

Überwachtes Lernen: Klassifizierer

	Idee/Ansatz	Voraussetzungen/Vorgehen	Vorteil (+) / Nachteil (-)	R-Code [Default]
	Preprocessing für praktische Anwendung von LDA und QDA • **Normalverteilung**: Für Feature wird Normalverteilung vorausgesetzt. Kann helfen, univariate Verteilungen jeder Variable zu überprüfen und zu transformieren (z.B. log/Wurzel für Exp.verteilung) • **Ausreisser entfernen:** Ausreisser können die Schätzungen verzerren. Ggf. hilft es, Ausreisser zu entfernen.			

Lineare Diskriminanzanalyse LDA	• Verwendung Bayes Theorem
	• Verteilungen der Klassen ist normalverteilt
	• Ausgelegt auf numerische Features.
	• Setzt keine Unabhängigkeit voraus.
	• Zusammenhang wird modelliert.
	Parameterschätzung mit Trainingsdaten:
	• Erw.wert μ_I wird mit Mittelwert aller Trainingsdaten aus Klasse I geschätzt. $\hat{\mu}_I = \frac{1}{n_I}\sum_{i:y_i=I} x_i$
	• Var. σ^2 wird als gewichteter Mittelwert Var. aller k-Klassen geschätzt. $\hat{\sigma} = \frac{1}{n-k}\sum_{I=1}^{k}\sum_{i:y_i=I}(x_i - \hat{\mu}_I)^2$
	• π_I wird als Anteil der Trainingsbeobachtungen geschätzt, die der I-ten Klasse angehören
	Multivariate Merkmalsverteilung
	• jede Klasse wird durch multivariate Normalverteilung mit Erwartungswertvektor $\mu\left(\mu_{x_1}, \mu_{x_2}, \ldots, \mu_{x_p}\right)$ und
	• Kovarianzmatrix Σ (alle Klassen gleich) definiert.
		• Numerische Features
		• Keine Parameter zum Tunen!
		• Gleiche Varianz: Es ist fast immer gute Idee, Daten vor Verwendung zu standardisieren.
		
		Vorgehen:
		1. Schätzen Prior der Klassen gem. Klassenhäufigkeit im Trainingsdatensatz oder Vorwissen
		2. Schätzen Parameter für zugrundeliegende Normalverteilungen (μ_1, μ_2, σ^2) → Annahme LDA: Prädiktoren für jede Klasse sind normalverteilt mit **unterschiedlichem Erwartungswert**, aber **gleicher Varianz**
		3. Bestimmen Entscheidungsgrenze nach Bayes-Regel: $P(Y = C_I|X) = \frac{\pi_I f_I(x)}{\sum_{i=1}^{K}\pi_i f_i(x)}$ mit $f_i(x)$: Dichte von X für Klasse k, π_I: Prior für Klasse I
			+ Keine Unabhängigkeitsannahme
			+ Methode funktioniert erstaunlich gut, selbst bei nichtnormalverteilten Gruppen.
			
			– Annahme Normalverteilung
			– Nur für numerische Features
			– Annahme: Varianzen aller Klassen sind gleich
			– Ist für Datensätze mit vielen Merkmalen weniger geeignet
				lda_fit <- train(y= trainDat$**Species**, x = trainDat[, **1:4**], method = "lda", trControl = ctrl, **preProcess = c("center", "scale"))**
				
				# Vorhersage
				confusionMatrix(predict(lda_fit, testDat), reference = testDat$**Species**)
				
				#Teilweise muss für DA zuerst Dimensionsreduktion durchgeführt werden:
				res.pca <- prcomp(trainDat[, -**1**], scale = F)
				v_80 <- which(cumsum(res.pca$sdev^2)/sum(res.pca$sdev^2)>0.8)[1]
				
				train_reduced <- data.frame(label = trainDat$**label**, res.pca$x[,1:v_80])
				m.lda <- train(**label** ~ ., data = train_reduced, method = "lda", trControl = ctrl)

Quadratische DA	• Erweiterung LDA auf QDA
	• Gibt für jede Klasse eigene Kovarianzmatrix Σ
	• X geht nun quadratisch in die Diskriminanzfunktion $\delta k(x)$ ein → Klassengrenzen können jetzt eine quadratische Form haben.
	• Bei QDA müssen wesentlich mehr Parameter geschätzt werden.
		• Numerische Features
		• Keine Parameter zum Tunen!
		Vorgehen:
		1. Modellierung Verteilung Features X der versch. Gruppen individuell in Trainingsdaten
		2. Bayes Theorem, um bedingte Wahrscheinlichkeit → $P(Y = C_I|X)$ zu bestimmen.
		3. Ordne die Beobachtung der Klasse mit der höchsten A-posteriori Wahrscheinlichkeit zu.
			+ Keine Unabhängigkeitsannahme
			+ Verschiedene Varianzen in Klassen möglich
			+ Methode funktioniert gut, selbst bei nichtnormalverteilten Gruppen.
			– Annahme Normalverteilung
			– Nur für numerische Features
			– Hohe Berechnungskomplexität
			– Ist für Datensätze mit vielen Merkmalen weniger geeignet
				qda_fit <- train(y = trainDat$**Species**, x = trainDat[, **1:4**], method = "qda", trControl = ctrl, **preProcess = c("center", "scale")**)
				
				# Vorhersage
				confusionMatrix(predict(qda_fit, testDat), reference = testDat$Species)

Log. Regression	• $\log\left(\frac{p(y|x)}{1-p(y|x)}\right) = \beta_0 + \beta_1 \cdot x$ • Schätzung Parameter $\hat{\beta}_0, \hat{\beta}_1$ mit Max. Likelihood I: $I(\hat{\beta}_0, \hat{\beta}_1) = \prod_{i:y_i=1} p(y_i|x_i)\prod_{i:y_i=0}(1 - p(y_i|x_i))$ • Linearer Prädiktor (rechte Seite) kann weitere erklärende Variablen haben (vgl. multiple Regression) • Logistische Regression hat keinen Fehlerterm. Streuung Ergebnisse wird Bernoulli-Verteilung modelliert. • Geschätzen Modellparameter kann man analysieren. • Koeffizienten sind log-Odds. Können diese in Odds umwandeln. Z.B. exp(res.glm$coefficients[2])	• kein Tuningparameter **Multinomiale log Regression (Mehr als 2 Klassen)** • Eine der K Klassen wird zur «Referenzkategorie» k_0 (erste Faktorstufe, optimal wenn es grösste Klasse ist) • Für alle anderen $K-1$ Klassen wird eine seperate logistische Regression gegenüber der Referenzkategorie modelliert. $\log\left(\frac{P(Y_i=k|X_i)}{P(Y_i=0|X_i)}\right) = logit(p_k)$ • Gilt $\sum_{k=0}^{K-1} p_k = 1$ → $p(Y_i = k|X_i) = \frac{\exp(\beta_k \cdot X_i)}{1+\sum_{j=1}^{K-1}\exp(\beta_j \cdot X_i)}$	+ Methode geht auch für unbalancierte Daten, Konfusionsmatrix ist dann jedoch heikel + Gibt auch Ansätze für ordinale Variablen (method = "polr" in caret). + – Ist für Datensätze mit vielen Merkmalen weniger geeignet	res.glm <- train(y = trainDat$**y**, x = trainDat$**x**, method = "**glm**", family = "binomial") summary(res.glm)$coefficients confusionMatrix(predict(res.glm, newdata = testDat), testDat$**Aktiv**)$table res.glm <- train(y = **y**, x = **x**, method = "**multinom**", trControl = ctrl, tuneLength = 7, trace = F)

Logistische Regression, k-NN und Diskriminanzanalyse sind für Datensätze mit vielen Merkmalen weniger geeignet. ⇒ Alternative Klassifikationsbäume

• Diskriminanzanalyse und logistische Regression (insbes. multinomial): Anzahl zu schätzenden Parameter wird schnell zu gross.
• Obwohl k-NN keine Modellparameter hat, leidet es unter Fluch der Dimensionalität → im hochdimensionalen Raum werden viel mehr Punkte für gleiche Abdeckung benötigt. Beispiel: Wenn 10 Beobachtungen 10% des Raums in einer Dimension abdecken, sind 100 Beobachtungen in 2 Dimensionen erforderlich.
• kNN und Diskriminanzanalyse funktionieren nur begrenzt mit kategorialen Merkmalen.

Klassifikationsbäume

Idee/Ansatz

- Intuitives Verfahren, um Objekte mittels «ja/nein» oder «wahr/falsch» Fragen einer Klasse zuzuordnen.
- Nächste Frage hängt von Antwort der Frage zuvor ab.
- Fragen/Klassenzuordnung werden im Baum gezeigt
- Basis für Ensemble-Verfahren: Bagged Trees, Random Forest, Boosting (Adaboost)
- Verwendung auch für Regressionsprobleme
- Kann mehrere Bäume geben, die zu Daten passen

Einschub: Odds

- Wahr' als Chancenverhältnisse ausdrucken → Odds
- Kennt man aus Umgangssprache oder Wetten
- Z.B. Chance stehen 5 zu 3.
- Geschrieben als Odds $\frac{5}{3} = 1.7$
- Als Wahr' $\frac{5}{8} = 0.625$, Gegenwahr': $1 - 0.625 = 0.375$
- Odds aus Wahr' $Odds = \frac{p}{1-p} = \frac{0.625}{0.375} = 1.7$
- Gewinnchancen kleiner bei Odds zwischen 0 und 1, Gewinnchancen grösser bei Odds zwischen 1 und ∞ → unsymmetrisch → Alternative log(Odds)

logit-Funktion

- Wahr' zwischen 0 und 1
- Odds zwischen 0 und ∞
- log(Odds) zwischen $-\infty$ und ∞
- Modellierung der log-Odds (logit-Funktion) mit linearen Prädiktor. $\log\left(\frac{p(y|x)}{1-p(y|x)}\right) = \beta_0 + \beta_1 \cdot x$
- $p \rightarrow p(y|x) = \frac{1}{1+\exp(-(\beta_0-\beta_1 \cdot x))}$ → Sigmoid Funktion

Wo unterteilen?

- Fragen so stellen, dass Teilmengen möglichst einheitlich sind (d.h. nur Objekte der gleichen Klasse).
- Ist zweckmässiger mit Verunreinigung I (impurity)
- Welche Verunreinigung an Knoten N in zwei Untergruppen (N_L und N_R) so stark wie möglich verringert
- $\Delta I(N) = I(N) - \alpha_L \cdot I(N_L) - (1 - \alpha_L) \cdot I(N_R)$
- mit α_L Anteil Beobachung in Untergruppe N_L · Häufigste Verunreinigungsmassnahme Gini-Index: Erwartete Fehlerrate Knoten N, falls Label zufällig von den am Knoten vorhandenen Klassen selektiert werden.
- $Gini = 1 - \sum_{i=1}^{g} p_i^2$, p_i: relative Häufigkeit der Klasse i in diesem Knoten · Alternative Entropy $Entropy = \sum_{i=1}^{g} p_i \cdot \log_2(p_i)$

Voraussetzungen/Vorgehen

Vorgehen (Baumkonstruktion):

- Ausgangslage: gesamter Raum
- Unterteilungen Raum mit binären Fragen, welche sich auf eine Variable beziehen → nur orthogonale Aufteilungen möglich (→ nur gerade Linien)
- Aufteilung bis keine weiteren Unterteilungen mehr möglich sind (auch andere Abbruchkriterien möglich).
- Klassenzuordnung gemäss Mehrheit im Endknoten.

Stoppregel

- Baum wachsen bis $\Delta I(N) \leq 0$ → muss nicht reine Knoten geben → häufig überangepasst Bäume (Overfitting)

Mögliche Stoppregeln:

- $\Delta I(N)$ soll **Schwellenwert** nicht unterschreiten → direkt kontrollierbar in rpart über Parameter cp in caret ist das der Hyperparameter. $Schwellenwert = cp \cdot I(N_{start})$
- **Minimale Anzahl Objekte**, bei denen ein Knoten noch aufgeteilt wird → train(., method = "rpart", control = rpart.control(minsplit = 20))
- **Mindestanzahl Objekte**, welche in Endknoten noch vorhanden sein müssen → rpart(., method = "rpart", control = rpart.control(minbucket = round(minsplit/3))

CP mit KV optimieren! Cost-complexity pruning:

- Baum zuerst vollständig wachsen lassen
- cp mit Kreuzvalidierung optimieren
- Dann vollständigen Baum stutzen zu optimalem cp

Was ist die richtige Komplexität eines Modells?

- Bias–Variance Tradeoff: Fehler = Bias + Varianz + unreduzierbarer Fehler

Cost-complexity Pruning (kürzen, stutzen)

- Abwägung Genauigkeit und Komplexität des Baums:
- $R(\alpha) = Fehlerrate + \alpha \cdot |T|$
- T: Anzahl Endknoten und α: Komplexitätsparameter
- Für $\alpha = 0$ ist der grösstmögliche Baum optimal.
- Wie finden «optimale» α? Führen K-fache KV für versch. Werte von α für $i = 1, \ldots, K$ wie folgt durch:
  1. Lass Baum vollständig auf Trainingsdaten i wachsen
  2. Schneide Knoten Schritt für Schritt ab und berechne $R(\alpha)$ mit den Validierungsdaten i.
  3. Speichere optimale $R(\alpha)$.

→ Mittlere Fehlerquote als Funktion der α

- Endgültiger Baum: Gestutzter Baum mit allen Trainingsdaten, der dem optimalen Baum entspricht α.
- $cp = \frac{\alpha}{R(\infty)}$, (R(∞) Cost complexity an Wurzel)

Vorteil (+) / Nachteil (-)

+ anschauliche Darstellung, gut für Kommunikation mit Fachleuten und Auftraggebern mit wenig Interesse für statistische Details
+ Interpretierbarkeit
+ Keine Verteilungsannahmen
+ Funktioniert mit kategorialen Daten
+ Robust gegenüber Ausreissern
+ Keine Variablentransformation notwendig (oft doch sinnvoll !!!)
+ Kann nichtlineare Strukturen & lokale Wechselwirkungen erfassen
+ Erwartungsgetreuer Schätzer
+ Kleiner Fehler, wenn aussagekräftige Eingabevariablen vorhanden sind und Baum ausreichend tief ist

– neigt zu Overfitting * hohe Variation → selten für Vorhersagen verwendet
– Bäume sind instabil → Die Interpretation kann sich ändern.
– Erfordert grosse Datenmengen, um additive Strukturen oder lineare Strukturen zu erfassen.

R-Code [Default]

```
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))

# Visualisierung des Baums
plot(r.tree$finalModel, margin = 0.2, uniform = TRUE)
text(r.tree$finalModel, use.n = TRUE)

# Alternative für schöner Visualisierung
library(rattle)
fancyRpartPlot(r.tree$finalModel)

# Vorhersage für Test-Daten
confusionMatrix(predict(r.tree, newdata = testDat), testDat$Species)

# Alternative direkt mit rpart
library(rpart)
tree <- rpart(Species ~., data = trainDat, control = rpart.control(xval = 10, cp = 0))
printcp(tree)

# 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 = 0.01))
plot(tree)
```

Bias-Variance Tradeoff

Bias-Variance Tradeoff

- Bias/Varianz wesentlich bei Modellbewertung
- Bias ist Differenz zwischen mittleren Vorhersage des Modells und richtigen Wert, den wir vorhersagen wollen ($Bias(\hat{Y}) = E(\hat{Y}) - Y$).
- Modell mit hohem Bias schenkt Trainingsdaten sehr wenig Aufmerksamkeit, was zu Modell führt, welches zu stark vereinfacht ist. Es hat hohen Fehler bei Trainings- und Testdaten.
- Varianz ist die Variabilität der Modellvorhersage für einen bestimmten Datenpunkt.
- Modell mit hoher Varianz schenkt Trainingsdaten zu viel Aufmerksamkeit und verallgemeinert nicht auf ungesehene Daten. Wenn Trainingsdaten geringfügig ändern, ändert auch Modell. Modelle mit hoher Varianz schneiden bei Trainingsdaten sehr gut ab, haben aber hohe Fehlerquoten bei ungesehenen Daten.
- Entscheid.bäume = hohe Varianz, klein Bias

Warum Tradeoff?

- Wenn Modell zu einfach, kann es hohen Bias und geringe Varianz aufweisen.
- Wenn Modell zu komplex, wird es hohe Varianz und geringen Bias aufweisen.
- Komplexitätstradeoff ist Grund, warum Tradeoff zwischen Bias/Varianz besteht.
- Algorithmus kann nicht gleichzeitig komplex und weniger komplex sein.
- → Müssen richtiges Gleichgewicht zwischen Unter- und Überanpassung finden.
- $Gesamtfehler = Bias^2 + Varianz + Irreduzierbarer\ Fehler$

Umgang Tradeoff

- **Modellauswahl:** Testen div. Modelle, um optimale Lsg. für Problem zu finden. Durch richtige Wahl finden gutes Gleichgewicht Bias/Varianz.
- **Kreuzvalidierung:** Bewertung, wie Modell im Training auf ungesehene Daten reagiert, und Anpassung Hyperparameter.
- **Regularisierung**: Beinhaltet Strafterm, um zu verhindern, dass Modell zu sehr an Trainingsdaten anpasst und somit zu Überanpassung führt.
- **Feature engineering**: Bearbeitung Eingabefeatures zur Verbesserung Modellleistung.
- **Ensemble-Methoden**

Ensemble-Methoden

Ensemble-Methoden

- Ansatz, um Klassifizierer noch leistungsstärker zu machen
- Hypothese, dass Kombination mehrerer Modelle oft leistungsfähigeres Modell ergeben kann
- Viele Instanzen v. gleichen Klassifizierers oder versch. Klassifizierer zusammen verwenden.
- Gesamtklassifizierer → z.B. Mehrheitsabstimmung aller einzelnen Klassifizierer.
- Erfolgreich, wenn einzelne Klassifizierer im Schnitt unverzerrt
  - Reduktion der Variabilität gegenüber den einzelnen Klassifizierern
  - gut wenn einzelne Klassifizierer stochastisch unabhängig

Beispiel Ensemble-Methode

- Bsp. mit 25 Basisklassifizierer
- Jeder der Klassifizierer hat eine Fehlerrate $\epsilon = 0.35$
- Angenommen alle Klassifizierer sind unabhängig (fragwürdigste Annahme)
- Nehmen Mehrheitsentscheid
- Mehrheitsentscheid ist falsch, wenn 13 oder mehr der einzelnen Klassifizierer falsch liegen.
- $X \sim Bin(size = 25, p_0 = 0.35) \Rightarrow P(X \geq 13) = \sum_{i=13}^{25} \epsilon^i (1-\epsilon)^{25-i} = 1 - pbinom(12, 25, 0.35) = 0.06$
- Ensembles sind nur besser als einzelner Klassifizierer, wenn jeder Klassifizierer besser ist als zufälliges Raten!

Pro (+) / Contra (-)

+ Verbesserung Genauigkeit
+ Macht Modell robuster
+ Ermöglicht gleichzeitige Erfassen von einfacheren lineare und komplexeren Beziehungen in Daten
– Reduziert Modell Interpretierbarkeit
– Zeitaufwändiger
– Auswahl Modelle für Erstellung Ensembles ist nicht immer einfach und braucht Erfahrung

Ensemble-Ansätze

Ensemble-Ansätze

- **Stacking**: heterogene schwache Klassifizierer werden parallel trainiert und mittels Metamodell kombiniert, um Vorhersage basierend auf Vorhersagen der versch. schwachen Modellen zumache
- **Bagging** (bootstrapping & aggregieren): homogene schwache Klassifizierer, lernen parallel anhand unabhängiger Bootstrap-Samples und werden aggregiert. → Bagged Trees und Random Forest (Bagged trees mit «speziellen Trick»)
- **Boosting** (iteratives Verfahren zur adaptiven Änderung der Trainingsdaten durch stärkere Fokussierung auf zuvor falsch klassifizierte Datensätze). → Gradient boosting, ADABOOST und XGBoost

Stacking → Beispielcode siehe Seite 14

1. Daten aufteilen in 3 Datensätze: Train1-, Train2-Daten und Testdaten
2. Training mit verschiedene Algorithmen auf den Train1-Daten (Hyperparameter-Tuning mittels Kreuzvalidierung). → erste Ebene
3. Wenden Modelle auf Train2-Daten an und speichern vorgesagten Wahrscheinlichkeiten jedes dieser Algorithmen.
4. Trainiere Klassifikator mit vorhergesagten Wahrscheinlichkeiten aus der ersten Schicht (Feature) und ursprünglichen Zielvariable → Toplayer
5. Die Wahrscheinlichkeitsvorhersagen werden für neue Daten (Testdaten) mit den Modellen der ersten Schicht gemacht, die als Eingangsvariablen für das Modell der obersten Schicht verwendet werden. → endgültige Vorhersage

(Verschiedene Anpassungen sind möglich: z.B. Mehrheitsentscheidung der ersten Schicht, Neutraining der ersten Schicht nach Training der obersten Schicht mit allen Trainingsdaten, weitere Schichten oder noch komplexere Ansätze)

Idee Bagging

1. Mehrere Datensätze anhand originale Trainingsdaten erstellen
2. Erstelle mehrere Klassifizierer
3. Kombiniere Klassifizierer

Grundidee (Beispiele Bäume):

1. Züchten viele Bäumen auf Bootstrap-Stichproben aus Trainingsdaten
2. Minimiere Fehler, indem man Bäume ausreichend tief wachsen lässt
3. Reduzierung Varianz der Variablen der unverfälschten Bäume durch Mittelwertbildung

→ Hochgradig nichtlineare Schätzer wie Bäume profitieren am meisten vom Bagging.

Klassifizierung

Wie klassifizieren neue Beobachtung?

I. Jeder Baum hat «Siegerklasse». Verwende Klasse, die am häufigsten gewonnen hat.
II. Mitteln Wahr': $p(c|v) = \frac{1}{T}\sum_t^T p_t(c|v)$

Kategoriale Features

Allgemein

- Nicht alle Modelle können kategoriale Variablen verarbeiten (z.B. k-NN oder DA)
- Workarounds mit nachfolgenden Methoden.

Label Encoding

- Jedes Level kategorialen Variable durch Zahl ersetze
- Bei k-NN ist dies jedoch kritisch (Abstände sind nicht mehr korrekt).
- Random Forest funktioniert besser, ist aber auch nicht effizient.

One Hot Encoding

- besser → kategoriale Variable in mehrere Variablen aufteilen (eine Variable pro Level)

+ Ermöglicht Modell, die von Haus aus keine kat. Var. verarbeiten können (z. B. k-NN), dies effektiv zu tun; geht von keiner Ordnung zwischen Levels aus
– Nachteil: Erhöht Dimensionalität Merkmalsraum, was Modelltraining erschwert; kann Merkmalsauswahl in Random Forests beeinträchtigen.

Target Encoding

- wenig bekannte, aber wirksame Transformation von kat. Var.
- Jede Stufe eines kat. Merkmals wird durch durchschn. Antwort in Zielvariablen ersetzt.

Vorgehen

1. Gruppieren Trainingsdaten nach Stufen einer kategorischen Variable
2. Berechnen Durchschnitt/Anteil der Zielvariablen für jede Stufe
3. Weisen Durchschnitt jeder Beobachtung zu, die zu dieser Stufe gehört

+ Geringere Dimensionalität des Merkmalsraums; Erfassung des Einflusses von Kategorien auf Zielvariablen, nützlich in datenreichem Kontext
– Ggf. Verzerrung mean durch schlecht repräsentierte oder stark variable Kat.

- Data leakage: Wenn Zielvariable in Codierung verwenden, kann Overfitting geben.
- Implementiert in Funktion ranger(…,respect.unordered.factors = "order")

Beispiel Target Encoding

Feature category Cat und Zielvariable Buy:

```
td <- data.frame(Cat = c("A", "B", "A", "C", "B", "B", "C", "A"),
Buy = c("y", "y", "y", "n", "n", "n", "y", "y"))

(ratio_buy <- tapply(td$Buy, td$Cat, FUN = function(x) mean(x == "y")))

td$EncCat <- ratio_buy[td$Cat]
```

One-Hot-Encoding ist ideal für Modelle, die eindeutige, separate kategoriale Merkmale benötigen. Target-Encoding besser für direkte Modellierung Beziehung zwischen Kategorien und Zielvariablen.

Random Forest

Idee/Ansatz

Gegeben: N Trainingsdaten und M Features/Merkmale

Evaluation Random Forest: Out of Bag (oob) Fehlerrate
- Bei jedem Baum werden nicht verwendeten Testdaten jeweils zur Evaluation der Fehlerrate verwendet.
- Mitteln über alle Bäume → **KV on the fly!**

Kategoriale Variablen mit Random Forest
- Bäume können i.A. gut mit kategorialen Var. umgehen
- Bei vielen Faktorstufen wird dies numerisch komplex (Permutation Aufteilungsmöglichkeiten) und langsam.
- R randomForest kann nur mit 53 Stufen umgehen.
- One-hot-encoding ist nicht optimale Strategie für RF. Problem: zufällige Variablenauswahl in Knotenpunkten. Effekt: andere Merkmale werden herunter gewichtet und feature importance ist nicht mehr korrekt.
- In caret wird One-Hot-Codierung automatisch mit Formelnotation durchgeführt! Sogar mit nur ein paar Faktorstufen. (kann dies verhindern, indem ohne Formelschreibweise oder direkt mit randomForest arbeitet.

Voraussetzungen/Vorgehen

- RF ist nicht immer die beste Methode - aber oft funktioniert sie einfach!

Jeder Baum wird mit Algorithmus konstruiert:
1. Trainingsset indem N mal mit Zurücklegen aus allen N verfügbaren Trainingsfällen gezogen wird (Bootstrap) → restliche Daten für Validierung
2. Erzeuge ungestutzten Klassifikationsbaum mit Modifikation: Bei jedem Knoten wird beste Aufteilung nur unter zufälligen Auswahl von mtry Features gesucht. $mtry << M$ (default Klassifikation $mtry = \sqrt{M}$) Entkorrelieren mit allen Mitteln! → Maximierung der Varianzreduktion indem Korrelation zwischen Bäumen möglichst klein gehalten wird → Verhindert Overfitting
3. Vorhersage aus Mehrheitsabstimmung aller Bäume

Vorteil (+) / Nachteil (-)

+ Einfache Handhabung - keine Verteilungsannahmen/Transformationen
+ Perfekte Benchmark-Methode
+ Liefert kostenlose oob Fehlerrate
+ Kann mit tausenden von Eingangsvariablen umgehen, auch wenn viele Störvariablen darunter sind
+ Guter Umgang mit Wechselwirkung
+ Robust gegen Ausreisser
+ Neigt nicht zu Overfitting (Beobachtungen müssen unabhängig sein!)
+ Kann unbalancierte Daten durch Bootstrap-Sampling verarbeiten
+ Anwendung auf Regressionsproblem
+ Kann verwendet werden, um fehlende Daten zu imputieren
+ Bietet Methoden zur Evaluierung der Bedeutung der Variablen und Abhängigkeitsdiagramme
– Nur 53 Faktorstufen

R-Code [Default]

```
ctrl <- trainControl(method = "oob")
r.rf <- train(Species ~., data = trainDat, method = "rf", trControl = ctrl, tuneLength = 3, ntree = 1000) #ntree Anzahl Bäume
r.rf$results
plot(r.rf$finalModel, las = 1)

# Konfusionsmatrix
r.rf$finalModel$confusion

# Alternative mit library(ranger)
r.out <- ranger(formula = survived ~., data = trainDat, num.trees = 1000, mtry = 2, respect.unordered.factors = "order")
#"order" = target encoding
#sonst level encoding (Standard)

confusionMatrix(predict(r.rf, data = testDat), testDat$survived)
```

Unbalancierte Daten

Allgemein

Haben unbalancierten Datensatz mit mehr Beobachtungen in Klasse 1 als in Klasse 2 (bspw. 85 % F zu 15 % T). Für hohe Genauigkeit, reicht aus, jede Beobachtung Klasse 1 zuzuweisen.
- Sensitivität (True-Positive-Rate) Anteil "Positiven", die korrekt identifiziert werden.
- Spezifizität (True-Negativ-Rate) Anteil "Negativen", die korrekt identifiziert werden.
- Erhalten in diesem Beispiel hohe Spezifität, aber tiefe Sensitivität. Je nach Fragestellung kann es andersherum sein.

Ansätze Umgang

Was machen, um Identifikation zu verbessern?

Alternativer Cutoff
- Wahrscheinlichkeitsvorhersagen und dort alternativen Cutoff-Wert wählen (Standard 50 %), neu z.B. 20 %
- Cutoff-Wert von Fragestellung/Anforderungen abhängig
- Auswertung auf Validierungsdaten
- Trade-of zwischen verschiedenen Arten von Fehlern. → ROC-Kurve

```
pred_prob <- predict(res.glm, newdata = testDat, type = "prob")
pred_neu <- factor(ifelse(pred_prob[,1] > 0.03, "Yes", "No"), levels = c("Yes", "No"))
#levels: Klassen, Cutoff neu 3 %
confusionMatrix(data = pred_neu, reference = testDat$default)
```

ROC-Kurve

- ROC-Kurve (Receiver Operator Characteristic): Sensitivity (True-Positive Rate (tpr)) gegen 1-Specificit (False-Positive-Rate (fpr)) für verschiedene Cut-Off-Werte
  - Modell mit perfekter Trennung geht durch den Punkt (1, 1)
  - Möglicher Cutoff-Wert nächster Punkt dazu auf der Kurve
- ROC-Kurve zeigt, dass Modellvergleiche auf Basis Accuracy irrführend sein können, da bessere Cutoff-Wert möglich sind.
- ROC kann auch zur Modellevaluierung verwendet werden

```
library(pROC); r_roc <- roc(testDat$Churn, pred_prob[,1], las = 1, levels = c("F", "T")); plot(r_roc, las = 1)
(coord <- coords(r_roc, x = "best", best.methd = "closest.topleft"))
pred_alt <- factor(pred_prob[,1] > coord[,1], levels = c("T", "F"))
confusionMatrix(data = pred_alt, reference = testDat$Churn)
```

Anpassung Prior

- Einige Klassifizier können Prior-Wahrscheinlichkeit einstellen (Satz Bayes). → Naive Bayes-Klassifikator oder Diskriminanzanalyse
- Standardmässig wird Prior-Wahrsch' aus Verteilung in Trainingsdaten bestimmen. Kann das auch anpassen.
- ACHTUNG: Wahrscheinlichkeiten stimmen nicht mehr und dürfen nicht mehr interpretiert werden

```
lda_fit <- train(Churn ~., method = "lda", prior = c(0.5,0.5))
conf$table; conf$byClass["Sensitivity"]
```

Sampling Methoden

Prior Sampling
- Falls Dysbalance bekannt, Datenaufnahme anpassen (balancierte Trainingsdaten, Testdaten sollten für korrekte Evaluation, trotzdem tatsächlichen Verteilung entsprechen)

Up-Sampling
- Ziehen aus kleineren Klasse bis Klassen angeglichen sind
- Gibt ausgeklügeltere Ansätze, z.B. SMOTE (synthetic minority over-sampling technique)

Down-Sampling
- Beobachtungen aus Mehrheitsklasse löschen, um Anzahl Minderheitsklasse anzugleichen. (Optimal: Matching Verfahren) → Datenverlust
- spannend im Kontakt von Bootstrapsamples → Stratifizierte Samplen beim Random Forest

Bemerkung: Resampling erst nach Aufteilen Trainings/Testdatensatz durchführen.

```
down <- downSample(x = trainDat[, -15], y = trainDat$Churn, yname = "Churn")
dim(down); #774 15 table(down$Churn) # 387 387
rf_d <- train(x = down[, -15], y = down$Churn, method = "rf", trControl = ctrl)
pred_d <- predict(rf_d, newdata = testDat)
confusionMatrix(pred_d, testDat$Churn)
up <- upSample(…) # down überall durch up ersetzen
```

Random Forest

Down-Sampling mit Random Forest → **Stratifiziert Bootstrap**-Stichprobe

```
ctrl <- trainControl(method = "oob")
r.rf <- train(x = trainDat[, -15], y= trainDat$Churn, method = "rf", tuneLength = 3, trControl = ctrl, strata = trainDat$Churn, sampsize = rep(387, 2))
#strata = Zielvar.
#sampsize = Vektor mit länge = Anz. Klassen + Anz. Elemente v kleinster Klasse aus Trainingset
table(trainDat$y)
```

Interpretierbares Machine Learning

Interpretation Modelle

- Machine Learning Algorithmen, insbesondere Ensemble-Verfahren, haben Ruf Black-Box-Modelle zu sein.
- Das ist irreführend, man kann auch bei komplexen Algorithmen Zusammenhänge der Variablen analysieren (ähnlich wie linearen Regression).
- Explainable Machine Learning ist aktives Forschungsfeld. Licht in Modelle bringen.

Fokus auf 2 Aspekte:

- **Wichtigkeit Variablen**
- **Einfluss Variablen**

Variable Importance VI – Wichtigkeit der Variablen

Allgemein

Wie wichtig ist eine Variable bei einem bestimmten Modell, um genaue Vorhersagen zu machen. → je mehr sich Modell auf eine Variable verlässt für Vorhersagen, desto wichtiger ist sie.

Wieso ist das wichtig?

- Wollen sehen, ob Modell realistisch ist und Erwartungen entspricht. Haben Möglichkeit Modell zu verbessern?
- Feature-Selektion (welche Variablen für gutes Modell)

Ansätze

Modellspezifisch

- Ansatz direkt mit Modell verknüpft
- Nicht für alle Algorithmen verfügbar
- Nicht über verschiedene Algorithmen vergleichbar

Modellunabhängig

Flexible, für alle Klassifikationsalgorithmen verwendbar.

Modellspezifisch Ansätze

- caret hat modellspez. Ansätze (Rand. Forest)
- Bei jedem Split, bei dem Variable verwendet wird, werden Werte der Variablen im Out-of-Bag-Sample zufällig gemischt, wobei alle anderen Variablen gleichbleiben.
- Gemessen wird Abnahme Vorhersagegenauigkeit. Differenz zwischen den beiden Genauigkeiten wird dann über alle Bäume gemittelt.

→ Idee auch modellunabhängig umsetzbar

```
ctrl <- trainControl(method = "oob")
r.rf <- train(x = trainDat[, -15], y = trainDat$Churn, method = "rf", importance = TRUE)
varImp(r.rf, scale = FALSE)
```

scale = TRUE skaliert, dass max. Wert bei 100 liegt. train importance = TRUE, wird bei mehr als 2 Klassen Importance für Klassen einzeln ausgegeben, sonst werden diese entsprechend Klassenhäufigkeit gewichtet.

Modellunabhängige Ansätze

- Permutationsansatz v. Random Forest kann auf beliebige Modell verallgemeinert werden (ohne Out-of-Bag-Daten).
- Trainieren eines beliebigen Algorithmus
- Loop über alle m Variablen
  1. Werte der $i-$ten Variable in Daten zufällig mischen. Werte der restlichen $m-1$ Variablen bleiben fix
  2. Vorhersage für modifizierte Datensatz.
  3. Messen Abnahme Vorhersagegenauigkeit.
- Ranking basierend auf Vorhersageabnahme bei einzelnen Variablen (je stärker Abnahme, desto wichtiger Variable).

```
library(vip);  q_fit <- train(Churn ~., data = trainDat, method = "qda")
r_vi <- vi(q_fit, train = testDat, target = "ZIELVARIABLE", method = "permute", metric = "accuracy", pred_wrapper = predict, nsim = 3) #nsim Anz. Wiederholungen (Zufallsprozess)
vip(r_vi, geom = "boxplot")
```

ICE Plot

- Idee: “Flache” Linien haben keinen Einfluss (Variable unwichtig), Linien, die stark ändern (Variable wichtig)
- Veränderungsmass Linie: Varianz
- Variablen Importance nicht zwingend gleich für verschiedene Ansätze.
- Variablen Importance bei kategoriellen Variablen nicht immer aussagekräftig.

```
library(vip)
res_vi <- vi(q_fit, train = trainDat, method = "firm", target = "Churn", metric = "accuracy", pred_wrapper = predict, ice = TRUE) #firm für VI
vip(res_vi)
```

Allgemeine Einschränkung: Alle Ansätze hängen von Modellqualität ab. Wenn Modell schlecht ist, ist Variable Importance nicht aussagekräftig.

Einfluss einer Variable

ICE-Plots (Individual Conditional Expectation)

- Intuitiver Ansatz, um Einfluss einer Variable auf Klassifizierung zu analysieren (ICE oder Partial Dependence Plot PDP)
- Jede Beobachtung wird Werte Variable über ganzen Wertebereich variiert, während alle anderen Variablen festgehalten werden, und dafür eine Vorhersage erstellt. → Linie für jede Beobachtung. X-Achse zeigt Wertebereich und y-Achse wie sich Wahrscheinlichkeit für Zielklasse ändert.
- Rote Line zeigt Mittelwert → Partial Dependence Plot (PDP)
- ICE-Plot ist reinen PDP vorzuziehen, da sich dort gegenläufig Effekt ausmitteln können und verschwinden.
- **Vorsicht** Wechselwirkungen zwischen Variablen werden ICE-Plot nicht berücksichtigt. Kann falsche Schlussfolgerungen

R-Code [Default]

```
library(pdp);  r_par <- partial(q_fit, pred.var = "ERKL. VARIABLE", type = "classification", which.class = "WERT ZIELKLASSE", prob = T, ice = T)
plotPartial(r_par, main = "ICE Plot")
```

pred.var: Welche erklärende Variable soll genommen werden?
type: Auch für Regressionsprobleme verwendbar
which.class: Welche Klasse der Zielvariable soll gezeigt werden?
prob: Wahrscheinlichkeit, sonst logit-Werte, ice: ICE, sonst PDP-Kurve

Eigenes Ensemble-Modell mit Stacking erstellen

```
intrain <- createDataPartition(y = churn$Churn, p = 0.8, list = F)
train <- churn[intrain,];  test <- churn[-intrain,]
intrain2 <- createDataPartition(y = train$Churn, p = 0.7, list = F)
train1 <- train[intrain2,];  train2 <- train[-intrain2,]

ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
m_knn <- train(Churn ~., data = train1, method = "knn", trControl = ctrl, tuneLength =10)
m_lda <- train(Churn ~., data = train1, method = "lda", trControl = ctrl)
m_qda <- train(Churn ~., data = train1, method = "qda", trControl = ctrl)
m_log <- train(Churn ~., data = train1, method = "glm", family = "binomial", trControl = ctrl)
results <-resamples(list(KNN = m_knn, LDA = m_lda, QDA = m_qda,  LOG = m_log))
summary(results);  bwplot(results);  dotplot(results)  #Modellvergleich

pred_knn <- predict(m_knn, train2, type = "prob")[, 1];
pred_lrg <- predict(m_log, train2, type = "prob")[, 1]
pred_lda <- predict(m_lda, train2, type = "prob")[, 1]
pred_qda <- predict(m_qda, train2, type = "prob")[, 1]
input_toplayer <- data.frame(pred_knn = pred_knn, pred_lrg = pred_lrg,
                             pred_lda = pred_lda, pred_qda = pred_qda)

#Verwenden Wahr’ aus ersten Layers als Feature für Top Layer mit einer log Regression
input_toplayer$Churn <- train2$Churn
(model_top <- train(Churn ~ ., data = input_toplayer, method = 'glm', trControl = ctrl))

pred_knn <- predict(m_knn, test, type = "prob")[, 1] #Mit Testdaten auswerten
pred_lrg <- predict(m_log, test, type = "prob")[, 1]
pred_lda <- predict(m_lda, test, type = "prob")[, 1]
pred_qda <- predict(m_qda, test, type = "prob")[, 1]
input_toplayer <- data.frame(pred_knn = pred_knn, pred_lrg = pred_lrg,
                             pred_lda = pred_lda, pred_qda = pred_qda)
confusionMatrix(predict(model_top, newdata = input_toplayer), test$Churn))
```

Boosting

Allgmein

- Im Unterschied zum Bagging, werden verschiedenen homogenen Klassifizierer nicht mehr unabhängig voneinander, sondern **sequenziell trainiert**.
- Grösseres Gewicht auf Beobachtungen, welche im vorherigen Modell falsch vorhergesagt wurden.
- Unterschiedliche Gewichtung Modelle im Ensemble.
- Hauptaugenmerk auf Reduzierung Bias → Basismodelle, Modelle mit geringer Varianz, aber hohem Bias (Unterschied zu Bagging!). Diese Modelle sind weniger rechenaufwendig.
- Modelle meist einfache, flache Bäume

AdaBoost

- Erste praktische Implement.
- Binärer Klassifikator

3 Hauptideen

- Kombination von vielen schwachen Lerner → Bäume; häufig mit nur einer Stufe (stamps).
- Nicht alle Modelle sind gleich gut. Bessere Modelle erhalten im Ensemble mehr Gewicht.
- Jedes Modell berücksichtig die Fehler vom vorgängigen stärker.

Berücksichtigung Gewichte in Klassifizieren

- Gewichteter Gini-Index (d.h. stärkere Berücksichtigung der Beobachtungen mit grösserem Gewicht)
- Neues Datensample → ziehen mit Zurücklegen aus Originaldaten mit Gewicht entsprechen den Wahr'
- → Neuer Klassifiziere mit Sample (gleiche Gewichtung)

```
ctrl <- trainControl(method = "cv", number = 5)
res.ada <- train(label ~ ., data = trainDat, method = "AdaBoost.M1", trControl = ctrl, tuneGrid = data.frame(mfinal = 50, maxdepth = 1, coeflearn = "Breiman"))
confusionMatrix(predict(res.ada, newdata = testDat), reference = testDat$label)
```

mfinal: Anzahl Bäume
maxdepth: max Tiefe einzeln Bäume
coeflearn: Versch. Ansätze Gewichtung Ensembles (α)

Gradient Boosting

- Schätzen Zielvariable mit schwachen Algorithmus (Regression/Klassifikation)
- Schrittweises pushen Vorhersage gegen beobachtete Zielvariable anhand Informationen aus Residuen.
- $\hat{y} = F_0(x) + \sum_{m=1}^{M} \lambda \cdot f_m(x) = F(x)$
- $\hat{y}$: Vorhersage
- $F_0(x)$: Initialisierungsmodell für Zielvariable
- $\sum_{m=1}^{M}$ ...: Kumulative Verfeinerung Modell mit M Iterationen
- $f_m(x)$: Weak learner (bei Klassifikation einfache Klassifikationsbäume) mit Residuen (gradient) aus Schritt $m-1$
- λ: Lernrate zur Steuerung Schrittweite
- $F(x)$: finales Modell

Algorithmus:

- Initialisierung $F_0(x)$ ist i.d.R. konstanter Schätzer. Mittelwert bei Regression. Frequenzen bei Klassifikation.
- Funktionen $f_m(x)$ verwenden i.d.R. kleine Regressionsbäume.

1. Initialisieren Vorhersagemodell: $\hat{y} = F_0(x)$
2. Von $m = 0$ bis M
   a. Residuen berechnen: $r = y - F_m(x)$
   b. Residuen modellieren: $\hat{r} = f_m(x)$
   c. Neues Modell: $F_{m+1}(x) = F_m(x) + \lambda f_m(x)$

+ Sehr leistungsfähig, flexibel
– Viele Hyperparameter
– Anfällig auf Overfitting
– Interpretierbarkeit

Verlustfunktion

- Verlustfunktion definiert, was Fehler zwischen vorhergesagten Werten $\hat{y}$ und tatsächlichen Werten y angesehen wird. Quantifiziert, wie gut Modell $F(x)$ der Zielvariable annähert.
- Vorhersagen völlig falsch → Verlustfunktion hohen Wert, wenn gut → niedrigere.
- Variabel anwendbar; z.B. Regression (kontinuierliche Zielvariable) oder Klassifikation (kategoriale Zielvariable).
- Ziel → Verlustfunktion minimieren, z.B. mit Algorithmus.
- Regression:
  - $MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$, $MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}$
- Klassifikation:
  - 0-1 Verlust $Genaugikeit = \sum_{i=1}^{n} \begin{cases} 1: y_i \neq \hat{y}_i \\ 0: y_i = \hat{y}_i \end{cases}$
  - Negativ Log Likelihood (NLL)/Cross-Entropie für Wahr'vorhersage $\hat{p}$ von M Klassen: $NLL = \sum_{i=1}^{n}\left(-\sum_{c=1}^{M} y_{i,c} \log(\hat{p}_{i,c})\right)$

Verbesserung Vorhersage

- Schauen für jeden Datenpunkt, in welche Richtung verbessern könnten. Vorhersagen werden in diese Richtung "gedrückt".
- Praxis: wissen nicht, wo Optimum liegt. Nutzen Gradienten, um Richtung zu bestimmen. Für Vorhersage steht Gradienten nicht zur Verfügung.
- $f(x)$ modelliert Gradienten in Abhängigkeit Daten. $-Gradienten \approx f(x)$
- können für zu schätzenden Gradienten Residuen $r_i = y_i - \hat{y}_i$ einsetzen!
- Klassifikation Log Loss $\mathcal{L}(\hat{y}_i) = y_i \log(\hat{p}_i)$ → $\frac{d\mathcal{L}}{d\log(odds_i)} = -y_i + \hat{p}_i = r_i$

Stochastic Gradient Boosting

- Stabilere Optimierung und geringere Gefahr «Feststecken» lok. Optima.

Algorithmus:

1. Initial. Vorhersagemodell: $\hat{y} = F_0(x)$
2. Von $m = 0$ bis M
   a. Residuen berechnen: $r = y - F_m(x)$ und zufälliges Ziehen daraus (Bspw. 50 %)
   b. Residuen modellieren: $\hat{r} = f_m(x)$
   c. Neues Modell: $F_{m+1}(x) = F_m(x) + \lambda f_m(x)$

Parameter:

- M: Anz. Bäume (1000-10000)
- λ: Lernrate (0-1); kleine Rate reduziert Effekt einzelner Bäume, verbessert aber Vorhersage auf lange Sicht (braucht mehr Bäume). Kleinere Werte liefern bessere Performance. Mit kleineren Werten steigen Computerressourcen. Empfehlung 0.01-0.001.
- Baumtiefe: i.A. kleine Bäume, Empfehlung 2-5.
- Min Anz. an Beobachtungen in Endknoten der Bäume (z.B. 10)
- **Gefahr** Gradient Boosting: Neigt schnell zu **Overfitting**, da kein Strafterm!

```
ctrl <- trainControl(classProbs = TRUE, method = "cv", number = 5, summaryFunction = mnLogLoss)
gbm_fit <- train(Spe ~., data = trainDat, method = "gbm", trControl = ctrl, distribution = "multinomial", metric = "logLoss", verbose = FALSE, tuneGrid = data.frame(expand.grid(n.trees = c(200, 500, 1000), interaction.depth = seq(2, 5), shrinkage = 0.01, n.minobsinnode = 10)))
#Verteilung "bernoulli"/"gaussian"
confusionMatrix(predict(gbm_fit, newdata = testDat), testDat$Spe)
```

XGBoost Extreme Gradient Boosting

- erfolgreiche Implementation von Gradient Boosting
+ Bietet Regularisierungsoptionen, um Modellkomplexität zu reduzieren → verringert Overfitting.
+ Berücksichtigt bei Baumentwicklung reduzierte Anzahl Features (vergleichbar zum Random Forest).
+ Betracht Gradienten 2. Ordnung → hilft besser Richtung zur Minimierung Lossfunktion zu finden.

Parameter:

- M (nrounds): Anz. Bäume, mit CV optimieren.
- Baumtiefe (max_depth): Wie komplex einzelne Bäume max werden. 2-8
- λ (eta): Learning rate, wie gross sind die Schritte (wie gbm), 0.1-0.001, 0.01
- CP (gamma): Regularisierung Baumkomplexität, (vgl. Entscheidbaum), 0-0.1
- Variablenanteil (colsample_bytree): Wie viel Prozent der Variablen pro Baum genutzt werden
- min Samples je Node (min_child_weight): Mindestanzahl Beobachtungen in einer Partition
- Subsample: Prozent zufälligen Beobachtungen, die für Erstellung Baum benutzt werden.

```
ctrl <- trainControl(classProbs = TRUE, method = "none", summaryFunction = mnLogLoss)
#braucht one-hot encoding
xgb_fit <- train(Species ~., data = trainDat, 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))
confusionMatrix(predict(xgb_fit, newdata = testDat), testDat$Species)
```

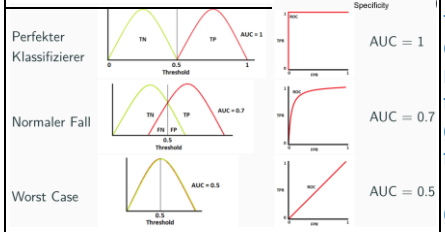
Modellevaluierung 2.0

Allgemein

Accuracy
+ Leicht zu verstehen und zu interpretieren; direkt anwendbar.
– Irreführend bei unbalancierten Klassen
– berücksichtigt keine Wahr'lichkeiten
• Accuaray ist nicht immer gutes Mass für Performanceevaluation Modell.
• Gibt bessere Masse vorhergesagten Wahr' nutzen: AUC und Log-Loss
• Wahrscheinlichkeit anstelle von Klassifikation: Beim Umwandeln von Wahr' zu Klassen verlieren wir Information!

ROV Kurve Allgemein

Werte liegen auf Winkelhalbierenden = Modell bringt keine Erkenntnis
Wenn Werte unterhalb liegen, dann wurde Zuordnung (T/F) vertauscht



AUC (Area Under ROC Curve)

• ROC-Kurve: (Receiver Operator Characteristic)-Kurve: Sensitivity (tpr) gegen 1-Specificity (fpr) für verschiedene Cut-Off-Werte
• AUC (Area under the Curve) Modellbeurteilung anhand der Fläche unter der Kurve (AUC) → zeigt, wie gut Modell Klassen unterscheiden kann (unabhängig Schwellenwert). Höherer AUC-Wert → besseres Modell
+ Effektiv bei unbalancierten Klassen
+ berücksichtigt Rangfolge Vorhersagen
– Nicht empfindlich gegenüber den absoluten Werten der Vorhersage
– Irreführend sein bei sehr unbalancierten Datensätzen

```
ctrl <- trainControl(classProbs = TRUE, method = "cv", number = 10, summaryFunction = multiClassSummary)
iris.knn <- train(Species ~., data = trainDat, method = "knn", trControl = ctrl, tuneLength = 3, metric = "AUC")
```

ROC-Kurve Vergleich zweier Modelle

```
library(pROC) #levels/Klassen im Dat sind hoch und tief
res.glm <- train(crime ~., data = trainDat, method = "glm", family = "binomial")
ctrl <- trainControl(method = "none")
r.tree <- train(crime ~., data = trainDat, method = "rpart", trControl = ctrl, tuneGrid = data.frame(cp = 0))
glm_pred <- predict(res.glm, testDat, type = 'prob')
tree_pred <- predict(r.tree, testDat, type = 'prob')

glm_roc <- roc(testDat$crime ~ glm_pred$'hoch')
tree_roc <- roc(testDat$crime ~ tree_pred$'hoch')
glm_roc$auc;  tree_roc$auc  #für Erhalt AUC Werte

plot(glm_roc, las = 1)
lines(y = tree_roc$sensitivities, x = tree_roc$specificities, col = "red", lwd = 2, lty = 2)
legend("bottomright", legend = c(paste("glm auc:", round(glm_roc$auc, 2)), paste("tree auc:", round(tree_roc$auc, 2))), lwd = 2, lty = c(1, 2), col = c("black", "red"), bty = "n")
```

Güte einer Wahrscheinlichkeit (Wahr' = Wahrscheinlichkeit)

Statt Wahr' mittels Grenzen in Klassen umzuwandeln, kann Güte der geschätzten Wahr'keiten direkt beurteilt werden.

Kalibrierungskurve

• Grafik, die beurteilt, ob Modell gut kalibriert ist. Modell ist gut kalibriert, wenn vorhergesagte Wahrscheinlichkeit mit tatsächlichen Häufigkeit Ereignisse, die es vorhersagt, übereinstimmen. D.h. wenn Modellvorhersage 80 % Wahr' zutrifft, sollte Ereignis, das es vorhersagt, tatsächlich in ca. 80 % der Fälle eintreten.
• Kalibrierungskurve vergleicht geschätzte Wahrscheinlichkeiten mit echten Frequenz in beobachteten Daten.
• Wahrscheinlichkeiten werden in Gruppen eingeteilt und mit den jeweiligen Mittelwerten gearbeitet.
• Vorhergesagte Wahrscheinlichkeiten der Klasse m, $\hat{p}(y = Klasse\ m|x)$, werden in Gruppen eingeteilt (Standard = 10 Gruppen)
• x-Achse sind gemittelten, vorhergesagten Wahrscheinlichkeiten der Beobachtungen pro Gruppe.
• y-Achse sind Anteile Beobachtungen in Gruppen, die zur Klasse m gehören.
• Perfekte Kalibrierung: Alle Punkte liegen auf Winkelhalbierenden (oberhalb = Modell überschätzt W'keiten, unterhalb = Modell unterschätzt W'keiten).

```
library(predtools)
calibration_plot(data, obs = 'y', pred = 'prob')
```

Log Loss

• Metrik, welche Güte der Kalibrierung misst. (häufig verwendeten Metriken für Klassifikation) Auch als Cross-Entropie oder negative LogLikelihood bekannt.
• Gibt an, wie nahe Vorhersagewahrscheinlichkeit an tatsächlichen Klasse liegt. Je mehr Vorhersagewahrscheinlichkeit abweicht, desto höher ist Log-Loss-Wert.
• Log-Loss ist **datenspezifisch**! Zwei Modelle, die auf verschiedenen Datensätze angewendet werden, können nicht verglichen werden.
• N = Anzahl Beobachtungen, M = Anzahl Klassen
• y_{ij}: 1 wenn Beobachtung zur Klasse j gehört, sonst 0
• $\hat{p}_{ij}$: vorhergesagte Wahrscheinlichkeit, dass Beobachtung zur Klasse j gehört
• $LogLoss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(\hat{p}_{ij})$
+ Sinnvoll für Wahr'schätzungen und nützlich für Optimierung
– Schwer zu interpretieren und ausreiseranfällig

```
ctrl <- trainControl(classProbs = TRUE, method = "cv", number = 10, summaryFunction = multiClassSummary)
iris.knn <- train(Species ~., data = trainDat, method="knn", trControl = ctrl, tuneLength = 3, metric = "logLoss")
```

Log Loss Vergleich zweier Modelle

```
library(pROC) #levels Dat hoch und tief
res.glm <- train(crime ~., data = trainDat, method = "glm", family = "binomial")
ctrl <- trainControl(method = "none")
r.tree <- train(crime ~., data = trainDat, method = "rpart", trControl = ctrl, tuneGrid = data.frame(cp = 0))
glm_pred <- predict(res.glm, testDat, type = 'prob')
tree_pred <- predict(r.tree, testDat, type = 'prob')

glm_pred$obs <- testDat$crime
tree_pred$obs <- testDat$crime

mnLogLoss(glm_pred, lev = levels(testDat$crime))
mnLogLoss(tree_pred, lev = levels(testDat$crime))
```

Welche Metrik sollten Sie verwenden? ⇒ Verwenden Sie mehrere Metriken, um ein umfassendes Bild der Modellperformance zu erhalten.

Ablauf Klassifizierung

- Datenaufbereitung: Kontrolle Balancierung Daten, Zielvariable als Faktor hinterlegt mit der richtigen Reihenfolge (Faktorstufe von Interesse an erster Stelle)?
- `Crime$crime <- factor(Crime$crime, levels = c("tief", "hoch"))`
- Variablenkategorien? Numerisch, Factor, binär??? → korrekt hinterlegt
- Verteilung numerische Variablen? → Skalieren??? `boxplot(dat)`
- `pairs(trainDat[, 2:4], col = c("red", "blue")[trainDat$default])` #Korrelation zwischen den einzelnen Variablen als Plot darstellen

Feature Engineering

Allgemein

- Prozess, in welchem Rohdaten in Features umgewandelt werden, die in Algorithmen des maschinellen Lernens, z. B. in Vorhersagemodellen, verwendet werden können.
- Je nach Format Rohdaten gibt es unterschiedliche Techniken.
- Um gut Feature erstellen zu können, braucht man Fachwissen und muss die Daten verstehen!
- Feature-Engineering hat i.d.R. massiv grösseren Einfluss auf Qualität Ergebnisse als Algorithmus, welchen man verwendet!
- ML Projekten Datenaufbereitung und Feature-Engineering rund 80% der Zeit beanspruchen.

Textdaten

- Tokenisierung → Text aufteilen
- Stoppwörter entfernen → Uninformative Wörter entfernen
- Stemming oder Lemmatisierung → Wörter basierend auf Regeln oder auf Struktur Sprache (Wörterbücher) auf Stammform zurückführen
- Document-Term Matrix → Tabelle mit Häufigkeiten der Wörter
- Entfernung weiterer seltene oder sehr häufige Wörter.
- Bag-of-Words → Vektor mit Anzahl Beobachtungen für jedes Wort in Wortliste

Erweiterungen:

- Transformation des Bag-of-Words in TF-IDF
- word2vec: Ansatz, um automat. Zusammenhänge zwischen Wörtern und Bedeutung aus Textkorpus zu lernen. Token (Wörter/Unterwörter/Multiwort-Ausdrücke) lassen sich als Zahlen ausdrücken, um sie für Modelle zu verwenden.
- Vortrainierte Transformer-Modelle

Bilddaten

- Für Klassifikation Bildern wird Deep Learning DL verwenden, welche fähig sind relevante Merkmale automatisch zu lernen.
- Vortrainierte Modelle nutzen, um Feature aus Bildern zu extrahieren. Feature sind Vektoren, welche für klassische ML Modelle verwenden.

Zeitreihen

- Frequenzen extrahieren mittels Fourier-Transformation.
- Können DL Methoden, welche Feature selbstständig lernen sehr erfolgreich sein.

Mehrere Messungen

- Hat man zu einer Beobachtung mehrere Messungen der gleichen Variable (zum Beispiel kurze Zeitreihe) kann es sinnvoll sein, Kennzahlen (min, max, Anzahl Peaks, Median, mean) zu extrahieren und diese als Feature für die Modellierung zu verwenden.
- I.d.R. je mehr Domainwissen für Feature Engineering verwenden, umso besser.

Fehlende Daten

Allgemein

- sind häufiges Problem bei Data Science Projekten.
- Gründe sind vielfältig und häufig nicht unter unserer Kontrolle
- fehlende Daten müssen bei Datenaufbereitung adressieren, da Modelle damit i.d.R. nicht umgehen können.
- Gibt keine richtige Handhabung, aber Möglichkeiten, die je Anwendungsfall verwenden können.

➔ Wenn fehlende Werte mit falschen Daten auffüllen, fügen wir einen Bias hinzu.

Arten von fehlenden Daten

- **Missing completely at Random (MCAR):** keinen Zusammenhang zwischen Fehlen Datenpunkts mit restlichen Werten im Datensatz. Fehlende Daten sind zufällige Teilmenge.
- **Missing at Random (MAR):** Fehlende Werte haben Zusammenhang mit beobachteten Daten, aber nicht mit fehlenden Wert selbst (z.B. Umfrage über psychische Störungen bei Männern und Frauen. Männern melden Depressionsstatus vielleicht weniger, das hat aber nichts mit dem Grad ihrer Depressionen zu tun).
- **Missing Not at Random (MNAR):** Beziehung zwischen fehlendem Wert und dessen Wert (Z.B. Depressionsumfrage; Befragte mit höheren Depressionswerten füllen Umfrage aufgrund ihres Depressionsgrades häufiger nicht aus).

Umgang mit fehlenden Daten

Ansatz: Complete Case Analysis (nur vollständige Beobachtungen verwenden)

- Entfernen von Beobachtungen (Zeilen): train(..., **na.action = na.omit**)
- Wenn einzelnen Featuren (Spalten) sehr viele Daten fehlen (z.B. 60%) kann es auch Sinn machen ganze Spalte zu entfernen.
- In den Fällen MCAR und MAR ist dies in der Regel okay. Im Fall MNAR kann dies zu Bias im Modell führen.
- Sollte versuchen zu ergründen, wieso Werte fehlen. I.A. ist Weglassen Daten nicht beste Lösung, da wir potenziell nützliche Informationen verlieren.

⇒ Besser zu **imputieren**.

Imputation fehlende Daten

Imputation: Statistische Verfahren, um fehlende Daten in Datenmatrix zu vervollständigen. Aktuelles Forschungsfeld. Möglichkeiten:

- Einsetzen konstanten Wertes, der sich von allen anderen Werten unterscheidet (z.B. neue Faktorstufe für NAs bei kategoriellen Variable)
- Einsetzen Mittelwert, Median oder Moduswert für diese Spalte (**nicht zu empfehlen**)
  - Mittelwert: nicht optimal, zerstört Struktur, reduziert Varianz
  - Zufallswert: zerstört Struktur, Beibehalt Varianz
  - Nächstem Nachbar: Benachbarte Punkte müssen über andere, vorhandene Variablen gefunden werden.
- Schätzen Wert mit eigenen prädiktiven Modell
- Mehrfache Imputation

Nützliche Modelle für Imputation

- KNN: kNN im Paket VIM. Imputation basierend auf Variation Gower-Distanz für numerische, kategoriale, geordnete und semikontinuierliche Variablen.
- Random Forest: z.B. missForest im Paket missForest: geeignet um gemischte Daten mit komplexen Interaktionen und nichtlinearer Beziehungen zu imputieren.
- Funktion preProcess() für Imputation:
  - "knnImpute" kNN-Klassifizierer → Daten werden automatisch zentriert und skaliert
  - "bagImpute" Bagged trees
  - "medianImpute" Imputation mit Median Trainingsdaten
- Ansatz muss vorgängig ans Training durchgeführt werden:

```
imp <- preProcess(data, method = "knnImpute")
data_miss_imp <- predict(imp, data)
```

Multiple Imputation

- Warnung: Imputation werden so behandelt, als seien es Messungen → Unterschätzung der Unsicherheit
- ⇒ Multiple Imputation (mehrere Datensätze, welche zum Trainieren Modell verwendet werden, analysieren der Varianz).

Idee

- Erzeugen mehrerer imputierter Datensätze
- Passen Modell an jeden Datensatz an
- Zusammenführen der Ergebnisse der verschiedenen Modelle
- Beispiel: MICE (Multivariate Imputation by Chained Equations): Imputiert jede Variable stochastisch unter Verwendung eines Regressionsmodells. R-Paket mice

Support Vector Machine (SVM)

Idee/Ansatz

- Flexibles Klassifikationsverfahren mit guter Performance. Bspw. geeignet, wenn Anzahl Features sehr gross oder Anzahl Beobachtungen limitiert ist.

Grundidee:

- Jede Beobachtung → p-dimensionaler Vektor
- SVM konstruiert Hyperebene (Hyperplane), um 2 Klassen zu trennen.

Maximale Margin Klassifikator

- Maximale Margin Klassifikator wählt Ebene, welche Daten so trennt, dass Abstand von Hyperebene zum nächsten Trainingspunkt maximal ist.
- kleine Margin → Gefahr, dass Trainingsdaten überanpasst werden (Overfitting)
- grosse Margin → Erwartung, dass auch Testdaten korrekt abgebildet werden

Support Vektors

- Support Vektor: Trainingsdaten, welche nächsten an Trenngrenze liegen, bestimmen Hyperebene
- Restliche Trainingsdaten tragen nicht zur Festlegung der Grenze bei!
- Haben einen Abstand von $\leq M$ zur Hyperebene

Hyperplane bestimmen

- Hyperplane definiert: $\beta_0 + \beta_1 \cdot X_1 + \cdots + \beta_p \cdot X_p = 0$
- Punkte $O \rightarrow \beta_0 + \beta_1 \cdot O_1 + \cdots + \beta_p \cdot O_p > 0$ und $O^* \rightarrow \beta_0 + \beta_1 \cdot O_1^* + \cdots + \beta_p \cdot O_p^* < 0$ liegen auf unterschiedlichen Seiten der Hyperebene ⇒ Notationstrick: Zielvariable mit $y = \pm 1$ definieren.

Optimierungsproblem:

- $y_1 \cdot (\beta_0 + \beta_1 \cdot x_1 + \cdots + \beta_p \cdot x_p) \geq M$ für alle $i = i, \dots, N$
- Gesucht β-Vektor, welcher M maximiert. Für eindeutige Lösung benötigen wir Nebenbedingung: $\sum_{j=1}^{p} \beta_j^2 = 1$ da ansonsten jede Multiplikation β's mit Konstante ein Lsg

Nicht eindeutig trennbare Daten

- Vielfach sind Daten mit linearen Grenze nicht trennbar.
- Idee: Erlauben einige Fehlklassifizierungen (Support Vector Classifier) $y_i \cdot (\beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_p \cdot x_{ip}) \geq M \cdot (1 - \epsilon_i)$, ϵ sind Slack-Variablen, welche Beobachtungen erlauben auf falschen Seite Margin oder Hyperebene zu liegen.
- $\epsilon_i \geq 0$, $\sum_{i=1}^{n} \epsilon_i \leq \frac{1}{C} \Rightarrow C > 0$ (Cost): Tuningparameter, entspricht Kosten Fehlklassifikation in Trainingsdaten

Voraussetzungen/Vorgehen

-

Kleines C:

- Mehr Fehlklassifikationen da geringe Kosten → ermöglicht grössere Margin
- Gefahr Übergeneralisierung Trainingsdaten
- Wenn Trainingsdaten leicht ändern, neigen Grenzen dazu, stabil zu bleiben
- Klassifikator mit geringer Variabilität und grossem Bias

Grösseres C:

- Weniger Fehlklassifikationen da grössere Kosten → kleinere Margin
- Gefahr Überanpassung an Trainingsdaten
- Wenn die Trainingsdaten leicht ändern, können Grenzen stark variieren
- Klassifikator mit grosser Variabilität und kleinem Bias

⇒ mehr Fehlklassifikationen zulassen, dafür aber grössere Margin zu erhalten.

Nichtlineare Grenzen

- Manchmal lineare Hyperebene nicht funktionieren, egal was C gewählt wird.

Feature Erweiterung Idee:

- Featureraum vergrössern, d.h. weitere erklärende Variablen hinzufügen, indem Transformationen der ursprünglichen Variablen als zusätzliche Variablen verwendet werden (z.B. $X_1^2, X_1^3, X_1X_2, X_1X_2^2$) → Übergang von p- zum M-dimensionalem Raum ($M > p$).
- Anwendung SVM-Klassifizierer auf vergrösserten Raum → ergibt nicht-lineare Entscheidungsgrenzen in originalen Dimension.

Kernel Trick ⇒ SVM

- Polynome manuell hinzufügen, wird schnell schwierig handelbar. → Bessere Lösung, um Nichtlinearität einzubinden sind Kernels.
- Kernel → Funktion, die Ähnlichkeit zwischen 2 Beobachtungen quantifiziert.
- Ansatz Skalarprodukt $\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij}, x_{i'j}$
- Alternative Schreibweise für Linear SV Klassifizierer $f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$ Um Parameter α_i und β_0 zu schätzen braucht man $\binom{n}{2}$ Skalarprodukte zwischen allen Paaren an Trainingsbeobachtungen, wobei meisten $\hat{\alpha}_i$ 0 sind. Relevant ist Supportset S (Support Vectors).
- Daraus SV Klassifizierer: $f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$

SVM: Radialer Kernel

- $K(x_i, x_{i'}) = exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)$
- γ: Hyperpara, definiert Einflussbereich Trainingsdaten
- kleines γ grosser Einflussbereich (Grenzen linearer)
- grosses γ kleiner Einflussbereich (flexible Grenzen)
- → Schätzung durch Kreuzvalidierung

SVM: Mehr als 2 Klassen:

- **OVA**: One versus All: Fit K verschiedene SVM-Modelle: jeweils Klasse k gegen den Rest. Klassifiziere eine neue Beobachtung in die Klasse mit der grössten Distanz zur Hyperebene.
- **OVO**: One versus One: Fit alle $\binom{K}{2}$ paarweise SVM-Modelle. Klassifiziere Beobachtung in Klasse, welche meisten paarweisen Vergleiche gewinnt.

→ Welche Ansatz wählen? Falls K nicht zu gross → OVO

Pro (+) / Contra (-)

+ Klassifikationsmethode mit hoher Vorhersageperformance (Erweiterung auf Regressionsprobleme möglich).
+ Funktioniert auch mit relativ wenigen Trainingsdaten und vielen Features.
+ Auch bei mehr als 2 Klassen anwendbar
+ Kernel-Trick (neu konstruierte Features) erlaubt es, nichtlineare Trenngrenzen zu finden.

– Separierung Klassen durch Hyperebene und Margin basierend auf SV. Übrige Trainingsdaten tragen nichts zur Hyperebene bei.
– Liefert standardmässig keine Wahrscheinlichkeitsvorhersagen.

R-Code [Default]

Linearer Kernel

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

svm_lin <- train(y ~ ., data = trainDat, method = "svmLinear2", trControl = ctrl, tuneGrid = data.frame(cost = c(0.1, 1)))
svm_lin $bestTune #Ausgabe C
```

Bemerkung: Daten werden in Funktion defaultmässig standardisiert. Was auch wichtig ist.

```
confusionMatrix(predict(svm_lin, testDat) , reference = testDat$y)
```

Gaussian bzw. radialer Kernel

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

svm_rad <- train(y ~ ., data = trainDat, 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")
```

Polynomieller Kernel

```
svm_poly <- train(y ~ ., data = trainDat, method = "svmPoly", tuneGrid = expand.grid(degree = 1:3, scale = c(0.1, 1, 2, 5, 10), C = c(0.1, 1, 5, 8, 10, 15)), trControl = ctrl)
svm_poly$bestTune
plot(svm_poly)

confusionMatrix(data = predict(svm_poly, testDat), reference = testDat$y)
```

Deep-Learning DL

Was neu an DL?

- Traditionelle ML extrahieren handgefertigter Merkmale und verwenden Merkmale zum Modell Training.
- Deep-Learning DL (End-to-End-Ansatz) Deep-Neuronale Netze starten mit Rohdaten und lernen während Training, geeignete hierarchische Merkmale zu extrahieren, um diese für Klassifizierung zu verwenden.

Einordnung Modelle

- Regelbasierte Systeme: Input → handdesigntes Programm → Output
- Klassisches Maschine Learning: Input → handdesigntes Features → Mapping von Features → Output. Extrahieren von handgefertigten Featuren, welche zum Modelltraining (z.B. SVM, RF) verwendet werden.
- Representation/Feature Learning (z.B. Deep Learning): Input → Features → Mapping von Features → Output
- Deep Learning: Ausgehend Rohdaten lernt Modell selbstständig, geeignete Feature zu extrahieren, welche der Vorhersage dienen. Für grössere Aufgaben ist oft grössere Rechenkapazität notwendig (GPU/Server-Lösung).

Power of Deep Learning

- Ziel neuronalen Netzes ist Funktionen $y = f(x)$ finden, die Attribute x auf Ausgabe y abbilden. Wieso funktionieren das? ⇒ Universal Approximation Theorem: Sei f beliebige kontinuierliche Funktion auf $[0,1]^d$ und φ eine beliebige sigmoidale Aktivierungsfunktion, dann lässt sich f uniformverteilt durch endliche Summe Form $\hat{f}(x) = \sum_{j=1}^{m} w_j^{(2)} \varphi\left(\mathrm{w}_\mathrm{j}^{(1)T} x + w_{j,0}^{(1)}\right)$ approximieren.
- Neuronales Netzwerk mit mindestens einem Hidden Layer und einer ausreichenden Anzahl von Neuronen ist in der Lage jede stetige Funktion auf einem kompakten Intervall mit einer beliebigen Genauigkeit anzunähern.
- Theorem macht keine Aussage über Effizienz Training oder Anzahl der benötigten Neuronen. Es besagt lediglich, dass ein solches Netzwerk theoretisch existiert. In der Praxis kann es jedoch Herausforderungen geben, die mit Overfitting oder anderen Faktoren zusammenhängen können.

FCNN

Fully Connected Neural Networks (FCNN)

- Neuronale Netze: Modell aufbauend auf einer Netzwerkstruktur mit künstlichen Neuronen.
- Simple Neural Network besteh aus einem Hidden Layer, während Deep Learning Neural Network mehrere Hidden Layers hat.
- Grosser Aufschwung, da fähig, grosse Mengen unstrukturierten Daten (z.B. Bilder oder Text) besonders gut auswerten zu können.

Klassifikationsnetzwerk

- w: Gewicht, b: Bias
- x: Inputvektor
- $z_1 = b + \sum_{i=1}^{n} x_i \cdot w_{i1} = b + x_1 \cdot w_{11} + x_2 \cdot w_{21} + \cdots + x_n \cdot w_{n1}$
- Output $\hat{y}$ letzten Layers wird durch Softmax-Funktion in Wahrsch'vektor $\hat{p}$ für verschiedenen Klassen transformiert: $\hat{p}_i = \frac{e^{\hat{y}_i}}{\sum_{j=1}^{m} e^{\hat{y}_j}}$ (Summe 1) $m =$ Anzahl Klassen
- Nicht-lineare Aktivierungsfunktion:
  - Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$ (s förmig)
- ReLU $R(z) = \max(0, z)$ (lineare Gerade)

Netz trainieren

- Gewichte Netzwerk sind Modellparameter → Deep Learning Modelle haben viele Parameter (Moderne Netzwerke können Billionen (10^9) Gewichte haben.)
- Anpassung Gewichte durch Optimierung Loss Funktion mit Gradientenverfahren.
- Steigung Lossfunktionen sind lokale Gradienten. (Ableitung)

Iterativer Prozess

- Schrittweises trainieren mit zufälligen Sample aus Trainingsdaten (**Batch**).
- Wird wiederholt, bis alle Trainingsdaten einmal durch sind (**Epoche**).
- Repetiert für mehrere Epochen → Je mehr Beispiele Netz für Training bekommt und je öfter es diese gesehen hat, desto besser wird die Performance.

Ablauf Training

- Zufällige Indizierung Gewichte für erste Runde.
- Forward Pass: Input-Werte werden ins Netz eingespiesen und durch Netz propagiert → Modelloutput
  - Vergleich Output mit bekannten Ergebnissen des Datensatzes.
  - Verlustfunktion (z.B. Klassifikation: Log-Loss/Kreuzentropie) $Loos = -\frac{1}{N}\sum_{i=1}^{N} y_I(\log(\hat{y}))$
- Backward Pass: Jedes Gewicht und jeder Bias Term wird kleines Stückchen in Richtung angepasst, die Fehler kleiner macht. (→ Gradientenverfahren)
- Grösse Anpassung wird über Lernrate (Hyperparameter) gesteuert. Ist Rate zu gross, kann Minimum Verlustfunktion verpasst werden. Ist Rate zu klein, dauert Training lange).
- Forward und Backward Pass werden als iterativer Prozess wiederholt.

Lernrate

- ist ein wichtiger Parameter im Deep Learning.
- hat enormen Einfluss auf Modellanpassung. Deep Learning Modell kann optimal auf Problem zugeschnitten sein und dennoch kann alles schief gehen. → zu gross oder zu kleine Lernrate

Overfitting

- Wenn Netz während Training alle Trainingsdaten sehr oft gesehen hat, kann sein, dass es Daten eher auswendig lernt, statt abstraktes Konzept zu lernen. → Overfitting
- Um abstrahierendes Modells zu erstellen, werden Training Daten in Training und Validation aufgeteilt. Die Gewichte werden jeweils nur auf Trainingsdaten angepasst. Optimierung folgt auf Trainings- und Validierungsdaten. → Finaler Test an ungesehenen Testdaten, welche gar nicht für Training verwendet wurden.
- Dropout: Während Training werden z.B. 50 % Neuronen eines Layers einfach abgeschaltet. Die Abschaltung erfolgt pro Durchlauf jeweils auf zufälliger Basis. Alle Neuronen des Layers werden damit gezwungen, weniger spezielle und mehr abstrakte Konzepte zu lernen, um trotz reduzierten Anzahl gut zu performen.
- Für finale Modell (für Vorhersagen) stehen schlussendlich wieder alle Neuronen zur Verfügung. Ergebnisse werden i.d.R. robuster und besser im Hinblick auf neue unbekannte Daten.

Keras Workflow

Workflow

1. Definieren Netzwerk → Layerweise Definition Netzwerk
2. Kompilieren Netzwerk → Loss und Optimierer definieren
3. Fitten Netzwerks → Trainieren Modell mit Trainingsdaten
4. Evaluieren Netzwerk mit Testdaten
5. Anwendung finales Modell

1. Definieren Netzwerk

```
library(keras3)
model <- keras_model_sequential()
layer_dense(model, units = 16, input_shape = c(4), activation = 'relu')
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)
```

- units: Anzahl Neuronen im Layer
- input_shape: Dimension Inputs (nur im ersten Layer notwendig)
- activation: Aktivierungsfunktion (relu, sigmoid oder softmax)
- rate: Anteil Auszulassende Gewichte
- Alternativ Modellaufbau %>% dplyr.

2. Kompilieren Netzwerk

```
model <- compile(model, optimizer = optimizer_adam(learning_rate = 0.001), loss = 'categorical_crossentropy', metrics=c('accuracy')) #AUC
```

- optimizer: Adaptive Moment Estimation (Adam; optimizer_adam) oder Stochastic Gradient Descent (SGD; optimizer_sgd).
- learning_rate: Lernrate.
- loss: categorical_crossentropy entspricht Log-Loss. Bei Regression mean_squared_error

Preprocessing Daten

```
library(caret)
index <- createDataPartition(iris$Species, p = 0.7, list = F)
index <- sample(index)
trainDat <- as.matrix(iris[index, 1:4])
testDat <- as.matrix(iris[-index, 1:4])
preP <- preProcess(trainDat, method = c("center", "scale"))
train_scale <- predict(preP, trainDat)
test_scale <- predict(preP, testDat)
train_y <- as.numeric(iris[index, 5])-1
train_labels <- to_categorical(train_y)
test_y <- as.numeric(iris[-index, 5])-1
test_labels <- to_categorical(test_y)
head(train_labels, n = 2)
```

- Input benötig numerische Matrizen
- Empfehlung Daten skalieren (verhindert Stabilitätsproblem, sorgt effiziente Gewichtanpassung, Regularisierungseffekt)
- One Hot Encoding für kat. Daten notwendig (Zielvariable).

3. Fitten Netzwerk

```
history <- fit(model, x = train_scale, y = train_labels, epochs = 100, batch_size = 32, validation_split=0.2, verbose=0)
plot(history)
```

- epochs: Epochen zum Trainieren Modell
- batch_size: Anzahl Stichproben pro Gradienten-Update
- validation_split: Anteil Daten, welche nur Validieren und nicht zum Fitten verwendet werden.
- verbose: 0 Silent, 1 Progressbar, 2 eine Linie pro Epoche.

4. Evaluation Netzwerk

```
evaluate(model, x = testDat, y = test_labels, verbose = 0)
```

5. Anwendung

```
predict(model, testDat[1:10, ])
```

Netzwerk Architekturen

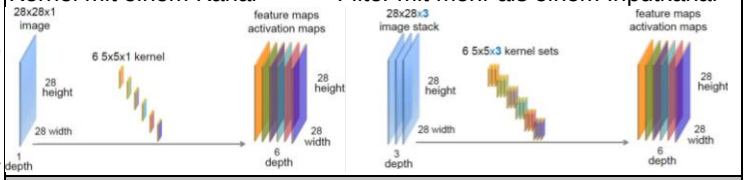
- Bis jetzt auf Fully Connected Neurale Netzwerke (FCNN) beschränkt.
- Diese eignen sich für tabellarische Daten. Reihenfolge Eingabe spielt keine Rolle.
- Geordnete Daten (z.B. Bilder, Zeitreihen) werden «geflattet». Geordnete Information geht verloren. → Gibt bessere Alternativen.
- Für Bilder sind Convolutional Neural Networks (CNNs) geeignet, (verg. ImageNet-Competition)
- Während FCNN Gewichte für jedes einzelne Pixel haben, werden bei CNNs Filter mit geteilten Gewichten verwendet, welche über das ganze Bild gleiten.
- Weitere Architekturen: z.B. RNNs (Rekurrentes neuronales Netz) bzw. LSTMs (Long short-term memory) für Zeitreihen.

Beispiel CNN-Layer

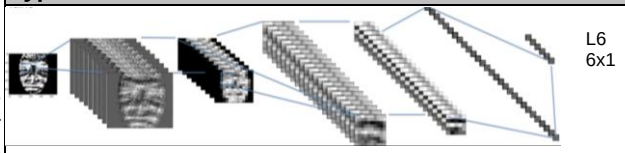
- An jeder Position des Inputs werden die gleichen Gewichte verwendet.
- Soll die Grösse des Bildes erhalten bleiben, kann z.B. Null-Padding angewendet werden (Rand wird um 0-Werte erweitert).
- Maxpooling: Maximum in Filtergrösse; reduziert die Grösse der Bilder → weniger Gewichte.

Convolution Layer mit mehreren Filtern

Kernel mit einem Kanal — Filter mit mehr als einem Inputkanal



Typisches klassisisches CNN-Modell



Bilden 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)
```

- filters: Anzahl Filters
- kernel_size und pool_size: Filtergrösse

Tricks bei der Anwendung

- Deep Learning braucht nicht viel Preprocessing, aber Standardisierung Daten hilft oft. Achtung die Aktivierungsfunktion muss zum Wertebereich der Daten passen.
- Haben nie genügend Daten für Modelltraining → Data Augmentation. Dabei werden, um Datenmenge zu vergrössern, leicht modifizierte Kopien von bereits vorhandenen Daten erstellt (Bild rotieren).
- Was kann man bei limitierten Daten, Computerressourcen und Zeit machen? → Verwenden von vortrainierten Modellen zur Merkmalsgenerierung.
  - Neu trainiert wird nur der/die letzte/n Layer/s (Transfer Learning). Alternative mit einfacherem parallelen Modell (Side Learning).
  - Verwenden eines klassischen Klassifikator (SVM) zur Klassifikation basierend auf den extrahierten Features.
  - Insbesondere für Bilderkennung stehen viele vortrainierte Modelle zur Verfügung. Verwendet werden sollte ein Modell, das für einen möglichst ähnlichen Task entwickelt wurde.

Stand der Arbeit

SW	Vorlesung	AB/Aufgabe	R-Skript	ZF
1	☒	☒	~~☐~~	☒
2	☒	☒	☒	☒
3	☒	☒	☒	☒
4	☒	☒	☒	☒
5	☒	☒	☒	☒
6	~~☐~~ Ostern	☒	☒	~~☐~~
7	☒	☒	☒	☒
8	☒	☒	☒	☒
9	☒	☒	☒	☒
10	☒	☒	☒	☒
11	☒	☒	☒	☒
12	☒	☒	☒	☒
13	☒	☐	☐	☒
14	~~☐~~	~~☐~~	~~☐~~	~~☐~~

Modulauflagen

Obligatorische Leistungsnachweise für die Modulbewertung:

- Praktische Arbeiten, 40%
- Endprüfung (90 Minuten), 60%
- Die Modulendnote setzt sich somit zusammen aus: 0.4 x Praktische Arbeiten + 0.6 x Endprüfung.
- Für die praktische Arbeit müssen 4-mal im Semester die erlernten Methoden auf einen eigenen Datensatz angewendet und in einem kurzen Bericht dokumentiert und interpretiert werden. Die Berichte werden bewertet (Gewichtung jeweils 8%). Zudem muss einmal im Semester eine vorgestellte Methode inklusive Übungsaufgabe zusammengefasst und der Klasse präsentiert werden (Gewichtung 8%). Die Tasks können in 2er-Gruppen bearbeitet werden. Details zu den Semestertasks sind auf Moodle und in den Vorlesungsunterlagen ausgeführt.

SEP-Hilfsmittel

- Erlaubte Hilfsmittel für Endprüfung: Eine zehnseitige Zusammenfassung auf Papier ~~und ein (elektronisches) File mit R / Python Befehlen~~. Laptop mit R und/oder Python. Alle benötigten Bibliotheken müssen installiert sein. Es sind keine weiteren Hilfsmittel und elektronischen Kommunikationsmittel wie z.B. Mobiltelefone und WLAN erlaubt.

Statistisches Data Mining – Zusammenfassung

Allgemeines

- Data Mining ist die Anwendung geeigneter Methoden zur (automatisierten) **Entdeckung** von **Strukturen** und **Zusammenhängen** in **grossen Datenmengen**.
- Die Datenbestände werden dabei nach Regelmässigkeiten, Mustern und Strukturen, Abweichungen, jeglicher Art von Zusammenhängen und gegenseitigen Beeinflussungen untersucht.
- Data Mining ist durch den Begriff Data Science in den Hintergrund gerückt. Die Tätigkeit Data Mining ist allerdings nur ein Teil von Data Science und wird dort oft unter dem Begriff Data Analytics subsummiert.
- Data Mining hat grosse Ähnlichkeiten zur statistischen Datenanalyse und verwendet u.a. auch Methoden des Maschinellen Lernens.

Data Mining

Ziele:

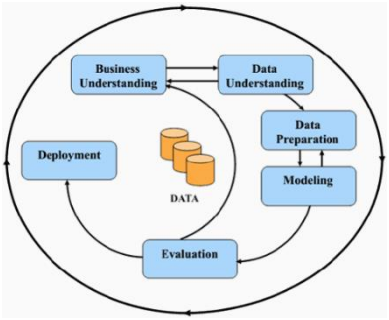
- Daten verstehen/visualisieren, Muster finden und klassifizieren → Querverbindungen/Trends erkennen.
- Vorhersagen machen → Fokus liegt auf guten Vorhersagen. Das Modell ist im Allgemeinen nicht so wichtig.
- Hypothesen formulieren.

Spezielle Merkmale:

- Anwendung statistischer Methoden auf **grosse Datenbestände** («Big Data»).
- Von einer Beobachtungseinheit (z.B. ein Patient) können zig-tausend Merkmale erhoben werden, so dass die **Anzahl** der **Merkmale** die **Anzahl** der **Beobachtungseinheiten übersteigen** kann.
- Charakteristisch, dass Daten **nicht** für bestimmten **Zweck** - z.B. Überprüfung Hypothese - erhoben wurden.
- **Methoden anwendbar, wenn traditionelle Techniken versagen** (d.h. Datenmenge zu gross, zu heterogen oder Dimensionalität zu hoch)

Data Mining Prozess

Data Mining Prozess für datengetriebene Lernen. Cross-industry standard process data mining (CRISP-DM) ↓



- Business Understanding: Was will der Auftraggeber? Wer sind die Empfänger? Was sind die Ressourcen?
- Data Understanding: Verfügbarkeit, Meta-Daten, Struktur
- Data Preparation: Umgang mit fehlenden Werten und Ausreissern, Standardisierung, Transformation, Aggregieren und Selektion von Variablen.
- Modeling: Auswahl des Data Mining Verfahrens, Prognosen, Auswertungen
- Evaluation: Statistische Evaluierung des Modells, qualitative Beurteilung
- Deployment: Umsetzung in die Praxis
- Modellierung/Evaluation stehen im Kurs im Zentrum, auch wenn Data Understanding/Preparation 80 % Zeit ausmachen.

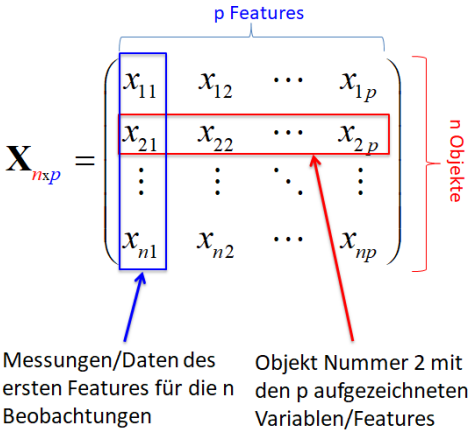
Daten sind der Schlüssel

Prozess	**Was kann falsch gehen?** → garbage in, garbage out (GIGO)
• Business Understanding	Problem nicht verstanden
• Data Understanding	Falsche Daten
• Data Preparation	Fehlende Daten
• Modeling	Umgang mit grossen Datenmengen
• Evaluation	Falsche Methoden / Annahmen
• Deployment	Kommunikation

- Data Mining / Machine Learning lernt direkt von Daten. Um es gut zu machen, brauchen i.d.R. viele Daten!

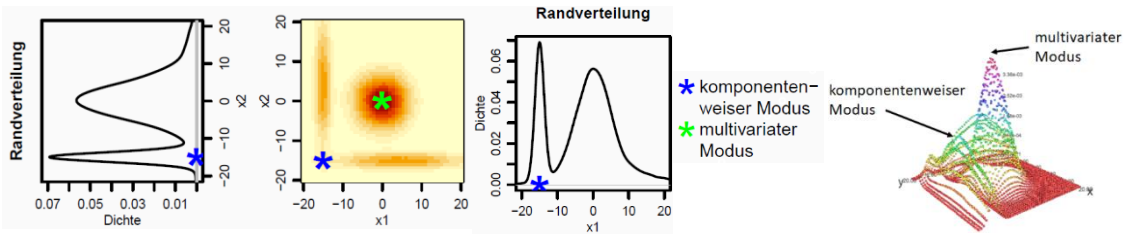
Allgemeine Struktur der Daten

- Datenobjekte und deren Attribute (Flat Table wie Excel Sheet)
- Variable/Features sind Eigenschaften eines Objekts
  - Beispiele: Geschmackskategorie (Whisky), Transaktionsmerkmale, Wörter, Pixel, etc.
  - Für Klassifizierung gibt es ein ausgezeichnetes Attribut (Klassenattribut): z.B. Betrug ja oder nein
- Eine Sammlung von Attributen beschreibt ein Objekt
  - Objekt auch: Individuum, Fall, Datum oder Rekord
  - Beispiele: Whisky, Transaktion, Mail, Bild
- Teilweise müssen die Daten zuerst in die entsprechende Struktur gebracht werden → Feature Engineering



Fallstricke bei univariaten Analysen

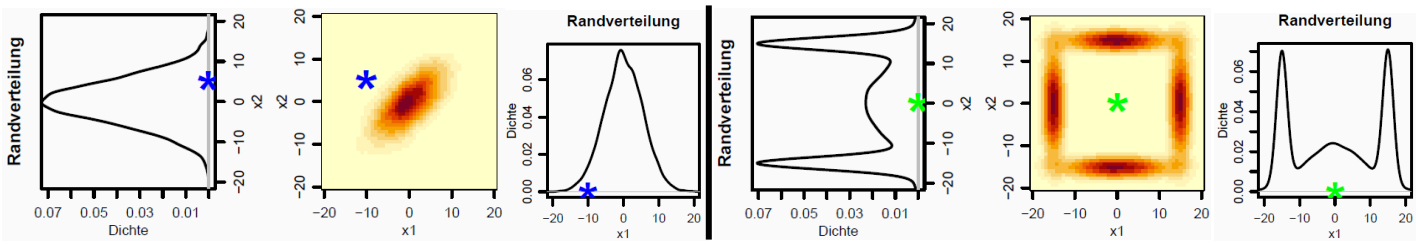
- Wieso wollen wir multivariable Daten analysieren? Was ist das Problem, wenn man die Variablen eines multivariaten Datensatz einzeln betrachtet?
- Beispiel Modus: Multivariater Modus entspricht wahrscheinlichsten Realisierung p-dimensionalen Merkmalsvektors.



- Der mulitvariate Modus kann nicht über die Moduswerte der Randverteilung abgeleitet werden!
- Beim Median gibt es nicht einmal eine allgemein akzeptierte Definition für den multivariaten Median.
- Die weitest verbreitete Version für den multivariaten Median ist der **L1-Median**. Er ist definiert als Punkt mit minimaler Summe der absoluten Abstände zu allen anderen Punkten (**L1median**, Paket **robustX**).
- Der komponentenweise Median ergibt nicht dasselbe.
- Immerhin ist Mittelwert gutmütig. Der komponentenweise Mittelwert entspricht multivariaten Schwerpunkt.

Ausreisser im multivariaten Raum

- Ein multivariater Ausreisser ist nicht unbedingt auch ein Ausreisser in den Randverteilungen!
- Ausreiser in blau respektive grün in den Grafiken.



Curse of dimensionality (Fluch der Dimensionalität)

- Wie viele Daten werden für eine bestimmte Abdeckung benötigt?
- Bei uniform-/gleichmässig verteilten Daten gilt zudem:
  - In hochdimensionalen Räumen «extreme Merkmalswerte» nicht selten
  - Kaum vorstellbar, aber in hohen Dimensionen wahr: Alle Punkte haben fast gleichen Abstand zueinander

Dimension	Anz. Beobachtung
1	10
2	100
3	1000
…	…
6	10^6
p	10^p