Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorthesis

Moritz Knüppel

Evaluation von Data-Mining-Algorithmen am Beispiel einer Anwendung zur Parkplatzsuche

*Fakultät Technik und Informatik*
*Department Informations- und Elektrotechnik*

*Faculty of Engineering and Computer Science*
*Department of Information and Electrical Engineering*

Moritz Knüppel

# Evaluation von Data-Mining-Algorithmen am Beispiel einer Anwendung zur Parkplatzsuche

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Henning Dierks
Zweitgutachter : Prof. Dr.-Ing. Karin Landenfeld

Abgegeben am 5. September 2017

**Moritz Knüppel**

## Thema der Bachelorthesis

Evaluation von Data-Mining-Algorithmen am Beispiel einer Anwendung zur Parkplatzsuche

## Stichworte

Data-Mining, Classifier, Klassifikationsanalyse, Assoziationsanalyse, Clusteringverfahren, Weka, ARFF, KDD, Confusion Matrix, Cross-Validation

## Kurzzusammenfassung

Diese Arbeit befasst sich mit der Machbarkeit einer Anwendung, die Vorhersagen über das Auffinden von Parkplätzen trifft. Hierzu werden verschiedene Data-Mining-Verfahren und entsprechende Evaluationsverfahren vorgestellt, angewendet und ausgewertet.

**Moritz Knüppel**

## Title of the paper

Evaluation of Data Mining Algorithms at the example of an application for finding parking spots

## Keywords

data mining, classifier, statistical classification, association rule learning, clustering, Weka, ARFF, KDD, confusion matrix, cross-validation

## Abstract

This thesis investigates the possibility of an application that makes predictions on the likelihood of finding parking spaces. For this, various data mining and corresponding evaluation approaches are presented, applied and assessed.

# Inhaltsverzeichnis

# Tabellenverzeichnis

# Abbildungsverzeichnis

# 1. Einführung

Das Ziel dieser Arbeit ist die Untersuchung der Machbarkeit einer hypothetischen mobilen Anwendung, die Autofahrer beim Finden eines Parkplatzes unterstützen soll. Die Vision der Anwendung ist es, anhand von aktuellen Umgebungsdaten wie bspw. der Uhrzeit, dem Wetter und der Geoposition und basierend auf existierenden Bestandsdaten statistische Vorhersagen über das Auffinden von Parkplätzen in nahegelegenen Straßen treffen zu können. Der Use-Case wurde wie folgt vorgegeben:

*„Du sitzt im Auto einige Straßen entfernt von deinem Ziel an einer Kreuzung und drückst in der App auf einen Knopf. Daraufhin sagt sie dir, in welcher Straße du mit welcher Wahrscheinlichkeit einen Parkplatz findest. Falls du noch etwas weiter vom Ziel entfernt bist, bestimmt sie dir die Route mit der höchsten Wahrscheinlichkeit, einen Parkplatz zu finden.“*

Es ist somit zu ermitteln, ob es durch Einsatz von Methoden des Data Mining möglich ist, Wahrscheinlichkeiten über das Auffinden von Parkplätzen in einem bestimmten Bereich vorherzusagen, die weiterführend in einer mobilen Anwendung sinnvoll verwendet werden können. Die Untersuchung umfasst hierbei sowohl die Überprüfung der theoretischen Machbarkeit - Proof of Concept - als auch eine Bewertung, ob eine solche Anwendung mit aktuell bestehenden Möglichkeiten und beschränkten Ressourcen realistisch umsetzbar ist - Proof of Value. Dabei steht die gesamte Untersuchung unter der Annahme, dass die Verfügbarkeit von Parkplätzen zwar oberflächlich betrachtet ein stochastischer Prozess ist, der nur von Zufall oder vom “Glück” des Parkplatzsuchenden abhängig ist, tieferblickend aber statistische Trends existieren, die maßgeblich die Verfügbarkeit beeinflussen. Sehr plakativ lässt sich diese Annahme am Beispiel eines öffentlichen Parkplatzes nahe der HAW veranschaulichen, welcher unter der Woche zu Arbeitszeiten nahezu immer vollständig belegt ist, sonntagnachts aber beinahe vollkommen leersteht.

Somit werden im Folgenden verschiedene Verfahren innerhalb des Bereichs Data Mining dargelegt, Daten-Faktoren erörtert, die für eine solche Anwendung eine Rolle spielen, Algorithmen praktisch evaluiert und abschließend eine Bewertung der Machbarkeiten vor dem Hintergrund der Vision dieser App abgegeben.

# 2. Einführung ins Data Mining

## 2.1. Definition von Data Mining

Das Oxford English Dictionary definiert Data Mining wie folgt: „The practice of examining large pre-existing databases in order to generate new information“, also die Untersuchung von existierenden Datenbanken bzw. Datenbeständen mit dem Ziel neue Informationen zu generieren bzw. gewinnen. Der amerikanische Data Scientist Usama Fayyad fasst Data Mining zusammen als die nicht-triviale Entdeckung gültiger, neuer, potenziell nützlicher und verständlicher Muster in Datenbeständen. (Fayyad u. a., 1996, S. 6)
Der niederländische Professor Adriaans zieht bildhaft eine Analogie von Data Mining zum Bergbau (auf Englisch „Mining“). In letzterem werden durch großen Aufwand enorme Massen zutage gefördert und untersucht, mit dem Ziel wertvolle Rohstoffe zu finden, in ersterem werden durch großen Aufwand enorme Datenmengen durchsucht und verarbeitet mit dem Ziel, den wertvollen Rohstoff Information zu gewinnen. (Adriaans und Zantinge, 1996, S. 8)

## 2.2. Einordnung von Data Mining

Die Wissenschaft der Datenanalyse unterscheidet zwei Kategorien von Problemen: (Knobloch, 2000, S. 9)

1. Hypothesengetriebene Probleme, bei denen das Ziel die Verifikation oder Falsifikation bestehender Hypothesen durch Auswertung von Datenbeständen ist. Dieses Verfahren wird als Top-Down-Approach bezeichnet, da zu Beginn eine Hypothese steht, auf die hingehend Daten untersucht werden.

2. Hypothesenfreie Probleme, bei denen keine Hypothese geprüft wird, sondern versucht wird, aus den bestehenden Daten mittels verschiedener Methoden Erkenntnisse zu gewinnen. Dieses Verfahren wird als Bottom-Up-Approach bezeichnet, da aus einem Bestand von Daten Erkenntnisse, bzw. Erkenntnis-Hypothesen, gewonnen werden sollen.

Gemäß Fayyads Definition von Data Mining ist dieses somit in die Kategorie der hypothesenfreien Probleme einzuordnen. Ein Beispiel für ein hypothesengetriebe Herangehensweise ist das sogenannte OLAP (Online Analytical Processing), bei dem im Rahmen von z.B. Business Intelligence Anwendungen bestehende Daten aus verschiedenen Blickwinkeln („Dimensionen“) betrachtet werden können, um eine schnelle und flexible Analyse zu ermöglichen.

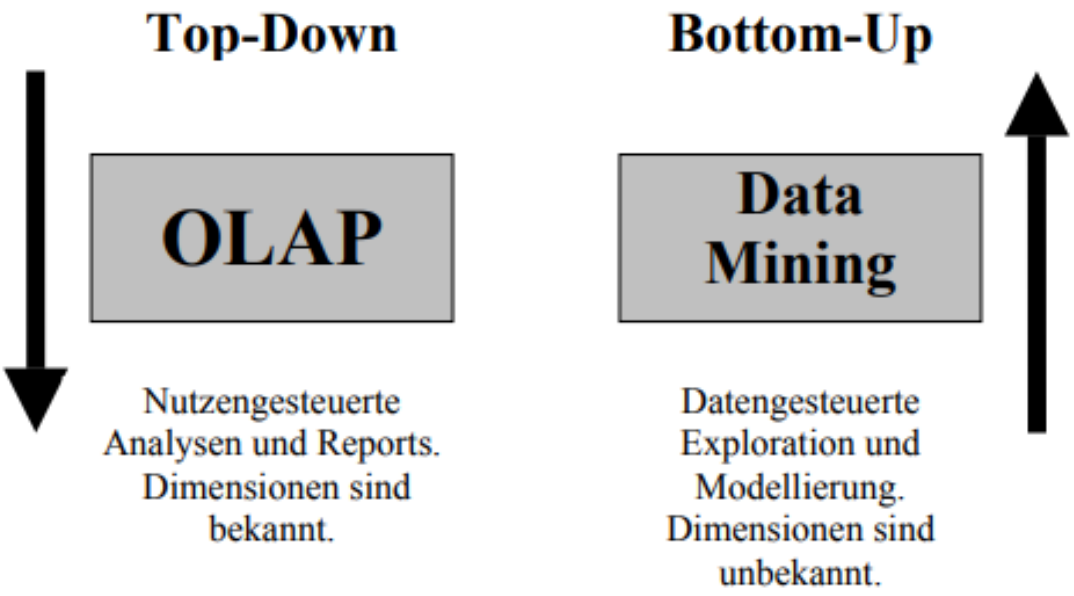


Abbildung 2.1.: Vergleich zwischen Top-Down- und Bottom-Up-Analyse, OLAP und Data Mining. (Gottermeier, 2003, S. 12)

Hierbei ist wichtig anzumerken, dass OLAP und Data Mining sich nicht gegenseitig ausschließen, sondern ergänzen. Erkenntnisse aus Data Mining lassen sich mittels OLAP verifizieren und durch OLAP verifizierte, aber noch unerklärte Korrelationen lassen sich mittels Data Mining potentiell ergründen.

## 2.3. Abgrenzung zu Knowledge Discovery in Databases

Die Begriffe Data Mining und Knowledge Discovery in Databases sind eng miteinander verknüpft und werden in der Literatur häufig synonym zueinander verwendet. Da diese sich aber durchaus unterscheiden, ist es notwendig hier eine klare Abgrenzung zu schaffen:

Knowledge Discovery in Databases (KDD) bezeichnet einen gesamtheitlichen Prozess zur Erschließung neuer Erkenntnisse aus meist großen Datenbeständen und erweitert somit

sein vermeintliches Synonym Data Mining um mehrere kritische Schritte. Das Prozessmodell nach Fayyad, Piatetsky-Shapiro und Smyth visualisiert dies deutlich:

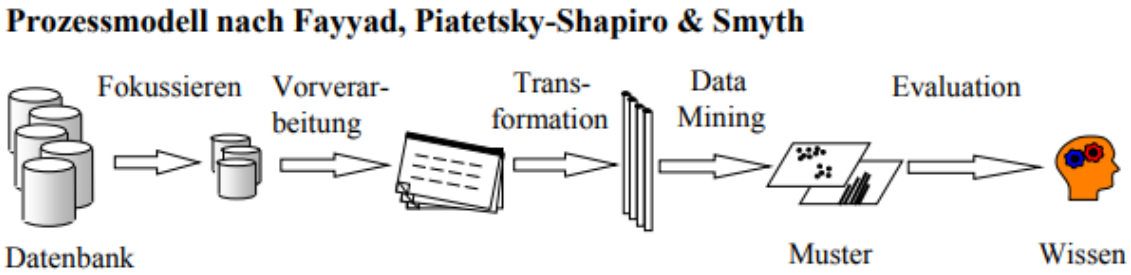


Abbildung 2.2.: Prozessmodell nach Fayyad, Piatetsky-Shaüiro & Smyth. (Böhm u. a., 2003, S. 7).

Hierbei sind die dargestellten Schritte wie folgt zu verstehen:

- Fokussieren: Datenbeschaffung und Selektion relevanter Daten.
- Vorverarbeitung: Möglicherweise Zusammenführung von Daten mehrerer Quellen, Vervollständigung und Konsistenzprüfung mit dem Ziel, die Datenqualität zu erhöhen.
- Transformation: Umwandlung der Daten in für Data Mining bearbeitbare Formate, sowohl Dateiformat und -struktur, als auch z.B. möglicherweise die Diskretisierung von numerischen Merkmalen.
- Data Mining: In diesem Schritt werden Algorithmen verwendet, um Muster zu erkennen und/oder Modelle zu entwickeln. Hierauf liegt der Schwerpunkt dieser Bachelorarbeit.
- Evaluation: Bewertung der möglichen Relevanz der Erkenntnisse und Validierung der der erzeugten Modelle mit statistischen Methoden.

## 2.4. Einsatzgebiete von Data Mining

Data Mining findet in Wissenschaft und Forschung Anwendung; so wird es beispielsweise im Bereich der Humangenetik verwendet, um Hypothesen über zu generieren, welche DNA-Sequenz das Risiko beeinflussen, an Krankheiten wie Krebs zu erkranken (Zhu und Davidson, 2007, S. 18) oder im Bereich der Analyse von medizinischen Versuchsergebnissen, um bspw. die Leistungsfähigkeit neuer Behandlungsmethoden zu untersuchen. (Zhu und Davidson, 2007, S. 31-48)

Aber auch in der freien Wirtschaft wird Data Mining genutzt. So finden bspw. Analysen von persönlichem Kaufverhalten im Internet - wo Käufer über ihren Account mit Käufen assoziiert werden können - aber auch offline - wo diese Assoziation bspw. durch die Verwendung von Kunden- bzw. Treuekarten ermöglicht wird - von Seiten der Verkäufer bzw. Plattformen statt.

Ein Beispiel, das 2012 mediale Beachtung fand, ist der Fall der Vorhersage einer Schwangerschaft durch den amerikanischen Einzelhandelsdiscounter Target Golgoski (2012). Dieser hatte einer jungen Dame mit einer Kundenkarte Werbung für Babyprodukte zukommen lassen. Der sich hierüber beschwerende Vater erfuhr erst dadurch von der Schwangerschaft seiner Tochter. Basierend auf ihren Einkäufen bei Target wie Vitaminpillen und größerer Mengen Cremes war Target in der Lage, die Schwangerschaft zu identifizieren, bevor der eigene Vater dies konnte.

Ein für die meisten alltägliches Beispiel von Data Mining sind Produktvorschläge etwa bei Amazon basierend auf vorherigen Einkäufen und Produktsuchen oder Werbeanzeigen basierend darauf, welche Websites in der nahen Vergangenheit besucht wurden und wonach bei Suchmaschinen gesucht wurde. Bei manchen Webshops, etwa einigen Buchungsportalen für Flüge, wird sogar davon ausgegangen, dass sie ihre angezeigten Preise basierend auf Erkenntnissen aus unter anderem Suchhistorie, Betriebssystem und Browser-Cookies verändern Collinson (2010). Dem zugrunde liegt eine Zielgruppenanalyse die, mit hoher Wahrscheinlichkeit mit Werkzeugen der Data Minings identifiziert wurden.

In der Wirtschaft eröffnen sich mit rasant zunehmenden Datenmengen, etwa durch Digitalisierung der Industrie und des persönlichen Alltags durch multisensorische Smartphones, Smart Homes, Smart Cars und die steigende Nutzung von elektronischen Zahlungsmitteln, sowie die Nutzung von sozialen Netzwerken immer bessere Möglichkeiten Kunden zu identifizieren, analysieren und Vorhersagen über den Erfolg etwa von Werbemaßnahmen zu treffen. Die Maßgabe scheint hier oft zu sein: *Wir wissen, was der Kunde will, bevor er es selbst weiß*.

# 3. Arten von Data-Mining-Verfahren

Der Begriff Data Mining umschließt eine Vielzahl von Methoden mit deren Anwendung Informationen aus Datenbeständen gezogen werden sollen. Diese lassen sich grob in folgende drei Kategorien einteilen:

- Assoziationsverfahren: Diese sollen Korrelationen zwischen verschiedenen Merkmalen einer Datenmenge entdecken.
- Clusteringverfahren: Diese teilen Datenmengen in Gruppen, genannt Cluster, ein, die ähnliche Eigenschaften aufweisen. Man spricht hier auch von der Entdeckung von Ähnlichkeitsstrukturen.
- Klassifikationsverfahren: Diese dienen zur Vorhersage eines oder mehrere Merkmale auf Grundlage anderer Merkmale einer Inputmenge.

Die untenstehende Grafik visualisiert diese Einteilung und ordnet jene Algorithmentypen ein, die im weiteren genauer erörtert werden:

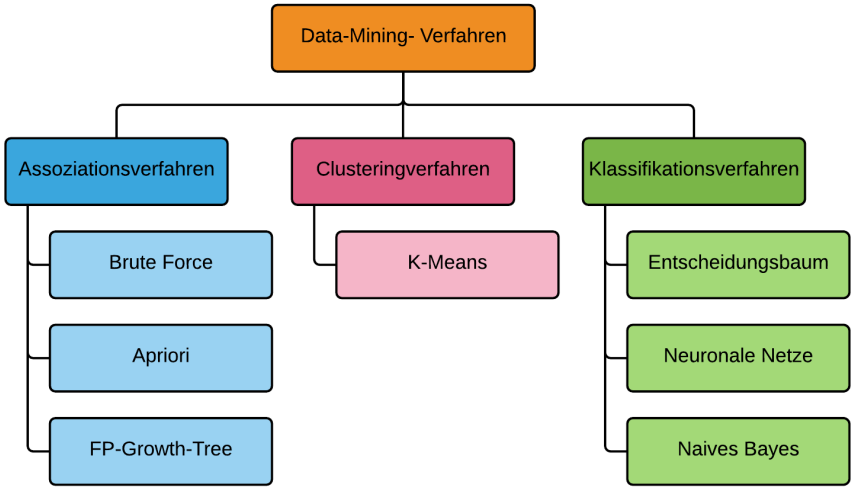


Abbildung 3.1.: In der Arbeit dargelegte Verfahren und Algorithmen

## 3.1. Assoziationsanalyse

Assoziationsverfahren versuchen, aus einem Datenbestand Regeln zu extrahieren, etwa in Form von Wenn-Dann-Beziehungen. Das wohl bekannteste Beispiel hierfür ist die Warenkorbanalyse, bei welcher versucht wird, auf Basis von großen Datenmengen von Einkäufen Verhaltensregeln abzuleiten. Denkbar wäre hier bspw. eine Regel wie „Wenn der Kunde Bier kauft, kauft er zu 80% beim selben Einkauf auch Kartoffelchips“. Durch Nutzung von Kundenkarten lassen sich nicht nur einzelne Einkäufe betrachten, sondern mehrere Einkäufe eines Kunden entlang einer zeitlichen Achse untersuchen, woraus sich etwa Regeln entwickeln lassen wie „Wenn der Kunde eine Waschmaschine gekauft hat, wird er zu 40% innerhalb der darauffolgenden 10 Wochen auch einen Trockner kaufen“. Solche Untersuchungen, bei denen die Reihenfolge der untersuchten Datensätze eine wichtige Rolle spielen werden als Sequenzanalysen bezeichnet.

Zur formalen Beschreibung eines Assoziationsproblems wird eine Datenmenge $D$ betrachtet, die aus Transaktionen $t$ besteht. Jede Transaktion $t$ besteht wiederum aus einer Menge unterschiedlicher Elemente mit unterschiedlichen Wahrscheinlichkeiten. Eine Assoziationsregel wird in Form von impliziten Schlüssen $X \rightarrow Y$ formuliert, was bedeutet, dass Verhalten $X$ zu Verhalten $Y$ führt. $X$ wird hierbei als Regelrumpf bezeichnet, was dem „Wenn“-Teil einer Wenn-Dann-Beziehung entspricht, $Y$ als Regelkopf, was dem „Dann“-Teil entspricht. (Schinzer u. a., 1999, S. 118)

Assoziationsregeln lassen sich anhand von drei wichtigen Kriterien bewerten: Support, Konfidenz und Lift.

**Support** Der Support gibt die relative Häufigkeit der Transaktionen an, die die Regel erfüllen. Berechnet wird dieser durch Division der Anzahl der die Regel erfüllende Transaktionen durch die Gesamtzahl alle Transaktionen in $D$.

$$\text{supp}(X \rightarrow Y) = \frac{|\{t \in D | (X \cup Y) \subseteq t\}|}{|D|}$$

Je nach Anwendungsfall kann es sinnvoll sein einen Mindest-Support zu wählen um sicherzustellen, dass die gefundenen Assoziationen relevant genug sind.

**Konfidenz** Die Konfidenz gibt an, für welchen Anteil der Transaktionen, in denen $X$ gegeben ist, die Regel $X \rightarrow Y$ zutrifft. Dieser wird berechnet durch Division der Anzahl der Transaktionen, für die die Regel zutrifft, durch die Gesamtzahl der Transaktionen die, $X$ enthalten.

$$\text{conf}(X \rightarrow Y) = \frac{|\{t \in D | (X \cup Y) \subseteq t\}|}{|t \in D | X \subseteq t|}$$

Wie für den Support kann es auch für die Konfidenz sinnvoll sein eine Mindestgröße zu wählen, etwa um sicherzustellen dass es sich bei der gefunden Regel nicht um zufällige Korrelationen handelt.

**Lift** Der Lift gibt an, wie hoch die Konfidenz der Regel den Erwartungswert übertrifft und ist somit ein Maß der Bedeutung der Assoziation. Zur Berechnung wird der Support der Regel durch das Produkt der Supports der assoziierten Verhalten dividiert.

$$\text{lift}(X \to Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) * \text{supp}(Y)}$$

Kaufen bspw. 30% aller Kunden $\{\text{Eier, Milch}\}$, 40% $\{\text{Eier}\}$ und 50% $\{\text{Milch}\}$, so ergäbe sich ein Lift von:

$$\text{lift}(\text{Eier} \to \text{Milch}) = \frac{\text{supp}(\text{Eier} \cup \text{Milch})}{\text{supp}(\text{Eier}) * \text{supp}(\text{Milch})} = \frac{0.3}{0.4 * 0.5} = 1.5$$

Würden nur 25% aller Kunden sowohl $\{\text{Eier}\}$ als auch $\{\text{Milch}\}$ kaufen, ergäbe sich ein Lift von:

$$\text{lift}(\text{Eier} \to \text{Milch}) = \frac{\text{supp}(\text{Eier} \cup \text{Milch})}{\text{supp}(\text{Eier}) * \text{supp}(\text{Milch})} = \frac{0.25}{0.4 * 0.5} = 1.25$$

Ein höherer Lift ist somit Indikator für die Ungewöhnlichkeit der Assoziation, allerdings nicht notwendigerweise für die Relevanz. Er darf hierfür nur als Indikator in Kombination mit den anderen Faktoren betrachtet werden, da ein hoher Lift auch durch sehr geringe Supports der einzelnen Verhalten entstehen kann. Kaufen bspw. 1% aller Kunden $\{\text{Eier, Milch}\}$, 2% $\{\text{Eier}\}$ und 2% $\{\text{Milch}\}$, so ergäbe sich ein Lift von:

$$\text{lift}(\text{Eier} \to \text{Milch}) = \frac{\text{supp}(\text{Eier} \cup \text{Milch})}{\text{supp}(\text{Eier}) * \text{supp}(\text{Milch})} = \frac{0.01}{0.02 * 0.02} = 25$$

Es besteht also eine starke Assoziation, welche aber nur auf wenig Fälle zutrifft, was etwa am geringen Support von 0.01 erkennbar ist.

### 3.1.1. Brute-Force

Wie etwa bei Entschlüsselungsverfahren basieren auch in der Assoziationsanalyse Brute-Force-Algorithmen (aus dem Englischen „brute force“ für „rohe Gewalt“) auf der Idee der systematischen Erprobung aller möglicher Kombinationen. Hierbei werden für jede mögliche

Kombination Support und Konfidenz ermittelt und die Assoziation gegebenenfalls als potentiell interessant bewertet, wenn sie angegebene Mindestwerte für Support und Konfidenz überschreiten. Insbesondere bei komplexeren Analysen, also solchen mit mehreren Faktoren und/oder der Möglichkeit der Berücksichtigung mehrelementriger Regeln, wie etwa $\{\text{Brot}, \text{Butter}\} \rightarrow \{\text{Milch}\}$, kann das Brute-Force-Verfahren in der Regel nicht praktikabel verwendet werden, da der Rechenaufwand schnell stark ansteigt.

### 3.1.2. Apriori-Algorithmus

Ein verbessertes Verfahren der Assoziationsanalyse stellt der Apriori-Algorithmus dar. Dieser basiert auf der Annahme, dass Teilmengen einer häufigen Menge selbst häufig auftreten und benötigt angegebene Mindestwerte für Support und Konfidenz.

Der Algorithmus arbeitet nach dem folgenden Prinzip:

1. Entferne alle Mengen mit je einem Element für die gilt $\text{supp}(\text{Menge}) < \text{Mindestsupport}$
2. Erstelle aus den verbleibenden Mengen alle möglichen Kombinationen, die zwei Elemente enthalten ($k = 2$)
3. Entferne alle Mengen für die gilt $\text{supp}(\text{Menge}) < \text{Mindestsupport}$
4. Erstelle aus den verbleibenden Mengen alle möglichen Kombinationen, die drei Elemente enthalten ($k = 3$).
5. Entferne alle Mengen die Teilmengen mit $k = 2$ Elementen enthalten, die nach Schritt 4 nicht mehr in der Menge enthalten waren
6. Wiederhole Schritte 3 bis 5 mit jeweils inkrementiertem $k$ bis entweder $k$ der Gesamtzahl aller Elemente entspricht oder die gefilterte Menge leer ist.

Alle in den jeweiligen Schritten nicht entfernten Mengen bilden nun die Basis, auf derer Assoziationsregeln generiert werden. Der Vorteil dieses Vorgehens liegt darin, dass in jedem Schritt zwar mehr Aufwand durch die Entfernung nicht relevanter Kombinationen betrieben werden muss, dadurch aber zahlreiche Kombinationen entfallen, die in zukünftigen Schritten untersucht werden müssten.

### 3.1.3. FP-Growth-Algorithmus

Der Frequent-Pattern-Growth-Algorithmus verwendet zur Ermittlung möglicher relevanter Assoziationen ein Verfahren unter Zuhilfenahme einer Baumstruktur, die als FP-Tree bezeichnet wird. Das Verfahren ist in zwei Schritte geteilt: zunächst wird der FP-Tree mit den vorhandenen Transaktionen erstellt, anschließend dieser analysiert. Wie bereits beim Apriori-Algorithmus ist es notwendig, Mindestwerte für Support und Konfidenz anzugeben.

In Schritt 1 werden alle Sets mit genau einem Element bestimmt, die den Mindestsupport erfüllen. Diese werden nach ihrer Häufigkeit absteigend sortiert und in eine Tabelle, die Frequent Item Header Table, geschrieben. Nun wird ein Wurzelelement für den Baum definiert. Anschließend werden für jede Transaktion der Datenmenge folgende Aktionen unternommen:

1. Die Elemente einer Transaktion werden gemäß der Reihenfolge des Frequent Item Header Tables sortiert.
2. Die Transaktion wird in den Baum eingehangen. Dies erfolgt, indem für das erste Element überprüft wird, ob ein Ast existiert, welcher dem Element entspricht. Falls nicht, wird ein entsprechender Ast erstellt und dessen Zählvariable instanziiert. Von diesem Ast ausgehend wird nun das zweite Element nach dem gleichen Verfahren eingehangen usw.
   Steht ein potentiell einzuhängendes Element nicht im Frequent Item Header Table, so wird dieses nicht eingehängt und es wird zur nächsten Transaktion übergegangen.
3. Bei jeder Überstreichung eines Knotens wird dessen Zählvariable inkrementiert.

Am Ende dieser Aktionen steht ein Baum, dessen Knoten jeweils Zählerwerte besitzen. Dieser wird nun zurechtgestutzt (im Englischen ‘pruning’ genannt), indem Äste ab dem Punkt entfernt werden, ab dem der Knoten nicht mehr der geforderten Mindestkonfidenz entspricht. Auf Basis dieses gestutzten Baumes werden nun Assoziationen ermittelt.

Zur Verdeutlichung ist im Anhang in der Sektion Beispiel FP-Growth Tree auf Seite 69 ein gekürztes Beispiel auf Basis eines Blogeintrages des Singularities-Blogs angeführt.

## 3.2. Clusteringverfahren

Bei Clusteringverfahren wird versucht, Objekte auf Basis von Distanzmaßen mehrerer Merkmale in Gruppen, sogenannte Cluster, einzuteilen. Hierbei sollen die Cluster nach innen

homogen sein, also alle Objekte möglichst ähnlich sein, nach außen hin möglichst heterogen, was bedeutet, dass die Cluster untereinander möglichst unähnlich sind; so sollte es in der grafischen Darstellung möglichst keine Überlappungen der Cluster geben.

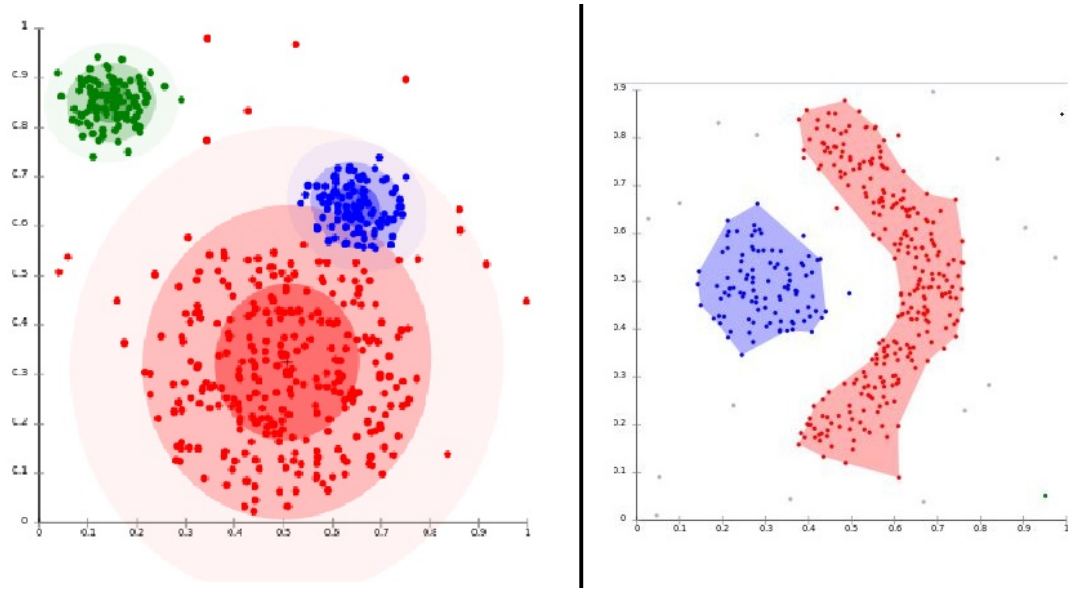

Abbildung 3.2.: Links: sich teilweise überschneidende Cluster/Clumps. Rechts: sauber voneinander abgegrenzte Cluster. Quelle: eigene Darstellung aus Wikimedia-Material

Zur Anwendung von Clusteringverfahren wird in folgenden Schritten vorgegangen:

1. Absteckung der Rahmenbedingungen abhängig von den Zielen der Analyse. Ein Beispiel ist die Klärung, ob ein Objekt ausschließlich in einem Cluster vorkommen darf, die Menge also in disjunkte Gruppen eingeteilt werden soll, oder es in mehreren Clustern vertreten sein darf; in letzterem Falle spricht man von Fuzzy Clustering oder Clumping. Ein weiteres Beispiel ist die Festlegung, ob die Anzahl der Cluster im Vorfeld definiert wird oder durch den jeweiligen Algorithmus selbst bestimmt werden darf. In diesem Fall muss festgelegt werden, wie viele Elemente notwendig sind, um einen Cluster zu bilden und ggf. wieviel Rechenzeit der Algorithmus verwenden darf.
2. Ermittlung und Bereitstellung der Daten. Hierbei kann es, wie in der Sektion Abgrenzung zu Knowledge Discovery in Databases auf Seite 10 dargelegt, sinnvoll sein, vorhandene Daten aufzubereiten, etwa durch Auslassen von wenig relevanten Merkmalen oder statistische und stochastische Verfahren wie die Normalisierung von Werten und das Entfernen von Ausreißern.

3. Festlegung von Proximitätsmaßen, also anhand welcher Maße Objekte als ähnlich zusammengruppiert - geclustert - werden. Die Wahl ist hierbei abhängig vom Format der vorliegenden Daten, welche sich in folgende drei Kategorien einteilen lassen:

   a) Nominale Daten, etwa binäre Angaben wie das Geschlecht[1], Student/Nicht-Student, und so weiter. Hierbei wird als Ähnlichkeitsmaß die Anzahl der Übereinstimmungen zwischen Objekten, welche unterschiedlich normiert werden. So können bspw. Übereinstimmungen einzelner Merkmale stärker gewichtet werden als anderer.

   b) Ordinale Daten, also Daten mit Merkmalen die eine zugrundeliegende Ordnung bzw. Reihenfolge besitzen. Ein Beispiel hierfür sind Zufriedenheitsstufungen wie „sehr zufrieden“ $>$ „eher zufrieden“ $>$ „eher unzufrieden“ $>$ „sehr unzufrieden“. Diese lassen sich anhand der Reihenfolge in metrische Daten überführen.

   c) Metrische/Rationale Daten, also Daten, die in numerischer Form vorliegen, wie etwa Kilometerstände bei Fahrzeugen oder das Jahreseinkommen von Personen. Hierbei lässt sich als Distanzmaß die euklidische Distanz verwenden.

   Bei der Behandlung von Daten mit Merkmalen mehrerer Kategorien gibt es grundsätzlich zwei Möglichkeiten:

   a) Transformation der Daten auf das geringste gemeinsame Niveau, also etwa die Diskretisierung sämtlicher rationaler Werte in binäre größen wie „größer als 200m“ und „kleiner als 200m“, was einen hohen Informationsverlust bedeuten kann, oder

   b) die getrennte Berechnung der Ähnlichkeiten der jeweiligen Merkmale einer Kategorie und die anschließende Aggregation, wobei hier auf eine sinnvolle Gewichtung der Ergebnisse zu achten ist.

4. Auswahl eines Clustering-Algorithmus. Hierbei ist auf die Anforderungen an das Clusterverfahren zu achten, etwa das Umgehen mit Ausreißern, die benötigte Rechenleistung bei verschieden großen Datensätzen und die Funktionalität abhängig von der Anzahl der Inputparameter.

5. Anwendung des Clustering-Algorithmus. Hierbei ist zu beachten, dass die meisten Algorithmen heuristisch vorgehen, also nicht die optimale Lösung finden, sondern eine Lösung ermitteln, die sowohl brauchbar als auch mit vertretbarem Aufwand bestimmbar ist. Verdeutlicht werden kann die Notwendigkeit hierfür daran, dass bereits für die Aufteilung von 10 Elementen in zwei mögliche Cluster 511 Kombinationen vorliegen,

[1] In dieser Arbeit wird von nur zwei existierenden Geschlechtern ausgegangen

bei 100 Elementen und zwei Clustern bereits über $10^{29}$ Möglichkeiten, was einen naiven Ansatz wie Brute Force mit anschließender Bewertung unpraktikabel macht. Stattdessen stellen die meisten Algorithmen eine Zielfunktion auf, die optimiert werden soll.

6. Nach Ablauf kann die Clusterisierung mit weiteren Daten getestet werden und ggf. auf kontextuell relevante Aussagen hin interpretiert werden.

Anwendung finden Clustering-Verfahren z.B. im Marketing in der Zielgruppenanalyse und -definition, aber auch bswp. in der Mustererkennung bei automatischer Bilderkennung. Ein weiterer Anwendungsfall ist die Erstellung von Clustern zur Reduktion von Datenmengen für weitere Untersuchungen. Da Cluster per Definition eine gewisse Homogenität darstellen, können weitere Verfahren wie Assoziationsanalysen oder Klassifikationsverfahren, aber auch andere statische Analysen innerhalb von Clustern angewendet werden und genauere Ergebnisse mit geringerem Aufwand zu ermitteln.

Clusteringverfahren lassen sich in mehrere Kategorien einteilen, von denen die zwei bekanntesten hier dargelegt werden.

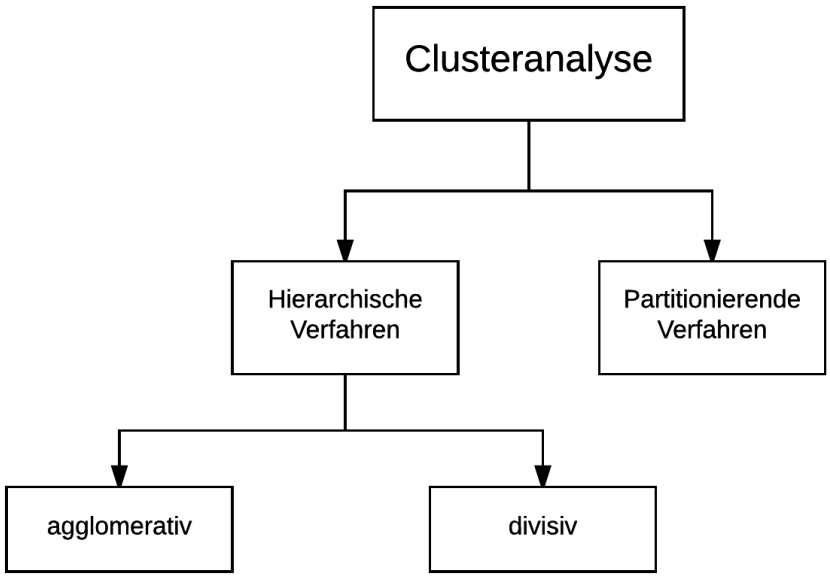


Abbildung 3.3.: Arten von Clusteranalyseverfahren

### 3.2.1. Hierarchische Verfahren

Hierarchische Verfahren unterteilen eine Menge von Objekten basierend auf geringer Distanz bzw. hoher Ähnlichkeit in Cluster ein. Hierbei wird basierend auf der Vorgehensrichtung zwischen zwei Typen unterschieden:

Bei agglomerativen hierarchischen Verfahren bildet zunächst jedes Objekt einen eigenen Cluster. Diese werden miteinander verglichen; wird eine Ähnlichkeit anhand der definierten Ähnlichkeitsmaße festgestellt, werden Cluster zusammengelegt. Dieser Vorgang wird wiederholt bis eine Abbruchbedingung erreicht ist, etwa die vordefinierte Clusterzahl erreicht ist, oder bis sich alle Objekte in einem Cluster befinden.
Das Gegenteil hierzu bilden divisive hierarchische Verfahren; hier befinden sich zunächst alle Objekte in einem Cluster. Sukzessiv werden bestehende Cluster anhand von Ähnlichkeitsmerkmalen in immer kleinere Cluster aufgeteilt.

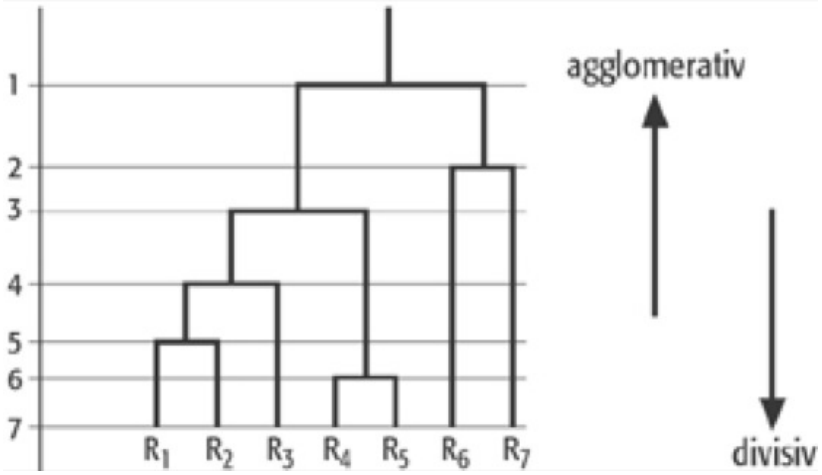


Abbildung 3.4.: Vergleich agglomerativer und divisiver Verfahren: Die Y-Achse beschreibt die Anzahl der Cluster, die Pfeile die Vorgehensrichtung der jeweiligen Verfahrensarten. Quelle: http://www.spektrum.de/lexika/images/geogr/hier_gru_w.jpg (Abfrage 21.8.2017)

Hierarchische Verfahren lassen sich mit Dendrogrammen, also Visualisierungen mit Baumdiagrammen, gut darstellen.

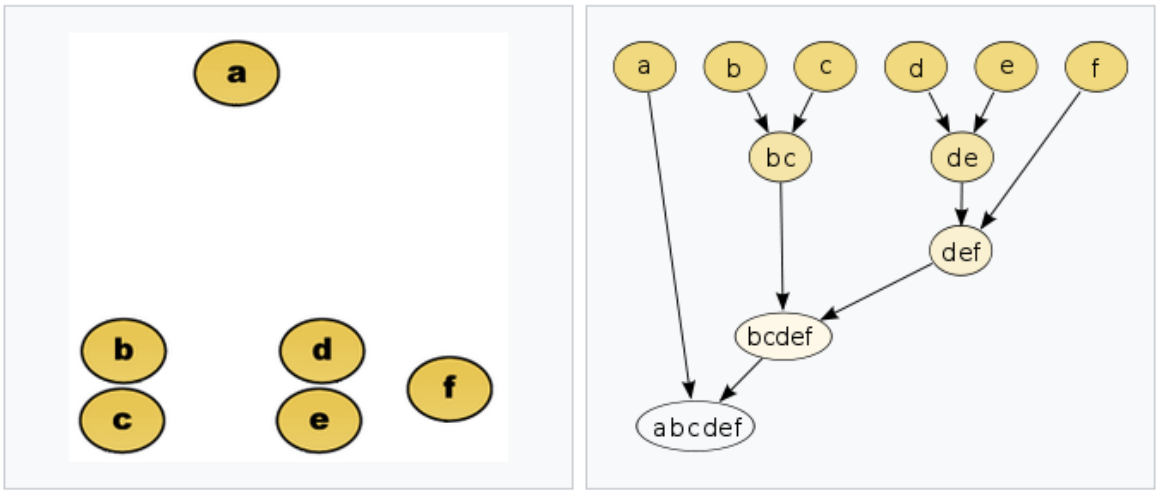


Abbildung 3.5.: Beispiel eines agglomerativ gebildeten Dendrogramms: Datensatz (links) und darauf basierendes Dendrogramm (rechts). Quelle: https://de.wikipedia.org/wiki/Hierarchische_Clusteranalyse (Abfrage 21.8.2017)

### 3.2.2. Partitionierende Verfahren

Hierbei wird von einer zufälligen Startpartitionierung der Objekte in eine vorgegebene Anzahl von Clustern ausgegangen. Die Zentren diese Cluster werden schrittweise so verschoben, dass die Güte der Gruppierung immer weiter verbessert wird. Ein klassisches Beispiel hierfür ist der k-Means-Algorithmus, der wie folgt vorgeht:

Zunächst liegen alle Objekte in einem n-dimensionalen Raum; ihre Position wird durch den Vektor ihrer Eigenschaften beschrieben. Zufällig im Raum werden nun die Mittelpunkte der Cluster verteilt, wobei deren Anzahl vorher definiert worden sein muss.

Alle Objekte werden nun dem Cluster zugeordnet, dessen Mittelpunkt sich ihnen auf Basis einer Distanzfunktion am nächsten befindet.

Die Positionen der Mittelpunkte wird neu berechnet, sodass er im Schwerpunkt aller Punkte innerhalb eines Clusters liegt.

Nun werden erneut die Punkte dem Cluster zugeordnet, dessen Mittelpunkt sie am nähesten sind.

Die letzten beiden Schritte werden so lange wiederholt bis eine definierte Iterationstiefe erreicht wurde oder sich die Güte der Unterteilung nicht weiter verbessert.

## 3.3. Klassifikationsverfahren

Klassifikationsverfahren ermitteln Muster, anhand derer Objekte Klassen zugeordnet, also klassifiziert, werden. Anders als bei Clustern, bei denen die Einordnung in einen Cluster prinzipiell als Klassifikation angesehen werden kann, sind bei Klassifikationsverfahren die Klassen vorher bekannt. Jedes Objekt kann genau einer Klasse zugeordnet werden, insofern unterscheiden sich Klassifikation darüber hinaus von Fuzzy-Clustering-Verfahren. Eine Klasse ist hierbei die Ausprägung eines bestimmten Merkmals, ob es sich beispielsweise bei einem bestimmten Kraftfahrzeug um einen LKW, einen PKW oder ein Motorrad handelt. Die Kraftfahrzeugart mit den Optionen LKW, PKW und Motorrad wäre hier somit die Klasse.

Die zu lösende Problemstellung bei Klassifikationsverfahren kann wie folgt formuliert werden: „Welcher Klasse kann ein neues Objekt zugeordnet werden, basierend auf dem Wissen um bereits bekannte Objekte, deren Merkmale und Klassen“.

Zur Lösung dieses Problems werden drei wesentliche Schritte unternommen:

1. Die Trainingsphase, in welcher ein Classifier erstellt wird. Hierbei wird mittels eines Algorithmus auf Basis der Menge bekannter Objekte oder einer Teilmenge dieser ein Modell, bspw. ein Regelsatz, gebildet. Dieses Modell ist in der Lage, jedes noch nicht klassifizierte Objekt anhand seiner Merkmalskombination genau einer Klasse zuzuordnen. Dieses Modell wird als Classifier, also zu Deutsch etwa Klassifizierer oder Klassifikator, bezeichnet.

2. Die Testphase, in welcher die Performance des Classifiers geprüft wird. Abhängig von den Gegebenheiten der Datenmenge, wie etwa Größe und Anzahl der Merkmale, bilden verschiedene Algorithmen verschieden gute Classifier. Daher ist es wichtig, die Performance der Classifier in Hinblick auf korrekte Klassifizierung zu prüfen. Hierzu werden Testsets verwendet; Mengen von Objekten, deren jeweilige Klassen bekannt sind. Diese werden klassifiziert und die Ergebnisse der Klassifizierung mit den zuvor bekannten Klassen verglichen. Auf verschiedene Verfahren und Kriterien der Testung und zur Bildung von Testsets wird im Kapitel Algorithmen-Evaluation ab Seite 49 weiter eingegangen.

3. Die Klassifikationsphase, in welcher neue Objekte klassifiziert werden. Hat die Testphase ergeben, dass der Classifier hinreichend gut klassifiziert, können mit Hilfe dessen neue Objekte klassifiziert werden.

Anwendung finden Klassifikationsverfahren unter anderem in Risikoanalysen von Versicherungen. So werden die Höhen der Prämien bei Lebensversicherungen auf Basis von zahlreichen Faktoren wie Beruf, Gesundheitszustand, Wohnort und Geschlecht anhand von Modellen bestimmt, die auf Basis von Kundendaten und evtl. zugekauften Daten erstellt wurden. Ein stark vereinfachtes Beispiel wäre folgendes:

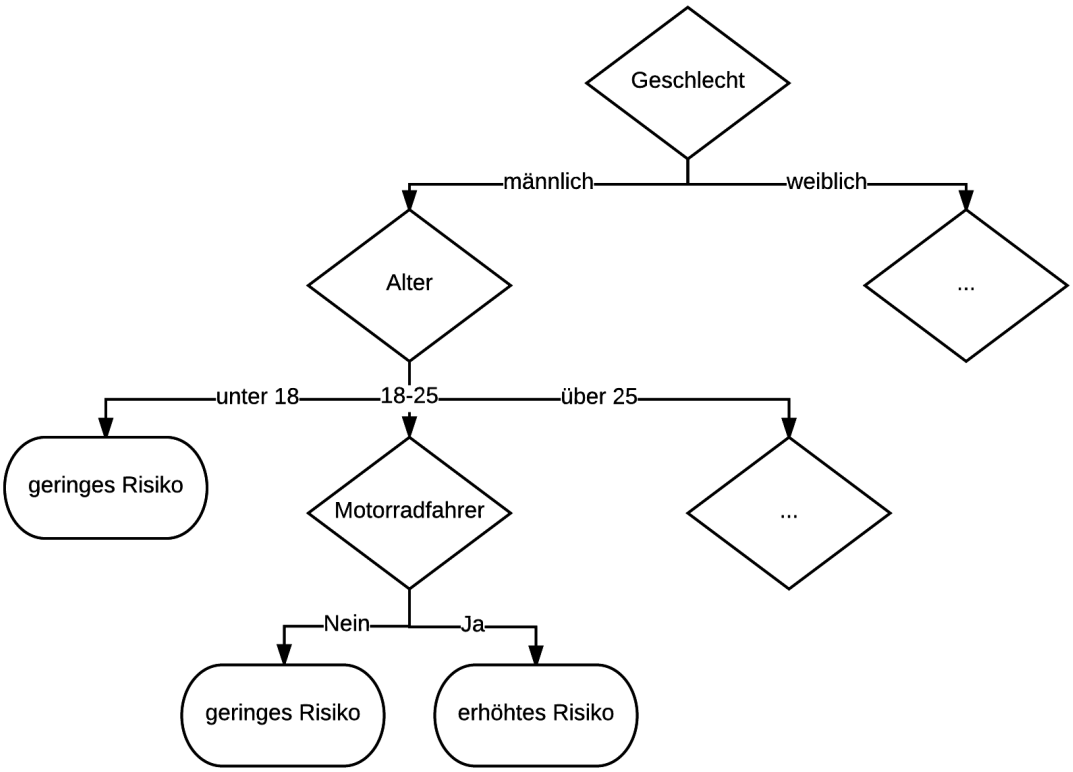


Abbildung 3.6.: Vereinfachtes Beispiel eines Entscheidungsbaumes für Versicherungen

Hierbei handelt es sich um einen Entscheidungsbaum, welches ein Verfahren zur Klassifikationsanalyse darstellt. Im Folgenden wird auf dieses, sowie zwei weitere Verfahren eingegangen.

### 3.3.1. Entscheidungsbäume

Bei Entscheidungsbäumen wird rekursive Partitionierung benutzt, um die Datenmenge zu unterteilen. Zunächst werden alle Objekte als eine Menge betrachtet. In dieser werden die bedingten Häufigkeitsverteilungen in Abhängigkeit der Attribute mittels eines Auswahlmaßes analysiert. Das am besten bewertete Attribut wird als Testattribut ausgewählt, anhand dessen die Menge in mehrere Teilmengen aufgeteilt wird. Da hierbei die Bewertung daraufhin ausgelegt ist, die bestmögliche Unterteilung zu erstellen, spricht man von „gierigen“ (im Englischen „greedy“) Algorithmen. Die Unterteilung der Mengen kann entweder in zwei Teilmengen erfolgen oder in mehrere. Im ersten Falle spricht man von binären Bäumen, im

letzteren von N-Way-Trees. Auf die nun gebildeten Teilmengen wird das gleiche Verfahren rekursiv angewendet, bis kein Attribut mehr zu einer Verbesserung der Klassifikation führt oder eine Aufteilung entlang aller Attribute erfolgt ist. Auch können weitere Abbruchbedingungen wie bspw. eine maximale Tiefe des Baums definiert werden.

Als Auswahlmaß zur Selektion des Testattributes wird unter anderem der Informationsgewinn (im Englischen „information gain“) verwendet, welcher misst, wie viel Information durch die Feststellung des Wertes des Testattributes gewonnen wird. Hierzu wird das Entropiemaß bei der Berechnung verwendet:

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

Der Informationsgewinn entspricht der Entropieverminderung beim Übergang in eine möglich neue Verteilung entlang des Testattributs und wird wie folgt berechnet: (Gottermeier, 2003, S. 30)

$$I_{gain}(C, A) = H_C - H_{C|A} = H_C + H_A - H_{CA} =$$

$$-\sum_{i=1}^{n_C} p(x_i) \log_2 p(x_i) - \sum_{j=1}^{n_A} p(x_j) \log_2 p(x_j) + \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} p(x_{ij}) \log_2 p(x_{ij})$$

Hierbei ist $H_C$ ist die Entropie der Klassenverteilung, $H_A$ die Entropie der Attributwerteverteilung und $H_{CA}$ die Entropie der gemeinsamen Verteilung. $n_C$ steht für die Anzahl an Klassen, $n_A$ für die Anzahl an Attributwerten.

In den meisten Fällen wird ein fertig konstruierter Baum anschließend gestutzt. Bei diesem als Pruning bezeichneten Vorgehen werden Entscheidungsknoten mit geringem Anteil an der Klassifikationsgüte entfernt. Zwar führt ein tieferer, also besser ausdifferenzierter, Baum zu einer höheren Klassifikationsgüte gemessen am Trainingsset; es verringert sich allerdings ab einem gewissen Punkt die Generalisationsfähigkeit, also die Möglichkeit mit dem Baum neue Objekte korrekt zu klassifizieren. Verdeutlichen lässt sich dies wie folgt:
Gehen wir von einer Datenmenge aus, deren Objekte keine Widersprüche enthalten, also keine zwei Objekte die bei gleichen Merkmalen unterschiedlich klassifiziert sind. Es ist nun prinzipiell möglich, einen Baum zu erstellen, der jedes Objekt korrekt klassifiziert. Dieser Baum wäre der ideale Baum für eine, zumeist kleine, Datenmenge. Versuchen wir aber nun, neue Daten mit diesem stark spezialisieren Classifiers klassifizieren zu lassen, kann dies zu schwerwiegenden Fehlern kommen, da unser Classifier keine möglicherweise zugrunde liegende Logik abbildet, sondern lediglich die Ausgangsmenge perfekt klassifizierte. Unser vermeintlich perfekter Baum kann sich so als unbrauchbar für neue Daten und somit als ungeeigneter Classifier herausstellen. Bei dieser zu starken Spezifizierung auf den

vorliegenden Datensatz spricht man von Overfitting. Dieses wird versucht durch Pruning zu vermeiden, wobei man zwischen Pre-Pruning und Post-Pruning unterschiedet. Ersteres bezeichnet die Begrenzung der Tiefe des Baumes zu Beginn des Trainings, letzteres das angesprochene Entfernen von Entscheidungsknoten mit wenig Auswirkung auf die Güte.

### 3.3.2. Künstliche Neuronale Netze

Künstliche Neuronale Netze, im Englischen „artificial neural networks (ANN)“ genannt, stellen einen Zweig der künstlichen Intelligenz dar und werden verwendet, um nicht-lineare funktionale Beziehungen zu modellieren. Hierzu wird ein Netzwerk aus Neuronen geknüpft, welches in drei Sektionen geteilt wird: die Inputneuronen, welche einen Eingangsvektor entgegennehmen, Outputneuronen, welche einen durch das Netzwerk berechneten Ausgangsvektor ausgeben und eine oder mehrere Schichten von inneren Neuronen. Letztere werden als „hidden“ bezeichnet, da sie keine direkte Kommunikation nach außerhalb des Netzes führen. Die Verbindungen zwischen den Neuronen der verschiedenen Schichten (Topologie) und deren jeweilige Gewichtungen sowie die Schwellenwerte für die Eingangsgrößen, ab denen ein Neuron „feuert“ beschreiben hierbei die Funktion des Netzes.

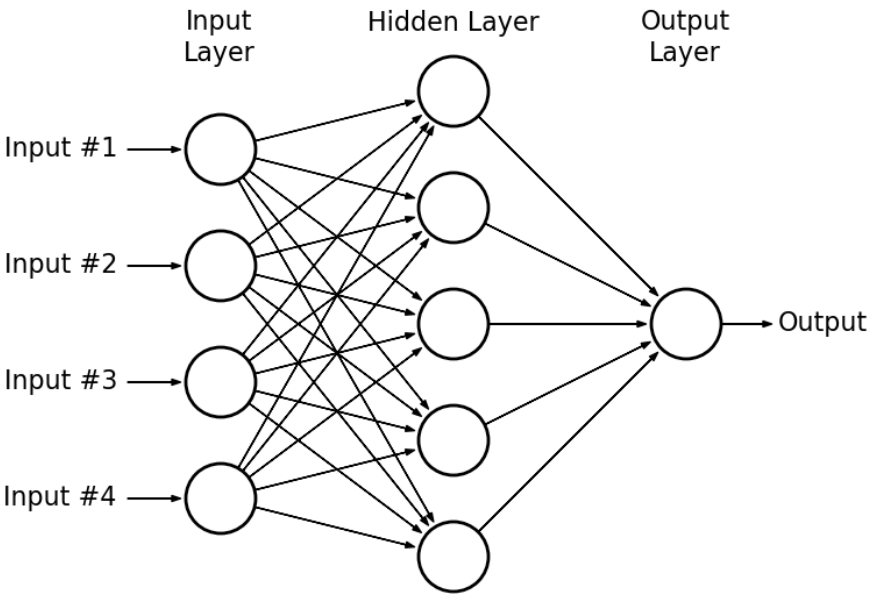


Abbildung 3.7.: Beispieltopologie eines AKK. Quelle: http://alexminnaar.com/implementing-the-distbelief-deep-neural-network-training-framework-with-akka.html

Die Topologie wird zumeist zu Beginn des Lernvorgangs eines Netzes definiert; für die Bestimmung der Gewichtungen werden Anfangswerte definiert, welche mit der Zeit durch einen

zu wählenden Lernalgorithmus verändert werden. Beim sogenannten überwachten Lernen (im Englischen „supervised learning“) werden Trainingssets bestehend aus In- und Output-Vektoren bereitgestellt. Aus dem Inputvektor wird über das Netz ein Outputvektor berechnet und dieser mit dem tatsächlichen Outputvektor verglichen. Die Gewichte werden durch den Algorithmus mit dem Ziel verändert, in jedem Schritt die Unterschiede zwischen berechnetem und tatsächlichem Outputvektor zu minimieren. Dieser Vorgang läuft solange, bis eine definierte Abbruchbedingung erreicht wurde, zumeist eine hinreichende Genauigkeit. Idealerweise ist das so erstellte Netz mit seinen optimierten Gewichtungen in der Lage, einen passenden, sinnvollen Ausgangsvektor für jeden Eingangsvektor zu ermitteln.

Ein großer Nachteil von neuronalen Netzwerken liegt darin, dass das letztlich erzeugte Resultat, also das Netz mit all seinen Gewichtungen, nicht nachvollziehbar ist. Zwar lässt sich die Topologie visualisieren - sofern die teils hohe Anzahl der Schichten und darin enthaltenen Neuronen dies nicht unpraktikabel macht - und die Gewichtungen der Verbindungen auslesen; dies ist aber weitaus weniger anschaulich als etwa die Visualisierung eines Entscheidungsbaumes. Die implizit erstellten Regeln lassen sich, insbesondere bei komplexeren Netzen, in denen unter anderem Rekursionen vorkommen können, nicht einfach auslesen. Daher können neuronale Netzwerke zumeist nur als Black-Boxen angesehen werden, was die Überprüfung der zugrundeliegenden Logik unmöglich und das Verfahren somit anfällig für das Prinzip „Garbage In, Garbage Out“ macht.

### 3.3.3. Naives Bayes

Ein verbreitetes Klassifikationsverfahren ist das naive Bayes, welches nach Thomas Bayes, einem britischen Mathematiker, benannt wurde, da es aus dessen Satz von Bayes abgeleitet ist. Als naiv wird es bezeichnet, da dieses Verfahren von einer Unabhängigkeit der Merkmale eines Objekts ausgeht. Obwohl dies in der Regel keine korrekte Annahme ist, zeigt sich bei Datensätzen mit weniger stark korrelierten Attributen, dass die Klassifikationen dennoch häufig gute Ergebnisse liefert. Bedingt durch diese „naive“, vereinfachende Annahme zeichnet sich naives Bayes durch einen geringen Rechenaufwand und damit schnelle Generierung von Modellen aus. Aufgrund der Schnelligkeit wird naives Bayes teilweise nicht als eigenständiger Classifier bentuzt, sondern nur verwendet um einen ersten Überblick über die Beziehungen der Attribute und der Klasse zu gewinnen, um unter Berücksichtigung dieser Erkenntnisse andere Klassifizierungsverfahren zu verwenden.

Bei der Bayes-Klassifikation wird ein Objekt der Klasse zugeordnet, die als wahrscheinlichsten für seine Merkmalskombination ist. Zur Bestimmung wird die Größe $p(C_i|X)$ verwendet, die die Wahrscheinlichkeit angibt, dass ein Objekt mit einem Merkmalsvektor $X$ in die Klasse $C_i$ gehört. Gesucht wird nach der Klasse, für die bei einem gegebenen Objekt mit dem Vektor $X$ die Wahrscheinlichkeit $p$ am größten ist, welche über den Satz von Bayes berechnet

wird:

$$p(C_i|X) = \frac{p(X|C_i)p(C_i)}{p(X)}$$

Hierbei gilt:

- $p(X)$ ist die Wahrscheinlichkeit des Auftretens des Vektors X in der Datenmenge
- $p(X|C_i)$ ist die Wahrscheinlichkeit für das Auftreten des Vektors X bei gegebener Klasse $C_i$ in der Datenmenge
- $p(C_i)$ ist die Wahrscheinlichkeit des Auftretens der Klasse $C_i$ in der Datenmenge

Zur Veranschaulichung der Berechnung nach dem Satz von Bayes befindet sich in der Sektion Beispiel Satz von Bayes auf Seite 72 ein Beispiel.

Unter der „naiven“ Annahme der Unabhängigkeit der Elemente im Vektor X lässt sich die Formel vereinfachen. Hierbei wurde $p(X)$ vernachlässigt, da dieser Wert für den Vektor $X$ konstant ist und somit für die hier angestrebte Optimierung von $p(C|X)$ nicht relevant ist.

$$p(C|X) \approx p(x_1|C) \times p(x_2|C) \times \ldots \times p(x_n|C) \times p(C)$$

$$\approx p(C) \prod_{i=1}^{n} p(x_i|C)$$

# 4. Evaluation von Klassifikationsverfahren

Für die in dieser Arbeit betrachteten Anwendungsfälle werden Algorithmen der Klassifikationsanalyse betrachtet, weshalb im Folgenden auf Methoden und Parameter der Evaluation von Verfahren dieser Art eingegangen wird.

Wie bereits in der Sektion zu Entscheidungsbäumen ab Seite 24 dargelegt, tendieren einige Algorithmen zu Overfitting, also der Erstellung eines Modells, welches zu speziell auf die vorhandene Datenmenge zugeschnitten und daher für auf weitere Datensätze nicht generalisierbar ist. In der Bewertung der Korrektheit der Klassifizierung - auf deren Bewertung später genauer eingegangen wird - ist es somit wichtig, das Modell nicht auf Basis des Trainingssets zu bewerten. Hierbei existieren einige grundlegende Verfahren, die unten erörtert werden. Wichtig ist in jedem Fall, dass die Größe der Datenmengen hinreichend groß ist, wobei diese Bewertung von Algorithmus zu Algorithmus unterschiedlich ist.

## 4.1. Bereitstellung eines Testsets

Hierbei stellt der Anwender dem verwendeten Tool einen separaten Datensatz zur Verfügung anhand dessen das Modell evaluiert wird. Es existieren also zwei separate Datensätze, zum einen das Trainingsset, zum anderen das Testset. Hierbei ist für die Aussagekraft des Tests wichtig, dass beide Datensätze unter gleichen Bedingungen entstanden sind, also das Testset prinzipiell die gleiche zugrundeliegende Logik aufgrund einer gemeinsamen Quelle besitzt. Ein Testset, dessen Datensätze mit denen des Trainingssets keinen oder nur einen geringen Zusammenhang besitzen, werden in den meisten Fällen schlechte Evaluationsergebnisse liefern, obwohl das Modell potentiell die Logik der Datenquelle des Trainingssets adäquat nachgebildet hat. Die Qualität des Testsets hat somit direkten und massiven Einfluss auf die Aussagekraft der Evaluation.

## 4.2. Aufteilung der Datenmenge

Besteht nur eine Datenmenge, so ist es möglich diese in zwei disjunkte Teilmengen zu unterteilen, wobei eine als Trainingsset und eine als Testset verwendet wird. Dieses Verfahren wird als „Hold Out“ bezeichnet, da man dem Algorithmus einen Teil der Daten bei der Konstruktion eines Classifier-Modells vorenthält (im Englischen „to hold out“). Zumeist wird angegeben, wie viel Prozent des Datensatzes für das Trainingsset verwendet wird, der Rest fungiert anschließend als Testset. Man spricht daher hier von einem Percentage Split.

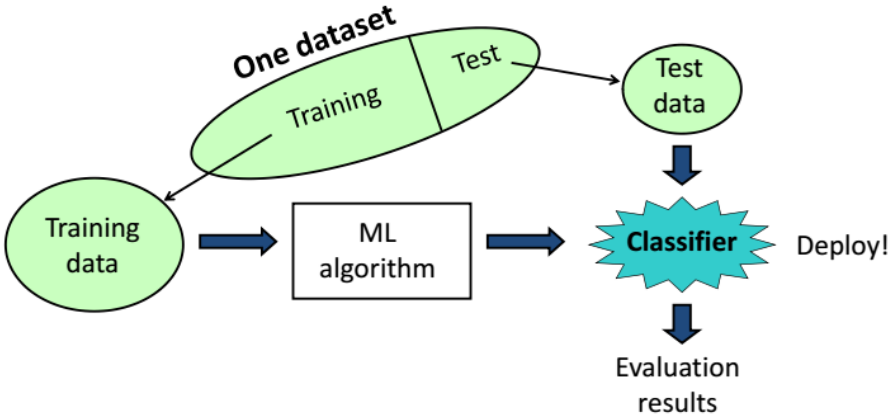


Abbildung 4.1.: Funktionsprinzip des Hold-Out-Verfahrens. „ML“ steht dabei für „Machine Learning“. Quelle: (Witten, 2013, S. 8)

Bei diesem Verfahren wird vor Nutzung der ersten N% der Gesamtmenge diese zufällig neu sortiert (im Englischen: „shuffling“). Dies geschieht, um sicherzustellen, dass die Reihenfolge der Datensätze zufällig ist und diese somit keinen Einfluss auf das Modell hat.
Bestünde eine Datenmenge beispielsweise aus Einträgen über Personen, wären nach Geschlecht geordnet und existierten gleich viele männliche wie weibliche Einträge[1], so würde ein undurchmischter 50%-Split dazu führen, dass nur ein Geschlecht im Trainingsset und nur das andere im Testset Beachtung finden würde, wodurch sowohl der Classifier als auch die Evaluation potentiell nutzlos wären.

Das Risiko bei Percentage Split besteht darin, dass es stark von der Zufälligkeit des durchmischten Datensatzes abhängt, weshalb in der Evaluation die Ergebnisse für mehrere Durchgänge mit unterschiedlichen Random Seeds betrachtet werden sollte und bei diesen ggf. Ausreißer gesondert behandelt werden sollten.

Für die Auswahl des Prozentsatzes gibt es keine eindeutigen Regeln. Diese hängt unter anderem von der Größe des gesamten Datensatzes und der anzunehmenden Komplexität des

[1] In dieser Arbeit wird von nur zwei existierenden Geschlechtern ausgegangen

Modells ab; ist der Datensatz bereits zu Beginn relativ klein, kann ein geringer Prozentsatz dazu führen, dass zu wenig Daten vorhanden sind, um ein adäquates Modell zu konstruieren. In der Regel bewegen sich die Werte des Percentage Splits zwischen 66% und 90%, womit zwischen einem Drittel und einem Zehntel als Testsets verwendet werden.

## 4.3. Kreuzvalidierung

Eine Technik, die auf der Hold-Out-Methode aufbaut ist die Kreuzvalidierung. Hierbei wird die Menge in $k$ möglichst gleich große, disjunkte Untermengen unterteilt. In $k$ Durchgängen wird nun jede Mengen einmal als Testset und $k-1$ mal als Teil des Trainingssets verwendet. Zumeist wird k in der Benennung der Kreuzvalidierung angegeben, so spricht man im Englischen bei einer Kreuzvalidierung mit 10 Teilmengen von einer „10-fold cross-validation“. Wird bei der Aufteilung der Gesamtmenge darauf geachtet, dass in jeder Untermenge die Verhältnisse der Klassen ungefähr denen in der Gesamtmenge entsprechen, so wird von „stratified k-fold cross validation“ gesprochen, wobei stratified hier soviel bedeutet wie „geschichtet“.

In einem $k+1$-ten Durchgang wird nun auf Basis der Gesamtmenge ein Classifier-Modell erstellt. Die Ergebnisse der $k$ vorherigen Evaluationen werden gemittelt und die errechneten Gesamtwerte, die somit hoffentlich robuster gegen einzelne Ausreißer sind, werden verwendet um die Performance des Gesamtmodells zu beschreiben.

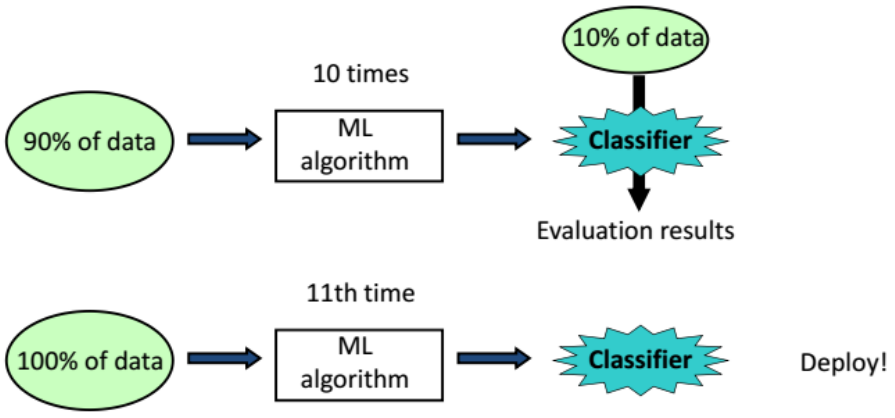


Abbildung 4.2.: Funktionsprinzip der Kreuzvalidierung. „ML“ steht dabei für „Machine Learning“ Quelle: (Witten, 2013, S. 25)

Im Allgemeinen gilt Kreuzvalidierung als besseres Verfahren im Vergleich zum einfachen Hold-Out, ist allerdings durch die insgesamt $k+1$ Modellerstellungen und $k$ Testdurchläufe

um etwa Faktor $k$ aufwändiger. Insbesondere bei sehr großen Datensätzen, bei denen das einfache Hold-Out-Verfahren in der Regel sehr gute Resultate liefert und eine Erhöhung des Rechenaufwands um Faktor $k$ gravierender wäre, wird auf Kreuzvalidierung verzichtet. Für die meisten Fälle ist Kreuzvalidierung allerdings die Regel, da es genauere Resultate mit geringerer Varianz liefert.

## 4.4. Confusion Matrix

Ein zentrales Werkzeug bei der Bewertung von Klassifikationsverfahren ist die Betrachtung der Confusion Matrix. In dieser wird angegeben, wie viele Datensätze mittels eines Classifiers korrekt und inkorrekt klassifiziert wurden. Hierzu wird eine quadratische Matrix erstellt, deren Ordnung der Anzahl der verschiedenen Möglichkeiten von Klassen entspricht. Die Spalten geben die Zuordnung zu einer Klasse an, die Zeilen die tatsächliche Klasse. Die Hauptdiagonale beschreibt somit die korrekt klassifizierten Datensätze, alle anderen Felder eine inkorrekte Zuordnung.

Verdeutlichen lässt sich diese Bewertung an einem Beispiel, bei welchem lediglich zwei Klassen existieren, bspw. „P“ für positiv und „N“ für negativ. Somit erstellen wir eine 2x2-Matrix, bei der jeweils die erste Spalte und Zeile positive Ergebnisse, klassifiziert und tatsächlich, darstellen, die jeweils zweite negative.

| | | **Predicted class** | |
|---|---|---|---|
| | | $P$ | $N$ |
| **Actual Class** | $P$ | True Positives (TP) | False Negatives (FN) |
| | $N$ | False Positives (FP) | True Negatives (TN) |

Abbildung 4.3.: Darstellung einer binären Confusion Matrix. Quelle: https://git.io/v5ne1

In dieser Matrix gibt das linke, obere Feld die Anzahl der Datensätze an, die korrekterweise vom Classifier als der Klasse „P“ zugehörig klassifiziert wurden. Bei diesen korrekt als po-

sitiv klassifizierten Datensätzen spricht man von „True Positives“. Das linke, untere Feld gibt an, wieviele Datensätze False Positives sind, also solche, die tatsächlich negativ sind, vom Classifier aber für positiv befunden wurden. Das rechte, untere Feld gibt die True Negatives an, das rechte, obere die False Negatives.

Die Confusion Matrix ist ein nützliches Instrument zur Visualisierung von korrekt und inkorrekt klassifizierten Datensätzen, da sie schnell verständlich darlegt, welche Fehler der Classifier macht, etwa ob er bspw. dazu tendiert bei einer binären Klassifizierung eher False Positives als False Negatives zu generieren oder bei einer höheren Klassenanzahl Datensätze einer bestimmter Klasse ungewöhnlich häufig einer anderen Klasse zuzuordnen.

Mit Hilfe der Confusion Matrix lässt sich eine Vielzahl potentiell nützlicher Werte des Classifiers bestimmen, insbesondere bei binären Matrizen. Einige wichtige hierbei sind:

- Die True Positive Rate (TPR), auch „sensitivity“ genannt, welche angibt welcher Anteil der tatsächlich positiven Datensätze auch positiv klassifiziert werden:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- Die True Negative Rate (TNR), auch „specificity“ genannt, welche angibt welcher Anteil der tatsächlich negativen Datensätze korrekterweise als negativ klassifiziert werden:

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

- Die False Positive Rate (FPR), manchmal „false alarm rate“ genannt, welche angibt welcher Anteil der tatsächlich negativen Datensätze fälschlicherweise als positiv klassifiziert werden:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}}$$

- Der Positive Prediction Value (PPV), auch „precision“ genannt, welcher angibt welcher Anteil der als positiv klassifizierten Datensätze korrekt positiv klassifiziert wurden:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

- Die „accuracy“ (ACC), welche angibt, welcher Teil der Daten korrekt klassifiziert wurde:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}}$$

## 4.5. ROC und AUC

Die meisten Classifier sind in der Lage ein Objekt nicht nur zu klassifizieren, sondern auch anzugeben, wie sicher sie sich sind, dass diese Klassifikation korrekt ist. Man spricht hierbei von der „prediction probability“. Dies ist möglich, da diese Classifier zunächst die Wahrscheinlichkeit berechnen, dass ein Objekt einer Klasse zuzuordnen ist und dann anhand eines Grenzwertes, Threshold genannt, entscheidet, ob dieses Objekt dieser Klasse zugeordnet werden soll, oder nicht - was bei zwei möglichen Klassen eine Zuordnung zur anderen Klasse zur Folge hat.

Erstellen wir in einem Diagramm zwei Histogramme, eines für die tatsächlich negativen und eines für die tatsächlich positiven Objekte, wobei die X-Achse die Wahrscheinlichkeit des Datensatz als positiv klassifiziert zu werden darstellt, so erhalten wir beispielsweise folgende Grafik:

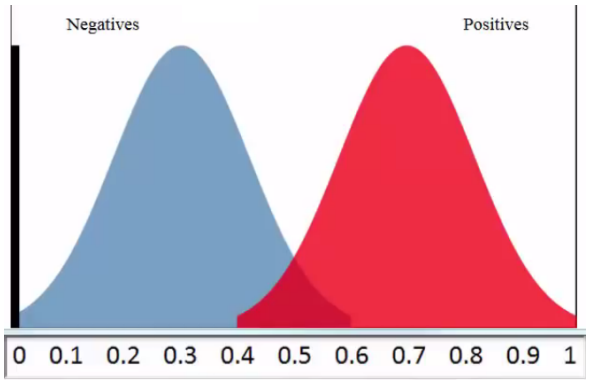


Abbildung 4.4.: Beispiel eines Classifiers. Blau: negative Objekte. Rot: positive Objekte. Quelle: https://youtu.be/OAl6eAyP-yo?t=1 (Abfrage 21.8.2017)

In diesem Beispiel handelt es sich um zwei Normalverteilungen, die blaue repräsentiert die tatsächlich negativen Datensätze, die rote die tatsächlich positiven Datensätze. Im lilafarbenen Bereich überschneiden sich beide Kurven (diese werden hier aber nicht aufsummiert!), was bedeutet, dass hier tatsächlich negative Objekte existieren, die eine höhere Wahrscheinlichkeit besitzen als positiv klassifiziert zu werden als manche tatsächlich positiven Objekte.

Definieren wir den Threshold als 0.6, so werden alle im Diagramm rechts davon stehenden Objekte als positiv klassifiziert, alle links davon stehenden als negativ, wodurch diejenigen tatsächlich positiven Objekte mit einer Wahrscheinlichkeit von unter 0.6 fälschlicherweise als negativ klassifiziert werden, also False Negatives werden.
Definieren wir den Threshold als 0.5, so werden zwar weniger False Negatives klassifiziert,

dafür aber alle über 0.5 liegenden tatsächlich negativen Werte als False Positives.
Für jeden Threshold lassen sich TPR und FPR, also jeweils die Anteile der korrekt bzw. inkorrekt als positiv klassifizierten Datensätze bestimmen. Plotten wir für eine hinreichend große Anzahl von Thresholds die FPR entlang der X- und TPR entlang der Y-Achse, so entsteht eine Kurve die immer bei 0/0 beginnt und bei 1/1 endet.

Ein idealer Classifier, also einer der immer korrekt klassifiziert, wäre derjenige, dessen Kurve den Punkt 0/1 erreicht, also für den es möglich ist einen Threshold zu wählen, bei dem alle positiven Objekte korrekterweise als positiv und alle negativen als negativ klassifiziert würden. Ein denkbar schlechter Classifier, der im gleichen Maße True und False Positives klassifiziert - denkbar wäre eine zufällige Zuordnung mit jeweils 50%iger Wahrscheinlichkeit - würde einer 45-Grad-Gerade entsprechen.

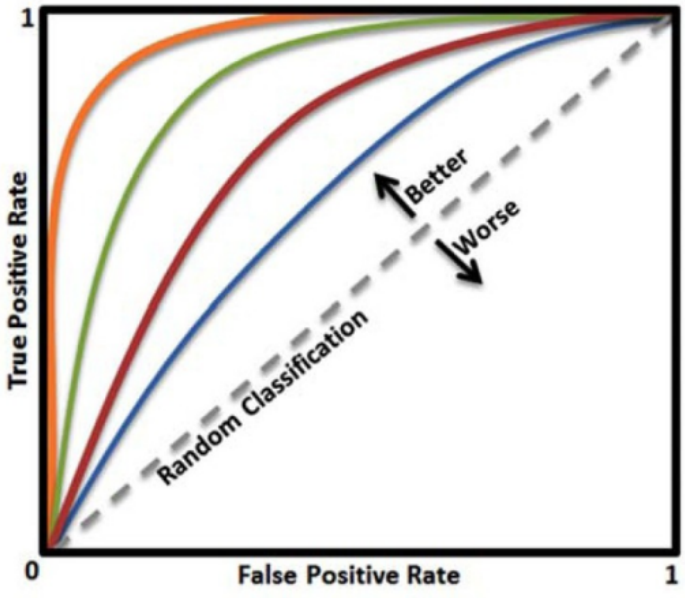


Abbildung 4.5.: Verschieden ROC-Kurven. Quelle: https://openi.nlm.nih.gov/imgs/512/261/3861891/PMC3861891_CG-14-397_F10.png (Abfrage 21.8.2017)

Diese erstellten Kurven nennt man Receiver Operating Characteristic Curve, kurz ROC Curve. Aus dieser lässt sich eine Kennzahl zum vereinfachten Vergleich mehrerer Kurven bestimmen. Ein Classifier ist dann besser, wenn sich seine ROC Curve dem Punkt 0/1 möglichst stark nähert, was zur Konsequenz hat, dass die Fläche unterhalb der Kurve größer wird. Daher wird zum Vergleich häufig AUC, was für „Area Under the Curve“ steht - also ganz

wörtlich die Fläche unterhalb der Kurve - verwendet, deren Wert sich theoretisch zwischen 0 und 1, praktisch aber zwischen 0.5 und 1 bewegt.[2]

Auch lassen sich ROC Curves und somit AUC-Werte für das Verhältnis True Negatives und False Negatives erstellen. Somit bietet eine binäre Klassifikation zwei AUC-Werte, anhand derer die Performance des Classifiers gemessen werden kann. Zumeist werden diese Werte (gewichtet) gemittelt.

[2] Ein Wert zwischen 0.5 und 0 kann durch logische Negation zu einem Wert zwischen 0.5 und 1 umgewandelt werden. Relevant ist daher der möglichst große Betrag des Abstandes der AUC zu 0.5

# 5. Das Datenmodell

## 5.1. Problem fehlender verfügbarer Daten

Wie dargelegt, setzen Data Mining Methoden die Existenz eines gewissen Datenbestandes auf Basis dessen (hoffentlich) sinnvolle Aussagen getroffen werden können. Im konkreten Fall liegen uns diese allerdings nicht vor, da öffentliche Datensätze über das Auffinden von Parkplätzen nicht existieren. Zwar besteht durch die Nutzung von Navigationssystemen - sowohl in Form von Stand Alone Devices als auch in Form von Anwendungen für Mobiltelefone - und die wachsende Anzahl an Smart Cars das Potential, solche Daten zu erheben, allerdings sind diese (noch) nicht (öffentlich) verfügbar.
Ein Grund hierbei könnte sein, dass die Identifikation des Prozesses der Parkplatzsuche, also das Feststellen, ob ein Fahrer zur Zeit nach einem Parkplatz sucht anstatt die Straße nur zu durchfahren, ein schwieriger Vorgang ist. Zwar ist dieser vermutlich möglich, bedeutet aber einen Aufwand, der nur dann gerechtfertigt ist, wenn dieses Datum wirtschaftlich rentabel genutzt werden kann. Aufgrund der fehlenden Verfügbarkeit echter Datensätze beschäftigt sich diese Arbeit mit der Evaluation von Algorithmen an selbst erstellten Datensätzen. Zur Generierung dieser wurde ein Programm geschrieben, das nach einer im Folgenden weiter erörterten internen Logik die benötigten Datensätze erzeugen kann.

## 5.2. Trennung zwischen Datenmodell und Evaluation

Die Phasen der Erstellung des Programms zur Generierung von Datensätzen und der Evaluation wurden strikt getrennt. Bei der Evaluation soll eine möglichst objektive Anwendung und Auswertung stattfinden, auf Basis derer verschiedene Verfahren miteinander verglichen werden können. Diese Objektivität setzt voraus, dass Zwischenergebnisse der Evaluation nicht die dem Modell zugrundeliegende Logik beeinflussen. Sollten die (Weiter-) Entwicklung des Modells und die Evaluation parallel zueinander geschehen, existiert die Gefahr, das Modell den Zwischenergebnissen der Evaluation - unter Umständen auch nur unbewusst - anzupassen.

## 5.3. Vorüberlegungen zum Datenmodell

Es ist anzunehmen, dass eine enorme Vielzahl von Faktoren das Auffinden eines Parkplatzes in einem bestimmten Gebiet beeinflusst. Basierend auf Beobachtungen und politischen Infrastrukturmaßnahmen in Hamburg ist anzunehmen, dass folgende Faktoren auf das Auffinden einen nennenswerten Einfluss haben:

- Wetter
  - Windstärke
  - Temperatur
  - Niederschlag
- Zeit
  - Tageszeit
  - Wochentag
  - Jahreszeit
- Lage des Platzes
  - Wohn- oder Arbeitsgebiet
  - Umliegende Parkhäuser
  - Anzahl allgemein verfügbarer Parkplätze
- Besondere Ereignisse
  - Sperrung des ÖPNV
  - Straßensperrungen z.B. durch Bauarbeiten
  - Temporäre Parkverbote
- Infrastruktur
  - Existenz von ÖPNV
  - Anbindung an das Straßennetz
  - Ausbau der Radwege im Umkreis

Obwohl diese Liste mit Sicherheit unvollständig ist, erlaubt sie, gewisse Aussagen über den Aufbau unseres Modells zu treffen:

1. Der Einfluss der Faktoren ist nicht unkorreliert, sondern hängt maßgeblich von Kombinationen mehrer Faktoren ab. So ist bspw. das Auffinden eines Parkplatzes zu üblichen Geschäftszeiten in Wohngebieten sehr wahrscheinlich, in Arbeitsgebieten dagegen sehr unwahrscheinlich. Der Faktor Uhrzeit hat also abhängig vom Gebiet komplett gegenteilige Auswirkungen und lässt sich nicht alleine betrachten.
2. Die Auswirkung der Faktoren sind nicht allgemeingültig, sondern bestenfalls auf ein Gebiet beschränkt. So führt bspw. ein Temperaturwert unter 0 Grad Celsius in einigen Teilen der Erde zu einem kompletten Verkehrsstillstand, während es in anderen Teilen ein gewöhnliches Vorkommnis ist, das wenig Einfluss auf den Verkehr hat.
   Selbst in benachbarten Straßen verhält sich das Finden von Parkplätzen aufgrund einer praktisch unmöglich erfassbaren Vielzahl weiterer Faktoren in bestimmten Maßen unterschiedlich.

## 5.4. Umsetzung

### 5.4.1. Konzipierung

Für das Modell wurden folgende Faktoren gewählt:

- Lage des Orts: Wohn- oder Arbeitsgebiet
- Straße: Eindeutige Identifikation der Straße
- Zeit: Wochentag, Tageszeit
- Wetter: Temperatur, Niederschlag
- Besondere Ereignisse: Sperrung des ÖPNV

Die Möglichkeit eines Modells, bei dem jeder Faktor einzelnd die Gesamtwahrscheinlichkeit erhöht oder senkt ist, aufgrund der dargelegten, teils gegenteiligen Effekte einzelner Faktoren in Abhängigkeit anderer Faktoren nicht möglich. Als praktikabelste Lösung wurde hier ein Logikbaum verwendet. Dieser wurde an manchen Stellen zurechtgestutzt - das zuvor beschriebene „Pruning“, - um Äste ohne nennenswerten Informationsgewinn der Einfachheit halber zu vermeiden. Somit wurde für jede Kombination oder Gruppe von Kombinationen explizit Wahrscheinlichkeiten für das Auffinden vorgegeben.

Die Auswirkung der einzelnen Faktoren entspringt einer definierten Logik, die auf folgenden Definitionen über das gesamte modellierte Gebiet basiert:

- Es existiert eine klare Abgrenzung zwischen Wohn- und Arbeitsgebiet. Das Groß der Menschen sind bspw. Industriearbeiter im nahegelegenen Industriegebiet.

- Das Groß der Menschen arbeitet Montag-Freitag im Zeitraum von 9 bis 17 Uhr.
- Einige Menschen arbeiten auch Samstags.
- Sonntag ist in der fiktiven Gemeinde Ruhetag und es wird wenig Auto gefahren.
- Es existieren annehmbare Fahrradwege und die Gemeinde fährt teilweise Rad.
- Es existiert ein ÖPNV, der aber nicht zu 100% verlässlich ist und manchmal ausfällt.
- Die Stadt verfügt über einen nahegelegenen See, der an heißen Tagen besucht wird.
- Nachts ist allgemein in der Stadt wenig los, es ist ruhig und kaum jemand fährt mit dem Auto.
- Der Einfluss von Pendlern wird nicht gesondert betrachtet.
- Es gibt zahlungspflichtige Parkflächen und abgelegene Reserveparkflächen, die Prozentzahlen beziehen sich allerdings lediglich auf die kostenfreien Platze, die schnell zu finden sind.
- Feiertage werden nicht gesondert berücksichtig.
- Die Arbeitszeiten sind an allen Tagen für alle Personen gleich. Kurze Freitage existieren nicht.

Das Wohn- und Arbeitsgebiet in unserer Modellstadt wird durch jeweils mehrere Straßen repräsentiert. Diese verhalten sich allgemein ähnlich, allerdings verfügt jede Straße über gewisse definierte Eigenheiten in Bezug auf die Wahrscheinlichkeit einen Parkplatz zu finden:

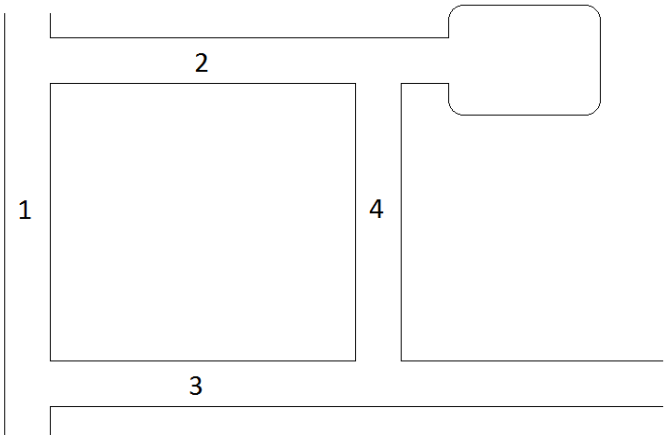


Abbildung 5.1.: Skizze der Straßen des Wohngebiets

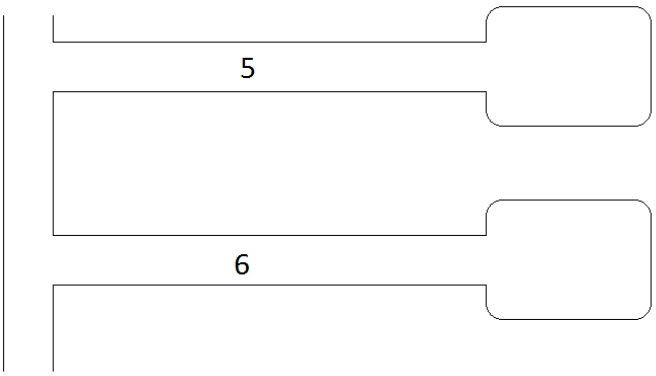


Abbildung 5.2.: Skizze der Straßen des Arbeitsgebiets

Die Wohnstraßen folgen somit weiterhin der Logik des Wohngebietes, haben aber jeweils individuelle Varianzen:

- An der Kreuzung von 1 und 3 befindet sich die Bushaltestelle. Menschen, die in diesen Straßen wohnen, nehmen häufiger den Bus und sind somit stärker von ÖPNV-Sperrungen beeinflusst.
- In Straße 3 leben mehr am Samstag arbeitende Menschen, somit sind dort am Samstag zur Arbeitszeit mehr Parkplätze verfügbar.
- Straße 4 grenzt nur auf einer Seite an Häuser, auf der anderen an einen kleinen Park. Daher sind hier mehr „Parkplätze pro Kopf“ zu finden.
- Straße 1 ist eine Hauptstraße, dort sind Parkplätze immer schwerer zu finden.
- Straße 2 hat eine Fahrrad-Mietstation, daher fahren dort mehr Menschen mit dem Fahrrad, wodurch sie regen- und temeraturanfälliger werden.

Die Arbeitsstraßen folgen weiterhin der Logik des Arbeitsgebietes. Die Varianz:

- Straße 5 ist weiter vom Werk entfernt, somit lässt sich eher ein Parkplatz zur Arbeitszeit finden als in Straße 6.

### 5.4.2. Programmierung

Die programmtechnische Umsetzung des Modells zur Datengenerierung erfolgte mit zwei Java-Klassen. Da keine Notwendigkeit zur Erstellung mehrerer Objekte bestand, wurde die Umsetzung vor allem in prozeduraler Manier vorgenommen. Die Quellcodes sind im Anhang zu finden.

### Model.java

Model.java enthält hierbei die Logik unseres Modells. Ziel war es, eine Klasse mit nur einer public-Funktion `getProbability(...)` zu erstellen, die die oben definierten Parameter entgegennimmt und eine Wahrscheinlichkeit für das Auffinden eines Parkplatzes zurückgibt. Für die Umsetzung wurden folgende Schnittstellendefinitionen getroffen:

Inputs:

- String area - dieser soll entweder „living“ für ein Wohngebiet oder „working“ betragen.
- String streetID - dieser soll die Bennungen 1 bis 6 der Straßen [s. oben] im String-Format enthalten.
- String weekday - dieser soll den Wochentag im englischen Kurzformat „Mon“, „Tue“, ... , „Sun“ enthalten.
- float time - dieser soll die Uhrzeit enthalten. Statt eines Datumswertes wird hierbei ein float-Wert verwendet, so entspricht 13:30 dem Wert 13,5 (bzw. 13.5).
- float temperature - dieser soll den Temperaturwert in Grad Celsius enthalten
- boolean precipitation - dieser soll die Information enthalten, ob Niederschlag existiert.
- boolean specialOccurrence - dieser soll die Information über das besondere Ereignis einer Sperrung des ÖPNV enthalten.

Rückgabe:

1. Die Funktion gibt eine Fließkommazahl im Bereich von 0 bis 100 zurück. Diese entspricht der prozentualen Wahrscheinlichkeit für das Auffinden eines Parkplatzes. Wird die Zahl -1 zurückgegeben, gab es einen Fehler.

Die Funktion `getProbability(...)` bestimmt den Rückgabewert für einen bestimmten Eingangsvektor in zwei Schritten:

Zunächst wird die Wahrscheinlichkeit für das Finden eines Parkplatzes im definierten Gebiet, also Wohn- oder Arbeitsgebiet, mithilfe der Funktion `getProbabilityByArea(...)` bestimmt. Diese private-Funktion nimmt die gleichen Eingangswerte wie `getProbability(...)` entgegen mit Ausnahme von `streetID`. Diese Funktion ruft zunächst drei private Hilfsfunktionen auf, die jeweils den Wert von weekDay, time und temperature kategorisieren. Die Variable des Wochentages wird entweder zu einem regulären Arbeitstag (`Workday`), einem Samstag (`Saturday`) oder einem Sonntag (`Sunday`) klassifiziert, die Uhrzeit anhand definierter numerischer Grenzen in Arbeitszeit (`WorkingHours`), Abend (`Evening`) und Nacht (`Night`) kategorisiert, und die Temperaturvariable in die Kategorien kalt (`cold`), mild (`mild`) oder warm (`warm`) eingeordnet. Mithilfe der drei Kategorien und der Parameter

`area`, `precipitation` und `specialOccurrence` wird nun die Wahrscheinlichkeit in einem durch verschachtelte if-elseif-Bedingungen implementierten Logikbaum ermittelt. Hierbei handelt es sich um die allgemeine Wahrscheinlichkeit in dem besagten Gebiet.

```
if(area == "living"){
    if(weekdayCategory == "Workday"){
        if(timeCategory == "WorkingHours"){
            if(temperatureCategory == "cold"){
                /* It's a working day during working hours and it is cold
                 *People drive to work by car rather than bike. */
                if(precipitation){
                    if(specialOccurrence){
                        return 95;
                    }
                    else{
                        return 85;
                    }
...
```

Listing 5.1: Auszug der Funktion `getProbabilityByArea(...)` aus Model.java

Im zweiten Schritt werden die einzelnen Faktoren der Straßen [s. oben] berücksichtigt und der zuvor ermittelte Wahrscheinlichkeitswert abhängig von streetID und den weiteren Faktoren noch einmal verändert.

```
switch(streetID){
    case "1": //street 1
        /* The only bus station is on the crossing of streets 1 and 3.
           People living in those streets take the bus more frequently.
        /* Thus they are more affected by a specialOccurrence */
        if(specialOccurrence){
            float probabilityWithoutSpecialOccurrence =
                getProbabilityByArea(area, weekday, time, temperature,
                precipitation, false);
            float deviation = probability -
                probabilityWithoutSpecialOccurrence;
            probability = (float) (probabilityWithoutSpecialOccurrence +
                deviation * 1.3);
        }

        //street 1 is a major road, therefore it is always more difficult
            to find a parking space
        probability -= 5;
        break;
```

```
14  ...
```

Listing 5.2: Auszug der Funktion `getProbability(...)` aus Model.java

Abschließend wird der nun gerechnete Gesamtwert der Wahrscheinlichkeit als Funktionswert zurückgegeben.

#### Generator.java

Die Klasse Generator.java ist für die Erstellung der Dateien abhängig von gesetzten Parametern zuständig und enthält auch die main-Funktion. Zunächst wird eine neue .arff-Datei erstellt und die notwendigen Headerinformationen gesetzt. Auf das Attribute-Relation File Format (.arff) wird in der Sektion Attribute-Relationship File Format auf Seite 44 tiefer eingegangen. Nun werden in mehreren Durchgängen Datensätze generiert und in die Datei geschrieben. Die Anzahl ist dabei durch die statische Variable `iterations` definiert. In jedem Durchgang wird für jeden der oben definierten Parameter ein Zufallswert aus einem festgelegten Bereich oder einer festgelegten Menge bestimmt. Mit diesen Zufallswerten wird die Funktion `getProbability(...)` eines Model-Objekts aufgerufen, welche wie oben dargelegt eine Wahrscheinlichkeit für das Auffinden eines Parkplatzes zurückgibt. Auf Basis dieser Wahrscheinlichkeit wird nun für den konkreten Datensatz ermittelt, ob ein Parkplatz gefunden wurde oder nicht. Abschließend werden sämtliche Parameter sowie die Information, ob ein Parkplatz gefunden wurde oder nicht, in die Datei geschrieben.

In einem statischen Array `censorParameters[]` kann definiert werden, ob und falls ja mit welcher prozentualen Wahrscheinlichkeit ein bestimmter Parameter im Datensatz durch ein „?“ ersetzt wird. Ein Fragezeichen entspricht einer Unbekannten und entspräche in unserem Anwendungsfall der Aussage, dass ein bestimmter Wert, z.B. der aktuelle Temperaturwert, nicht ermittelt werden konnte. Diese Funktion ist für die Untersuchung der Auswirkungen von Datenrauschen auf die Aussagekraft von Algorithmen implementiert worden.

### 5.4.3. Attribute-Relationship File Format

Die Evaluation der Algorithmen in dieser Arbeit geschieht mithilfe des Programms „Weka“, auf welches im Kapitel Weka ab Seite 46 detailierter eingegangen wird. Für das Datenmodell maßgeblich ist das Format, in welchem Weka Daten entgegennimmt. Obwohl Weka in der Lage ist, CSV-Dateien mit manueller Unterstützung einzulesen, ist die gängiste Methode zur Dateneingabe das eigens für Weka entwickelte Attribute-Relationship File Format, bekannt unter dem Akronym ARFF oder seiner zugehörigen Dateiendung .arff. Hierbei handelt es sich um auf ASCII basierendes Format, welches in zwei Teile unterteilt ist.

Zunächst muss ein Header angegeben werden, in dem die Struktur der Daten beschrieben wird. Hierzu wird mit `@relation relation-name` eine neue Beziehung definiert, welche durch Attribute beschrieben wird, die mit `@attribute attribute-name datatype` deklariert werden. Hierbei besteht neben Datentypen wie `string` und `numeric` auch die Möglichkeit eine eigene Enumeration in geschweiften Klammern zu definieren. Weka geht davon aus, dass das letzte angebene Attribut die Klassifizerung darstellt. Anschließend wird mit `@data` der Block begonnen, der die Datensätze enthält. Hierbei werden die Attribute in der zuvor definierten Reihenfolge durch Komma getrennt, zeilenweise angegeben.

Im vorliegenden Programm hat dies zur Konsequenz, dass in Generator.java die neu erstelle Datei zunächst mit den Headerdaten gefüllt wird und erst anschließend die Datensätze zeilenweise geschrieben werden.

```
@RELATION parking
@ATTRIBUTE street               {"1", "2", "3", "4", "5", "6"}
@ATTRIBUTE weekday              {Mon, Tue, Wed, Thu, Fri, Sat, Sun}
@ATTRIBUTE time                 NUMERIC
@ATTRIBUTE temperature          NUMERIC
@ATTRIBUTE precipitation        {true, false}
@ATTRIBUTE specialOccurrence    {true, false}
@ATTRIBUTE result               {"Y", "N"}
@DATA
"6",Mon,16.966707,2.8191743,false,false,"N"
"2",Thu,12.235365,6.8040104,false,false,"Y"
"2",Fri,22.831238,28.955322,true,false,"N"
"5",Tue,19.425102,23.760174,false,false,"Y"
"5",Sat,15.696143,20.664135,true,false,"N"
"2",Sun,12.078556,31.895004,true,false,"N"
"4",Fri,3.8297575,9.613866,true,false,"N"
"1",Fri,13.197587,10.798707,true,false,"N"
"1",Thu,4.6168146,11.047333,false,false,"N"
"5",Wed,11.613608,15.776719,false,false,"Y"
"5",Fri,23.69413,16.909882,false,true,"Y"
"3",Sat,2.1340926,10.750453,true,false,"Y"
"6",Sun,0.2213207,27.299204,false,false,"Y"
"6",Mon,13.681532,16.117867,false,false,"N"
```

Listing 5.3: Auszug aus einer vom Programm generierten .arff-Datei

# 6. Weka

Zur Evaluation der Algorithmen in dieser Arbeit wird das Waikato Environment for Knowledge Analysis, kurz „Weka“ verwendet. Hierbei handelt es sich um ein an der neuseeländischen University of Waikato entwickeltes und in Java programmiertes Open-Source Softwaretool, welches eine Reihe von Data-Mining-Algorithmen implementiert sowie hierzu Visualisierungswerkzeuge und eine GUI bereitstellt.

Abbildung 6.1.: Das Thumbnail von Weka zeigt den in Neuseeland beheimateten Vogel Kiwi

Das Herzstück von Weka und zugleich der in dieser Arbeit verwendete Teil ist der Weka Explorer, welcher sich in folgende Panels einteilt:

- Preprocess: bietet die Möglichkeit, Daten aus verschiedenen Quellen wie CSV-Dateien oder Datenbanken einzulesen und vor der Analyse aufzubereiten, etwa durch Löschung von Attributen oder anderen Filterungen.
- Classify: bietet eine Vielzahl von Klassifikationsalgorithmen sowie Evaluationsmöglichkeiten dieser an. Dieses Panel wird Schwerpunkt in der Analyse sein.
- Cluster: bietet eine Vielzahl an Clusteringverfahren an, unter anderem eine Ausprägung des zuvor behandelten k-Means-Algorithmus.
- Associate: bietet die Möglichkeit Assoziationsverfahren auf den geladenen Datensatz anzuwenden. Standardmäßig sind unter Anderem die zuvor diskutierten Apriori- und FP-Growth-Algorithmen implementiert.

- Select attributes: bietet Verfahren an, mit denen Aussagen getroffen werden können, welche Attribute die einflussstärksten sind.
- Visualize: bietet eine Visualisierung des Datensatzes mittels Streudiagramm-Matrix an, in der jede Kombination von Attributen als Streudiagramm bereitgestellt wird.

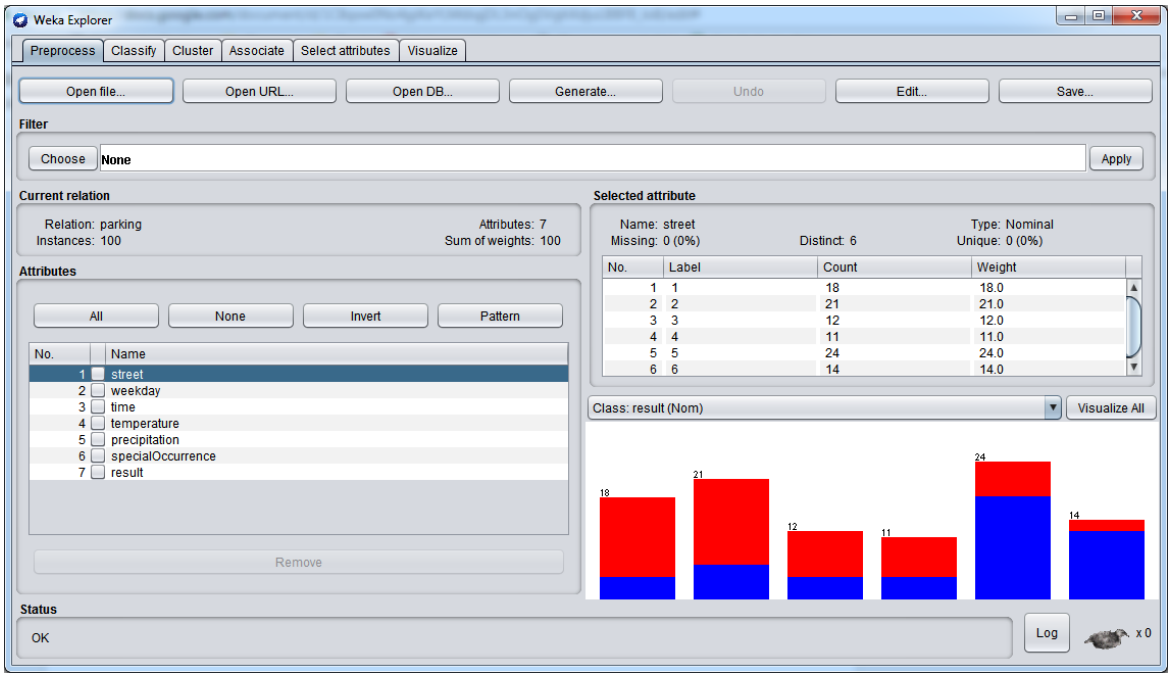


Abbildung 6.2.: Screenshot des Weka Explorers mit geladenem Datensatz.

Im Classify-Panel besteht neben der Auswahl des zu verwendenden Algorithmus samt Feineinstellungen zu diesen auch die Möglichkeit zwischen verschiedenen Testverfahren, die in der Sektion Evaluation von Klassifikationsverfahren ab Seite 29 erläutert wurden, auszuwählen, sowie die zu evaluierenden Parameter einzustellen und sogar den gebildeten Classifier als Java-Quellcode auszugeben.

Per Default werden folgende bekannte Maße ausgegeben:

- Korrekt und inkorrekt klassifizierte Objekte, jeweils in absoluten und prozentualen Werten
- True Positives Rate, der Anteil der korrekt klassifizierten Objekte
- False Positives Rate, der Anteil der inkorrekt klassifizierten Objekte
- Precision, welche hier angibt, welcher Anteil der Objekte einer Klasse dieser korrekt zugeordnet wurden

- ROC Area, wobei es sich um die zuvor diskutierte Area Under the Curve (AUC) handelt
- Confusion Matrix, die angibt wie viele Datensätze wie klassifiziert wurden

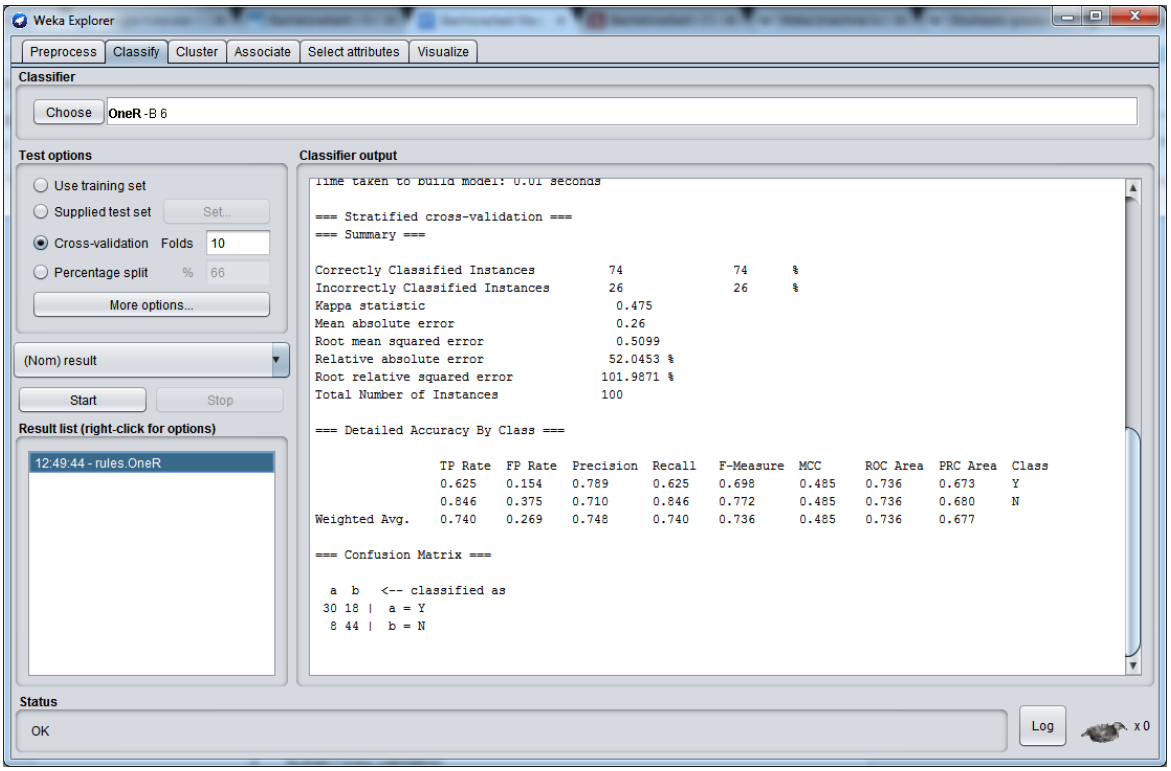


Abbildung 6.3.: Screenshot des Classify-Panels mit angewendetem OneR-Algorithmus

Darüber hinaus ist es möglich, für jedes Objekt aus dem Testset die Vorhersagesicherheit, welche in der Sektion ROC und AUC auf Seite 34 diskutiert wurde, auszugeben.

# 7. Algorithmen-Evaluation

Untersucht wird die Performance von insgesamt vier Algorithmen. Zum einen wird repräsentativ für Entscheidungsbaumverfahren der J48-Tree verwendet. Hierbei handelt es sich um eine für Weka erstelle Java-Implementierung des C4.5-Algorithmus, welcher als bester Data MIning Algorithmus bezeichnet wird[1]. Zum anderen wird der zuvor beschriebene Naive-Bayes-Algorithmus verwendet. Darüberhinaus Verwendung findet der Multilayer-Perceptron-Algorithmus, bei welchem es sich um einen Algorithmus der Klasse künstlicher neuronaler Netze (ANN) handelt. Letzlich wird stets der ZeroR-Algorithmus verwendet. Hierbei handelt es sich um einen Benchmarking-Algorithmus, welcher lediglich zur relativen Bewertung der Performance der anderen Algorithmen benutzt wird. Der ZeroR-Algorithmus ermittelt aus den vorhandenen Daten die häufigste Klasse und klassifiziert alle neuen Objekte in diese Klasse ein.

Untersucht werden im Lichte der hypothetischen Endanwendung folgende Kriterien:

- **Accuracy (Acc)**: Hierbei handelt es sich um das Maß der Genauigkeit des Classifiers, also welcher Anteil der Daten korrekt klassifiziert wird. Für die Anwendung ist dieses Maß insofern von Bedeutung, als dass es ausschlaggebend für die Richtigkeit der gegebenen Vorhersagen ist.
- **AUC**: Die Area Under the Curve ist ein Qualitätsmaß des Classifiers, welches hilft zu bestimmen, wie klar dieser die Klassen unterscheidet. Ein hohes AUC-Maß gibt Hinweise, wie „zuversichtlich“ der Algorithmus bzgl. seiner Klassifikationen ist.
- **Laufzeit (t)**: Die benötigte Zeit zur Erstellung des Classifiers ist in Anbetracht der möglichen Dimensionen einer Endanwendung relevant. Die Auswahl eines (zeitlich) effizienten Algorithmus kann die Notwendigkeit für teure Hardware oder zugekaufte Rechenzeit potentiell dramatisch reduzieren.
- **FPR**: Die False Positives Rate gibt hier an, wie häufig ein Parkplatz prognostiziert wurde, tatsächlich aber nicht existiert. Dies ist in Anbetracht der Endanwendung insofern wichtig, als dass dies ein Maß ist, das stark mit der Unzufriedenheit des Kunden korrelieren wird. Der Endbenutzer würde es hinnehmen, einen Parkplatz zu finden, der ihm nicht vorhergesagt wurde, sicherlich aber weniger erfreut sein, häufig Parkplätz

[1] http://www.cs.umd.edu/ samir/498/10Algorithms-08.pdf

vorhergesagt zu bekommen, die dann doch nicht existieren. Dieses Maß ist immer nur in Relation mit der Accuracy zu betrachten, da eine geringe FPR bei gleichbleibender Accuracy lediglich eine Verlagerung der Fehlerquelle in die FNR bedeutet.

- **Certainty**: hierbei handelt es sich um einen Durchschnittswert der Sicherheit, welche in der Sektion ROC und AUC auf Seite 34 beschrieben wurde. Hierzu wird die berechnete Sicherheit aller Testobjekte gemittelt. Die Endanwendung gibt dem Benutzer prozentuale Wahrscheinlichkeiten über das Auffinden von Parkplätzen in der Umgebung. Die Certainty ist hierbei der durchschnittlich angezeigte Wert. Ähnlich wie die FPR ist dieses Kriterium wichtig für die Kundenpsychologie. Wird stets eine 95%ige Sicherheit angegeben, ist das Nicht-Auffinden eines Parkplatzes für den Kunden ärgerlicher als bei 75%. Liegt der Wert stets bei 50%, so würde sich dem Benutzer die Frage stellen, ob eine Entscheidung per Münzwurf nicht einfacher als die Benutzung der Anwendung wäre.

## 7.1. Untersuchungen mit rauschfreien Datensätzen

Für diese Untersuchung wurden Datensätze generiert, bei denen alle Faktoren bis auf die Art des Gebiets (Wohn/Arbeit) bekannt sind, sich also keine „?“ in den Datensätzen finden lassen. In der Praxis entspräche dies der Annahme vollständiger Datensätze ohne fehlende Parameter wie etwa Temperaturwerten ohne vorherige Clusterung. Es wurden 5 Datensätze der Größen $10^n$ für $n = 2...6$ generiert.
Sofern nicht anders angegeben, finden die Untersuchungen für $n = 2...5$ mit einer 10-fold Cross-Validation statt, für $n = 6$ mit einem 90%-Split. Die Classifier-Outputs aus Weka sind im Anhang zu finden.

### 7.1.1. ZeroR

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty(%) |
|---|---|---|---|---|---|
| 2 | 52.00 | 0.468 | 0.00 | 100 | 52.0 |
| 3 | 54.60 | 0.495 | 0.00 | 100 | 54.6 |
| 4 | 54.46 | 0.500 | 0.01 | 100 | 54.5 |
| 5 | 54.69 | 0.500 | 0.01 | 100 | 54.7 |
| 6 | 54.48 | 0.500 | 0.08 | 100 | 54.4 |

Tabelle 7.1.: Ergebnisse des ZeroR ohne Rauschen

An den Daten ist die Funktion des ZeroR klar zu erkennen. Unsere generierten Datensätze verfügen über etwas mehr als 50% positive Datensätze, weshalb alle Objekte als positiv klassifiziert werden. Die leichten Schwankungen sind als zufällige Abweichungen zu betrachten. Insbesondere der kleinste Datensatz mit 100 Objekten weicht etwas stärker ab, was diese Vermutung bestärkt. Da es zum Erzeugen des Classifiers lediglich notwendig ist, die Anzahl der Objekte der einzelnen Klassen aufzusummieren und zu vergleichen, ist von einem linearen Wachstum der benötigten Laufzeit auszugehen, was durch die scheinbar konstante Laufzeit von nahezu 0 Sekunden bestätigt wird.

### 7.1.2. J48

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) | Blätter | Knoten |
|---|---|---|---|---|---|---|---|
| 2 | 63.00 | 0.599 | 0.03 | 47.9 | 74.1 | 13 | 16 |
| 3 | 69.30 | 0.723 | 0.11 | 28.0 | 83.0 | 63 | 106 |
| 4 | 81.48 | 0.871 | 0.41 | 17.9 | 85.6 | 238 | 391 |
| 5 | 82.85 | 0.898 | 3.15 | 15.8 | 83.6 | 240 | 375 |
| 6 | 83.20 | 0.901 | 56.12 | 15.7 | 83.4 | 226 | 332 |

Tabelle 7.2.: Ergebnisse des J48 ohne Rauschen

Gemessen an der Performance des ZeroR zeigt der J48 hier respektable Ergebnisse. Es ist zu erkennen, dass mit zunehmender Menge an verfügbaren Objekten für das Erstellen eines Classifiers zunehmend bessere Bäume erstellt werden können. Die Bäume werden zunehmend komplexer und benötigen daher steigende Laufzeiten zum Erstellen, liefern aber auch deutlich bessere Resultate. Der Anstieg der Laufzeit erscheint vergleichsweise gering, womit sich der J48 potentiell für größere Datenmengen eignen könnte.

Interessant zu beobachten ist, dass für $n = 6$ die Anzahl der Blätter und Knoten wieder leicht sinkt. Es ist zu vermuten, dass die höhere Anzahl an Objekten keine weitere Ausdifferenzierung ermöglicht, aber die Möglichkeiten des Prunings verbessert, wodurch der Baum insgesamt kleiner wird, ohne dadurch schlechtere Resultate zu liefern.

```
1 street = 1
  |   time <= 8.987825
3 |   |   weekday = Mon: N (74.0/3.0)
  |   |   weekday = Tue: N (91.0/6.0)
5 |   |   weekday = Wed: N (93.0/2.0)
  |   |   weekday = Thu: N (86.0/2.0)
7 |   |   weekday = Fri: N (66.0/2.0)
  |   |   weekday = Sat: Y (75.0/19.0)
```

```
9 |   |   weekday = Sun: N (77.0/23.0)
  |   time > 8.987825
11|   |   time <= 16.890295
```

Listing 7.1: Auszug des J48-Classifier-Baumes für 10000 Objekte

Bereits mit $10^4$ Objekten, also einer relativ geringen Anzahl, lassen sich für die Praxis verwendbare Ergebnisse erzielen. Betrachten wir den erstellten Baum für $n = 4$, so erkennen wir, dass die im Datenmodell definierten Grenzwerte für Zeit und Temperatur relativ gut erfasst werden, so teilt dieser beispielsweise die zu Straße 1 gehörigen Datensätze im zweiten Einteilungsschritt nach der Zeit auf, wobei der Grenzwert der Einteilung 8.987825 beträgt, was einer Abweichung von dem im Modell definierten Grenzwert 9.0 von deutlich unter einem Prozent.

### 7.1.3. Multilayer Perceptron (ANN)

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 2 | 65.0 | 0.652 | 1.07 | 35.4 | 91.1 |
| 3 | 70.0 | 0.766 | 4.90 | 33.3 | 88.6 |
| 4 | 80.1 | 0.876 | 51.29 | 21.7 | 83.4 |
| 5 | 81.1 | 0.886 | 488.59 | 21.1 | 83.0 |

Tabelle 7.3.: Ergebnisse des Multilayer Perceptron ohne Rauschen

Der Multilayer Perceptron liefert im Vergleich zum ZeroR gute Resultate, bereits bei nur 100 Objekten im Trainingsset werden fast 2 von 3 neuen Objekten korrekt klassifiziert. Für $n > 2$ ähnelt die Performance im Punkt Accuracy derer des J48, allerdings bei signifikant höheren Kosten der Laufzeit. Bereits bei $10^5$ Objekten dauerte die Erstellung des Modells mehrere Minuten, wohingegen es beim J48 lediglich einige Sekunden dauerte.

Da eine 10-fold Cross-Validation die Konstruktion von insgesamt 11 Modellen benötigt, wurde bereits für $n = 5$ ein 90%-Split verwendet um die insgesamt benötigte Zeit der Simulation zu begrenzen.

Interessant ist die mit zunehmender Datenmenge sinkende Sicherheit der Vorhersagen, die beim J48 tendenziell eher gestiegen war.

Die Laufzeit scheint linear zu steigen, allerdings mit einem sehr hohen Anfangswert. Bei $10^6$ Datensätze dürfte die benötigte Zeit im Bereich von etwa einer Stunde liegen.

Insgesamt erscheint der Multilayer Perceptron aufgrund der hohen Laufzeiten und relativ zum J48 schlechteren Performance bei größeren Datenmengen eher für kleine Datensätze geeignet, bei welcher er relativ zum J48 bessere Resultate in hinnehmbar höherer Laufzeit erzeugt.

### 7.1.4. Naives Bayes

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 2 | 63.0 | 0.634 | 0.05 | 43.8 | 68.2 |
| 3 | 69.1 | 0.717 | 0.00 | 29.3 | 68.8 |
| 4 | 70.2 | 0.743 | 0.04 | 25.8 | 69.1 |
| 5 | 69.4 | 0.737 | 0.22 | 26.4 | 68.5 |
| 6 | 69.8 | 0.741 | 0.70 | 25.4 | 68.7 |

Tabelle 7.4.: Ergebnisse des Naive Bayes ohne Rauschen

Auch die Verwendung von naivem Bayes liefert relativ zum ZeroR-Algorithmus gute Ergebnisse. Im Gegensatz zum J48- und ANN-Algorithmus verbessert sich die Accuracy aber nicht wesentlich über 70% und ist somit deutlich schwächer als diese, die ihre Maxima bei etwas über 80% erreichen. Auch die AUC und die False Positive Rate sowie die Certainty sind im Vergleich zu den anderen beiden analysierten Algorithmen merklich schlechter.

Wie in der theoretischen Beschreibung des Naive Bayes dargelegt, zeichnet sich dieser durch eine sehr hohe Geschwindigkeit aus. Während der J48 bei $10^5$ Objekten bereits über drei Sekunden Laufzeit benötigt und der Multilayer Perceptron sogar 8 Minuten, bleibt der Naive Bayes auch bei $10^6$ noch deutlich unterhalb einer Sekunde. Somit ist es durchaus nützlich um im Classifier Output einen ersten Überblick über die Verteilungen der Klassen zu gewinnen und ggf. manuelles Clustering vorzunehmen.

## 7.2. Untersuchungen von geclusterten rauschfreien Datensätzen

Für diese Untersuchung wurden Datensätze generiert, bei denen auch die Art des Gebiets bekannt sind. Da die Straßen innerhalb eines Gebiets eine ähnliche Verhalten aufweisen (Homogenität) und die Gebiete untereinander sehr unterschiedlich sind (Heterogenität) kann die Angabe des Gebiets als Clustering bezeichnet werden.

In der Praxis entsprächen die nun generierten Datensätze also der Annahme vollständiger Datensätze ohne fehlende Parameter mit vorheriger Clusterung. Es wurden wieder 5 Datensätze der Größen $10^n$ für $n = 2...6$ generiert.
Sofern nicht anders angegeben, finden die Untersuchungen für $n = 2...5$ mit einer 10-fold Cross-Validation statt, für n = 6 mit einem 90%-Split. Die Classifier-Outputs aus Weka sind im Anhang zu finden.

### 7.2.1. ZeroR

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 2 | 55.0 | 0.449 | 0.00 | 100 | 54.4 |
| 3 | 52.0 | 0.500 | 0.00 | 100 | 54.8 |
| 4 | 54.8 | 0.500 | 0.03 | 100 | 54.8 |
| 5 | 54.7 | 0.500 | 0.03 | 100 | 54.6 |
| 6 | 54.3 | 0.500 | 0.12 | 100 | 54.5 |

Tabelle 7.5.: Ergebnisse des ZeroR ohne Rauschen mit Clustering

Wie bereits bei nicht-geclusterten Datensätzen zeigt sich das typische Verhalten des ZeroR. Die Klassifkation findet nur auf Basis der Häufigkeit der Klassen im Trainingsset statt, wodurch die Laufzeit gering ist, das Ergebnis aber auch entsprechend wenig nutzbar ist. Die Angabe der Art des Gebiets, also eines zusätzlichen Faktors der durch Clustering hätte entstanden sein können, hat bei diesem Algorithmus wie zu erwarten keinerlei Effekt.

### 7.2.2. J48

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) | Blätter | Knoten |
|---|---|---|---|---|---|---|---|
| 2 | 71.0 | 0.669 | 0.21 | 26.7 | 75.0 | 2 | 3 |
| 3 | 78.0 | 0.816 | 0.05 | 28.0 | 88.2 | 77 | 117 |
| 4 | 82.4 | 0.876 | 0.52 | 16.7 | 84.8 | 156 | 239 |
| 5 | 82.9 | 0.893 | 4.63 | 16.5 | 83.3 | 127 | 192 |
| 6 | 83.3 | 0.895 | 75.39 | 15.5 | 85.2 | 130 | 189 |

Tabelle 7.6.: Ergebnisse des J48 ohne Rauschen mit Clustering

Bei einem zuvor geclusterten Datenset weist der J48 im Bereich kleiner Datenmengen eine deutliche Perfomancesteigerung gegenüber dem ungeclusterten Set auf. Für $10^2$ und $10^3$

liegt eine Steigerung der Accuracy um 7 Prozentpunkte vor, die FPR ist für $10^2$ über 20 Prozentpunkte geringer als bei nicht-geclusterten Daten. Diese Effekte normalisieren sich ab $10^4$. Es ist davon auszugehen, dass der J48 den Informationsvorteil des geclusterten Sets ab einer hinreichend großen Datenmenge ausgleicht. Hierfür werden allerdings tiefergehende Verzweigungen benötigt, was die beim nicht-geclusterten Datenset deutlich höhere Anzahl an Blättern und Knoten erklärt. Bis auf einen Ausreißer bei $n = 3$ ist die Anzahl der Blätter und Knoten jeweils signifikant kleiner als beim Classifier des nicht-geclusterten Sets.

```
area = living
|   time <= 9.000393
|   |   weekday = Mon: N (3280.0/273.0)
|   |   weekday = Tue: N (3212.0/285.0)
|   |   weekday = Wed: N (3304.0/270.0)
|   |   weekday = Thu: N (3106.0/308.0)
|   |   weekday = Fri: N (3183.0/297.0)
|   |   weekday = Sat: Y (3184.0/502.0)
|   |   weekday = Sun: N (3164.0/979.0)
|   time > 9.000393
|   |   time <= 16.998379
|   |   |   weekday = Mon: Y (2836.0/732.0)
|   |   |   weekday = Tue: Y (2793.0/705.0)
|   |   |   weekday = Wed: Y (3012.0/774.0)
|   |   |   weekday = Thu: Y (2823.0/738.0)
|   |   |   weekday = Fri: Y (2944.0/817.0)
|   |   |   weekday = Sat
|   |   |   |   temperature <= 15.015023
|   |   |   |   |   street = 1
|   |   |   |   |   |   precipitation = true: Y (101.0/47.0)
|   |   |   |   |   |   precipitation = false
|   |   |   |   |   |   |   specialOccurrence = true
|   |   |   |   |   |   |   |   time <= 16.185028: N (21.0/6.0)
|   |   |   |   |   |   |   |   time > 16.185028: Y (4.0)
|   |   |   |   |   |   |   specialOccurrence = false: N (183.0/72.0)
|   |   |   |   |   street = 2: Y (350.0/101.0)
|   |   |   |   |   street = 3
|   |   |   |   |   |   precipitation = true: Y (118.0/51.0)
|   |   |   |   |   |   precipitation = false: N (243.0/111.0)
```

Listing 7.2: Auszug des J48-Classifier-Baumes für 100000 zuvor geclusterte Objekte

Bei der Betrachtung des gebildeten Baumes für $n = 5$ ist interessant zu beobachten, dass dieser der Entscheidungslogik unseres Datenmodells sehr nahe kommt. Zunächst wird nach Art des Gebiets unterschieden, welches im Datenmodell der gravierendste Faktor war, anschließend in mehreren Schritten das korrekte Zeitfenster bestimmt, wobei die Abweichung

der ermittelten Grenzwerte von denen im Modell vorgegebenen weit unter einem Promill liegen, anschließend findet eine Unterteilung nach den Wochentagen statt.

Die Unterschiede zwischen den einzelnen Straßen ist für den J48 stellenweise so gering, dass der Informationsgewinn durch die Unterteilung als zu gering bewertet und die entsprechenden Verzweigungen gepruned werden. Am Pruning des Baumes des geclusterten Datensets können wir ablesen, welche Faktoren den stärksten Einfluss auf die Klassifikation nehmen, in diesem Fall Gebiet, Uhrzeit und Wochentag. Die Auswirkungen von Sperrungen des ÖPNV sind im Baum stellenweise noch sichbar, weitestgehend aber durch Pruning entfernt.

Betrachten wir den nicht-geprunten Baum (s. Anhang), so lassen sich die Auswirkungen aller Faktoren beobachten, wodurch der Baum allerdings auch signifikant komplexer wird; im Falle von $n = 5$ besitzt dieser 720 Blätter und 1123 Knoten, also jeweils rund 6 mal so viele wie beim geprunten. Die Performance des ungeprunten Baumes ist in keinem Faktor unwesentlich abweichend von denen des geprunten Baumes.

### 7.2.3. Multilayer Perceptron (ANN)

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 2 | 62.0 | 0.679 | 0.85 | 35.6 | 90.6 |
| 3 | 71.9 | 0.786 | 5.80 | 29.8 | 90.2 |
| 4 | 79.6 | 0.869 | 57.54 | 17.8 | 83.2 |
| 5 | 80.7 | 0.889 | 564.55 | 24.7 | 82.7 |

Tabelle 7.7.: Ergebnisse des Multilayer Perceptron ohne Rauschen mit Clustering

Wie bereits im Falle des ungeclusterten Datensatzes wurde auch hier für $n = 5$ aufgrund der hohen Laufzeit ein 90%-Split statt einer 10-fold Cross-verwendet und auf die Analyse für $n = 6$ verzichtet, da davon auszugehen ist, dass diese etwa 90 Minuten benötigen würde.

Anders als beim J48 erhöhte sich für keinen Fall die Performance durch die Angabe des Clusterattributs. Die Accuracy ist zumeist leicht schlechter, ebenso die Laufzeit. Insgesamt lässt sich keinen nennenswerte Veränderung, insbesondere keine Verbesserung durch Angabe der Art des Gebiets beim Multilayer Perceptron feststellen.

### 7.2.4. Naives Bayes

| n | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 2 | 71.0 | 0.734 | 0.02 | 22.2 | 83.9 |
| 3 | 67.6 | 0.715 | 0.01 | 21.9 | 80.5 |
| 4 | 67.2 | 0.730 | 0.04 | 20.5 | 80.0 |
| 5 | 67.4 | 0.736 | 0.21 | 20.2 | 77.7 |
| 6 | 67.0 | 0.736 | 0.29 | 20.1 | 80.1 |

Tabelle 7.8.: Ergebnisse des Naive Bayes ohne Rauschen mit Clustering

Beim naiven Bayes ist für das geclusterte Datenset interessant zu beobachten, dass die Accuracy für das Datenset mit 100 Objekten merklich besser verhält, darüber hinaus aber um 67% pendelt, während die Accuracy des Classifiers des ungeclusterten Datensets eine Accuracy um 70% pendelte. Auch die AUC und die Laufzeit sind durch das Hinzufügen der Art des Gebiets zumeist merklich schlechter geworden, lediglich die FPR hat sich merklich verbessert.

Insgesamt führt die Angabe eines Clusterkriteriums beim Naiven Bayes zu keiner Verbesserung, was daran liegen könnte, dass ein Cluster der zuvor beschriebenen naiven Annahme des Naive-Bayes-Algorithmus zuwidergeht. Ein Cluster ist letztlich ein aus bekannten Kriterien abgeleitetes Kriterium, wodurch der Cluster stark mit anderen Faktoren korrelliert. Der naive Ansatz geht allerdings explizit davon aus, dass die Faktoren nicht korreliert sind. Somit war zu erwarten, dass ein Clusterkriterium nicht zu einer nennenswerten Verbesserung führen würde.

## 7.3. Untersuchung mit verrauschten Datensätzen

Bei dieser Untersuchung wird betrachtet, wie sich die Algorithmen verhalten, wenn einzelne oder mehrere Faktoren teilweise nicht angegeben sind. Hierfür wurden mehrere Datensätze mit je 1000 Datenobjekten generiert, für die jeweils gewisse Anteile bestimmter Faktoren durch ein „?“ ersetzt wurden.

### 7.3.1. Rauschen im Clusterkriterium

In den beiden vorhergehenden Sektionen wurden zum einen ein gänzlich ungeclustertes Datenset verwendendet, zum anderen ein komplett geclustertes Datenset. Diese Untersuchungen können als Extrema eine Untersuchung der Performance der Algorithmen bei Rauschen

im Clusterkriterium angesehen werden, wobei ein gänzlich ungeclustertes Datenset einem Rauschen von 100% und ein gänzlich geclustertes Set einem Rauschen von 0% entspricht. Die Zwischenräume zwischen diesen Extrema wurde nun in Schritten von je 20 Prozentpunkten untersucht.

**ZeroR**

| Noise (%) | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 0 | 54.46 | 0.500 | 0.01 | 100 | 54.46 |
| 20 | 53.51 | 0.500 | 0.02 | 100 | 53.50 |
| 40 | 54.85 | 0.499 | 0.00 | 100 | 54.85 |
| 60 | 54.80 | 0.500 | 0.00 | 100 | 54.80 |
| 80 | 53.94 | 0.500 | 0.00 | 100 | 53.90 |
| 100 | 54.76 | 0.500 | 0.03 | 100 | 54.76 |

Tabelle 7.9.: Ergebnisse des ZeroR mit Rauschen im Clusterkriterium bei $n = 4$

Der ZeroR ist per Definition unabhängig von jeglichen Kriterien mit Ausnahme der Klasse; somit überrascht es nicht, dass sich keiner der gemessernen Faktoren durch das Ansteigen des Rauschens so stark verändert, dass man von einer Korrelation ausgehen könnte. Der ZeroR ist somit robust gegenüber Rauschen.

**J48**

| Noise (%) | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) | Blätter | Knoten |
|---|---|---|---|---|---|---|---|
| 0 | 82.38 | 0.876 | 0.52 | 16.7 | 83.3 | 156 | 239 |
| 20 | 81.32 | 0.876 | 0.48 | 17.9 | 83.8 | 195 | 298 |
| 40 | 80.94 | 0.872 | 0.19 | 17.6 | 84.4 | 212 | 344 |
| 60 | 81.84 | 0.873 | 0.27 | 19.2 | 85.3 | 192 | 299 |
| 80 | 80.93 | 0.871 | 0.21 | 18.8 | 85.2 | 240 | 395 |
| 100 | 81.48 | 0.871 | 0.41 | 17.9 | 85.6 | 238 | 391 |

Tabelle 7.10.: Ergebnisse des J48 mit Rauschen im Clusterkriterium bei $n = 4$

Der J48 scheint für Veränderungen durch Rauschen im Clusterkriterium weitestgehend unanfällig zu sein. Die Accuracy, AUC, FPR verhalten sich alle relativ konstant. Die leichten Schwankungen in diesen Faktoren lassen keine Trends erkennen sind somit vermutlich auf

zufällige Schwankungen aufgrund der unterschiedlichen Datensätze zurückzuführen. Die Certainty lässt ein leichtes Ansteigen erkennen.

Auffällig ist allerdings die Komplexität des Baumes. Mit zunehmendem Rauschen wird die Anzahl der Blätter und Knoten größer. Wie in der Evaluation des geclusterten rauschfreien J48 auf Seite 54 vermutet, ist der J48 anscheinend in der Lage den Informationsverlust durch steigendes Rauschen bei ausreichend großer Datenmenge auszugleichen. Man könnte sagen, der J48 ist in der Lage die dem Clustering zugrundeliegende Logik selbst nachzubilden. Hierfür werden zusätzliche Verzweigungen benötigt, was sich mit der vorgefundenen Steigerung der Knoten- und Blätteranzahl deckt. Der J48 ist somit bzgl. Rauschen im Clusterkriterium gemessen an der Zuverlässigkeit seiner Aussagen robust.

#### Multilayer Perceptron (ANN)

| Noise (%) | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 0 | 79.60 | 0.869 | 57.54 | 17.8 | 83.2 |
| 20 | 80.33 | 0.875 | 67.68 | 20.2 | 82.1 |
| 40 | 79.58 | 0.875 | 67.86 | 20.8 | 83.3 |
| 60 | 80.68 | 0.875 | 79.15 | 18.7 | 84.0 |
| 80 | 79.91 | 0.877 | 73.94 | 21.9 | 84.4 |
| 100 | 80.10 | 0.876 | 51.29 | 21.7 | 83.4 |

Tabelle 7.11.: Ergebnisse des Multilayer Perceptron mit Rauschen im Clusterkriterium bei $n = 4$

Der Multilayer Perceptron zeigt nahezu keinerlei Veränderungen seiner gemessenen Parameter abhängig vom Rauschen. Die Accuracy schwankt in einem sehr kleinen Tunnel, die AUC ist nahezu konstant, die Laufzeit lässt keine Trends erkennen, die FPR steigt leicht, aber nicht signifikant an. Insgesamt ist dieser ANN-Algorithmus gegenüber Rauschen im Clusterkriterium robust.

**Naives Bayes**

| Noise (%) | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 0 | 67.20 | 0.730 | 0.04 | 20.5 | 80.0 |
| 20 | 68.34 | 0.732 | 0.07 | 21.7 | 78.8 |
| 40 | 68.10 | 0.719 | 0.02 | 22.5 | 75.5 |
| 60 | 68.62 | 0.720 | 0.02 | 25.4 | 72.5 |
| 80 | 68.99 | 0.726 | 0.02 | 25.1 | 70.5 |
| 100 | 70.20 | 0.743 | 0.04 | 25.8 | 69.1 |

Tabelle 7.12.: Ergebnisse des Naive Bayes mit Rauschen im Clusterkriterium bei $n = 4$

Während die Laufzeit wie zu erwarten vom Rauschen unabhängig und somit bis auf normales Schwanken konstant ist und auch die AUC relativ konstant bleibt, ist festzustellen, dass mit zunehmendem Rauschen die Accuracy wider erstes Erwarten steigt und die FPR ebenfalls steigt. Lediglich die Certainty fällt mit mit wachsendem Rauschen stetig ab.

Eine Betrachtung der anderen Verhältnisse auf Basis der jeweiligen Confusion Matrix zeigt, dass die FPR zwar steigt, dafür aber in etwas stärkerem Maße die False Negative Rate, also der Anteil der fälschlicherweise als negativ klassifizierten positiven Objekte sinkt. Die Steigerung der Accuracy trotz - bzw. gerade wegen - der Steigerung des Rauschens im Clusterkriterium liegt vermutlich, wie zuvor dargelegt, an der naiven Annahme des Naive Bayes, die davon ausgeht, dass die Faktoren untereinander unkorreliert sind, obwohl dies insbesondere beim Clusterkriterium nicht der Fall ist.

### 7.3.2. Rauschen im Zeitkriterium

Wie zuvor in der Sektion J48 auf Seite 54 festgestellt, spielt die Uhrzeit eine wichtige Rolle bei der Klassifikation. Daher wird nun in Schritten von je 20 Prozentpunkten die Performance der Algorithmen bei zunehmendem Rauschen auf diesem Faktor analysiert. Da bereits bekannt ist, dass das Rauschen eines Kriteriums keine Auswirkung auf die Performance des ZeroR hat, wird auf die Evaluation dessen verzichtet. Die Evaluation der übrigen drei Algorithmen wird anhand von 10000 Objekten großen Datensätzen unter Verwendung einer 10-fold Cross-Validation erstellt.

**J48**

| Noise (%) | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) | Blätter | Knoten |
|---|---|---|---|---|---|---|---|
| 0 | 82.38 | 0.876 | 0.52 | 16.7 | 84.8 | 156 | 239 |
| 20 | 78.76 | 0.855 | 0.45 | 20.6 | 77.3 | 118 | 183 |
| 40 | 74.79 | 0.820 | 0.68 | 23.6 | 72.6 | 131 | 192 |
| 60 | 71.75 | 0.747 | 0.35 | 27.9 | 70.0 | 44 | 63 |
| 80 | 69.40 | 0.707 | 0.32 | 27.4 | 69.5 | 14 | 18 |
| 100 | 69.57 | 0.718 | 0.25 | 33.5 | 70.4 | 68 | 106 |

Tabelle 7.13.: Ergebnisse des J48 mit Rauschen im Zeitkriterium bei $n = 4$

Für den J48 ist mit zunehmendem Rauschen eine merkliche Verschlechterung der Performance hinsichtlich Accuracy und AUC festzustellen. Auch die FPR steigt stark an und verdoppelt sich von 0% zu 100% Rauschen. Dies war zu erwarten, da das Zeitkriterium in der Logik des Datenmodells eine wichtige Rolle gespielt hat und - wie in den vorherigen Analyse des J48 festgestellt - für den Baum eines der Hauptkriterien zur Klassifkiations darstellte. Auch die Certainty fällt mit zunehmendem Rauschen um bis zu fast 15 Prozentpunkte ab. Vor diesem Hintergrund ist die Performance des J48, etwa im Vergleich zu den Benchmarking-Ergebnissen des ZeroR in den vorherigen Analysen, sogar noch relativ gut.

Interssant zu beobachten ist, dass die größe des geprunten Baumes mit zunehmendem Rauschen abnimmt, was daran liegen dürfte, dass zuvor das Zeitkriterium ein wichtiges Unterscheidungsmaß darstellt, dessen Relevanz nun bei immer stärkerem Rauschen so weit absinkt, dass der Informationsgewinn so gering wird, dass diese Verzweigungen im Prozess des Prunings entfernt werden.

**Multilayer Perceptron (ANN)**

| Noise (%) | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 0 | 79.60 | 0.869 | 57.54 | 17.8 | 83.2 |
| 20 | 77.43 | 0.854 | 52.01 | 26.3 | 81.0 |
| 40 | 72.93 | 0.842 | 52.11 | 18.4 | 80.5 |
| 60 | 74.64 | 0.824 | 53.53 | 41.7 | 78.6 |
| 80 | 69.63 | 0.784 | 58.56 | 32.1 | 79.5 |
| 100 | 68.37 | 0.736 | 52.72 | 28.6 | 72.2 |

Tabelle 7.14.: Ergebnisse des Multilayer Perceptron mit Rauschen im Zeitkriterium bei $n = 4$

Für den Multilayer Perceptron ist festzustellen, dass seine Performance mit steigendem Rauschen im Zeitkriterium sich ebenfalls merklich verschlechtert. Ebenso wie der J48 sinkt die Performance schrittweise auf einen Grenzwert knapp unterhalb von 70% Accuracy und einer AUC von ca. 0.72. Auch bei diesem Algorithmus sinkt die Certainty, wenn auch weniger stark als beim J48.

Die Laufzeit ist vom Rauschen wie zu erwarten nicht maßgeblich beeinflusst, lediglich die FPR und die Certainty zeigen ebenfalls Verschlechterungen. Dennoch ist auch hier anzumerken, dass die Accuracy trotz Totalausfall eines der kritischsten Kriterien im Vergleich zum Benchmark relativ hoch bleibt.

**Naive Bayes**

| Noise (%) | Acc (%) | AUC | t (s) | FPR (%) | Certainty (%) |
|---|---|---|---|---|---|
| 0 | 67.20 | 0.730 | 0.04 | 20.5 | 80.0 |
| 20 | 67.03 | 0.732 | 0.06 | 20.8 | 80.3 |
| 40 | 67.00 | 0.725 | 0.01 | 20.8 | 79.9 |
| 60 | 67.07 | 0.729 | 0.02 | 21.0 | 79.8 |
| 80 | 67.35 | 0.729 | 0.02 | 21.0 | 79.8 |
| 100 | 67.36 | 0.726 | 0.01 | 22.2 | 79.5 |

Tabelle 7.15.: Ergebnisse des Naive Bayes mit Rauschen im Zeitkriterium bei $n = 4$

Anders als die J48- und ANN-Algorithmen, bleibt der naive Bayes-Algorithmus vom Rauschen im Zeitkriterium unbeeinflusst. Die Accuracy schwankt um einige Nachkommastellen, lässt aber keine Trends erkennen, die AUC schwankt ledliglich um Promillewerte, die Schwankungen der Laufzeit sind in absoluten Werten so gering, dass davon auszugehen ist, dass hier eher Hintergrundprozesse des Betriebssystems für die Schwankungen verantwortlich sind. Auch die Certainty schwankt lediglich um einige Promill, was vermuten lässt, dass es sich hier eher um statistisches Rauschen handelt.

Der naive Bayes ist somit gegenüber Rauschen in diesem Falle weitestgehend unabhängig, liefert dafür aber auch in jedem Fall nur eine Accuracy, die ungefähr derer von komplett verrauschten J48 und ANN entsprechen. Somit verhält sich der naive Bayes bei Rauschen zwar relativ besser, absolut aber nicht signifikant besser als der Worst Case der anderen beiden Algorithmen.

## 7.4. Bedeutung der Ergebnisse für die Anwendung

Auf Basis der ermittelten Ergebnisse darf davon ausgegangen werden, dass eine Vorhersage über das Auffinden von Parkplätzen prinzipiell vermutlich möglich ist, womit der als Ziel dieser Arbeit erklärte vorläufige Proof of Concept erbracht wäre. Notwendig für eine nicht nur statistisch signifikante, sondern eine dem Nutzer einer mobilen Anwendung zumutbare Qualität der Aussage hinsichtlich Faktoren wie die Genauigkeit und Sicherheit sind nach den durchgeführten Tests folgende Parameter:

- **Menge der Daten:** Obwohl teilweise bereits mit kleinen Datenmengen gute Resultate erzielbar waren, kann in der Regel erst ab einer Menge von zwischen $10^3$, also 1000, und $10^4$, also 10000, Objekten von nutzbarer Qualität der Accuracy gesprochen werden.
- **Auswahl des/der Algorithmus/Algorithmen:** Wie in den Analysen erkennbar ist, verhalten sich unterschiedliche Algorithmen bei unterschiedlichen Umständen, wie etwa Unterschieden in der Datenmenge, Existenz von Clustering und Rauschen, unterschiedlich. Es kann daher sinnvoll sein, abhängig von Datenmenge und -qualität verschiedene Algorithmen zu verwenden. Im Falle unserer Anwendung könnte dies bspw. heißen, dass an einer Gabelung für die linke Straße, für welche viele Daten vorliegen, ein J48-Classifier verwendet wird, während für die rechte Straße, bei der nur einige hundert, teilweise verrausche Datensätze vorliegen, ein Naive Bayes Classifier zum Einsatz kommt.
- **Clustering:** In der Analyse war zu erkennen, dass der J48 und der Naive Bayes insbesondere im Bereich kleiner Datenmengen massiv von der Existenz eines Clusterkriteriums profitiert haben. Der zusätzliche Aufwand Cluster zu bestimmen, kann somit gerade dort sinnvoll sein, wo innerhalb eines Gebiets Unterschiede in den vorhandenen Datenmengen einzelner Straßen existieren. Stellen wir uns etwa drei im Dreieck angeordnete Straßen vor, in denen für zwei Straßen große Datenmengen existieren, für eine aber nur wenige. Auf Basis der Annahme, dass nahegelegene Straßen ähnliches Verhalten aufweisen, könnten diese drei Straßen zusammengeclustert werden, wodurch sich die Aussagequalität für die Straße mit geringer Datenmenge merklich erhöhen könnte.
- **Qualität der Datenmenge:** Neben der absoluten Menge an Datensätzen ist auch die Qualität der einzelnen Datenobjekte relevant. Hierbei zeigte die Analyse von Datensätzen mit Rauschen zweierlei:

    1. Verschiedene Faktoren tragen verschieden stark zur Qualität der Vorhersage bei. Betrachten wir die Auswirkungen von Rauschen im Cluster- und Zeitkriterium für den J48, so ist zu erkennen, dass Rauschen in verschiedenen Faktoren

verschieden schwere Auswirkungen auf die gemessenen Parameter hat. Diese Erkenntnis deckt sich mit der Evaluation des J48 für rauschfreie geclusterte Datensätze auf Seite 54, in welcher festgestellt wurde, dass weniger relevante Faktoren existieren, die in der Regel dem Pruning zum Opfer fallen. Für die hypothetische Application heißt dies, dass aus der Menge aller erfassten Faktoren diejenigen bestimmt und schrittweise entfernt werden müssen, die keine oder nur geringe Auswirkungen aus das Auffinden von Parkplätzen hat. Hierzu könnte möglicherweise der Naive Bayes verwendet werden, mit dem sich die Korrelationen zwischen Faktoren und der Klasse schnell ermitteln lassen.

2. Der Grad des Rauschens beeinflusst bei wichtigen Faktoren, wie in unserem Beispiel die Uhrzeit, das Ergebnis. Es ist somit darauf zu achen, dass die Erfassung der Daten möglichst vollständig erfolgt und verrauschte Daten nach Möglichkeit durch weitere Datenerfassung komplettiert werden. Denkbar wäre bspw. im Falle von verrauschten Einträgen zum aktuellen Wetter die Vervollständigung durch Ermittelns des Wetters aus externen Datenbanken unter Verwendung von Standort und Uhrzeit.

Angesichts dieser Anforderungen muss bezüglich der praktischen Machbarkeit - also dem Proof of Value - abgewogen werden, inwieweit dem Aufwand der Erfüllung dieser Kriterien die Möglichkeit gegenübersteht, die Anwendung zu monetarisieren.
Prinzipiell ist davon auszugehen, dass alle Faktoren in zufriedenstellendem Maße erfüllbar sind. Insbesondere die Erschließung von Datenquellen, die Zusammenführung von Daten aus mehreren Quellen (u.a. auch für die Komplettierung von Datensätzen) und die Konzipierung und Realisation eines Systems, welches eigenständig zum Datensatz passende Algorithmen zur Erstellung von Classifiern auswählt, dürften mit erheblichem Aufwand verbunden sein. Darüber hinaus wäre ein solches System auch bei Auswahl effizienter Algorithmen bei wachsenden Datenmengen sehr schnell rechenintensiv, wodurch zugekaufte Rechenzeit oder selbst verwaltete Hardware bezahlt werden müssten. Somit verbleibt der Proof of Value unter den genannten Anforderungen vor allem eine nicht-technische, sondern betriebswirtschaftliche Fragestellung, deren Beantwortung an anderer Stelle zu erörtern ist.

# 8. Zusammenfassung

## 8.1. Fazit

Im Verlauf dieser Arbeit wurde das Ziel dieser Arbeit als Ermittlung eines Proof of Concept und Proof of Value für eine potentielle Anwendung zur Unterstützung bei der Auffindung von Parkplätzen unter Zuhilfenahme von Data-Mining-Techniken definiert.

Der Begriff Data Mining wurde definiert und kontextuell eingeordnet und anschließend durch Darlegung verschiedener Verfahrensarten und jeweiliger Algorithmenbeispiele weitergehend erläutert. Auf die für die spätere Evaluation notwendigen Methodiken wurde durch Erläuterung von Hold-Out und Cross-Validation, der Confusion Matrix und den daraus ableitbaren Parametern sowie den ROC-Kurven und der entsprechenden Messgröße AUC eingegangen.

Anschließend wurde die Problematik fehlender vorhandener Daten dargelegt und ein Modell konzipiert und realisiert, welches die notwendigen Daten für die späteren Analysen generiert. Kurz wurde auf das verwendete Softwaretool Weka und das zugehörige .arff-Format eingegangen, welches für die Analyse verwendet wurde.

In der Evaluation wurden neben dem für Benchmarking verwendeten ZeroR die drei Algorithmen J48 aus der Klasse der Entscheidungsbäume, Multilayer Perceptron aus der Klasse künstlicher neuronaler Netze und Naive Bayes aus der Klasse der Bayes-Algorithmen verwendet. Untersucht wurde die Performance der Algorithmen unter verschiedenen Bedingungen anhand mehrerer zuvor definierter Kriterien. Bei den verwendeten Bediungungen handelte es sich um Änderungen der Datenmenge, Clusterung sowie Rauschen in mehreren Attributen.

Abschließend wurde die Bedeutung der evaluierten Ergebnisse für eine möliche Parkplatzapp dargelegt; konkret, dass eine solche vermutlich prinzipiell realisierbar sei, die Rechtfertigung für den notwendigen Aufwand und somit der Proof of Value aber noch zu erbringen ist.

## 8.2. Ausblick

Die Erbringung eines grundlegenden Proof of Value stellt den nächsten Schritt für eine mögliche Anwendung dar. Nur wenn die Realisation einer solchen Anwendung ökonomisch tragbar ist, sind weitergehende Schritte sinnvoll.

Sofern Modelle zur hinreichenden Monetarisierung existieren werden, kann es sinnvoll sein, auf Basis echter Datensätze die Performance der dargelegten Algorithmen, so wie weiterer Algorithmen, zu evaluieren. Die Ergebnisse der Performanceanalysen der Algorithmen in dieser Arbeit sind stark vom selbsterstellenden Datenmodell abhängig, weshalb ähnliche Evaluationen mit echten Daten zu wiederholen wären.
Auch sollte, abhängig von der für echte Daten ermittelten Relevanz, tiefergehend auf Clustering-Verfahren eingegangen werden um die theoretischen und praktischen Grenzen zu ermitteln und abzuwägen, ob dem Mehraufwand ein hinreichender Nutzen gegenübersteht.
Letztlich verbleibt es interessant, inwieweit die Performance der Algorithmen durch Feineinstellungen jeweiliger Parameter optimierbar sind, wie etwa Beschränkungen von Knotenanzahlen im Falle von Entscheidungsbäumen oder Topologieänderungen bei neuronalen Netzen. Diese Thematik, welche den Rahmen dieser Arbeit überstiegen hätte, bietet für jeden Algorithmus tiefergehende Möglichkeiten der Verbesserung von zeitlicher Performance und Ergebnisqualität und stellt somit ein weitläufiges Feld zur weiteren Vertiefung dar.

# Literaturverzeichnis


[Adriaans und Zantinge 1996] ADRIAANS, Pieter ; ZANTINGE, Dolf: *Data Mining*. Addison-Wesley Professional, 1996. – ISBN 0-2014-0380-3

[Aggarwal und Yu 1999] AGGARWAL, Charu C. ; YU, Philip S.: *Data Mining Techniques for Associations, Clustering and Classification*. S. 13–23. In: ZHONG, Ning (Hrsg.) ; ZHOU, Lizhu (Hrsg.): *Methodologies for Knowledge Discovery and Data Mining: Third Pacific-Asia Conference, PAKDD-99 Beijing, China, April 26–28, 1999 Proceedings*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1999. – URL https://doi.org/10.1007/3-540-48912-6_4. – ISBN 978-3-540-48912-2

[Böhm u. a. 2003] BÖHM, Christian ; ESTER, Martin ; JANUZAJ, Eshref ; KAILING, Karin ; KRÖGER, Peer ; SANDER, Jörg ; SCHUBERT, Matthias: *Skript zur Vorlesung Knowledge Discovery in Databases im Wintersemester 2003/2004*. 2003. – URL http://www.dbs.informatik.uni-muenchen.de/Lehre/KDD/WS0304/Skript/kdd-1-einleitung.pdf. – Zugriffsdatum: 21.08.2017

[Cabena u. a. 1996] CABENA, Peter ; HADJINIAN, Pablo ; STADLER, Rolf ; VERHEES, Jaap ; ZANASI, Alessandro: *Discovering data mining - from concept to implementation*. Prentice Hall, 1996. – ISBN 0-1374-3980-6

[Collinson 2010] COLLINSON, Patrick: *Beware the cookies: they can cost you money*. 2010. – URL https://www.theguardian.com/money/blog/2010/aug/07/computer-cookies-booking-online. – Zugriffsdatum: 21.08.2017

[Fayyad u. a. 1996] FAYYAD, Usama M. ; PIATETSKY-SHAPIRO, Gregory ; SMYTH, Padhraic ; FAYYAD, Usama M. (Hrsg.) ; PIATETSKY-SHAPIRO, Gregory (Hrsg.) ; SMYTH, Padhraic (Hrsg.) ; UTHURUSAMY, Ramasamy (Hrsg.): *Advances in Knowledge Discovery and Data Mining*. 1996

[Golgoski 2012] GOLGOSKI, Nina: *How Target knows when its shoppers are pregnant - and figured out a teen was before her father did*. 2012. – URL http://www.dailymail.co.uk/news/article-2102859/How-Target-knows-shoppers-pregnant--figured-teen-father-did.html. – Zugriffsdatum: 21.08.2017

[Gottermeier 2003] GOTTERMEIER, Christian: *Data Mining: Modellierung, Methodik und Durchführung ausgewählter Fallstudien mit dem SAS Enterprise Miner*, Universität Heidelberg, Diplomarbeit, 2003. – URL http://archiv.ub.uni-heidelberg.de/volltextserver/4073/1/Diplomarbeit_Christian_Gottermeier.pdf. – Zugriffsdatum: 21.08.2017

[Knobloch 2000] KNOBLOCH, Bernd: *Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten*. 2000. – URL http://www.ceus-bayern.de/forschung/downloads/%5BKnob00%5D.pdf. – Zugriffsdatum: 21.08.2017

[Krahl u. a. 1998] KRAHL, Daniela ; ZICK, Friedrich-Karl ; WINDHEUSER, Ulrich: *Data Mining: Einsatz in der Praxis*. Addison-Wesley Verlag, 1998. – ISBN 3-8273-1349-X

[Schinzer u. a. 1999] SCHINZER, Heiko ; BANGE, Carsten ; MERTENS, Holger: *Data Warehouse und Data Mining: Marktführende Produkte im Vergleich.* Vahlen, 1999. – ISBN 3-8006-2466-4

[Witten 2013] WITTEN, Ian: *Data Mining with Weka. Class 2 - Lesson 1*. 2013. – URL https://docs.google.com/file/d/0B-f7ZbfsS9-xdXpOUmgyaDdDR0U/edit. – Zugriffsdatum: 21.08.2017

[Zhu und Davidson 2007] ZHU, Xingquan ; DAVIDSON, Ian: *Knowledge discovery and data mining: challenges and realities*. IGI Global, 2007. – ISBN 1-5990-4252-5

# A. Beispiel FP-Growth Tree

Bei diesem Beispiel samt all seinen Bildern handelt es sich um leicht abgewandelte Variante eines Beispiels von singularities.com[1].

Sei unsere Menge: { {beer, bread, butter, milk} , {beer, milk, butter}, {beer, milk, cheese} , {beer, cheese, bread}, {beer, diapers, cheese} }.

Wir ermitteln die Häufigkeit der einzelnen Artikel: {beer: 5, bread: 2, butter: 3, milk: 3, cheese: 3, diapers: 1}

Definieren wir einen Mindestsupport der Artikel von 30%, so muss ein Artikel wenigstens 30% der Transaktionen, also mindestens 1.5 bzw. gerundet 2 Transaktionen erscheinen. Die Windeln (diapers) erfüllen dieses Kriterium nicht und werden entsprechend entfernt. Anschließend wird die Liste sortiert: {beer: 5, butter: 3, milk: 3, cheese: 3, bread: 2}. Dies ist unser Frequent Item Header Table.

Die Transaktionen werden nun in den Baum eingehangen. Bei jedem Überstreichen wird der Zähler des Knotens inkrementiert.

[1] https://www.singularities.com/blog/2015/08/apriori-vs-fpgrowth-for-frequent-item-set-mining

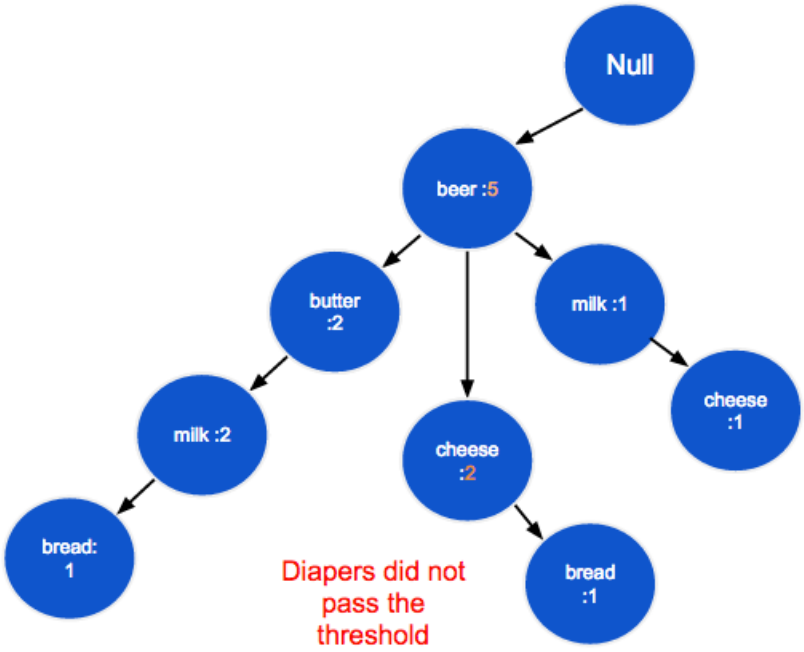


Abbildung A.1.: Der konstruierte Baum

Bei der Einsortierung von der letzten Transaktion {beer, diapers, cheese} sehen wir, dass Transaktionen, die Elemente enthalten die zuvor aussortiert wurden, ab dem Zeitpunkt abgebrochen werden. Somit ist oben der ungestutzte Baum zu sehen. [Anmerkung: im Originalbeispiel ist hier ein Fehler unterlaufen und im endgültigen Baum wurde beer:4 angegeben]

Nun gehen wir rekursiv durch den Baum und prüfen, ob Blätter existieren, deren Zählerwert unterhalb des Mindestsupport von hier 2 Transaktionen liegt. Wir sehen, dass der Knoten „bread: 1“ links unten unterhalb des Mindestsupport liegt, weshalb dieser entfernt wird. Selbiges gilt für das andere „bread: 1“ Blatt, sowie für den gesamten Ast „milk: 1 - cheese: 1“. Der Baum wird also heruntergestutzt auf Folgendes:

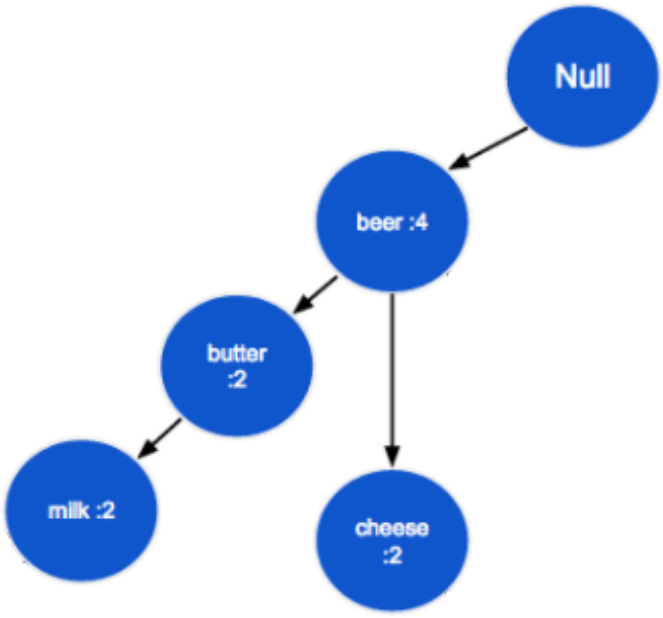


Abbildung A.2.: Der gestutze Baum

Hieraus lassen sich folgende Assoziationen schließen:

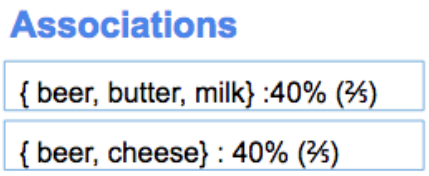


Abbildung A.3.: Aus dem Baum schließbare Assoziationen

# B. Beispiel Satz von Bayes

In unserem Beispiel geht es um die Vorhersage ob Golf abhängig von der Wettervorhersage gespielt werden sollte. Hierzu existiere folgende Tabelle, in welcher die Wettervorhersage der Eingangsvektor und „Golf spielen? - Ja/Nein“ die Klassifikation darstellt:

| Wettervorhersage | Golf spielen? | |
|---|---|---|
| | Ja | Nein |
| Sonne | 3 | 2 |
| Bewölkung | 4 | 0 |
| Regen | 2 | 3 |

Abbildung B.1.: Wertetabelle des Beispiels

Basierend auf dieser Tabelle bestimmen wir die Wahrscheinlichkeiten. Die zweite und dritte Spalte gibt jeweils an, wie häufig sich die Faktoren Sonne, Bewölkung und Regen innerhalb der als Ja bzw. Nein klassifizierten Fälle befinden. Die vierte Spalte gibt die allgemeine Häufigkeit der Faktoren Sonne, Bewölkung und Regen im gesamten Datensatz an. Die letzte Zeile gibt an, wie häufig eine Klasse („Ja/Nein“) eingetreten ist.

| Wettervorhersage | Golf spielen? | | |
|---|---|---|---|
| | Ja | Nein | |
| Sonne | 3/9 | 2/5 | 5/14 |
| Bewölkung | 4/9 | 0/5 | 4/14 |
| Regen | 2/9 | 3/5 | 5/14 |
| | 9/14 | 5/14 | |

Abbildung B.2.: In rot hervorgehoben: die zur Berechnung wichtigen Werte

Wollen wir nun die Wahrscheinlichkeit des Ereignisses „Ja“ (es wurde Golf gespielt) bei Sonne berechnen, so entspricht dies $p(C|X)$, was sich lesen lässt als „die Wahrscheinlichkeit,

dass C eintritt wenn X eingetreten ist“. Zur Berechnung verwenden wir den Satz von Bayes

$$p(C|X) = \frac{p(X|C)p(C)}{p(X)}$$

und bestimmen die Größen auf der rechten Formelseite aus unserer zweiten Tabelle.

- $p(X|C)$, hier $p(\text{Sonne}|\text{Ja})$, die Wahrscheinlichkeit dass die Sonne schien als Golf gespielt wurde, entspricht laut Tabelle 3/9, also 1/3.
- $p(C)$, hier $p(\text{Ja})$, die Häufigkeit der als „Ja“ klassifizierten Objekte, laut Tabelle 9/14
- $p(X)$, hier $P(\text{Sonne})$, die Häufigkeit der Objekte in denen Sonne existierte ist laut Tabelle 5/14.

Nach unserem Satz von Bayes gilt: $p(C|X) = p(\text{Ja}|\text{Sonne}) = (1/3 * 9/14)/(9/14) = 60\%$

# C. Quellcode, Datensätze, Classifier Outputs

Siehe Inhalt der beliegenden DVD.

# Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

| Hamburg, 4. September 2017 | |
|---|---|
| Ort, Datum | Unterschrift |