(1) Einleitung - Data Mining

Definition Data Mining

- Data Mining ist ein nicht-trivialer Prozess zur Identifizierung gültiger, neuartiger, potenziell nützlicher und letztlich verständlicher Muster in Daten.
- Extraktion von impliziten, neuen und hoffentlich nützlichen Mustern

Beispiele können:

- Klassifikation, Clustering, Regression, Assoziatensmuster, Time Series, Outlier-Analysis

Allgemein

- Anforderungen: Daten
- Aufgabe: Muster finden auf den zugrunde liegenden Regeln in Daten
- Data Mining Algorithmen, um Problem automatisiert zu lösen
- Herausforderungen:
    - Wie wird die Qualität der Ergebnisse gemessen?
    - Ist das Muster nützlich oder nicht?
- Gelernte Muster/Modelle können für Vorhersagungen genutzt werden

Typen für Vorhersageprobleme: Supervised/Unsupervised Learning

Supervised Learning

- "gelabelte" Traningsdaten, jedes Beispiel hat einen gewünschten Zielwert

Unsupervised Learning

- Keine bekannten Zeilwerte
- Auffällige/erklärende Merkmale durch Data Mining finden

Semi-supervised Learning

- Ähnlich zu supervised, manche Daten haben bekannte Zielwerte
- Z. B. Flickr: viele Bilder aber wenige davon sind getaggt

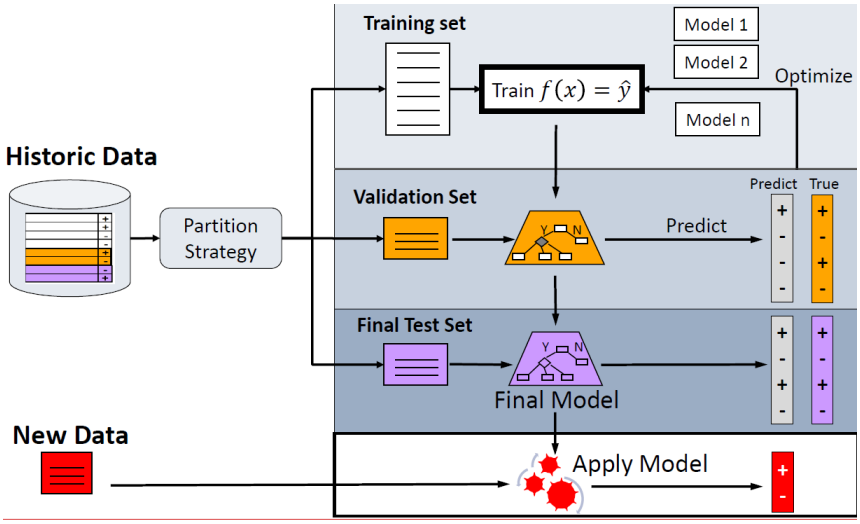
Reinforcement Learning

- Indirektes Feedback der Lernqualität
- Keine Antworten, nur sagen, ob das jetzige Ergebnis besser oder schlechter ist
- Feedback könnte verzögert eintreffen

Terminologie

- Object: Gegenstand des Interesses; Kunde, Produkt, Maschine etc.
- Features/Attributes: Ein Satz aus charakteristischen Features oder Attribute, die das Objekt beschreiben
- Label of target: Ein spezifisches Attribut, welches man vorhersagen soll
- Model: Ein Algorithmus oder eine Funktion, welches ein Outputwert des Zielattributes eines Objekts erzeugt. Diese Funktion minimiert eine error-Funktion oder optimiert eine Quality-Funktion

Training, Validierung und Testset



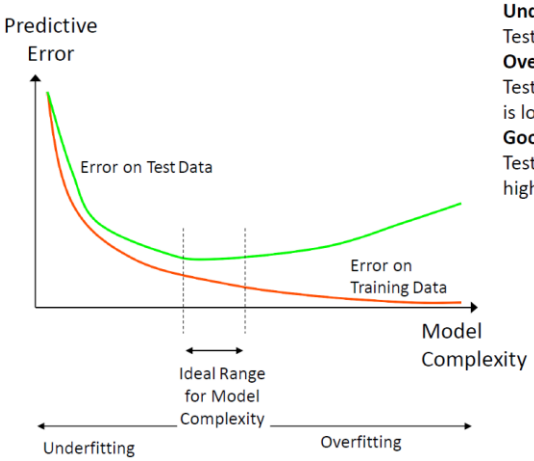
Hyperparameter: Gewichte auf Strafbänder. Z. B. Anzahl DTs bei Random Forest, Anzahl hidden nodes in einem NN

Das Modell wird mithilfe der Trainingsdaten und unterschiedlichen Hyperparameterwerten trainiert

Modell optimieren und selektieren, mit den geeignetsten Hyperparameter, welche die beste Performance auf den Validation-Datas zeigten (Training Error)

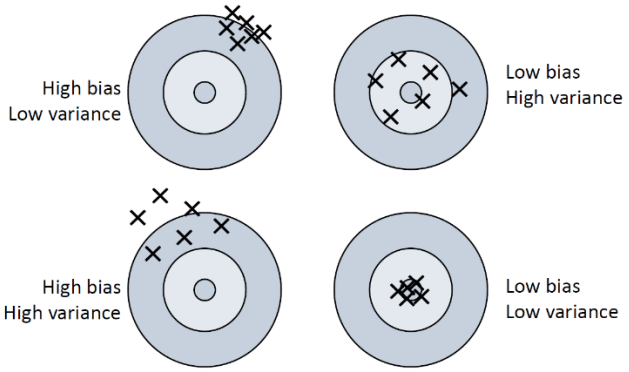
Der Error des optimierten Modells wird mit den Testdaten berechnet

How Overfitting affects Prediction

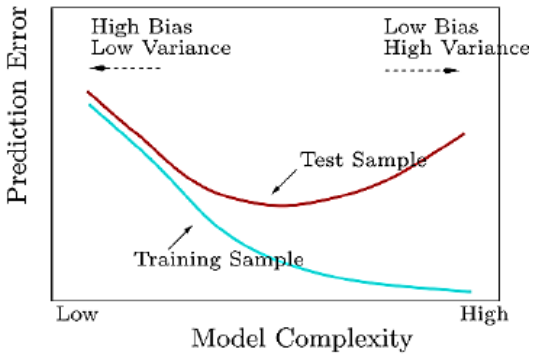


Underfitting
Test and training error are both high
Overfitting
Test error is high while training error is low
Good fit
Test error is low, and only slightly higher than the training error

Bias-Variance Tradeoff



Bias-Varianz Tradeoff



Bias (Systematischer Fehler/Abweichung)

- Der Error, der vom Modell verursacht wird
- Unterschied zwischen dem, was man erwartet und der Wahrheit
- Verringert man mit komplexer werdenden Modellen

Varianz (Auslenkung der Daten)

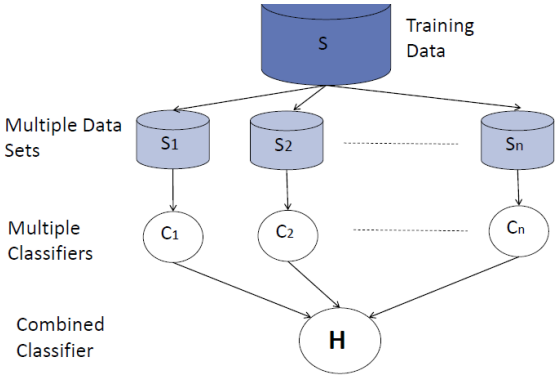
- Unterschied zwischen dem, was man erwartet zu lernen und dem, was man aus einem bestimmten Datensatz lernt
- Erhöht man mit komplexer werdenden Modellen
- (+) Hohe Varianz bedeutet: Modell ändert sich stark

Modell mit einem geringen Bias haben oft eine hohe Varianz: z. B. kNN, unpruned Decision Tree

Modell mit einem geringer Varianz haben oft einen hohen Bias: z. B. Decision Stump, linear Models

(2) Ensemble Methods, Random Forest und AdaBoost

Idee von Meta Learning



- Die Daten werden in mehreren Datensets aufgeteilt
- Die Datensets werden durch mehrere Klassifizierer eingeteilt
- Die Ergebnisse der einzelnen Klassifizierer werden in einem kombinierten Klassifizierer zusammengefasst

Bootstrapping

Original Data	1	2	3	4	5	6	7	8	9	10
Bootstrap 1	7	8	10	8	2	5	10	10	5	9
Bootstrap 2	1	4	9	1	2	3	2	7	3	2
Bootstrap 3	1	8	5	10	5	5	9	6	3	7

- Gegeben: ein Datenset mit n Instanzen
- Erzeuge ein Beispieldatensets mit derselben Anzahl an Instanzen und lege zurück
- Für jedes Set der Größe n gilt, dass die Wahrscheinlichkeit, dass ein gegebenes Beispiel enthalten ist, bei 0,6322 liegt
- Im Durchschnitt werden weniger als 2/3 der Beispiele in einem einzelnen Bootstrap-Sample angezeigt

Bagging – Aggregate Bootstrapping

Idee von Bagging

- Erzeuge Beispieldaten indem zufällig Datensätze zurückgelegt werden
- Lerne ein Modell mit jedem Beispieldatenset und kombiniere anschließend alle Modelle
- Dies reduziert die Schwankungen der Vorhersagen
- Funktioniert besser mit Predictions mit kleinem Bias und hohe Varianz

Vorgehen:

- Verwende den gleichen Basis-Lerner von Bootstrapversion des ursprünglichen Datensatzes
- berechne den Mittelwert aus den Vorhersagen der Modelle

Pseudo Code

```
1. for m = 1 to t // t … number of iterations
    a. Erzeuge ein Bootstrap sample Dm der Daten
    b. Lerne einen Klassifizierer Cm mit Dm
2. for each test example
    a. Teste alle Klassifizierer Cm
    b. Sage die Klasse vorher, die am höchsten bewertet ist
```

- Variationen sind möglich, z. B. durch Größe der Teilmengen, Beispiele, Zurücklegen
- Viele ähnliche Varianten: Sampling aus Features, nicht aus Instanzen; Lern ein Set aus Klassifizierer mit verschiedenen Algorithmen

Boosting

Idee von Boosting

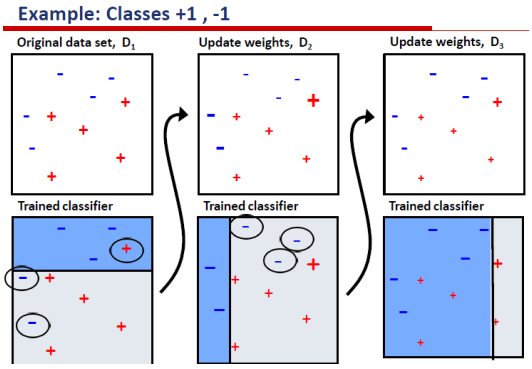
- Trainiere ein Set von Klassifizierer nacheinander
- Spätere Klassifizierer fokussieren sich auf Fehlerklassifikation der früheren Klassifizierer
- Gewichte die Vorhersagen der Klassifizierer mit ihrem Fehler
- (+) Entscheidungen treffen → konvergiert mit der Zeit, Permanente Anpassung der Gewichte

Realisierung

- Führe mehrere Iterationen durch
- Jedes Mal werden andere Beispielgewichte benutzt
- Gewichtungsanpassung zwischen den Iterationen
    - Erhöhe die Gewichte der falsch klassifizierten Beispielen → dadurch werden diese in der nächsten Iteration wichtiger: falsch klassifizierte Errors für diese Beispiele werden stärker gewichtet
- Kombiniere die Ergebnisse aller Iterationen → gewichtet mit ihren jeweiligen Fehlermaßen

Pseudo Code AdaBoost

1. Initialisiere die Gewichte
2. For m = 1 to B // B … Anzahl der Iterationen
    a. Lerne den Klassifizierer C_m mit den aktuellen Gewichten
    b. Berechne die gewichtete Fehlerrate
    c. Berechne die Gewichtung des Klassifizierers
    d. Verringere die Gewichtung für alle korrekten Datenpunkte
    e. Erhöhe die Gewichtung für alle fehlerhaften Datenpunkte
    f. Normalisiere die Gewichte, so dass ihre Summe 1 ergibt
3. Für jedes Testbeispiel
    a. Prüfe alle Klassifizierer C_m
    b. Sage die Klasse mit der höchsten Summe der Gewichte der Klassifizierer voraus



Bewertung

- Jeder Baum kann mit wenigen Endknoten sehr klein sein
- Der Lerner lernt langsam
- Tendenziell performen langsame statistische Learner-Ansätze gut
- Gefahr von Overfitting, wenn die Anzahl an Bäumen zu groß gewählt werden (nicht so bei Bagging und RF)
    - Kreuz-Validierung wird genutzt, um die Anzahl der Bäume zu wählen
- Beim Boosting hängt die Konstruktion der Bäume stark von den bereits bestehenden Bäumen ab

Random Forest

- Erzeuge den Baum aus Bootstrap-Sets
- Anstatt den besten Splittpunkt aller Attribute zu nutzen, wird die Teilung anhand einer Teilmenge von k Attributen genutzt
- Es entspricht Bagging, wenn k gleich der Anzahl der Attribute ist
- Bei jedem Baumknoten werden die Attribute zufällig ausgewählt → der Zielwert für jeden Datensatz wird anhand der Mehrheitsergebnisse aller Bäume bestimmt

Pseudo Code

1. For b = 1 to B // ~ 500
    a. Erzeuge ein Bootstrap Set Z^* der Größe N von den Trainingsdaten
    b. Erzeuge einen random forest T_b mit den Daten, indem rekursiv die folgenden Schritte für jeden Blattknoten ausgeführt werden, bis die minimale Knotengröße n_{min} erreicht wurde.
        i. Wähle m zufällige Variablen von den p Variablen
        ii. Wähle die beste Variable aus m
        iii. Teile die Knoten in zwei Tochterknoten
2. Gib die Kombinationen der Bäume aus

Out-of-bag

- Beobachtungen, die nicht Teil eines Bootstrap Samples sind, nennt man Out-of-bag
- Die Out-of-bag-Fehlerrate kann für jede Beobachtung über den gesamten Random Forests ermittelt werden
- Beim Random-Forest ist keine Cross-Validation notwendig

Vorteile

- Geeignet für hochdimensionalen Daten, bei denen die Menge an Attribute > Menge an Instanzen ist
- Robust gegenüber Ausreißer
- Quantifiziert die Wichtigkeit der einzelnen Variablen
- Gute Performance im Vergleich zu klassischen Ansätzen wie SVM und Neuronale Netze
- Einfache Abschätzung der Fehlerrate
- Konsequente Schätzung der Zielwerte
- Probleme einzelner Bäume (hohe Varianz) werden kompensiert: hohe Varianz, wenn falsche Entscheidung im hohen Knoten weitergeführt wird → viele Bäume reduzieren diesen Effekt

Unterschiede zum Decision Tree

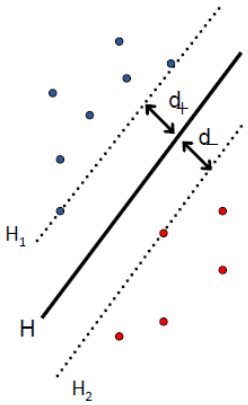
- Jeder Baum wird mit einem Bootstrap resample von Daten trainiert
    - Bootstrap resample von Daten mit n Beispielen: Mache neuen Datenset, indem n Samples zurückgelegt werden
    - Z. B.: manchmal kommen einige Samples mehrmals in neuen Datensets vor
- Für jeden Split werden zufällig m Attribute der gesamten p Attribute gewählt (m = sqrt(p))
- Kein Pruning → kein Abschneiden von unwichtigen Teilen der Bäume

(3) Support Vector Machines – SVM

SVMs als lineare Klassifizierer

- Lernt eine optimale lineare Separierung der Daten
- Ermöglicht eine Klassifikation der Testdaten auf Basis der gelernten Hyperebene
- Keine wirkliche Maschine
- Gelernt werden Linearkombinationen von Stützvektoren

Optimale Hyperebene



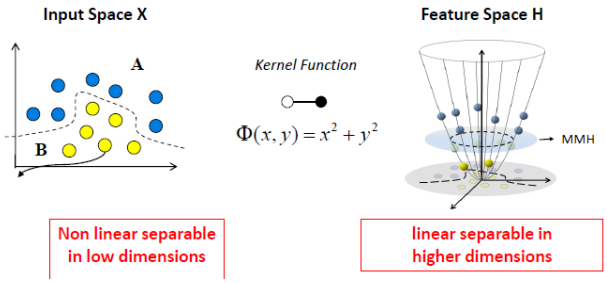
- Beobachtungen heißen linear separierbar, wenn eine Hyperebene H existiert, die die Beispiele in positiv und negativ aufteilt
- H ist eine optimale Hyperebene, wenn die Distanz d zu dem nächsten positiven und negativen Datenpunkt maximal ist

Wir wissen jetzt: das Lernergebnis ist eine lineare Kombination aus Stützvektoren, wir müssen nur die Skalarprodukte der Beispiele berechnen

- um auch Punkte auf der falschen Seite zuzulassen, werden Slack-Variablen eingeführt → diese **bestrafen** das Modell

Nicht linear separierbare Daten – Transformation in Feature Space H

- Es wird eine Kernel-Funktion angewandt, mit der die Datenpunkte in eine höhere Dimension verschoben werden
- In dieser Dimension sind die Daten meist separierbar
- Nach der Klassifikation wird eine Umkehrfunktion benötigt, um die Daten wieder in den ursprünglichen Feature-Raum zu transformieren



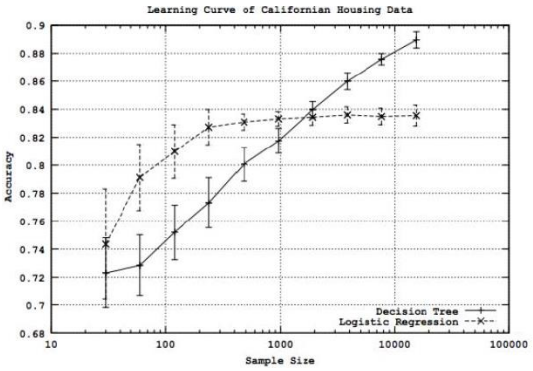
Kernel-Trick

- Der Feature-Raum ist nicht relevant, interessant ist nur die Kernel-Funktion als Maßangabe für die Ähnlichkeit
- Außerdem wissen wir nicht was im Kernel passiert, nur das Ergebnis ist interessant
- Wir haben einfach nur die geometrische Interpretation der separierenden Hyperebene
- Relevant ist nur das Ergebnis (Blackbox)

Zusammenfassung

- Kernel-Funktionen berechnen das Skalarprodukt der Beobachtungen in einem speziellen raum ohne eine tatsächliche Transformation in den speziellen Raum
- Die Mercer-Bedingung prüft, ob eine Funktion eine Kernel-Funktion ist, z. B. ob die Matrix positiv ist

(4) Model Evaluation



Lernkurven

```
Gegebene Trainings/Test Partition
Für jede Beispielgröße s der Lernkurve
  - Zufällige s Instanzen der Trainingsdaten wählen
  - Modell lernen
  - Model mit den Testdaten evaluieren, um die Genauigkeit a zu
    bestimmen
  - plot(s, a) oder (s, avg.accuracy und error bars)
```

Schätzungsmethoden

- Holdout: Reserviere 2/3 der Daten für das Training und 1/3 für das Testen
- Stratification: oversampling vs. undersampling
- Random subsampling: wiederholtes Holdout
- Cross Validation
  - Daten in k getrennte Teilmengen unterteilen
  - K-fold: trainiere mit k-1 Partitionen, getestet wird mit den verbleibenden
- Bootstrap: Sampling mit ersetzen / zurücklegen

Confusion Matrix für binäre Klassifizierer

2 Klassen: {Yes, No} → Zeilen: Predicted Class, Spalten: True Class

	Yes	No
Yes	TP = True Positive	FP = False Positive
No	FN = False Negative	TN = True Negative

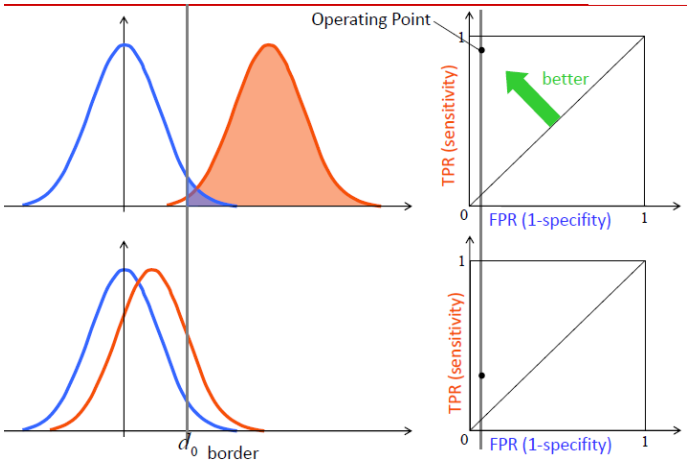
$$Accuracy = \frac{TP + TN}{P + N} \qquad Error\ rate = \frac{FP + FN}{P + N}$$

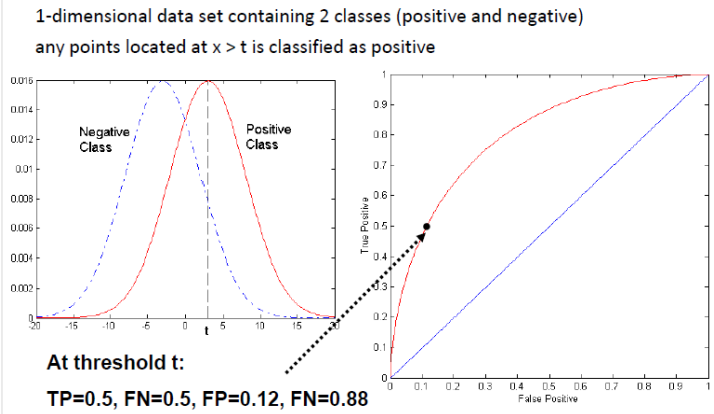
$$Recall = \frac{TP}{TP + FN} \qquad Precision\ (Sensitivity) = \frac{TP}{TP + FP}$$

$$Specifity = \frac{TN}{TN + FP}$$

ROC-Analyse – Receiver Operating Characteristics

- Ziel: Was ist die beste Methode um Parameter zu variieren?
- Methode:
  - Visualisierung der ROC-Kurve
  - Für und tpr für jeden binären Klassifizierer
  - x-Achse: FP-Rate (fpr), y-Achse: TP-Rate (tpr)
- (+) Trennpunkt | ← eher die eine oder die andere Klasse (TP)
- (+) Wie kommt man zur Kurve? Schwellwert varrieren → False-Positive Ranking
- (+) Bei guten Algorithmen: wenig False-Positive





(5) kNN, Curse of Dimensionality and PCA

k-Nearest Neighbor

- Finde die k-nächsten Nachbarn zu einer neuen Instanz z
  - Feature-Vectoren mit euklidischer Distanz bewerten
  - Wähle die k Vectoren mit der kleinsten Distanz zu z
- k = Anzahl der berücksichtigten Daten
- Entscheidungsset der k-nächsten Nachbarn für die Klassifikation
- Entscheidungsregel, wie entschieden wird, welche Klasse aus dem Entscheidungsset gewählt wird
- Distanzfunktion definiert die Ähnlichkeit der Objekte

Klassifikation / Vorgehen

- Ranking ergibt k Feature-Vektoren und ein Set aus k Klassenlabels
- Wähle das Label, das am häufigsten vorkommt
- Klassifiziere z als diese Klasse

„Training“ ist trivial: Verwende Trainingsdaten als Lookup-Tabelle und suche darin zum Klassifizieren → Lazy Learner

Methoden zum Normalisieren

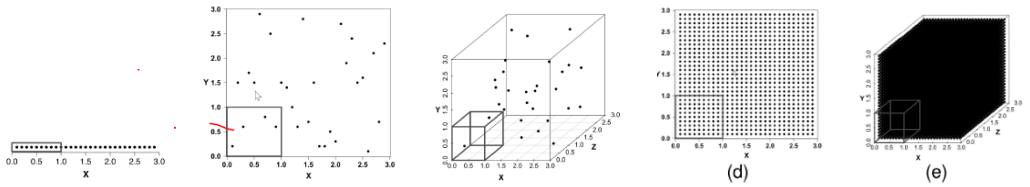
- Es kann zu ungenauen Vorhersagen kommen, wenn die Features unterschiedliche Wertebereiche haben
  - Das ist das gleiche, wie wenn Features verschiedene Varianzen haben
- Dies können wir mit Normalisierung anpassen
- (+) Unterschiedliche Wertebereiche auf Einheitsdimensionalität

z-transformation: $\forall x \in X: \quad z_i = \frac{x_i - \bar{x}}{s}$ bzw. $z_i = \frac{x_i - \mu}{\sigma}$

Normalization to interval [0, 1]: $\forall x \in X: x' = \frac{x - \min(x)}{\max(x) - \min(x)}$

Curse of Dimensionality

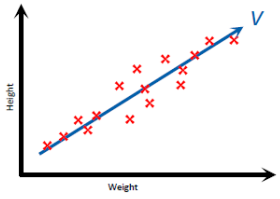
- 10^2 Datenpunkte im Unit-Intervall haben eine Distanz von 0.01
- Die gleiche Verteilung bei einer 10-dimensionalen Unit-Würfel mit Distanz 0.01 benötigt 10^{2*10} Datenpunkte
- Also: die Dichte der Unit-Würfel nimmt ab, wenn mehr Dimensionen dazukommen



(d) und (e) zeigt, was es kostet, um die Dichte beizubehalten → viel mehr Datenpunkte nötig

PCA – Principal Component Analysis

- Dimensionsreduktion
- Projiziert die Daten auf eine k orthogonalen Basisvektor v, der den Projektions-Error minimiert



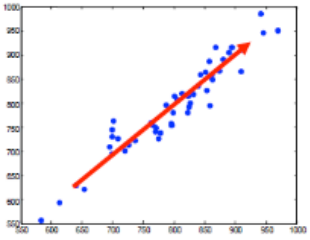
Example:
- original 2D dataset containing features weight and height
- projection on vector v

Gesucht: Vektor, der am besten x rekonstruieren kann

v aussuchen, um die residuale Varianz zu minimieren

v ist die Richtung der der maximalen Varianz

$$\min_{a,v} \sum_i \left(x^{(i)} - a^{(i)} \cdot v\right)^2$$



Aufschrieb: Grundidee

- Reduziert mehrere auf weniger Features
- Vektoren berechnen, die auf die Größte Varianz zeigen
- Wenn Streuung verloren geht, sind die Abstände der Hauptkomponente immer noch da

(6) Feature Selection

Ziele der Feature Selection

- Vorhersage verbessern (trotz Fluch der Dimensionen)
- Datenvisualisierung und Datenverständnis erleichtern
- Modellierungsprozess beschleunigen
- Das Verständnis für den zugrunde liegenden Prozess verbessern
- Speicherbedarf senken

Feature Selection: Ranking / Filter Methode

Algorithmus

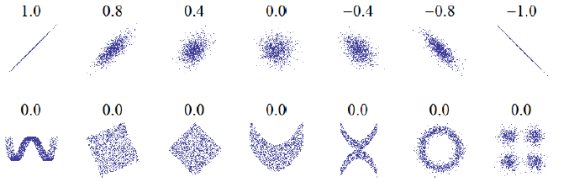
- Berechne den Score für jedes Feature xi gegen die Zielvariable y
- Benutze ein Scoring Kriterium S(i)
- Sortiere die Features nach dem Score
- Der Rang wird zum Messen der Relevanz der Variable genutzt

Rank	Feature	Score
1	B	10
2	A	5
3	C	4

Ranking – Korrelation

Criterion: Pearson correlation coefficient

$$Cor(x_i, y) = \frac{cov(x_i, y)}{\sqrt{var(x_i)var(y)}} = \frac{\sum_{i=1}^{n} (x_i - x)(y_i - y)}{\sqrt{\sum_{1}^{n} (x_i - x)^2}\sqrt{\sum_{1}^{n} (y_i - y)^2}}$$

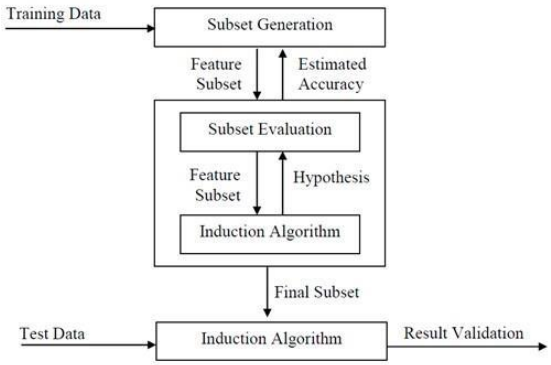


Ranking-Filter-Methode:

(+) Features rausnehmen und schauen, wie Modell performt

Wrapper-Methode

Nutzt die Vorhersage-Performance, um den Nutzen von Variablen-Teilmengen während der Suche zu bestimmen

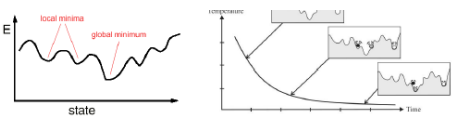


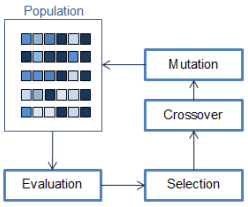
Folgendes ist zu definieren:

- Such-Methode
- Wie die Vorhersage-Performance bewertet wird, um die Suche zu steuern und zu stoppen
- Welcher Predictor benutzt wird

Such-Methoden

- Exhaustive Search (erschöpfende Suche) → alle Kombinationen ausprobieren
- Simulated Annealing
- Genetische Algorithmen





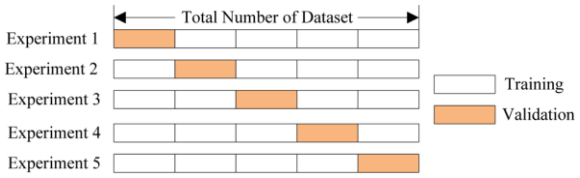
Greedy-Algorithmen

Zu intensives Suchen kann zu Overfitting führen. Einfachere und effizientere Strategien, wie Greedy Algorithmen, sollten verwendet werden

- Backward Elemination: Starte mit allen Features; die am wenigsten vielversprechendsten werden schrittweise eliminiert
- Forward Selection: Variablen werden schrittweise in immer größeren Teilmengen integriert

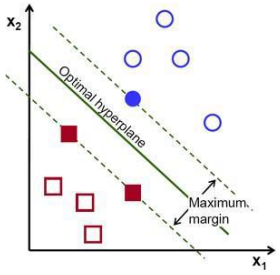
Performance-Bewertung

Performance-Messung:



- Genauigkeit auf dem Validierungsset
- Cross-Validation

Predictors



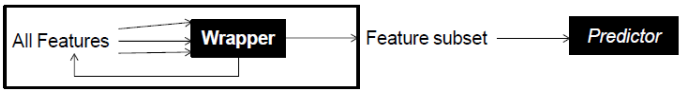
- Decision Tree
- Naive Bayes
- Least-Square Linear Predictors
- Support Vector Machines

Zusammenfassung - Wrapper

Vorteile

- Berücksichtigt die Beziehungen zwischen den Variablen
- Erzeugt gute Feature Subsets

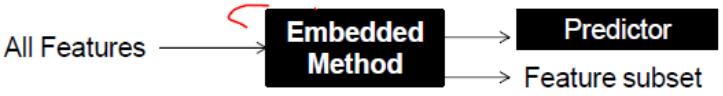
Nachteile



- Rechenintensiv
- Brute Force Methode
- Gefahr von Overfitting
- Varianz der Feature Subsets (Lösung: Bootstrap)

Embedded-Methode

- Feature Selection ist Teil des Lernalgorithmus
- Beispiel: Decision Tree, Random Forest
- Universeller und einfacher Ansatz
- Nutzt die verfügbaren Daten besser
- Effizienter als Wrapper-Methode



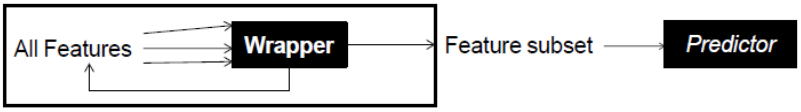
Feature-Selection – Zusammenfassung

Filter/Ranking

Rechnergestützt (rechentechnisch) effizient, führt zu redundanten Feature-Subsets

Wrapper

Rechentechnisch anspruchsvoll, berücksichtigt aber Beziehungen zwischen Features und führt zu nicht-redundanten Feature-Subsets



Embedded

(7) Applied Time Series Analysis

Was ist eine Zeitreihe?

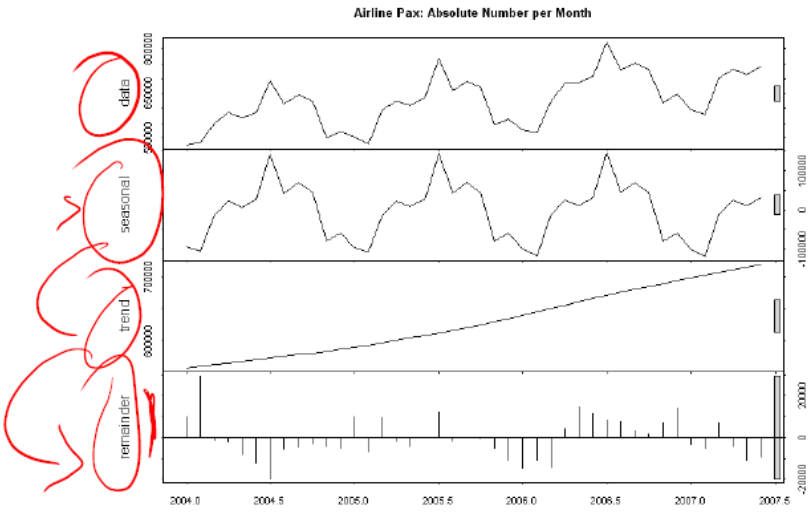
- eine Reihe von Beobachtungen, bei denen jede der Beobachtungen zu einem bestimmten Zeitpunkt gemacht wurde
- Das Set aus Zeiten T ist diskret und endlich
- Die Beobachtungen wurden in fixen Zeitintervallen gemacht
- Kontinuierliche und unregelmäßige Zeitreihen werden nicht enthalten

Grund für Zeitreihenanalyse: Vergangenheitsdaten analysieren und Zukunft vorhersagen

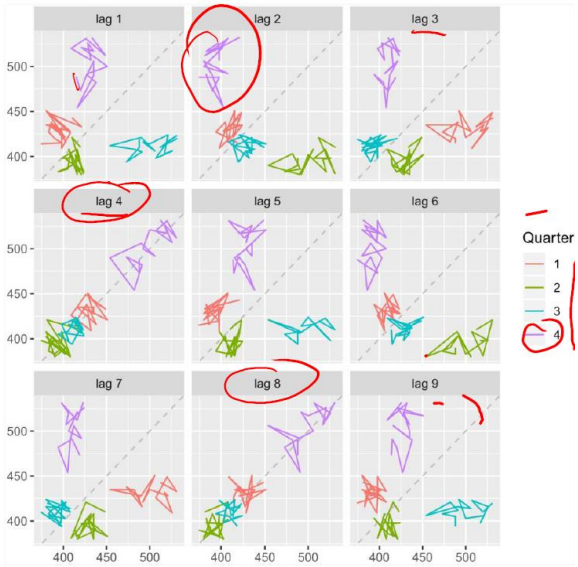
Zeitreihen-Patterns

- Trend: liegt vor, wenn auf lange Sicht gesehen ein Anstieg oder Abfall in den Daten erkennbar ist
- Cycle (zyklisches Muster): liegt vor, wenn die Daten Auf- und Abwärtsbewegungen aufweisen (üblich: über mindestens zwei Jahren)
    - Variable Länge, deutlich länger als seasonal
- Seasonal: liegt vor, wenn die Zeitreihen von saisonalen Faktoren beeinflusst werden (jedes Quartal, jeden Monat, Tag einer Woche)
    - Konstante Länge, vorhersagen von Spitzenwerten möglich

Die zeitlichen Spitzen und Täler sind mit saisonalen Daten vorhersehbar, aber langfristig unvorhersehbar mit zyklischen Daten.



Lag plots und Autokorrelation



Lagged scatterplot

Jeder Graph zeigt y_t grafisch dargestellt gegen y_{t-k} für verschiedene Werte von k

Die Autokorrelationen sind die Korrelationen, die mit diesen Streudiagrammen verbunden sind

Autokorrelation

- Kovarianz und Korrelation messen den linearen Zusammenhang zwischen zwei Variablen (x und y)
- Autokovarianz und Autokorrelation messen den linearen Zusammenhang zwischen lagged values (verzögerte / spätere Werte) einer Zeitreihe

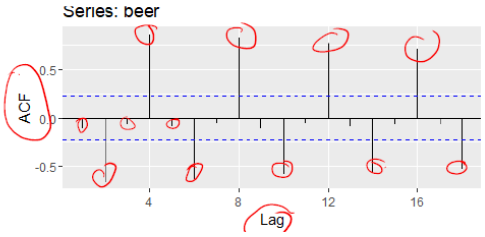
Es wird die Beziehung gemessen zwischen
- y_t and y_{t-1}
- y_t and y_{t-2}
- y_t and y_{t-3}
- etc.

Autokorrelation bei lag k $r_k = \frac{C_k}{C_0}$

- r_1 zeigt an, wie sich aufeinanderfolgende Werte von y auf andere Werte beziehen
- r_2 zeigt an, wie sich y-Werte, die zwei Perioden voneinander entfernt sind, zueinander verhalten
- r_k ist gleichbedeutend mit der Stichprobenkorrelation zwischen y_t und y_{t-k}

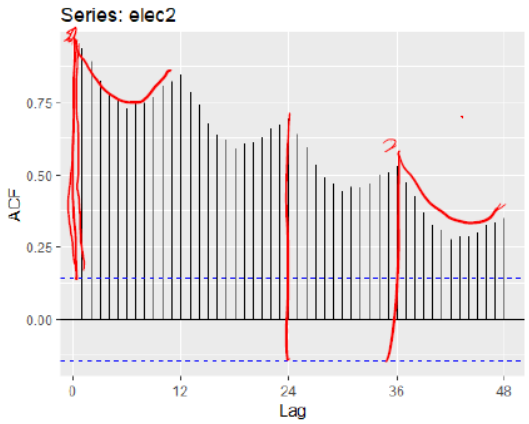
Results for first 9 lags for beer data:

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
-0.102	-0.657	-0.060	0.869	-0.089	-0.635	-0.054	0.832	-0.108



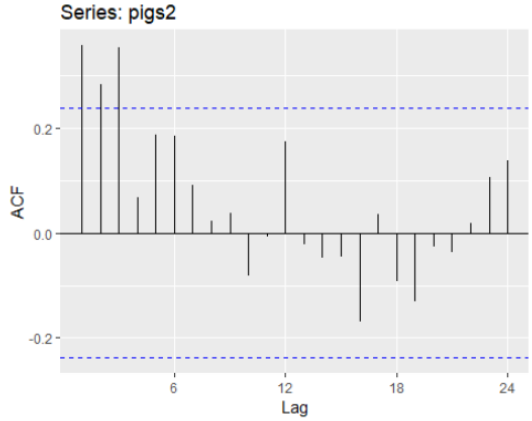
Trend und Saisonalität in ACF plots

- Wenn Daten einen Trend aufweisen, tendiert die Autokorrelation dazu, in kleinen Lags groß und positiv zu sein
- Bei saisonalen Daten ist die Autokorrelation in saisonalen Lags größer
- Trifft beides zu, ist eine Kombination der Effekte zu erkennen



- Trend und Saisonalität erkennbar
- Absteigender ACF zeigt einen Trend an
- Die Peaks in den Lags 12, 24 und 35 deuten auf eine Saison der Länge 12 hin

Pigs slaughtered



- Schwer zu beurteilen
- Deutliche Autokorrelation bei den Lags 1, 2 und 3
- Lag 12 ist relativ groß aber nicht signifikant. Dies könnte sich auf leichte Saisonalität hinweisen

(8) Applied Time Series Analysis – ARIMA

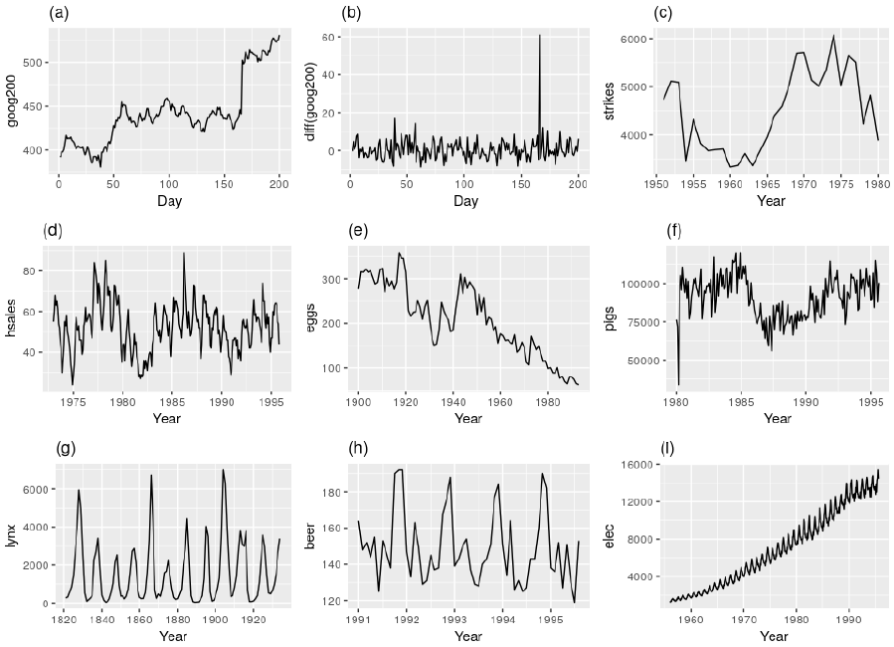
Stationarität

Eine stationäre Zeitreihe ist

- ungefähr horizontal
- konstante Varianz
- keine Muster auf lange Sicht erkennbar

Beweis für eine nicht-stationäre Zeitreihe

- Trend
- Saisonalität
- Keine konstante Varianz
- Keine konstante strukturelle Abhängigkeit



a) nein -> Trendlinie -> Veränderung über Zeit

b) ja

c) nein -> keine Konstanz über Zeit

d) nein

e) nein

f) nein

g) ja

h) unklar

i) nein

Stationarität

- Wenn $\{y_t\}$ eine stationäre Zeitreihe ist, dann ist die Verteilung von $y_t \ldots y_{t+s}$ für alle s nicht von t abhängig
- Transformationen helfen, die Varianz zu stabilisieren. Für die ARIMA-Modellierung müssen wir auch den Mittelwert stabilisieren

Strategien zur Erkennung von Nicht-Stationarität

Time Series Plot

- Kein konstanter vorhersagbarer Wert (Trend / saisonale Effekte)
- Änderungen in den Abhängigkeiten
- Keine Konstante Varianz

Korrelogramm

- Kein konstanter vorhersagbarer Wert (Trend / saisonaler Effekt)
- Änderungen in den Abhängigkeiten

Trick: die Zeitreihe in mehrere Teile aufzuteilen und die Teilstücke separat zu visualisieren

Nicht-Stationarität im Durchschnitt

Nicht-stationäre Zeitreihen identifizieren

- Time Plot
- Der ACF von stationären Daten geht sehr schnell gegen null
- Der ACF von nicht-stationären Daten verringert sich langsam
- r_1 ist bei nicht-stationären Daten oft groß und positiv

Differenzierung

- Differenzierung hilft den Durschnitt zu stabilisieren
- Die differenzierte Zeitreihe ist die zwischen jeder Beobachtung in der ursprünglichen Serie
- Die differenzierte Zeitreihe hat nur t – 1 Werte, da für die erste Beobachtung keine Differenz y_1' entstehen kann

Second-order Differenzierung

- Manchmal erscheinen die differenzierten Werte nicht-stationär, sodass eine erneute Differenzierung vorgenommen werden muss
- Die Zeitreihe hat dann im Vergleich zur Originalreihe zwei Werte weniger
- In der Praxis muss nie mehr als die zweite Differenzierung durchgeführt werden

$$\begin{aligned} y''_t &= y'_t - y'_{t-1} \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2} \end{aligned}$$

Saisonale Differenzierung

Es wird die Differenz zwischen dem aktuellen betrachteten Wert und dem Wert von vor einem Jahr/Monat/Quartal betrachtet.

$$y'_t = y_t - y_{t-m}$$

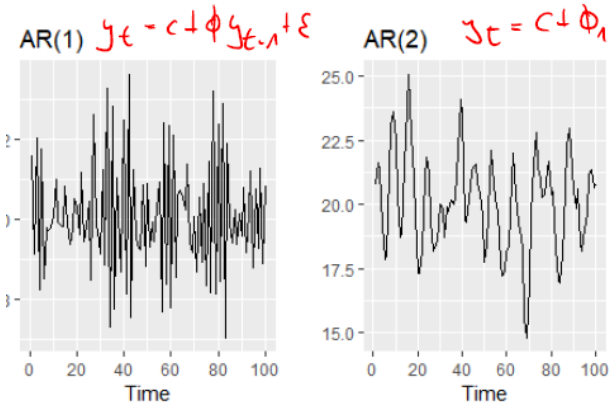
where m = number of seasons.
- For monthly data $m = 12$.
- For quarterly data $m = 4$.

ARIMA
- AR - autoregressive model
- MA - moving average model
- ARMA - Kombination von AR und MA
- ARIMA - nicht-stationäre ARMAs
- SARIMA - saisonale ARIMAs

ARIMA -> Mit Backshift-Operator

1. Autoregressives Model - AR(p)

- Autoregressive Modelle werden normalerweise nur auf stationäre Daten angewandt
- ε_t ist das weiße Rauschen. Dies ist eine multiple Regression mit lagged values auf y_t als Predictor
- p = wie weit in die Vergangenheit



AR(0) = C

AR(0)

The simplest model is the AR(1)-model

$$y_t = c + \phi y_{t-1} + \varepsilon_t$$

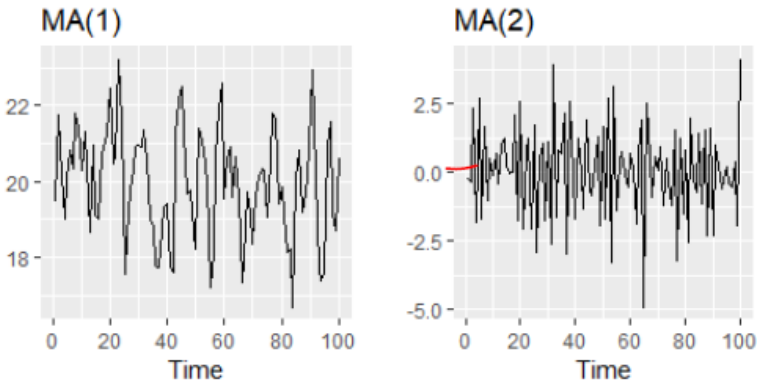
- When $\phi_1 = 0$,y_t is **equivalent to WN**
- When $\phi_1 = 1$ and $c = 0$, y_t is **equivalent to a RW**
- When $\phi_1 = 1$ and $c \neq 0$, y_t is **equivalent to a RW with drift**
- When $\phi_1 < 0$,y_t tends to **oscillate between positive and negative values**.

2. Moving Average (MA) Model

- ε_t ist weißes Rauschen. Dies ist eine multiple Regression mit vergangenen Fehlern als Predictors. Nicht verwechseln mit der durchschnittlichen Glättung

$$y_t = c + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} \dots + \theta_q\epsilon_{t-q}$$

Abweichung mit vorherigem Forecast berechnen und damit neue Vorhersage berechnen



Autoregressive Moving Average Model

$$y_t = c + \phi y_{t-1} + \cdots + \phi_p y_{t-p} + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} \dots + \theta_q\epsilon_{t-q} + \varepsilon_t$$

- Vorhersagen beinhalten lagged values und lagged errors (Verzögerungen)
- Bedingungen an die Koeffizienten stellen sicher, dass die Stationarität gewährleistet ist
- Bedingungen an die Koeffizienten gewährleisten Invertierbarkeit

Autoregressive Integrated Moving Average Model

- Kombiniere ARMA Model mit Differenzierung

(+) Differenzbildung → Glätten → somit stationär bekommen (Stationarität ist die Voraussetzung dafür, dass man ARIMA anwenden kann)

ARIMA(p, d, q) Model

- AR: p = Reihenfolge des autoregressiven Teils
- I: d = Grad der ersten Differenzierung, die involviert ist
- MA: Reihenfolge des gleitenden Durchschnitts (moving average part)

- White noise model: ARIMA(0, 0, 0) (kann man nicht vorhersagen, komplett zufällig)
- Random Walk: ARIMA(0, 1, 0) ohne Konstanten (nicht in Vergangenheit schauen, sondern nur Differenz von aktuellen und vorherigen ermitteln für Prognose)
- Random walk mit Drift: ARIMA(0, 1, 0) mit Konstante
- AR(p): ARIMA(p, 0, 0)
- MA(q): ARIMA(0, 0, q)

Backshift-Notation für ARIMA

ARMA model:

$$y_t = c + \phi_1 B y_t + \cdots + \phi_p B^p y_t + \varepsilon_t + \theta_1 B \varepsilon_t + \cdots + \theta_q B^q \varepsilon_t$$

or $(1 - \phi_1 B - \cdots - \phi_p B^p) y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q) \varepsilon_t$

ARIMA(1,1,1) model:

$$\underset{\text{AR(1)}}{(1 - \phi_1 B)} \quad \underset{\text{First difference}}{(1 - B) y_t} \quad = \quad c + \underset{\text{MA(1)}}{(1 + \theta_1 B) \varepsilon_t}$$

Written out:

$$y_t = c + y_{t-1} + \phi_1 y_{t-1} - \phi_1 y_{t-2} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

Backshift

$B y_t = y_{t-1}$

- B hat den Effekt, dass die Daten um eine Periode zurückgeschoben werden
- Für Monatsdaten, wenn wir die auf den gleichen Monat des Vorjahres verlagern wollen, dann wird $B^{12} y_t = y_{t-12}$ verwendet

ARIMA Model verstehen

Einschätzungen

Die Konstante c hat einen wichtigen Einfluss auf die Langzeitvorhersagen

1. c = 0 && d = 0 => Vorhersage geht gegen Null
2. c = 0 && d = 1 => Vorhersage geht gegen eine Konstante != 0
3. c = 0 && d = 2 => Vorhersage ist eine gerade Linie
4. c != 0 && d = 0 => Vorhersage geht gegen den Durchschnitt der Daten
5. c != 0 && d = 1 => Vorhersage ist eine gerade Linie
6. c != 0 && d = 2 => Vorhersage folgt einem quadratischen Trend

Aufschrieb:

- Wichtig: was ist Stationarität? Wie kann man es herausfinden?
- Wie erkannt man es? Plot / KSSP
- Wie kann man Graph von ARIMA verbessern? Durch Differenzierung
- ARIMA anwenden → Parameter schätzen
- Ohne C (Konstante) ist es für ARIMA schwer, gute Vorhersagen zu machen