Sebastian Mandl, BSc

# Echtzeitentscheidungen in der Supply Chain

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

Masterstudium Softwareentwicklung und Wirtschaft

eingereicht an der

**Technische Universität Graz**

*Betreuer:*

Dipl.-Ing. Dr.techn. Roman Kern

Institute of Interactive Systems and Data Science

Graz, Juli 2019

# EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

| | |
|---|---|
| Datum | Unterschrift |

# Kurzfassung

Prognosen in heutigen Lieferketten sind von immer mehr Einflussfaktoren abhängig und deshalb wird es immer schwieriger, die Laufzeiten vorherzusagen. Aus diesem Grund müssen oft externe Systeme abgefragt werden, was in der Regel ressourcenintensiv ist.
Ziele dieser Arbeit sind die Entwicklung und Einführung eines Entscheidungsbaumes, um die direkte Abhängigkeit von externen Services zu eliminieren und die Vorhersage anhand von historischen Daten durchzuführen. Über einen Datengenerator können synthetische aber auch konstante Testdaten erzeugt und somit die Performance des entwickelten Entscheidungsbaumes getestet werden.
Der Baum selbst unterscheidet zwischen Entscheidungsfragen und manuellen Fragen. Entscheidungsfragen werden vollständig in der Lernphase anhand der Parameter-Objekte definiert, wohingegen manuelle Fragen vorab programmiert werden. Eine Entscheidungsfindung basiert auf der Grundlage, dass so wenig Ebenen wie möglich erzeugt werden. Die Vereinfachung des Baumes wird anhand von mathematischen Operationen bzw. statistischen Werkzeugen, wie dem Ignorieren von unwahrscheinlichen Ergebnissen, erreicht.
In dieser Arbeit wird gezeigt, dass es möglich ist, eine NoSQL Datenbank für das Speichern von Entscheidungsmodellen zu verwenden. Darüber hinaus kann aufgezeigt werden, dass die Vorhersage des Zustelldatums in einem Online-Shop mittels Entscheidungsbaum möglich ist.

# Abstract

Today's supply chains depend on various internal and external influences. Due to this a prediction of run-time is difficult and queries to external systems leads to a high consumption of resources.
The main goal of the thesis is to design and implement an decision tree algorithm to eliminate the dependency on these external systems.
Firstly a data generator is used to provide stable test data to make a comparison of different executions possible. The tree differs between decision questions and manual ones. Decision question are built in the learning phase and relies on the given parameter object whereas manual question need to be programmed beforehand. The decision making process is based on the fact that the resulting tree should have as few levels as possible. Pruning of the tree is made by various mathematical and statistical tools like ignoring unlikely sub decisions.
The outcome of this thesis is the prove that saving decision models into a NoSQL database is possible and predicting the date of delivery for an online shop based on a decision tree is possible.

# Inhaltsverzeichnis

# Abbildungsverzeichnis



# Code Beispiele

# Tabellenverzeichnis

# 1. Vorwort

Durch meine langjährige Arbeit im Bereich der Webentwicklung mit dem Fokus auf Online-Shops in der Niceshops GmbH, war es mir relativ schnell möglich ein geeignetes Thema für die vorliegende Masterarbeit zu finden. Die Unterstützung des Unternehmens sowie die zahlreichen anregenden Diskussionen mit meinen Kollegen haben dazu beigetragen, dass die Arbeit ihre jetzige Gestalt angenommen hat.
Die interdisziplinäre Thematik der Programmierung und Wirtschaft verlangt ein breites Wissen in diesen beiden Fachgebieten und vereint meine beruflichen sowie privaten Interessen hervorragend. Die gewonnenen Erfahrungen aus dem Studium "Softwareentwicklung und Wirtschaft" bilden die ideale Grundlage für die Ausarbeitung meiner Masterarbeit.
Durch die tatkräftige Unterstützung von Herrn Dipl.-Ing. Dr.techn. Roman Kern konnte das Thema schlussendlich eingegrenzt und die Eckpfeiler schnell definiert werden. Mein Betreuer war stets für mich erreichbar und stand mir mit guten Tipps zur Seite.
Ein großer Dank gilt meiner Familie, da durch meine Eltern die Grundlage für mein technisches Interesse gelegt wurde. Ebenso möchte ich mich bei meinen Schwiegereltern bedanken, die mich seit Jahren unterstützen.
Der größte Dank gilt jedoch meiner Freundin Alexandra, die mich motiviert hat das Masterstudium an der TU Graz in Angriff zu nehmen und stets dafür gesorgt hat, dass ich mein Studium nun erfolgreich abschließen kann.

# 2. Einleitung

Entscheidungen in heutigen Lieferketten, wie zum Beispiel ob ein Produkt noch rechtzeitig geliefert werden kann oder ob der aktuelle Lagerstand noch ausreicht, um die Bestellungen bis zur nächsten Lieferung zu bedienen, werden immer komplexer und unvorhersehbarer. Durch diesen Umstand ist die Steuerung eines solchen Systems mit einem hohen Ressourcenaufwand verbunden. Diese Ressourcen stellen auf der einen Seite sogenannte Buffer zum Zwischenlagern von Materialien beziehungsweise Produkten oder personellen Reserven und auf der anderen Seite den Aufwand für die Planung dar. Ein weiterer Faktor, der im Zusammenhang mit diesem komplexen System steht, ist die Vorlaufzeit, welche verschiedene Ressourcen brauchen, um nachproduziert beziehungsweise geliefert zu werden. Hierbei treten oft hohe Schwankungen auf, welche zum Teil auch saisonal bedingt sind und durch Analyse der bereits bestehenden Daten erkannt werden müssen. Aber auch eine Produktion hat eine Vorlaufzeit betreffend der Rüstung von Maschinen und dem Kommissionieren von Materialien. Dadurch ist das Vorausplanen von Abläufen und die Einschätzung von kurzfristigen Änderungen eine wichtige Anforderung an ein solches Managementsystem. Das System selbst sollte jedoch nicht die genauen Arbeitsaufgaben vorgeben, sondern einen Rahmen schaffen, in dem die einzelnen Ressourcen, basierend auf deren Kompetenzen, die zu erledigende Arbeit selbstständig abarbeiten können.
Ziel einer Lieferkette ist, die kürzest mögliche Durchlaufzeit unter Berücksichtigung einer ausgewogenen Ressourcenverteilung zu erreichen. Des Weiteren sollen den Teilnehmerinnen und Teilnehmern die Informationen über Auslastung und Verfügbarkeit der Ressourcen beziehungsweise Wiederbeschaffungs- und Auslieferdauer zur Verfügung stehen. Insbesondere ist das Bereitstellen von qualitativ hochwertigen Parametern bezüglich Produktionsdauer und Lieferzeiten für die nachstehenden Teilnehmerinnen und Teilnehmer der Supply Chain, wie für einzelne Kundinnen und Kunden in einem Onlineshop, eine große Herausforderung für derzeitige Systeme. Durch ein ausgereiftes Reporting-System mit integrierten Vorhersagen, soll eine dynamische Grundlage für die Entscheidungsfindung geschaffen und durch eine Schnittstelle die gewonnen Daten in ein bestehendes System integriert werden.
Im Zuge der Arbeit wird ein Entscheidungsmodells, welches über mehrere Ebenen eine Abstraktion der Arbeits- und Beschaffungsschritte erlaubt entwickelt und durch eine Implementation die Durchführbarkeit bewiesen. Die Berechnung soll auf den obersten Ebenen in annähernd Echtzeit durchgeführt werden können. Erst wenn das Modell in

periodischen Abständen genauer durchgerechnet wird, werden alle Elemente und deren Abhängigkeiten betrachtet. In diesem Schritt sollen auch die Werte der Abstraktionsebenen angepasst und dadurch die Entscheidungen selbst in den abstrahierten Berechnungen verbessert werden. Die periodischen Abstände bilden den Takt, welcher als eine zeitliche Abgrenzung und als Event für das Erzeugen von Aufgaben gesehen werden kann. Darüber hinaus sollen Meilensteine im Entscheidungsmodell eingesetzt werden, um die Performance des Systems zu messen und ein Gegensteuern zu ermöglichen. Diese Meilensteine sind zum Beispiel erzielte Pakete für einen Lieferdienst von einem Tag oder bei wie vielen Paketen das versprochene Versanddatum eingehalten wurde. Gerade dieses automatische Controlling soll in jedem Tick [1] durchgeführt werden und das Entscheidungsmodell durch Verändern der Ressourcenverteilung angepasst werden. Hierbei sollen aber immer die Abhängigkeiten und Vorlaufzeiten von den einzelnen Teilnehmerinnen und Teilnehmern der Supply Chain berücksichtigt werden.

## 2.1. Forschungsfragen

Bei den täglichen Arbeiten in einem international tätigen Online-Handel, sind immer wieder Probleme bei Entscheidungsfindung im Bezug auf die Warenwirtschaft und deren Umfeld aufgetaucht. Daraus haben sich folgende zwei Forschungsfragen ergeben, welche im Zuge dieser Arbeit beantwortet werden sollen.
Wie können kurzfristige Änderungen in einem baumartigen Modell berücksichtigt werden und unter Zuhilfenahme von Abhängigkeiten zwischen den Knoten zu einer Veränderung der Entscheidungen in einer Supply Chain führen?
Ist eine NoSQL Datenstruktur für das Abbilden von dynamischen Modellen mit zeitabhängigen Knoten geeignet? Gibt es darüber hinaus die Möglichkeit eine derartige Struktur über eine Vorhersage aufzubauen?

## 2.2. Hypothesen

Entscheidungen können auf der Basis einer sich stetig und in kurzer Zeit ändernden Datenbasis durch Verwendung von Vorhersagen getroffen werden. Wird in diesem Zusammenhang die Berechnungszeit eingeschränkt, wird davon ausgegangen, dass die Genauigkeit der Entscheidung sinkt.
Mithilfe historischer Daten vom erwarteten Versand- beziehungsweise Zustelldatum lässt sich ein zu erwartender Entscheidungsbaum erstellen und somit ist sowohl eine fundierte

[1] Ein Tick ist eine fixe Zeitspanne, in welche die zu erledigende Arbeit eingeteilt werden kann.

Planung von Ressourcen als auch der einzelnen Abläufe möglich. Darüber hinaus können Vorlaufzeiten durch die auf den Vorhersagen basierenden Planungen minimiert und ein höherer Output erzielt werden.

# 3. Entscheidungen in der Supply Chain

In diesem Kapitel wird darauf eingegangen, welche Entscheidungen in einer Supply Chain getroffen werden und wie diese sich gegenseitig beeinflussen. Hierzu muss zuerst eine Supply Chain, wie sie zum Beispiel im Online-Handel üblich ist, aufgezeigt werden, um eine Diskussionsgrundlage für die anfallenden Entscheidungen zu schaffen.

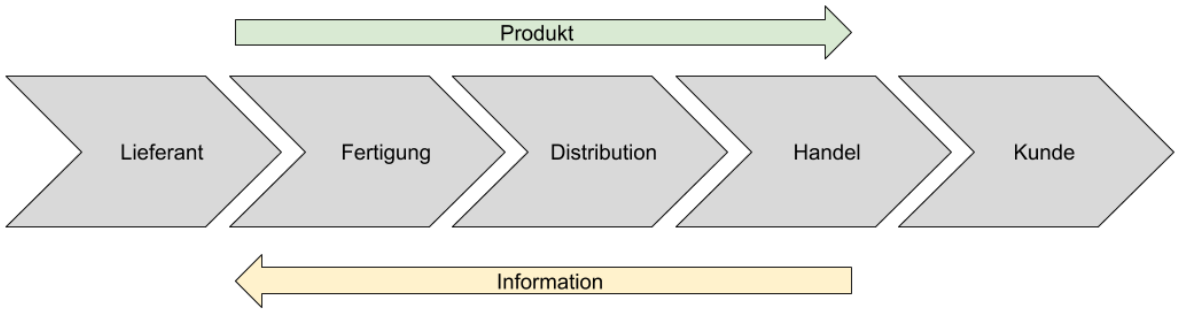


Abbildung 1: Bestandteile einer einfachen Supply Chain, eigene Abbildung

In der Abbildung 1 sind grob die fünf Phasen für die Supply Chain des Online-Handels skizziert vom Lieferanten bis hin zu der Zustellung bei der Kundin beziehungsweise beim Kunden. Hierbei wird zwischen dem Informationsfluss von der Käuferin und dem Käufer weg zurück in die Supply Chain und dem Produktfluss vom Ursprung der Produktion bis zur Kundin beziehungsweise zum Kunden unterschieden.

## 3.1. Lieferantin/Lieferant

Die Lieferantin bzw der Lieferant stellt die Rohstoffe für die nachgelagerten Fertigungsschritte zur Verfügung, hierbei werden neben reinen Rohstoffen auch bereits halbfertige Produkte bereitgestellt, welche in der Fertigung nur noch zu fertigen Produkten assembliert werden müssen. Hierbei kann die Beziehung zwischen Lieferantin beziehungsweise Lieferant und Fertigung auch als Kreislauf gesehen werden, da zur Herstellung der halbfertigen Produkte bereits eine Fertigung notwendig ist. Die wichtige Entscheidung im Bezug auf die Supply Chain ist, ob die Lieferantin beziehungsweise der Lieferant die benötigte Menge in ausreichender Qualität aufbringen kann. Dies kann oft aus den historischen Daten geschlossen werden, indem die Liefergenauigkeit beziehungsweise die tatsächlich gelieferte Menge betrachtet wird.

## 3.2. Fertigung

In der Produktionsphase werden vorab eingekaufte Rohstoffe in Veredelungsschritten zu einem fertigen Produkt. Diese Schritte hängen stark von der vorhandenen Ressourcenplanung ab, was die Hauptaufgabe der Produktionsplanung ist. Ziel ist es vorhandene Ressourcen wie Material, Maschinen und Menschen so effizient einzusetzen wie möglich, um den höchstmöglichen Output zu erreichen. Als Anfangsglied der Supply Chain ist die Produktion am meisten von den nachfolgenden Teilnehmerinnen und Teilnehmern der Supply Chain und deren Bedarf abhängig. Aus diesem Grund ist entweder ein großes Lager als Buffer oder eine funktionierende Vorhersage notwendig. Das Lager dient dazu, den Bullwipp Effekt abzufangen, welcher auftritt, wenn nachstehende Käufer Mengenrabatte ausnutzen und hierdurch immer Spitzen beim Bedarf entstehen.
Um eine Produktion durchführen zu können, muss nach [Hartmunt Stadtler, 2004] vorab ein Material Requirements Planning (MRP) durchgeführt werden, in diesem Plan werden alle Ressourcen und deren Menge festgehalten. Ein Teil davon ist zum Beispiel die Stückliste, nach welche der Einkauf vorgenommen wird. Damit die vorhandenen Maschinen optimal ausgenutzt und Vorlaufzeiten minimiert werden können, arbeiten Produktionslinien meist in Lots. Dabei handelt es sich immer um eine bestimmte Anzahl von Produkten derselben Art, welche am Stück produziert werden und somit kann die Vorlaufzeit der einzelnen Maschinen reduziert werden. Die Größe der Lots wird oft durch den Arbeitsplan und den geforderten Output definiert. Die einzelnen Arbeitsschritte werden in einem Prozessmodell abgebildet und dadurch alle Abhängigkeiten oder Bottlenecks sichtbar gemacht. Hierbei sind auch oft die Fehlerfälle von Interesse und werden in das Modell mitaufgenommen. In der Produktion selbst sind sehr viele Entscheidungen auf Basis des MRP, der verfügbaren Ressourcen und des gewünschten Outputs zu treffen.
Der Einkauf wird stark vom vorherrschenden Markt beeinflusst, gerade im Bezug auf Nachfrage und Angebot muss auf der einen Seite rasch reagiert werden, um ein Stockout zu verhindern. Auf der anderen Seite ist es von Vorteil in großen Mengen einzukaufen, um dadurch einen Mengenrabatt zu bekommen. Voraussetzungen hierfür sind jedoch die ausreichende Lagerkapazität und die nötigen liquiden Mittel.

## 3.3. Distribution

Im Bezug auf die Verteilung der Waren kann die Unterscheidung getroffen werden, ob der Handel in stationärer Form nachgelagert ist oder über Online-Shops die Distribution der Waren auch direkt an die Kundin beziehungsweise den Kunden stattfindet. Bei

der Nachlieferung in die einzelnen Standorte werden immer größere Mengen von einzelnen Produkten, oft sogar sortenrein, bestellt. Die Ware wird auf Paletten transportiert und hierdurch wird das Handling enorm erleichtert. Für die Verteilung von Paketen an die Endkundinnen und Endkunden ist der Prozess der Kommissionierung recht aufwendig, da hierfür große Distanzen in den Warenhäusern zurückgelegt werden müssen und eine geringe Stückzahl bestellt wird. In diesem Zusammenhang ist die Vorlaufzeit für die Kommissionierung und Verpackung zu berücksichtigen. Des Weiteren werden die Pakete und Paletten von den einzelnen Lieferdiensten zu vorab festgelegten Zeiten abgeholt. Ist ein Paket bis zu diesem Zeitpunkt nicht fertig verpackt, verzögert sich die Verteilung bis zur nächsten Abholung. Da nicht alle Lieferdienste zur selben Zeit die Abholung durchführen, ist die Planung der Kommissionierung eine wichtige Aufgabe in diesem Schritt. Die Zustellung bei Firmenstandorten wird meist avisiert und findet zu bestimmten Zeitpunkten statt, dies ist notwendig da gerade bei größeren Lieferungen am Bestimmungsort nicht genügend Ressourcen, wie zum Beispiel Docks für Lkws, zur Verfügung stehen könnten. Lieferungen an Endkundinnen beziehungsweise Endkunden hingegen sind für die Lieferdienste immer mit der Unsicherheit behaftet, dass der Kunde beziehungsweise die Kundin nicht zuhause ist und somit eine erneute Zustellung oder Zwischenlagerung stattfinden muss. Gerade bei der Zustellung an Privatpersonen hat sich in den letzten Jahren der Trend zur Vorabinformation über den erwarteten Zustelltag beziehungsweis Zeitpunkt immer weiter fortgesetzt und viele Lieferdienste wie zum Beispiel DHL oder die österreichische Post bieten diesen Service mittlerweile ohne Zusatzkosten an. In diesem Zusammenhang ist die Entscheidung, wie viele und welche Pakete in Zustellung gehen, unter Berücksichtigung der zur Verfügung stehenden Transportmittel, zu treffen.

## 3.4. Handel

Beim Handel wird, wie bereits im Abschnitt davor erwähnt, zwischen stationär und online unterschieden. Welche Produkte und vor allem wie viele gelagert werden, hängt bei jedem Standort davon ab, wie viel Platz vorhanden ist. Aus diesem Grund ist es besonders im stationären Handel von großer Bedeutung, dass das Produktsortiment auf die vorherrschende Kundengruppe abgestimmt wird. Sogenannte Stockouts wirken sich hierbei besonders auf den Umsatz aus, da Ware in den seltensten Fällen vorbestellt wird. Das bedeutet, dass die Vorhersage über die benötigten Produkte einwandfrei funktionieren muss. Der Online-Handel profitiert, im Vergleich zum reinen Geschäft, von erhöhter

Lagerfläche und geringeren Personalkosten. Jedoch müssen hier Prozesse zum Teil noch besser durchdacht werden, vor allem wenn Hoch- oder Außenlager eingesetzt werden. Diese Lagerarten haben eine hohe Vorlaufzeit im Vergleich zu normaler Regallagerung, bei welcher eine Person die Waren picken kann. Beim Picken wird das Produkt aus dem Lager entnommen und für die Kommissionierung bereitgestellt. Je größer ein Lager wird, desto wichtiger ist die Berücksichtigung eines eigenen Routings und der Möglichkeit einen Auftrag auf mehrere Personen aufzuteilen. Im stationären Handel entfällt der Prozess des Pickens komplett, hier nehmen die Kundinnen und Kunden selbst die Produkte aus den Regalen und es muss nur sichergestellt werden, dass diese ausreichend bestückt sind. Darüber hinaus hängt es bei der Lagerung stark vom Produkt ab, wie aufwändig diese ist und wie hoch die Vorlaufzeit ist, um neue Produkte wieder verpacken zu können. Herausforderungen bei der Lagerung sind unter anderem Kühlwaren, welche einen hohen Platz- und Energieaufwand erzeugen. Das System muss in diesem Zusammenhang die Entscheidungen unterstützen, welche Produkte an welchen Orten gelagert werden und wann ein Produkt aus einem Hoch- beziehungsweise Außenlager umgeräumt wird. Des Weiteren wird entschieden welche Bestellungen zusammen gepickt werden, um die optimalen Wege zu erreichen. Im stationären Handel soll das System bei der Entscheidung unterstützen, welche Produkte aus dem Zentrallager nachbestellt werden.

## 3.5. Kundin/Kunde

Das letzte Glied in der Supply Chain sind die Kundinnen und Kunden. Sie definieren den endgültigen Bedarf und somit auch den Markt für ein Produkt. Das Kaufverhalten online sowie auch stationär wird durch viele Faktoren beeinflusst beziehungsweise gesteuert. Durch geschickte Produktplatzierung in Filialen oder durch gezielte Werbekampagnen werden Käufe generiert und dadurch können Verkäufe bis zu einem bestimmten Grad vorhergesagt werden. Wichtig in diesem Zusammenhang ist, dass die Kundinnen und Kunden an ein Unternehmen gebunden werden und somit ein Teil der eigenen Supply Chain bleiben.

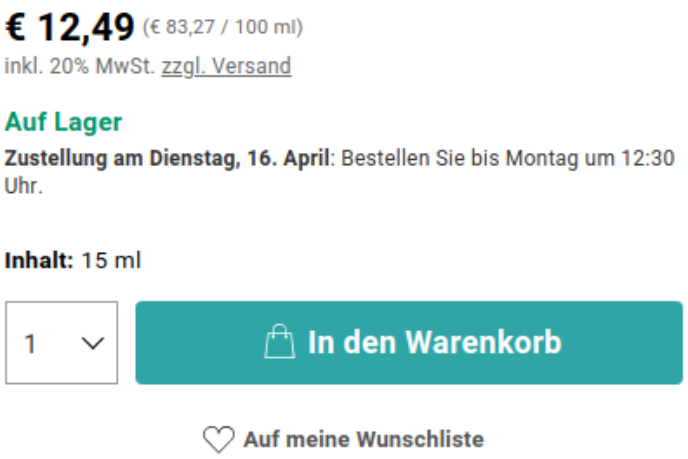


Abbildung 2: Ansicht der Lieferzeit [Niceshops, 2019]

Gerade in einem Online-Shop ist die Entscheidung von Interesse, ob ein Produkt heute noch versendet werden kann und wann die Lieferung schlussendlich bei der Empfängerin beziehungsweise beim Empfänger ist. Da die gesamte Supply Chain auf diese Frage Einfluss hat und dies sogar, wenn die Produktion oder Distribution herangezogen wird, in Echtzeit passieren kann, ergeben sich viele Ebenen und Abhängigkeiten, welche zu einem komplexen Entscheidungsbaum werden. Diese Information soll, wie in Abbildung 2 aufgezeigt wird, der Kundin beziehungsweise dem Kunden frühzeitig vermittelt werden um ein bestmögliches Einkaufserlebnis zu erzielen.

# 4. Forschungsstand

In diesem Kapitel wird auf den aktuellen Stand der Technik eingegangen und die für die Arbeit notwendigen Technologien näher betrachtet. Insbesondere sollen die verwendeten Programmiersprachen, Datenbanksysteme und Algorithmen aufgezeigt und deren theoretischen Grundlagen erklärt werden.

## 4.1. ERP-Systeme

Entscheidungen in Produktionsprozessen beziehungsweise von Lieferungen beruhen meist auf einem Enterprise Ressource Planning, kurz ERP, dies wird nach [Martin Hesseler, 2007] als firmenweite Ressourcenplanung verstanden und durch eine Standardsoftware unterstützt. Diese Software biete Funktionen zur Steuerung, Administration und Disposition in einem Unternehmen. Insbesondere muss so ein System Funktionen zur Finanzsteuerung und Materialwirtschaft bieten.


Abbildung 3: Bestandteile eines ERP-Systems [microtech GmbH, 2017]

Darüber hinaus besitzt laut [Jacob et al., 2008] ein ERP-System eine zentrale Datenbank mit allen Entitäten der verschiedenen Geschäftsfälle und eine tiefgreifende Prozessintegration im Unternehmen. Moderne Systeme für das Planen von Ressourcen in Großunternehmen über 1000 Mitarbeiterinnen und Mitarbeiter sind SAP mit einem Marktanteil von über 40 Prozent, Sage Group oder SSA Global. Microsoft beziehungsweise auch Open Source Software wie OpenERP oder odoo fokussieren sich auf Mittel- und Kleinbetriebe bis maximal 1000 Mitarbeiterinnen und Mitarbeitern. [Martin Hesseler, 2007]
In den verschiedenen Plattformen werden, wie in Abbildung 3, die Daten in die unterschiedlichen Module eingetragen. Die Produktion umfasst zum Beispiel die vorliegenden Prozesse, Ressourcen und Aufträge. Auf Basis der bereitgestellten Daten wird der Ablauf berechnet und der Output den anderen Modulen zur Verfügung gestellt. Diese Berechnung kann in Abständen von einem Tag bis zu einem Monat durchgeführt werden, je nachdem welche Varianz das zugrundeliegende System aufweist. Eine dynamische Planung in kleineren Zeiteinheiten und die Berücksichtigung von externen kurzfristigen Änderungen ist jedoch meist nicht möglich. Eine besondere Herausforderung für ERP-Software bildet die Erstellung von Produktvarianten, die explizit auf eine Kundin oder einen Kunden zugeschnitten werden und somit die Produktion direkt von der Kundin oder dem Kunden beeinflusst wird. ERP Systeme, die nur im Webbrowser laufen und vom Einkauf bis hin zu den Verpackungs- und Produktionsprozessen alle Schritte abbilden, sind eher selten. Jedoch bieten diese aber gerade in der Industrie 4.0, wo davon ausgegangen wird, dass jede Maschine in einem Unternehmen miteinander verbunden ist, ungeahnte Möglichkeiten, da sie die Unabhängigkeit von Endgeräten oder Betriebssystemen ermöglicht. Gerade aufgrund von diesem Umstand sind unternehmensübergreifende Zusammenarbeiten zur Optimierung der Supply Chain noch eher die Ausnahme.

## 4.2. Algorithmen

Der Einkauf wird meist anhand von historischen Daten berechnet und berücksichtigt selten die Auslastung der eigenen Ressourcen oder die Abhängigkeiten zwischen verschiedenen Typen von Ressourcen. Im schlimmsten Fall werden einfachste Algorithmen eingesetzt, welche nur auf dem Verkauf der letzten Monate beruhen. In diesem Zusammenhang wird teilweise ein gleitender Mittelwert eingesetzt, um die Tendenz einer Datenreihe über die Zeit zu berechnen und daraus Rückschlüsse über die zu erwartenden Verkäufe im nächsten Monat zu erlangen. ARIMA ist ein bekanntes Modell, welches neben autoregressiven Modellen auch das Moving Average einsetzt, um aus Beobachtungen

von vergangenen Zeitperioden Vorhersagen zu generieren.

### 4.2.1. Regressionsalgorithmen

Die Regressionsanalyse ermöglicht, unter Zuhilfenahme von verschiedenen Parametern, wie zum Beispiel Preis oder Werbungskosten für ein Produkt, die Berechnung von den Beziehungen zueinander und die daraus resultierende Auswirkung auf die Verkaufszahlen. Aus den vergangenen Zusammenhängen lässt sich eine Formel erstellen und durch Einsetzen der vorab definierten Parameter können die zukünftigen Verkaufszahlen berechnet werden. In Abbildung 4 ist das Ergebnis eines einfachen Regressionsmodells auf Basis von vorhergegangenen Umsätzen und deren Tendenzen zu sehen. Es basiert auf einem dreistufigen Modell, welches zuerst Umsätze pro Shop, Monat und Land berücksichtigt. Werden für diese Berechnung zu wenig Daten gefunden oder ist der Fehler auf dem Testset zu groß, rechnet das System in weiterer Folge ohne die Aufteilung auf das Lieferland. Das dritte Modell betrifft jene Shops, die relativ neu sind oder sehr wenig Bestellungen haben, in diesem Fall wird von einer linearen Verteilung über das Jahr ausgegangen und der Gesamtumsatz durch zwölf dividiert.

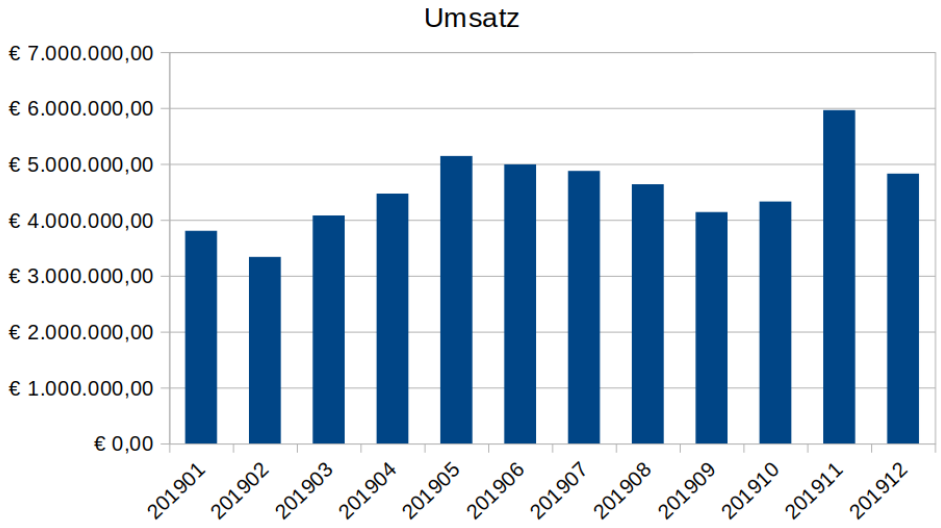


Abbildung 4: Erwarteter Umsatz 2019 [eigene Abbildung]

Bei der Berechnung wurden keinerlei Informationen über externe Abhängigkeiten wie zum Beispiel Werbung oder Preis der Mitbewerberinnen und Mitbewerber berücksichtigt. Auch sind Kampagnen wie zum Beispiel 20% Aktionen nicht erfasst und können daher nicht in die Berechnung einfließen. Ein lineares Regressionsmodell verwendet, wie von

[Von der Lippe, 2018] beschrieben, die belabelten Trainingsdaten, um ein Modell zu berechnen. Hierbei muss über Gradient Descent durch eine Annäherung an ein lokales bezhiehungsweise globales Minimum das optimale Ergebnis gefunden werden. Im einfachsten Fall ist die Hypothese, wie durch Formel 1 veranschaulicht, eine Gerade, welche die Daten beschreibt.

$$h(x) = \theta_0 + \theta_1 * x \tag{1}$$

Der Fehler von dem Modell kann über den Mean Square Error, der quadratischen Abweichung zu den Originaldaten, wie in der Formel 2, berechnet werden. Dieser Fehler wird in den einzelnen Durchläufen Schritt für Schritt kleiner und die Berechnung abgebrochen, sobald der Wert konvergiert. [Hastie et al., 2009]

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(h(x) - y)^2 \tag{2}$$

Dieses lineare Modell ermöglicht, wie in Abbildung 5 ersichtlich, nur das Schätzen von Tendenzen. Besonders bei der mittleren Abbildung ist die Schätzung über reine lineare Modelle mit zum Beispiel nur einem Feature[2] nicht sinnvoll und es erfordert ein Erweitern um polynomische Features, um den beliebigen Verlauf von Daten abzubilden. Wie gut die Annäherung ist, hängt stark von den vorhandenen Features und deren Korrelation mit der endgültigen Regression ab.

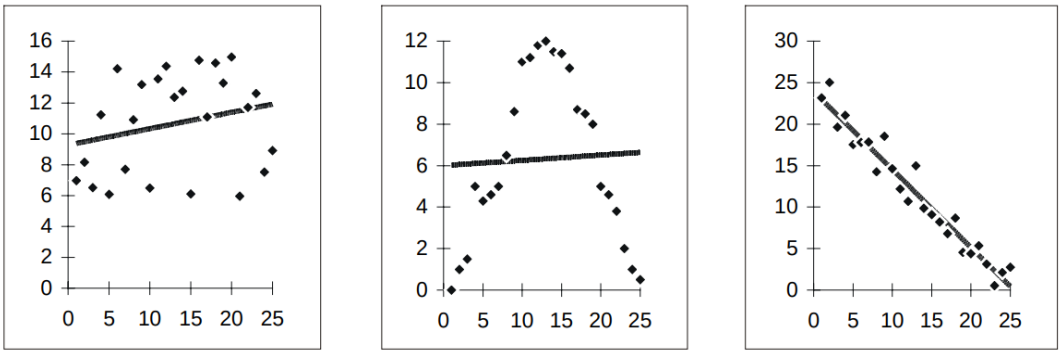


Abbildung 5: Lineare Regression [Von der Lippe, 2018]

Für die richtige Auswahl des Modells werden in verschiedenen Durchläufen die Modelle mit den vorhandenen Daten trainiert und der MSE für jedes Modell berechnet. Ausgewählt wird dann jenes Modell mit dem geringsten Fehler, wird dieser Vorgang nicht

[2]Ein Feature ist ein Merkmal, das die zugrundeliegenden Daten beschreibt. Ein Beispiel wäre die Größe einer Person.

reglementiert, führt dies schnell zum Overfitting, da die Parameter immer weiter ergänzt werden, um ein optimales Ergebnis bei den Trainingsdaten zu erlangen. Beim Overfitting wird auch oft vom Variance/Bias tradeoff gesprochen. Hierbei erhöht sich entweder die Streuung oder es führt zu einem Abdriften des Mittelpunktes am Testset. Die Abbildung 6 veranschaulicht das Problem des Overfittings bei polynomischen Features. Die Grundstruktur der Daten entspricht einem Polynom dritten Grades, was in der mittleren Kurve ersichtlich ist. In diesem Fall ist es laut [Shai Shalev-Shwartz, 2014] notwendig, zufällig Daten aus dem ursprünglichen Set in ein sogenanntes Validation-Set zu extrahieren und das Trainieren nur mit den Daten des Trainigssets durchzuführen.

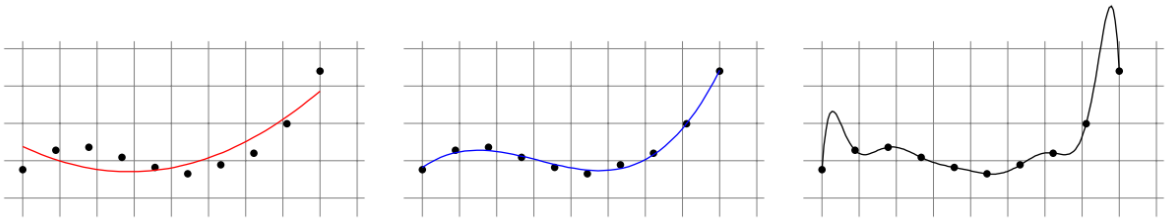

Abbildung 6: Polynomische Regression Grad 2, 3 und 10 [Shai Shalev-Shwartz, 2014]

Hierdurch können verschiedene Modelle trainiert werden, wie zum Beispiel jenes aus Abbildung 6 mit verschiedenen polynomischen Graden. Durch das Fehlen von Datenpunkten wird das Modell mit dem höheren polynomischen Grad die grundlegende Datenstruktur nicht gut abbilden können und somit einen höheren Fehler verursachen. In diesem Zusammenhang wird oft das Early Stopping eingesetzt, was bedeutet, dass man die Ausführung stoppt, sobald der Fehler am Validation-Set zu steigen beginnt. Dieser Fall wird in Abbildung 7 gezeigt. Durch das Erhöhen des polynomischen Grades $d$ sinkt auf der einen Seite der Fehler beim Trainings-Set stetig, auf der anderen Seite beginnt der Fehler am Validation-Set ab dem achten Grad zu steigen. [Shai Shalev-Shwartz, 2014]

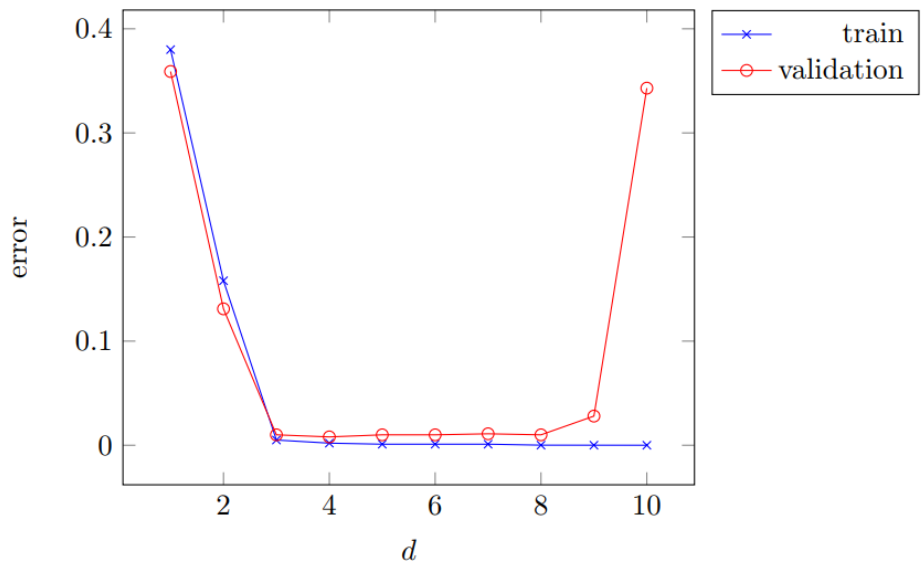


Abbildung 7: Training- vs. Validation-Fehler [Shai Shalev-Shwartz, 2014]

Im Idealfall stehen genügend Daten zur Verfügung, ist dies nicht der Fall, beschreibt [Hastie et al., 2009] mit der k-folds Cross-Validation eine Technologie, bei der das gesamte Set in $k$ Teile zerlegt wird und anschließend für jeden der $k$ Modelldurchläufe das Trainings-Set aus dem Gesamt-Set abzüglich dem Test-Set $k$ gebildet wird. Eine Spezialimplementierung von k-folds ist die leave-one-out, welche verwendet wird, wenn selbst für k-folds zu wenige Daten zur Verfügung stehen. Bei dem Testverfahren wird einfach bei jedem Durchlauf ein Datenpunkt aus dem Trainingsset entfernt und anschließend für diesen Punkt der Fehler berechnet.

### 4.2.2. Neuronale Netze

Moderne Systeme ersetzen oft die bestehenden Regressionsalgorithmen durch neuronale Netze. Bei dieser Technologie minimiert ein Netzwerk durch Lernen der optimalen Parameter oder Gewichte, unter Zuhilfenahme der Hidden-Nodes, den resultierenden Fehler. In der Regel wird bei Machine Learning laut [Shai Shalev-Shwartz, 2014] zwischen zwei Typen unterschieden. Auf der einen Seite gibt es Algorithmen, welche Supervised learning, was soviel wie überwachtes Lernen heißt, umsetzen. Hierzu müssen vorab die Daten mit den richtigen Ergebnissen gekennzeichnet werden. Die lineare Regression aus dem vorangegangenen Kapitel fällt zum Beispiel unter diesen Lerntyp. Auf der anderen Seite gibt es Algorithmen, die Unsupervised Learning ermöglichen. Dabei werden Daten ohne

vorhandene Ergebnissen betrachtet und daraus Rückschlüsse gezogen, wie es bei vielen clustering Algorithmen geschieht.

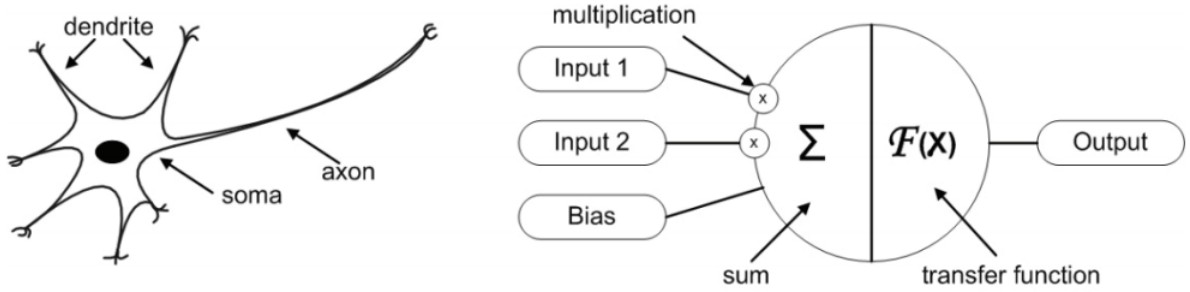


Abbildung 8: Biologischer und Künstlicher Neuron [Andrej Krenker, 2011]

Einem Neuron liegt das Perceptron zugrunde, welches für die einzelnen Inputs und dem zugrundeliegenden Schwellenwert ein Ergebnis berechnet, wie in Abbildung 8 rechts dargestellt. Der Multiplikator ist stellvertretend für die Gewichtung des einzelnen Inputvektors und stellt eine zentrale Komponente beim neuronalen Netz dar. [Andrej Krenker, 2011]
Für die Aktivierungsfunktion $F$ gibt es verschiedene Funktionen wie zum Beispiel den binären Aktivator, welcher auch als Heaviside Step Aktivator bezeichnet wird und durch den Sprung nicht differenzierbar ist. Darüber hinaus ermöglichen Sigmoid Funktionen durch den Einsatz von logistischen Methodiken das Erstellen von einer differenzierbaren Aktivierungsfunktion. Neben binären oder sigmoidalen Funktionen gibt es noch lineare Aktivatoren. [S N Sivanandam, 2006]

$$y(k) = F(\sum_{i=0}^{m} w_i(k) * x_i(k)) \tag{3}$$

Der Schwellenwert für die Funktion wird auch als Bias bezeichnet und kann, wie in Formel 3, direkt als Inputvektor $x_0$ gesehen werden. Das Gewicht für den Bias wird in diesem Fall mit Eins angegeben und die Funktion liefert nur dann ein Ergebnis $y$ zurück, wenn dieser Schwellenwert überschritten wird. [Braspenning et al., 1995]
Erst der Zusammenschluss von Preceptrons zu einem Netzwerk schafft die Voraussetzung, um komplexe Probleme zu lösen. Dabei werden drei verschiedene Typen von neuronalen Netzwerken unterschieden. Das Erste ist ein Single-Layer Feedforward Netzwerk, welches die einfachste Form ist und die Input Layer direkt mit der Ausgangsebene verbindet. Die zweite Art stellen die Multilayer Feedforward Netzwerke, wie in Abbildung

9 links dargestellt, dar. Hierbei werden die Eingangs- und Ausgangsebenen mit beliebig vielen Hidden-Layer verbunden. Je mehr Ebenen eingefügt werden, desto eher können Probleme höherer Ordnung gelöst werden. Ein vollständig verbundenes Netzwerk bedeutet, dass alle Nodes von einer Ebene mit den Nodes der nächsten Ebene verbunden sind.

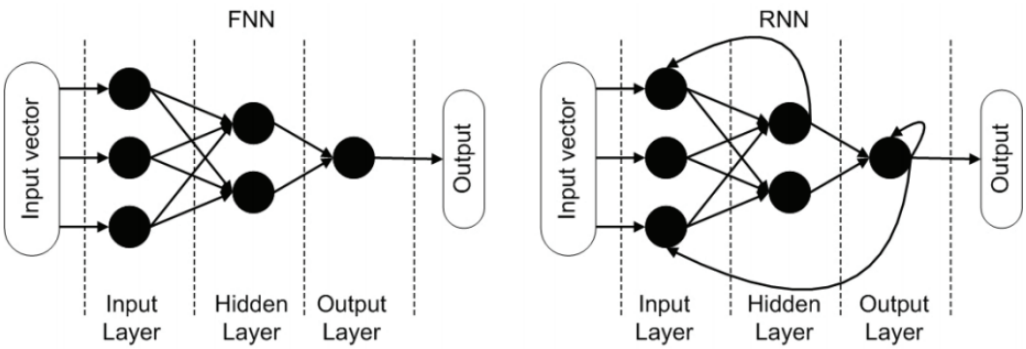


Abbildung 9: Mehrere Ebenen des Netzwerks [Andrej Krenker, 2011]

In der Darstellung 9 ist ein Feedforward Netzwerk links einem Recurrent Netzwerk, welches den dritten Typ darstellt und sich vom Feedforward Netzwerk durch das Vorhandensein von mindestens einer Schleife unterscheidet, gegenübergestellt.
Über Backpropagation wird laut [Braspenning et al., 1995] versucht, dass der Fehler über das gesamte Netzwerk durch Lernen und dem damit verbundenen Setzen der Gewichte von den einzelnen Nodes minimal wird. Dies wird, wie [Rojas, 2013] beschreibt, über das Gradient-Descent Verfahren unter Berücksichtigung der differenzierbaren Aktivierungsfunktionen ermöglicht. Hierbei wird zuerst der Fehler für einen Eingabevektor am Ausgang berechnet und anschließend über Backpropagation jedem Neuron (i) der Anteil an dem Fehler von jedem Sample (K) zugewiesen, wie in Formel 4 ersichtlich ist.

$$E^{(i)} = \frac{1}{2} \sum_{k=0}^{K} (z_k - y_k)^2 \tag{4}$$

$z_k$ beschreibt in diesem Zusammenhang das zu erreichende Ergebnis und $y_k$ das vorhergesagte Ergebnis. Basierend auf dieser Information wird das Gewicht für jede Kante unter Berücksichtigung einer Lernkonstante angepasst, sodass dem negativen Gradient gefolgt wird. Wie bei der linearen Regression empfiehlt sich beim Trainieren von neuronalen Netzen ein Training- und Test-Set einzusetzen, wie bereits im Abschnitt 4.2.1 erläutert wurde. Darüber hinaus existiert, wie [Chauvin and Rumelhart, 2013] beschreibt, die

Möglichkeit bei einem neuronalen Netz dem Overfitting entgegenzuwirken, indem ein Parameter für die Komplexität des Netzwerkes eingeführt wird. In diesem Zusammenhang ist die Komplexität des Netzwerk zum Beispiel durch die Anzahl der Hidden-Nodes definiert.
TensorFlow ist ein Machine Learning System basierend auf neuronalen Netzwerken, welches perfekt in heterogenen Systemen skaliert. Das neuronale Netz kann durch Anpassung der Gewichte an die komplexen Muster abstrakte Darstellungen erzeugen und Zusammenhänge erkennen. TensorFlow selbst ist ein Software-Framework, welches auf vielen Hardwareplattformen ausgeführt werden kann. Die Modelle werden wie Bausteine zusammengesetzt und darüber hinaus gesamte Modelle über Abstraktionsbibliotheken, wie zum Beispiel Keras, zur Verfügung gestellt. [Hope et al., 2018]

### 4.2.3. Entscheidungsbaum

Entscheidungsbäume bieten bei Machine Learning im Bereich der Klassifizierung eine Alternative zu neuronalen Netzen. Grundlegend kann ein Entscheidungsbaum, wie [Barros et al., 2015] beschreibt, einen gerichteten Graphen mit geordneten Paaren und exakt einem Root-Knoten aufweisen. Im Baum dürfen keine Schleifen vorkommen und jeder Knoten darf nur ein Eingangskante haben. Eine Klassifizierung wird, wie in [M. Oded, 2014] beschrieben, über Klassifizierungsbäume gelöst, welche in den Blättern die Klassen und in den Knoten die Entscheidungen halten. Um Regressionsprobleme zu lösen, definieren [de Ville and Neville, 2013] einen Regressionsbaum. Dieser Baum unterscheidet sich vom Klassifizierungsbaum durch eine lokale Suche unter Zuhilfenahme von Regeln. Darüber hinaus verwendet ein Regressionsbaum numerische und nicht kategorische Werte.
Das Beispiel in Abbildung 10 zeigt in jeder Ebene eine Frage anhand eines bestimmten Features, wie zum Beispiel das Alter oder das Geschlecht. In diesem Fall handelt es sich nur um einen binären Klassifizierer, es ist aber auch möglich, dass an jedem Blatt eine eigene Klasse hängt.

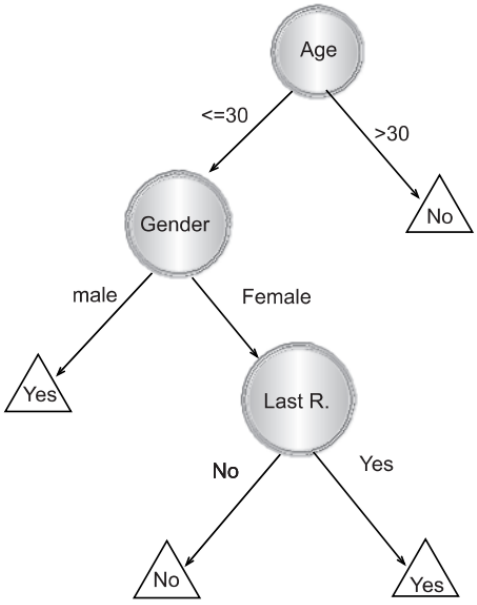


Abbildung 10: Klassifizierung mittels Entscheidungsbaum [M. Oded, 2014]

In [de Ville and Neville, 2013] ist lernen als anpassen von Entscheidungsgrenzen definiert. Wie diese Entscheidungsgrenze anzupassen ist, kann bei einer Top down Induktion of Decision Trees (TDIDT) durch Anpassen der einzelnen Knoten bestimmt werden. Dies ist im Beispiel in Abbildung 10 unter anderem die Binärentscheidung des Alters. Der Algorithmus wird von [Barros et al., 2015] als zwei Schritte-Rekursion definiert. Im ersten Schritt wird geprüft, ob die zu testenden Samples alle der gleichen Klasse angehören. Ist dies der Fall, kann direkt als Ergebnis dort die Klasse zurückgegeben werden. Trifft ersteres nicht zu, muss in einem zweiten Schritt ein Teilungskriterium gefunden werden, um das Set in kleinere Subsets zu unterteilen. In diesem Fall wird für jedes Ergebnis ein eigener Unterknoten erstellt und für jeden Pfad die Berechnung rekursiv weitergeführt, bis Schritt 2 nicht mehr ausgeführt werden muss. Als Hauptteilungskriterium gibt [M. Oded, 2014] die Entropie an. Hierdurch wird in jeder Ebene anhand des Features mit dem höchsten Informationsgehalt geteilt, wie es die Algorithmen ID3 und C4.5 machen.

$$\underset{a_j \leq a_j^R, j=1,\ldots,M}{\operatorname{argmax}} \left( \frac{information\_gain(a_i, S)}{entropy(a_i, S)} \right) \tag{5}$$

[Kozak, 2018] beschreiben das Teilungskriterium vom C4.5 Algorithum in Formel 5, wobei $a_i$ der aktuelle Wert des Attributes ist und $S$ das Datenset beschreibt. Darüber hinaus

ist es wichtig, die Höhe des Baumes einzuschränken, um einem Overfitting entgegenzuwirken. Dies kann einerseits durch das error-based pruning, wie es im C4.5 Algorithmus zum Einsatz kommt oder eine Early-Stopping Bedingung, wie zum Beispiel die maximale Baumtiefe, sein. Ein weiterer Alogrithmus ist, wie bei [Cloete and Zurada, 2000] beschrieben, CART. Dieser Algorithmus produzuiert einen reinen Binärbaum [3] und verwendet zum Beispiel Least-Square Regression zur Fehlerminimierung. Die beiden Algorithmen CART und C4.5 besitzen Pruning-Technologien, wie zum Beispiel Minimal Error Complexity Pruning, um den resultierenden Baum zu vereinfachen und redundante Pfade zusammenzuführen.

## 4.3. Datenbanken

Als Datenspeicher werden heutzutage zwei verschiedene Hauptarchitekturen eingesetzt. Auf der einen Seite werden Daten in relationalen Datenbanken in verschiedenen Tabellen mit fixen Abhängigkeiten gespeichert. Hierdurch wird die Struktur der Daten sehr strikt vorgegeben und eine Integrität ständig sichergestellt. Auf der anderen Seite werden in einer NoSQL Datenbank verschiedene Arten von Daten gespeichert, wie zum Beispiel Dokumente ohne fixe Relationen und Schema.

### 4.3.1. Relationale Datenbanken

Bei einer relationalen Datenbank werden laut [Unterstein and Matthiessen, 2012] alle Werte in Relationen dargestellt und Operationen für das Selektieren, Verbinden und Projezieren der Daten bereitgestellt.

| **Product_ID** | **Supplier_ID** | **Product_ArticleNr** | **Product_Stock** |
|---|---|---|---|
| 1 | 1 | PSF-001F | 10 |
| 2 | 1 | PSF-012B | 4 |
| 3 | 2 | HOV-001A | 99 |
| 4 | 2 | HOV-002A | 12 |

Tabelle 1: Auszug Produkt Tabelle

In Tabelle 1 ist ein typischer Auszug aus einer relationalen Datenbank zu sehen. [Sauer, 2002] beschreibt dabei die wichtigsten Objekte eines relationalen Modells mit dem Primärschlüssel, welcher in dem Beispiel die Product_ID ist. Darüber hinaus stellen

[3] Ein Binärbaum ist ein Baum, bei dem jeder Knoten exakt zwei oder keine ausgehende Kanten besitzt.

Fremdschlüssel (Foreign Keys) Referenzen auf Primärschlüssel anderer Tabellen oder der selben Tabelle bei einem Zirkelbezug dar. Die Wertebereiche, sogenannte Domänen, benennt [Sauer, 2002] mit Zahlen, Texten bis hin zu einzelnen Zeichen oder booleschen Werten. Des Weiteren existieren noch komplexe Domänen wie zum Beispiel Töne, Bilder oder das Datum. Für diese Werte existieren Repräsentationen und diese können oft nicht direkt von der Datenbank dargestellt werden.
Wie bereits in Tabelle 1 aufgezeigt, werden verschiedene Tabellen über Referenzen und den Fremdschlüsseln miteinander verknüpft. Eine relationale Datenbank muss zu jederzeit sicherstellen, dass jede Referenz wirklich existiert, dies stellt nach [Unterstein and Matthiessen, 2012] die referenzielle Integrität sicher. Hierbei werden beim Einfügen, Ändern und Löschen von Daten die Referenzen überprüft und gegebenenfalls die Operation unterbunden.
Damit dies sichergestellt ist, erfüllen relationale Datenbanksysteme strikt das ACID-Prinzip, was das Akronym für Atomicity, Consistency, Isolation und Durability ist. Die zentrale Bedeutung ist, dass jede Datenbank stets in einem donsistenten Zustand ist, jede Operation isoliert von den anderen durchgeführt wird und nach dem Abschluss die Änderung dauerhaft erhalten bleibt. Dies wird durch Transaktionen sichergestellt, welche nur ganz oder gar nicht abgeschlossen werden und somit, wenn die Datenbank vorher in einem konsistenten Zustand war, diese auch nach der Ausführung wieder konsistent ist. [Meier, 2017]
Relationale Datenbanken werden meist über die Structured Query Language, kurz SQL, gesteuert. Die Operation für das Erstellen neuer Tabellen ist in Code-Block 1 dargestellt. In diesem Zug müssen auch die Referenzen und Schlüssel, wie jener auf die Lieferantin beziehungsweise den Lieferanten (Supplier), angegeben werden.

```
1 CREATE TABLE Product (
2   Product_ID INT AUTO_INCREMENT PRIMARY KEY,
3   Supplier_ID INT NOT NULL,
4   Procuct_ArticleNr VARCHAR(255),
5   Product_Stock INT,
6   FOREIGN KEY (Supplier_ID) REFERENCES Supplier (Supplier_ID)
7 )
```

Code 1: SQL Create [Oracle, 2019b]

In weiterer Folge können über *INSERT*, wie in Code-Block 2 ersichtlich, Zeilen in die Tabelle aufgenommen werden. Die Product_ID muss in diesem Statement nicht angegeben werden, da vorab bei der Definition der Tabelle in Code-Block 1 für diese

Spalte ein $AUTO\_INCREMENT$ angegeben wurde. Dies führt dazu, dass die Spalte automatisch auf den nächst höheren Wert gesetzt wird, wenn sie $NULL$, also undefiniert, ist. $INTEGER$ Datenfelder sollen ohne Anführungszeichen eingegeben werden, wohingegen Datenfelder mit den Datentypen $VARCHAR$ oder $DATE$ unbedingt unter Anführungszeichen gestellt werden müssen.

```
1 INSERT INTO Product (Supplier_ID, Procuct_ArticleNr,Product_Stock)
2 VALUES (1, 'PSF-005X',10)
```

Code 2: SQL Insert [Oracle, 2019b]

Die einzelnen Zeilen können nun mittels $SELECT$, wie in Code-Block 3 veranschaulicht, abgefragt werden. Dabei werden alle Spalten durch die Angabe von $*$ aus der Datenbank zurückgegeben. Die Abfrage fügt über den $JOIN$ die zwei Relationen Produkt und Supplier zusammen und es sind in nur einer Abfrage bereits alle Informationen aus beiden Tabellen vorhanden. Die Ausgabe sieht dann wie in Tabelle 1 aus, nur um die Felder der Lieferantin beziehungsweise des Lieferanten erweitert. Der große Vorteil des gemeinsamen Ladens ist, dass die Daten gemeinsam in einer Abfrage verwendet werden können. Ein Nachteil ist jedoch, dass eine temporäre Tabelle mit der absoluten Breite über alle verbundenen Tabellen von der Datenbank erstellt werden muss und dies unter Umständen sehr ressourcenintensiv ist.

```
1 SELECT * FROM Product
2 JOIN Supplier ON Product.Supplier_ID = Supplier.Supplier_ID
3 WHERE Stock > 1
4 ORDER BY Stock DESC
5 LIMIT 10;
```

Code 3: SQL Select [Oracle, 2019b]

Darüber hinaus können mittels SQL die Ergebnisse weiter eingeschränkt beziehungsweise gefiltert und sortiert werden. In der $WHERE$ Bedingung wird ein Filter definiert, welcher direkt nach dem Verbinden der beiden Tabellen angewandt wird. Durch das $ORDER\ BY$ können komplexe Sortierungen umgesetzt werden und über $LIMIT$ ein Maximum an Zeilen beziehungsweise auch ein offset definiert werden. In weiterer Folge ist es auch möglich über eine Unterabfrage direkt in der übergeordneten Abfrage Daten für die Einschränkung oder die Ausgabe nachzuladen.

```
1 UPDATE Product SET Product_ArticleNr = 'PSF-001Z'
```

```
2 WHERE Product_ID = 1;
```

Code 4: SQL Update [Oracle, 2019b]

Der $UPDATE$ Befehl verwendet eine dem $SELECT$ sehr ähnliche Struktur. Es werden die selben Einschränkungen und Filter bereitgestellt, um die Daten zu ändern. Im Beispiel aus Code-Block 4 wird für alle Zeilen, bei denen die Bedingung in der Zeile zwei zutrifft, in der Produkttabelle die Artikelnummer geändert. Das $DELETE$ wird auch sehr ähnlich definiert und nur das $SET$ weggelassen.
Mysql ist eine der führenden Open Source Datenbanken für skalierbare Webanwendungen. Durch die Transaktionssicherheit und die vollständige Implementierung der ACID-Prinzipien ist sie sehr gut für unternehmenskritische Anwendungen geeignet. Dies ist dadurch möglich, dass im Hintergrund ein Bin-Log File geschrieben wird. Diese Datei wird außerdem auf alle Slave-Nodes verteilt und die darin enthaltenen Operationen nachgezogen. Des Weiteren kann über einen Cluster die Hochverfügbarkeit weiter gesteigert und der Single Point of Failure eliminiert werden. Die Datenbank selbst ist unabhängig von der darunterliegenden Architektur und läuft auf Windows, Linux oder Mac OSX. [Oracle, 2019a]

### 4.3.2. NoSQL

Neben relationalen Datenbanken haben sich dokumentenorientierte Datenbanken etabliert. Diese Datenbanken haben oft kein fixes Schema, was Erweiterungen vereinfacht, jedoch zur Laufzeit gewisse Überprüfungen erfordert. Hierdurch ist die vertikale Homogenität aufgehoben und es ist erlaubt, dass die Strukturen der Datensätze unterschiedlich sind. [Schildgen, 2016]

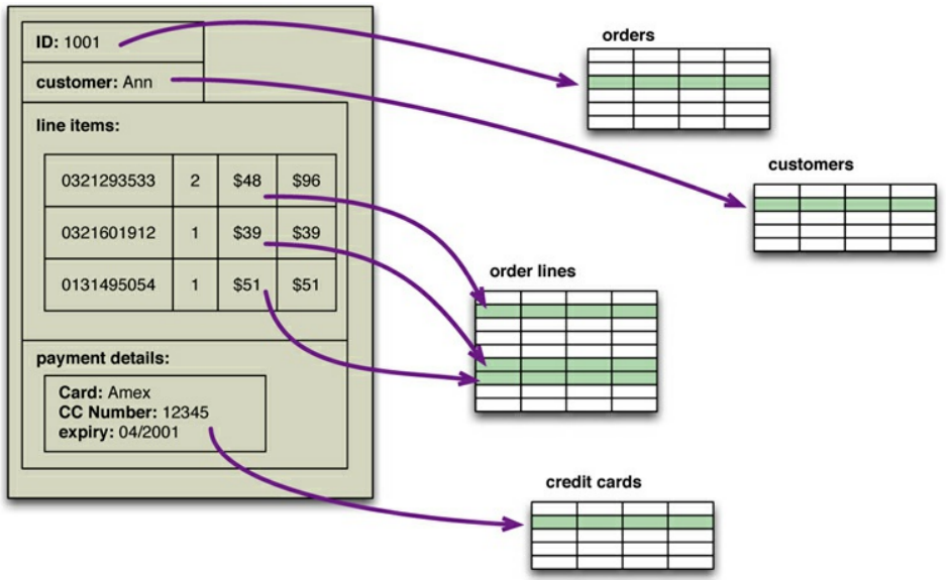


Abbildung 11: Dokument vs. relationale Datenbank, [Sadalage and Fowler, 2012]

In Abbildung 11 ist veranschaulicht, wie sich ein relationales Schema in einem Dokument vereinen lässt. Hierzu werden alle zu einer Bestellung gehörenden Daten direkt im Objekt der Bestellung gespeichert und es können in einer Abfrage, ohne betreffenden Relationen zu verbinden, alle Informationen schnell abgegriffen werden.
Laut [Vaish, 2013] beschrieb das Wort NoSQL in den Anfängen der Bewegung eine Entwicklung in Richtung Datenbanken, die gänzlich auf SQL verzichteten. Dies war dem Umstand geschuldet, dass Big Data Systeme mit reinen relationalen Datenbanken schnell an die Grenzen der vier V's (Volume, Velocity, Variety und Veracity) gestoßen sind. Mittlerweile steht der Begriff für "Not only SQL" und die Datenbanken unterscheiden sich oft nur durch den Wegfall der Relationen.
Unterteilt werden die Datenbanken in drei Kategorien. Eine Key-Value Datenbank ist laut [Schildgen, 2016] eine Tabelle mit nur zwei Spalten, wobei die erste immer der Schlüssel ist und die zweite immer einen Wert darstellt. Eine spezielle Form dieser Key-Value Datenbanken ist die Dokumentendatenbank. Hier werden Dokumente im JSON (Javascript Object Notation) in Spalte zwei gespeichert. Wide-Column-Stores verwendet zum Speichern der Daten einzelne Tabellen mit Zeilen und Spalten. Jedoch wird erst beim Einfügen einer Zeile die Anzahl der Spalten und deren Datentyp definiert. Darüber hinaus ist die Graphendatenbank durch die Beziehungen zwischen den einzelnen Elementen definiert.[Sadalage and Fowler, 2012]
Für das Suchen und Bearbeiten von Ergebnissen hat sich auf der einen Seite das Map/Re-

duce Prinzip und auf der anderen Seite die Technologie der Streams durchgesetzt. Beide Ansätze erlauben es, viele Aufgaben parallel in verteilten Datenbanken durchzuführen. In der Map Phase werden laut [Dean and Ghemawat, 2004] Key/Value Pärchen verarbeitet und ein neues Key/Value Set erzeugt. Dieses Set wird anschließend in der Reduce Phase mit dem selben Schlüssel unter der Zuhilfenahme von mathematischen Operationen zusammengefügt. Diese Funktionen werden oft in Java geschrieben und von der Datenbank direkt verarbeitet.

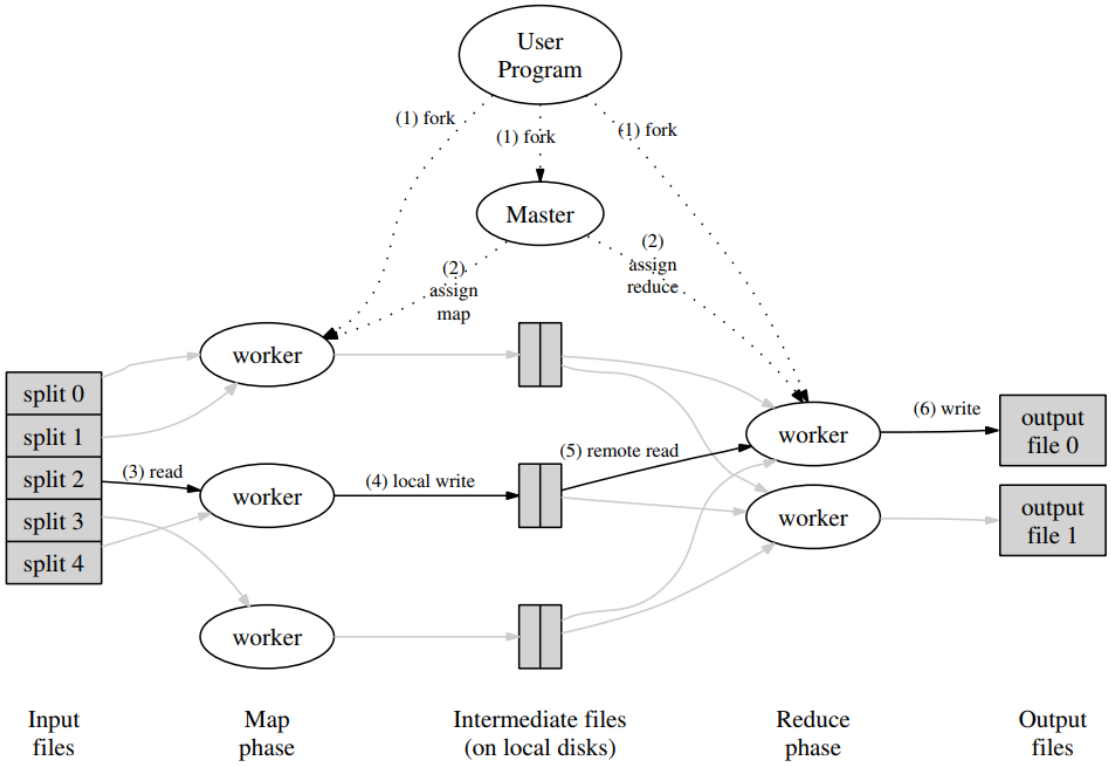


Abbildung 12: Map/Reduce Ablauf [Dean and Ghemawat, 2004]

In der Abbildung 12 wird der Ablauf und der Zugriff auf das Speichersystem in einer verteilten Umgebung dargestellt. Hierbei werden die verschiedenen Input-Dateien auf die einzelnen Worker für die Map Phase aufgeteilt und anschließend die berechneten Zwischendateien an weitere Worker für die Reduce Phase übergeben. Eine weit verbreitete Datenbank in diesem Bereich ist Apache Solr, welche häufig als Suchserver eingesetzt wird. Besondere Features sind hier im Datenprocessing angesiedelt, wie zum Beispiel der erweiterte Filter mit dem Phonetic Matching. Solr hat darüber hinaus ein Interface für die Structured Query Language (SQL), über welche einige Streaming Operationen

durchgeführt werden können. Darüber hinaus kann die Datenbank auch mit einem Schema betrieben werden, was die Datenintegrität verbessert. [Apache, 2019b]

```
curl http://localhost:8983/solr/techproducts/select?q=id:SP2514N&fl=id+
    name
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <responseHeader><status>0</status><QTime>2</QTime></responseHeader>
  <result numFound="1" start="0">
    <doc>
      <str name="id">SP2514N</str>
      <str name="name">Samsung SpinPoint P120 SP2514N</str>
    </doc>
  </result>
</response>
```

Code 5: Solr Abfrage [Apache, 2019a]

Das Beispiel in Code-Block 5 zeigt die Abfrage und Einschränkung von einem Datensatz mittels curl. Es werden hier über den Parameter $q$ die Filter und über den Parameter $fl$ die Rückgabefelder definiert. Das Konzept ist sehr ähnlich zu den $SELECT$, beziehungsweise $WHERE$ Bedingungen, in SQL. Als Suchkriterium kann auch ein Wildcard Zeichen wie ? oder $*$ und darüber hinaus auch ein bestimmter Wertebereich angegeben werden.
NoSQL Datenbanken werden laut [Sadalage and Fowler, 2012] oft in verteilten Systemen eingesetzt. Sharding ist dabei eine Technologie, die es ermöglicht verschiedene Partitionen von Daten auf unterschiedlichen Servern zu halten, um die Last der Anfragen annähernd homogen zu verteilen. Hierzu werden die Datensätze, welche oft zusammen abgefragt werden, auf einem Server nahe der physischen Position des Abfragenden gespeichert. Die Bestimmung der Shards kann, wie bei [Schildgen, 2016] beschrieben, anhand eines fixen Parameters, wie zum Beispiel dem Land einer Bestellung oder einer Hashfunktion über bestimmte Werte eines Feldes, durchgeführt werden.

## 4.4. Programmiersprachen

Es gibt derzeit viele objektorientierte Programmiersprachen, wie zum Beispiel Java, PHP, C++, C# und Swift, um Enterprise-Anwendungen, unter anderem auch Online-Shops, zu erstellen. All diese Sprachen werden immer ähnlicher und bieten, auch wenn

sich die Nomenklatur etwas unterscheidet, im Großen und Ganzen einen ähnlichen Funktionsumfang. Diese Programmiersprachen können laut [Loeckx et al., 2013] in maschinen- und problemorientierte Sprachen unterteilt werden, einige Sprachen existieren in beiden Varianten. Eine Maschinensprache wird extra für das Zielsystem in Maschinencode übersetzt und kann nur auf diesem ausgeführt werden. Bei einer problemorientierten Sprache, auch als höhere Sprache bezeichnet, wird der Sourcecode direkt vom Interpreter, welcher auf dem Zielsystem läuft, zur Laufzeit in Maschinencode umgewandelt. In diesem Kapitel werden die zwei für diese Arbeit verwendeten Programmiersprachen *PHP* und *Python* vorgestellt und deren Vor- beziehungsweise Nachteile herausgearbeitet.

### 4.4.1. PHP

Vor allem in der Webentwicklung haben sich die zwei großen Interpretersprachen Java und PHP durchgesetzt und ermöglichen Unternehmen ein flexibles Bereitstellen von Anwendungen. Beides sind Interpretersprachen, das heißt sie werden zur Laufzeit in Maschinencode umgewandelt. Dies wird aber durch das Zwischenspeichern von Opcode, wie der Maschinencode auch genannt wird, beschleunigt und somit wird der Maschinencode nur noch bei Änderungen umgewandelt. PHP ist hauptsächlich für den Einsatz auf Servern konzipiert und wird durch Apache oder Nginx Module nach außen zugänglich gemacht. [Group, 2019]
Auch die Anbindung an die in Abschnitt 4.3 beschriebenen Datenbanken funktioniert reibungslos. Vor allem Solr ist hier aber durch die REST-Schnittstelle sehr flexibel und kann problemlos in nahezu jedes System integriert werden. Andere Datenbanken, wie zum Beispiel Mysql, brauchen für die Integration in PHP ein eigenes Modul, welches bei der Installation angegeben beziehungsweise extra aktiviert werden muss.

```
1  <?php
2  class Example extends Calculator {
3    private $member = 0;
4
5    public __construct(){
6      $this->member = 1;
7    }
8
9    protected function calculate(){
10     $this->member = 2;
```

```
11    }
12
13    public function getMember(){
14      return $this->member;
15    }
16  }
```

Code 6: PHP Objekt Beispiel

Das Beispiel in Code-Block 6 veranschaulicht die Vererbung und Encapsulation, welche in PHP implementiert sind. Die Membervariable *member* darf nur von der aktuellen Klasse geändert werden, wohingegen die *calculate* Methode von jeder abgeleiteten Instanz von Example aufgerufen werden darf. PHP bietet darüber hinaus noch sogenannte magic methods, eine davon ist *__construct*, welche immer beim Erzeugen von einer Instanz aufgerufen wird. Public definiert eine frei zugängliche Methode oder einen Member welche wie im Beispiel von Code-Block 7 aufgerufen werden können.

```
1  $example = new Example();
2  print($example->getMember());
```

Code 7: PHP Objekt erzeugen

PHP bietet darüber hinaus eine Unterstützung für alle gängigen Betriebssysteme, was das Entwickeln und Ausführen von Source Code, wie bei Java Anwendungen, recht einfach gestaltet. All diese Vorteile sprechen für die Entwicklung von webbasierten Anwendungen mittels *PHP* in Kombination mit einem Webserver.
Des Weiteren unterstützt PHP Multithreading über die pthreads Erweiterung. Hierfür muss PHP neu kompiliert werden und die Zend Thread Safety vorab eingeschaltet werden. [Tyler, 2017]

### 4.4.2. Python

Im Bereich der Datenverarbeitung und -analyse haben sich jedoch die Sprachen Python sowie R durchgesetzt und eine breite Community entwickelt eine Vielzahl an Erweiterungen und Algorithmen. R wurde speziell für statistische Aufgaben entwickelt und wird entweder mittels einer Kommandozeilenumgebung oder der Entwicklungsumgebung RStudio ausgeführt. Python hingegen besitzt keine eigene Entwicklungsumgebung und wird generell über die Konsole aufgerufen. Der Interpreter, die sogenannte Python Virtual Machine, ist aber auch Bestandteil von vielen Entwicklungsumgebungen, wie zum Beispiel PyCharm oder Eclipse. Beide Sprachen orientieren sich eher an einfachen

funktionalen Scripts und es wird, ausgenommen bei der Implementierung von Library Modulen, selten auf die objektorientierten Möglichkeiten der Sprachen zurückgegriffen. Die Erweiterungen ermöglichen ein rasches Arbeiten und bieten meist für die jeweilige Programmiersprache optimierte Versionen der Algorithmen an. Gerade Python hat sich in den letzten Jahren zur ersten Wahl als Basis für die Datenanalyse und zur Entwicklung schneller Prototypen entwickelt. [Lutz et al., 2007]

```
1  class Example:
2    member = 0
3    def __init__(self):
4      self.member = 1
5
6    def calculate(self):
7      self.member = 2
8
9    def getMember(self):
10     return self.member
11
12 example = Example()
13 example.calculate()
14 print example.getMember()
```

Code 8: Python Beispiel

Besonders hervorzuheben ist hierbei nach [Theis, 2011] die einfache Syntax, welche sich eher an die natürliche Sprache anlehnt und eine klare Code-Struktur vorgibt. Dies ist besonders bei den Einrückungen wichtig, da keine Klammern notwendig sind und somit nur die aktuelle Ebene einen zusammenhängenden Code-Block definiert. Eine Encapsulation wie bei PHP gibt es in der Implementierung von den Objekten in Python nicht. Hierdurch ist das Verändern der gesamten Objekte zu jeder Zeit von außen möglich. Beim Erzeugen einer Instanz wird hier stets die *__init__* Funktion aufgerufen und kann somit als Konstruktor vom Objekt gesehen werden. Der Zugriff, auf die dem Objekt zugehörigen Member, geschieht über das Keyword self, welches bei jeder Methode des Objekts als erster Parameter übergeben werden muss. Python stellt darüber hinaus laut [Lutz et al., 2007] viele unterschiedliche Datentypen, wie zum Beispiel Zahlen, Strings, Listen, Dateien oder Mengen zur Verfügung und es ist auch erlaubt einige Typen zu mischen, ohne explizit eine Konvertierung durchzuführen. Es wird zum Beispiel ein Integer bei der Addition mit einem Float automatisch zu einem Fließkomma konvertiert und

das Ergebnis mit Nachkommastelle zurückgegeben.
Die Sprache selbst wird zur Ausführungszeit wie PHP und Java interpretiert und nicht vorab kompiliert. Dies ermöglicht eine weitgehende Unabhängigkeit vom Betriebssystem und den Einsatz als Komandozeilenprogramme oder mit grafischen Benutzeroberflächen.

# 5. Entscheidungsmodell

In diesem Kapitel wird das Entscheidungsmodell und dessen Bestandteile entworfen und die Implementierung aufgezeigt. Darüber hinaus sollen Einsatzgebiete genannt und mögliche Erweiterungen skizziert werden. In erster Linie wird das Modell durch eine Baumstruktur beschrieben. Es besteht aus unendlich vielen Ebenen, welche mehrere Entscheidungsfragen von unterschiedlichen Typen enthalten können.

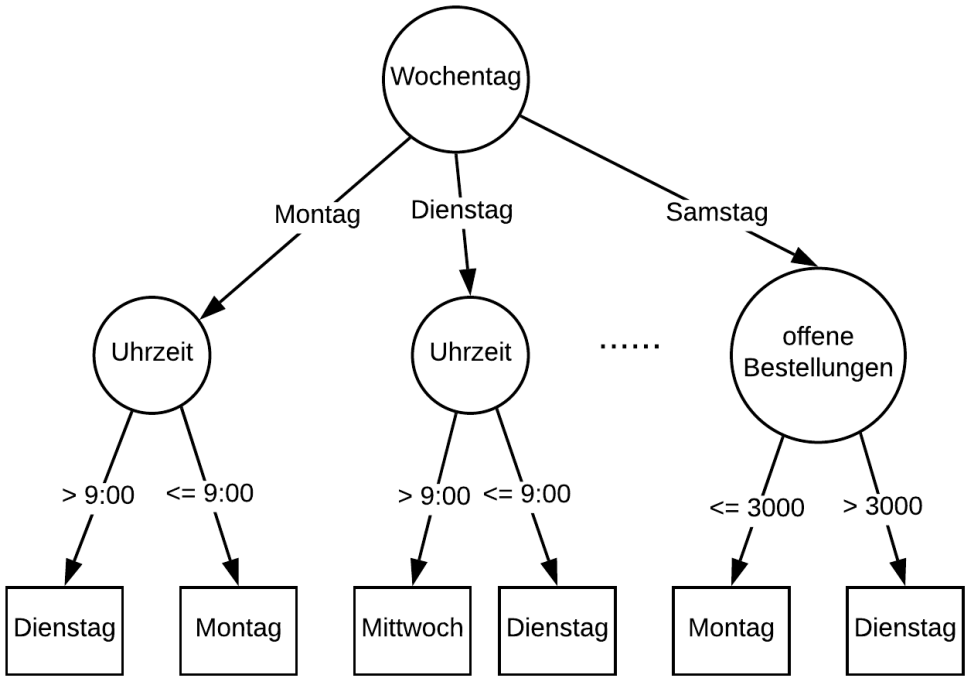


Abbildung 13: Entscheidungsmodell, eigene Abbildung

Das Beispiel in Abbildung 13 veranschaulicht einen vereinfachten Entscheidungbaum mit drei Ebenen für die Bestimmung, ob ein Paket am selben Tag noch versendet werden kann. Vernachlässigt wurde in diesem Beispiel die Berücksichtigung von Zahlungsarten und Lieferdiensten. Bestellt eine Kundin/ein Kunde am Montagvormittag vor 9 Uhr, geht das System davon aus, dass das Paket am gleichen Tag noch versendet wird. Wird die Bestellung jedoch am Samstag durchgeführt, ist von Interesse, ob nicht bereits mehr als 3000 offene Bestellungen im System sind, welche am Montag versendet werden müssen. Die Knoten stellen somit die einzelnen Fragen dar und die Antworten liegen in den Blättern des Baumes.

## 5.1. Ablauf

Für die Ausführung wird, wie bei [Gibbons, 1985] beschrieben, ein depth first Algorithmus implementiert, welcher im Gegensatz zu einem breath first zuerst alle Fragen eines Zweiges betrachtet. Im Prinzip arbeitete jeder Entscheidungsbaum nach diesem Prinzip, jedoch unterscheidet sich dieser Baum durch die manuellen Fragen von einem reinen Entscheidungsbaum. Hierdurch wäre es möglich, aber nicht sinnvoll, diese manuellen Fragen in einem breath first Algorithmus abzuarbeiten.[4]
Im ersten Schritt müssen der Baum und seine Parameter initialisiert werden. Dies kann entweder direkt durch das Anlernen der einzelnen Ebenen geschehen oder durch das Setzen von einem vorab berechneten Modell. Wird das Modell neu berechnet, muss zuerst eine Liste von Parameter-Objekten gesetzt und automatisch alle manuellen Fragen initialisiert werden. Der Algorithmus wählt als nächsten Schritt jenes Feature mit den meisten unterschiedlichen Schlüssel-/Ergebnis-Kombinationen. In weiterer Folge wird versucht, die einzelnen Schlüssel mit übereinstimmenden Ergebnissen zu gemeinsamen Pfaden zusammenzuführen. Anhand dieser Pfade wird im nächsten Schritt die Parameterliste verkleinert und eine neue Ebene mit der Auswahl des besten Features begonnen. Dieser Vorgang wird solange fortgesetzt, bis bei der Auswahl des Features eine leere Liste zurückgegeben wird oder bis beim Zusammenführen aufgrund von Heuristiken entschieden wird, dass ein Ergebnis zurückgeliefert wird.
Im Anschluss wird das Modell in der Klassenstruktur initialisiert und es kann eine Abfrage durchgeführt werden. Muss nun ein Ergebnis abgefragt werden, kann dies einfach über die Abfrage in Code-Block 9 geschehen. Wichtig ist hierbei, nicht auf den Aufruf von *calculate*() zu vergessen, da ansonsten kein Ergebnis berechnet wird. Dadurch wird auch sichergestellt, dass alle weiteren Ebenen des Entscheidungsbaumes, welche mit *on* definiert wurden, ausgeführt werden und schlussendlich das Ergebnis zurückliefern.

```
1 $test->setParameter($p)->calculate()->getResult();
```

Code 9: Entscheidungsbaum-Abfrage

Hinter der *calculate*() Methode versteckt sich eine Magic-Method von PHP, welche zuerst die *process*() Methode in der jeweiligen Frage aufruft. Diese Methode muss bei manuellen Fragen selbst implementiert werden, bei Entscheidungsfragen liefert diese jedoch immer das aktuelle Parameter-Objekt als Ergebnis zurück. Anschließend führt *calculate*() automatisch *processCompare*() aus, wodurch alle Evaluierungen, welche mit *getCompare*()

[4]Die parallele Ausführung ist nicht sinnvoll, da in den meisten Fällen, weitere darunterliegende Ebenen betrachtet werden müssen, um zu einem Ergebnis zu kommen.

oder in weiterer Folge mittels *on*() hinzugefügt wurden, ausgeführt werden.

## 5.2. Parameter

Das Parameter-Objekt bildet die Grundlage für jede Frage und ermöglicht ein dynamisches Erstellen von Entscheidungen. In jeder Frage können Platzhalter definiert werden, welche die Daten aus dem Parameter-Objekt laden und einsetzen. Das Objekt selbst ist ein einfacher Key-Value Store[5] und steht allen Ebenen der Entscheidungen zur Verfügung.

| **Tag** | **Lieferdienst** | **Zahlungsart** | **List<Items>** | **Ergebnis** |
|---|---|---|---|---|
| Montag | Post | Vorauskasse | [...] | 4 |
| Montag | Selbstabholung | Paypal | [...] | 0 |
| Sonntag | DHL | Paypal | [...] | 1 |
| Montag | UPS | Kreditkarte | [...] | 1 |

Tabelle 2: Parameter-Objekte

In Tabelle 2 ist die erste Ebene für mehrere Parameter-Objekte abgebildet. Jede Zeile stellt in diesem Zusammenhang eine konkrete Entscheidungsabfrage an das System dar. Objekte können verschachtelt sein, was in dem Beispiel über die Item-Liste aus Tabelle 3 skizziert wird. Das Ergebnis in der letzten Spalte dient zum Anlernen des Baumes und wird nur in der Lernphase evaluiert.

| **Produkt** | **Bestellt** | **Typ** |
|---|---|---|
| PSF-001A | 1 | Physisch |
| PSF-008Z | 1 | Physisch |
| PSF-005R | 2 | Physisch |
| PSF-001A | 1 | Physisch |

Tabelle 3: Parameter-Items

Diese Daten können dynamisch in manuelle Fragen eingesetzt werden, um zum Beispiel folgende Frage zu bilden: Wann ist die Bestellung vom Tag, per Zahlungsart bezahlt,

[5]Ein Key-Value-Store ist eine einefache Form einer NoSQL Datenbank und hält für jeden Schlüssel exakt einen Wert.

lieferbar. Dies ist in Formel 6 ersichtlich und hat den Vorteil, dass nicht für jede Kombination eine eigene Frage programmiert werden muss.

$$x = F(Tag,\ Zahlungsart) \tag{6}$$

Die Frage selbst muss überprüfen, ob die Parameter im Objekt richtig gesetzt sind und im Fehlerfall entscheiden, ob die Frage *undefiniert* als Ergebnis liefert oder der gesamte Baum einen Fehler aufweist. Im ersten Fall wird das Fehlen des Parameters durch andere Zweige des Baums kaschiert und das Ergebnis womöglich verfälscht. Ein Beispiel hierfür ist, wenn im Checkout kein Geburtsdatum eingegeben wird, die Betrugserkennung dies aber verwendet, um eine Risikogruppe anhand des Alters zu berechnen. Im zweiten Fall wird von einem Fehler bei der Implementierung der Knoten ausgegangen, dies kann bei der Wiederverwendung der Fragen in einem anderen Baum geschehen.

## 5.3. Entscheidungsfrage

Eine Entscheidungsfrage ist ein Knoten im Baum und liefert entweder sofort das Ergebnis zurück oder verweist auf die nächste Frage im Baum. Wie in Code-Block 10 ersichtlich, wird mittels *on* auf das Ergebnis der jeweiligen Entscheidungsfrage reagiert und entweder eine weitere Frage hinzugefügt oder das Ergebnis zurückgegeben.

### 5.3.1. Entscheidungsbaum

Der Entscheidungsbaum stellt eine vertikale Verkettung von Entscheidungsfragen dar und wird in der Berechnungsphase definiert oder mittels vorab entwickelten Fragen manuell aufgebaut, wie in Abbildung 13 bereits veranschaulicht wurde. Die Klasse des Entscheidungsbaumes wird im Prinzip durch dessen Root-Frage definiert, was im Beispiel vom Code-Block 10 die Abfrage des Wochentags ist. Hierdurch ist es möglich aus Entscheidungsbäumen kleinere partielle Entscheidungsbäume herauszulösen und getrennt zu betrachten.

```
1 public function init(){
2   $this->on(['Samstag', 'Sonntag'], (new OpenOrders())
3     ->on('<= 3000', "Montag")
4     ->on('> 3000', "Dienstag")
5   );
6   $this->default((new CutoffTime())
7     ->on('<= 9', function($p){return $p->getOrderDate();})
```

```
8       ->on('>9', function($p){return
9           $p->getOrderDate()->getNextWorkday();}
10      )
11    );
12    return parent::init();
13  }
14
15  public function process(){
16    return $this->getParameter()->getOrderDate()->getWeekday();
17  }
```

Code 10: Defintion Entscheidungfrage

In der Implementierung wurde die Vereinfachung mit dem *Default* Pfad eingeführt, was gerade den Konfigurationsaufwand minimiert, wenn nur eine Ausnahme existiert. Des Weiteren können in die Ergebnisberechnung auch Werte aus dem Parameter-Objekt einfließen. Hierzu muss nur eine *callable* Funktion mit einem Parameter als Rückgabewert, auf das zutreffen der Bedingung, definiert werden. Die Bedingung kann wiederum selbst eine Funktion oder ein Text sein, der Text kann mathematische Vergleichsoperatoren enthalten, welche evaluiert und in die entsprechende Funktion umgewandelt werden.

### 5.3.2. Berechnung

Soll der Entscheidungsbaum für den Versandtag automatisch berechnet werden, darf die *process* Methode nicht überschrieben werden. Darüber hinaus muss vorab über die Trainingsphase das Modell ähnlich zu jenem in Code-Block 10 aufgebaut und zwischengespeichert werden. Sollen bestimmte Entscheidungsfragen berücksichtigt werden, ist es notwendig diese über *use* bei der Initialisierung anzugeben.

```
1  public function learn($arrParamter)
2  {
3    $this->use(new IsProductInStock());
4    parent::learn($arrParamter);
5  }
```

Code 11: Versandtag automatisch berechnen

In erster Linie wird bei der Berechnung versucht anhand der definierten Parameter den Baum so zu entwerfen, dass in jeder Ebene eine Ergebnis-Klasse für einen Parameter gefunden wird mit der maximalen Anzahl an Treffern. Dies hat den Vorteil, dass für einen

Großteil der Anfragen nur wenige Ebenen des Baumes durchlaufen werden müssen. Wie in Abschnitt 4.2.3 beschrieben, wird über das Teilungskriterium der Baum aufgebaut. Die Berechnung des Teilungskriteriums hängt stark vom Datentyp des Parameter-Attributes ab. Handelt es sich um kategorische Daten, welche in den Parameter-Objekten immer als Texte dargestellt werden, kann nur der Gleichheitsoperator angewandt werden. Die zweite Art wird durch Zahlen repräsentiert und kann entweder wie ein kategorischer Parameter behandelt oder es muss ein Teilungskriterium anhand von mathematischer Funktionen gefunden werden.

**Datengenerator**

Für die Erstellung des Entscheidungsbaumes ist es notwendig, dass in einer Lernphase, anhand vorhandener Daten mit dem dazugehörigen Ergebnis, das Modell berechnet wird. Damit ausreichend Daten zur Verfügung stehen, kann der Generator in Code-Block 12 verwendet werden.

```
1  $pArray = [];
2  for($i = 0; $i < 1000; $i++){
3    $result = false;
4    $r = rand(0,10);
5    if($r == 2)  $result = true;
6
7    $t = "product";
8    if(rand(1,8) == 4){
9      $result = true;
10     $t = "mail";
11    }
12    if($i % 10 > 6 && $r == 5) $result = true;
13    $pArray[] = (new MyParameter())
14      ->setStock($i%10)
15      ->setOrderCount($r)
16      ->setType($t)
17      ->setResult($result);
18 }
```

Code 12: Test Parameter-Generator

Dieser Generator erzeugt eine Liste von Parameter-Objekten mit vorher definierten Parametern und zu erwartenden Ergebnissen. Es wurden mathematische Operationen ver-

wendet, um eine Abhängigkeit von Eingangsparametern zu dem endgültigen Ergebnis zu simulieren. Zum Beispiel liefert jeder Parameter mit der Bestellanzahl von zwei Stück immer wahr als Ergebnis zurück. Etwas komplexer ist die Berechnung in Zeile 15, hier wird die Abhängigkeit von zwei Parametern simuliert.

**Klassifizierer**

Bei der Klassifizierung werden in Rekursionsschritten alle Labels betrachtet und es wird in einem Knoten eine Weiche auf alle Werte eines Labels implementiert. Hierdurch wird je Ebene das Parameter-Set verkleinert und solange neu berechnet, bis das Parameter-Set nur noch ein Ergebnis beinhaltet oder 90 Prozent der Parameter-Objekte das selbe Ergebnis aufweisen.

```
1  (type == product)
2    (ordercount == 9) = 0
3    (ordercount == 3) = 0
4    (ordercount == 6) = 0
5    (ordercount == 4) = 0
6    (ordercount == 1) = 0
7    (ordercount == 2) = 1
8    (ordercount == 0) = 0
9    (ordercount == 5)
10     (stock == 4) = 0
11     (stock == 9) = 1
12     (stock == 7) = 1
13     (stock == 6) = 0
14     (stock == 2) = 0
15     (stock == 5) = 0
16     (stock == 0) = 0
17     (stock == 8) = 1
18     (stock == 3) = 0
19     (stock == 1) = 0
20   (ordercount == 10) = 0
21   (ordercount == 8)  = 0
22   (ordercount == 7)  = 0
23 (type == mail) = 1
```

Code 13: Klassifizierungsbaum

In Code-Block 13 ist der Entscheidungsbaum für den Datengenerator aus Code-Block

12 zu sehen. Sehr gut ist hier zu erkennen, dass für jeden Wert ein eigener Komparator hinzugefügt wurde und die Bedingung, dass der Lagerstand größer 6 ist, wenn die Bestellanzahl fünf ist, wird durch die Komparatoren der Zeilen 11, 12 und 16 abgebildet.

**Mathematischer Teiler**

Beim mathematischen Teiler wird versucht die Daten in zwei Bereiche zu teilen, um eine Performance von $O(n*log(n))$ ähnlich einem binären Suchbaum zu erreichen. Es handelt sich dabei um einen rekursiven Algorithmus, welcher in jeder Ebene eine Bedingung findet, um einen Knoten des Baumes zu bauen.
Das Beispiel von Code-Block 13 hat sehr viele Entscheidungen und ist bei Abweichungen der Eingangsparameter zu den angelernten Parametern nicht flexibel. Das Ergebnis von Code-Block 14 zeigt ein optimiertes und mit mathematischen Hilfsmittel berechnetes Ergebnis. Hierbei wurden zwei Annahmen getroffen, wenn mehr als 60 Prozent bei der Klassifizierung zu einem Ergebnis mit unterschiedlichen Variablen gehören, wird dies im *else* Zweig zusammengefasst.

```
1 (type == mail) = 1
2 (type == product)
3   (ordercount == 0) = 0
4   (ordercount == 2) = 1
5   (ordercount == 5)
6     (stock <= 6) = 0
7     (stock > 6) = 1
8   (else) = 0
```

Code 14: Klassifizierungsbaum

Darüber hinaus werden Zahlenbereiche anhand von deren Mittelwert zwischen dem Maximum der unteren Klasse und dem Minimum der oberen Klasse, wie beim Lagerstand ersichtlich, geteilt.

### 5.3.3. Speichern

Ein berechnetes Modell muss in einer Datenbank zwischengespeichert werden. Um dies zu erreichen, müssen nur die einzelnen Ebenen und deren Entscheidungen in ein transferierbares Format gebracht werden. Als Format eignet sich JSON sehr gut, da es direkt ohne Konvertierung in Relationen in einer dokumentbasierten Datenbank wie zum Bei-

spiel CouchDB[6] gespeichert werden kann.

```
{"key":"type","equals":{
  "mail":1,
  "product":
    {"key":"use#0","equals":{
      "out of stock":
        {"key":"ordercount","equals":{"2":1},
          "default":0
        },
      "in stock":
        {"key":"ordercount","equals":{
          "0":0,
          "2":1,
          "5":
            {"key":"stock",
            "less":{"6":0},
            "greater":{"6":1}
            }
          },
        "default":0
        }
      }
    }
  }
}
```

Code 15: JSON Modell

Das Modell in Code-Block 15 kann jederzeit aus der Datenbank geladen und bei dem Entscheidungsbaum angewandt werden. In diesem Zusammenhang kann ein einfacher Dokument-Store mit einem eindeutigen Identifikator für den Entscheidungsbaum im Source-Code, wie in Abschnitt 4.3.2 beschrieben wurde, verwendet werden. Die Daten werden im JSON Format gespeichert und können direkt von dem Entscheidungsbaum beim Laden geparsed werden, ohne das Modell neu zu berechnen müssen.
In der Datenbank werden somit alle Informationen über Vergleichsoperationen, deren Werte und Ergebnisse mit allen Ebenen gespeichert. Selbst manuelle Fragen, welche im

[6]Nähere Informationen sind auf der Webseite des Projekts http://couchdb.apache.org/ zu finden.

nächsten Abschnitt betrachtet werden, können in solch ein Modell aufgenommen und deren Ergebnisse dynamisch berücksichtigt werden.

## 5.4. Manuelle Fragen

In Code-Block 11 wurde bereits das Vorhandensein von manuellen Fragen, über das Hinzufügen mittels *use*(), angedeutet. Für diese Frage wird in der Lernphase für jedes Parameter-Objekt ein Ergebnis berechnet und anschließend für das Splitting-Kriterium berücksichtigt.

```
1  (type == mail) = 1
2  (type == product)
3    (IsProductInStock == out of stock)
4      (ordercount == 2) = 1
5      (else) = 0
6    (IsProductInStock == in stock)
7      (ordercount == 0) = 0
8      (ordercount == 2) = 1
9      (ordercount == 5)
10       (stock <= 6) = 0
11       (stock > 6) = 1
12     (else) = 0
```

Code 16: Manuelle Fragen

Die unterschiedlichen Fragetypen verhalten sich etwas anders als reine Entscheidungsfragen und werden, wie zum Beispiel die vereinenden Fragen, in diesem Abschnitt beschrieben.

### 5.4.1. Vereinende Fragen

Vereinende Fragen können, anders als Entscheidungsfragen, welche einen $XOR$ Charakter aufweisen, als $AND$ Struktur als Knoten im Entscheidungsbaum angesehen werden. Hierbei werden von der Frage ausgehend immer alle unterliegenden Fragen und deren Zweige betrachtet.

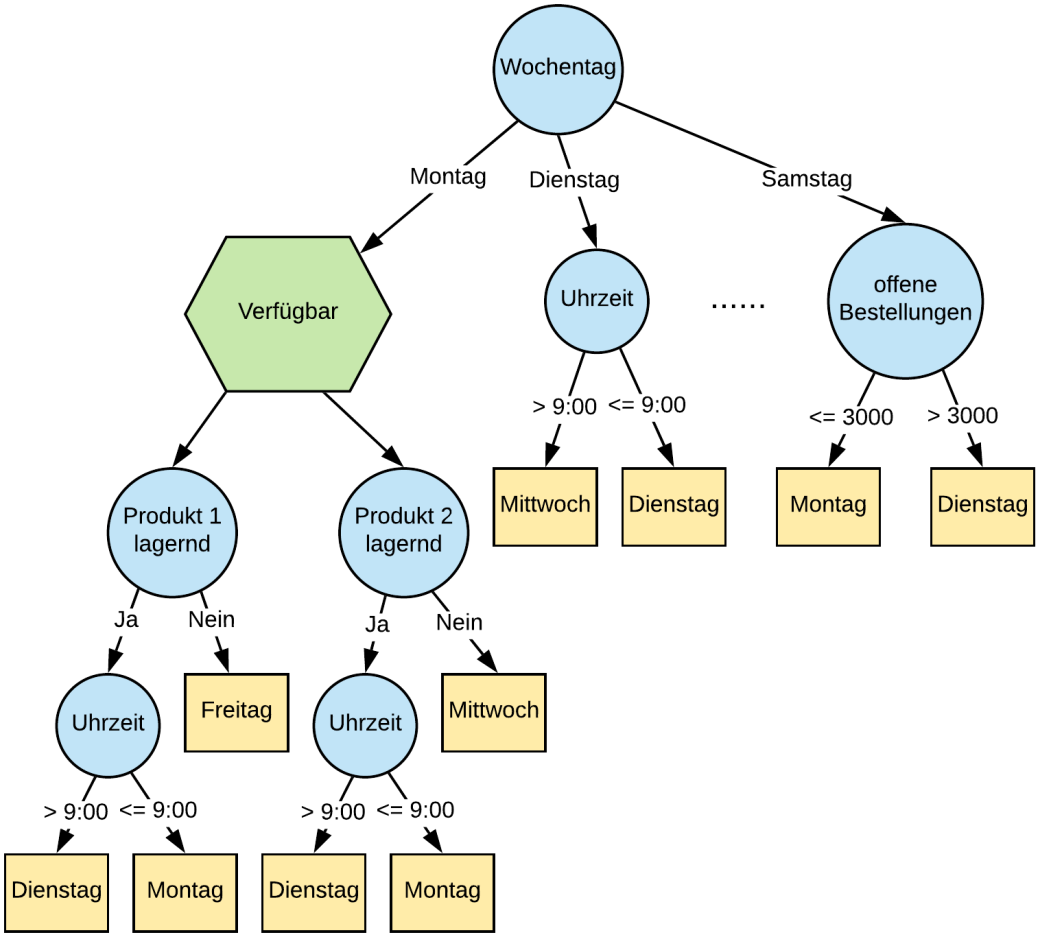


Abbildung 14: Entscheidungsbaum mit vereinender Frage, eigene Abbildung

In Abbildung 14 ist eine vereinende Frage in Grün dargestellt. Hierbei ist die Besonderheit, dass beide Produkte lagernd sein müssen, damit die Entscheidung "Montag" als Ergebnis liefern kann. In weiterer Folge können unter vereinenden Fragen wieder Entscheidungsbäume unterschiedlichster Art angehängt werden, was in dem Beispiel durch die Verschachtelung der Uhrzeit veranschaulicht wird.

**Glaubwürdigkeit**

Die Glaubwürdigkeit ist ein zentrales Element von vereinenden Fragen, da es immer möglich ist, dass die Unterfragen ein abweichendes Ergebnis liefern. Dies liegt daran, dass nur ein Teil der Fragen betrachtet wird oder grundsätzlich ein divergierendes Ergebnis in den Blättern der Sub-Fragen gefunden wird. Die Glaubwürdigkeit von einem Ergebnis

ergibt sich somit aus den vollständigen Berechnungen einer Frage oder durch das Setzen von einer unteren und oberen Schranke für das Ergebnis.

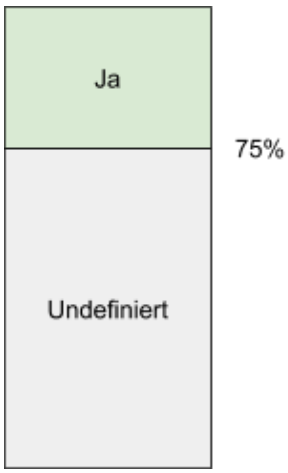


Abbildung 15: Obere Schranke, eigene Abbildung

Im einfacheren Fall kann die Schranke durch das Festlegen von einem Wert $\alpha$ für ein Ergebnis, wie in Abbildung 15 zu sehen ist, geschehen. Hierdurch ist ein Ergebnis erst dann zu werten, wenn die Wahrscheinlichkeit ($r$) die Schranke überschreitet. Erreicht keiner der gefundenen Werte diese Schranke, ist die Antwort $undefiniert$.

$$r > \alpha \tag{7}$$

$$r = \frac{n}{a} \tag{8}$$

$$r = \frac{\sum_{i=0}^{n} w_i}{\sum_{i=0}^{a} w_i} \tag{9}$$

Durch die Evaluierung der Schranken, wie in den Formeln 7 unter der Zuhilfenahme der Formeln 8 beziehungsweise 9 für die Bestimmung der Schwellenwerte der einzelnen Ergebnisse, kann die Laufzeit des Baumes signifikant verringert werden. In Formel 8 beschreibt $n$ die Anzahl der gleichen Ergebnisse einer Gruppe und $a$ die Gesamtanzahl an Ergebnissen. Hingegen wird in Formel 9 die Gewichtung $w$ der gleichen Ergebnisse einer Gruppe durch die Summe aller Gewichte angegeben. Weitere Fragen müssen nur noch berechnet werden, wenn die Schranke $\alpha$ noch nicht erreicht wurde. Wird als obere Schranke zum Beispiel 50 Prozent gewählt, kann bei fünf Fragen nach drei gleichen Ergebnissen abgebrochen und das Ergebnis kann mit einer Glaubwürdigkeit von 60 Prozent zurückgegeben werden.

$$\delta = \frac{\sum_{i=0}^{n} \delta_i}{a} \tag{10}$$

$$\delta = \frac{\sum_{i=0}^{n} w_i \delta_i}{\sum_{i=0}^{a} w_i} \tag{11}$$

Die Glaubwürdigkeit $\delta$ wird dann über die einzelnen Ebenen weitergegeben, entweder, wie in Formel 10 beschrieben, bei einer gleichwertigen Verteilung durch die Summe der Glaubwürdigkeiten der darunterliegenden Ebenen durch die Anzahl aller Ebenen oder, wie in Formel 11 ersichtlich, durch die gewichtete Summe über dieselben Ebenen. Der Parameter $a$ beschreibt hierbei die Anzahl aller Ebenen unter der betrachteten Frage. Der Aktivator $\alpha$ ist ein frei wählbarer Parameter von jeder kumulierenden Frage und definiert, ab welcher Glaubwürdigkeit die Antwort an die darüber liegende Ebene weitergegeben wird. Ist $r < \alpha$ wird $r$ auf 0 zurückgesetzt und die Frage liefert $undefiniert$ als Ergebnis zurück. Hierdurch wird sichergestellt, dass unvollständige Fragen das Ergebnis nicht verfälschen können.

**Vollständigkeit**

Die partielle Vollständigkeit $\lambda$ wird durch die betrachteten Fragen gegeben und gibt Aufschluss darüber, ob das Durchlaufen von weiteren Zweigen das Ergebnis und die Glaubwürdigkeit verbessert. Dies ist nur möglich, wenn der Entscheidungsbaum vereinende Fragen enthält und diese noch nicht vollständig evaluiert wurden. Dieser Parameter soll darüber hinaus verhindern, dass das Modell in eine Endlosschleife gerät, wenn alle Antworten unter der Schranke $\alpha$ stehen. Nur wenn $\lambda < 1$ gilt, werden noch weitere Zweige betrachtet.

$$\lambda = \frac{\sum_{i=0}^{n} \lambda_i}{a} \tag{12}$$

Um die Vollständigkeit zu berechnen muss, wie in Formel 12, die Vollständigkeit von allen bereits berechneten Fragen (n) einer vereinenden Frage durch die gesamte Anzahl der Fragen von dieser dividiert werden. In diesem Schritt wird immer nur eine Frage betrachtet und somit kann gesagt werden, dass durch das Herabsetzen der Schranke $\alpha$ die Vollständigkeit kleiner 1 sein darf. Erst wenn eine übergeordnete Frage durch das Kumulieren der Glaubwürdigkeit die Schranke nicht mehr erreicht wird, können bei nicht vollständig berechneten Fragen weitere Zweige evaluiert werden.
Wäre bei dem Beispiel aus Abbildung 14 die vereinende Frage mit $\alpha = 0.5$ gesetzt und liefert der erste Unterbaum bereits ein Ergebnis, kann auf die Berechnung des zweiten Zweiges verzichtet werden.

**Ergebnis**

Im einfachsten Fall kann das Ergebnis einer vereinenden Frage von binärer Natur sein und hierdurch kann ein Gesamtergebnis relativ einfach auf der einen Seite durch das Aufsummieren der unterliegenden Ergebnisse und dem anschließenden Dividieren mit der Anzahl der gesamten Ergebnisse von der Ebene darunter, wie in Formel 13, berechnet werden. Dies stellt eine gleichwertige Verteilung der Antworten dar.

$$x = \frac{\sum_{i=0}^{n} y_i}{n} \tag{13}$$

Auf der anderen Seite gibt es die Möglichkeit bei jeder Kante ein Gewicht zu definieren und es kann somit den Fragen eine Priorität zugewiesen werden. Die Berechnung der resultierenden Antwort ist in untenstehender Formel 14 über eine gewichtete Summe ersichtlich.

$$x = \frac{\sum_{i=0}^{n} w_i y_i}{\sum_{i=0}^{n} w_i} \tag{14}$$

Ist $x > \alpha$, wurde bereits eine ausreichende Glaubwürdigkeit erreicht und es können weitere Ausführungen übersprungen werden. In Code-Block 17 wird eine binäre Frage in Zeile 5 mit einer Gewichtung von 10 aufgerufen.

```
1  public function init(){
2    $this->setLimit(1);
3    $this->setTimeout(0.01);
4    //is_enough_in_stock
5    $this->all("Product")->addQuestion(new IsEnoughInStockQuestion())
6      ->setWeight(10);
7    //is_delivery_today
8    $this->addQuestion(new CutoffDayQuestion())
9      ->getCompare()->isEquals(date("Y-m-d"));
10   //is a capacity for that day
11   $this->addQuestion(new OpenPackagesQuestion())
12     ->getCompare()->is(function($r){
13       return ($this->getParameter()->getDeliveryDate()->getWorkdays()
14       * $this->getResult("max_packages_per_day")) < $r;
15     });
16 }
```

Code 17: Frage Initialisierung

Darüber hinaus sind in Code-Block 17 in Zeile 8 und 11 sogenannte offene Fragen definiert, welche als Ergebnis entweder eine Zahl, einen Text oder ein Objekt liefern können. Hierbei werden in erster Linie die Ergebnisse der Unterzweige kumuliert und jenes Ergebnis, das den Schwellenwert $\alpha$ überschreitet zurückgegeben. Es ist oft notwendig auf Ergebnisse einer Frage zu reagieren und einen neuen Rückgabewert zu definieren. In diesem Fall muss ein Komparator eingesetzt werden, um den Text in ein anderes Ergebnis umzuwandeln, welches an eine darüberliegende Frage weitergeleitet werden kann. Fragen mit einer Zahl als Antwort bieten mehr Möglichkeiten hinsichtlich der Berechnungen. Damit dies möglich ist, muss eine Kumulierungsfunktion, ähnlich zu jener bei den binären Fragen, definiert werden. Die Funktion selbst kann frei gewählt werden und zum Beispiel den Durchschnitt, ein Minimum oder Maximum beziehungsweise auch die Summe über die Antworten der Frage darunter bilden.

$$x = \sum_{i=0}^{a} f(y_i) \tag{15}$$

Das Ergebnis $x$ in der Formel 15 wird durch die Summe über die Funktion f von allen Antworten der Ebene darunter bestimmt. Die Funktion selbst kann beliebiger Natur sein und zum Beispiel die Gewichtung der Kante miteinbeziehen. Wichtig hierbei ist, dass jede Sub-Frage einer kumulierenden Frage mit einer Zahl als Antwort auch vom selben Typ sein muss und nicht zum Beispiel eine binäre Frage sein kann. Wird ein binäres Ergebnis anstatt einer Zahl benötigt, muss auch in diesem Fall ein Komparator eingesetzt werden, um das Ergebnis zu vergleichen.
Ein Komparator wird dann benötigt, wenn eine Entscheidung ein binäres Ergebnis benötigt, die darunterliegenden Fragen jedoch von offener Natur sind. Ein Komparator besteht aus drei Komponenten: dem Vergleichsoperator, einem linken und einem rechten Ergebnis. Der Vergleichsoperator, wie zum Beispiel $x < y$, bildet die Umwandlung zu einem binären Wert. In diesem Zusammenhang können die zwei Seiten entweder fix definiert oder ein Teil von einem Entscheidungsbaum sein.

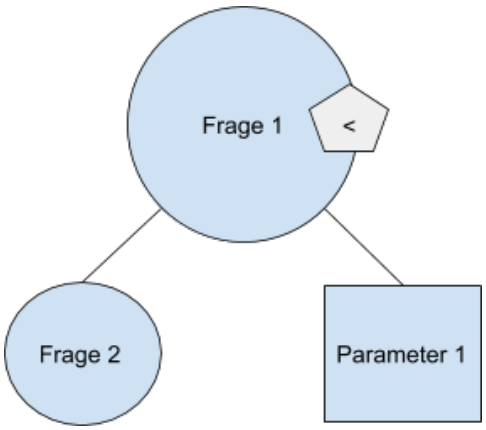


Abbildung 16: Frage mit Komparator, eigene Abbildung

In der Abbildung 16 ist die Hauptfrage in Ebene 1 mit einem < Vergleichsoperator definiert, dessen linke Seite durch das Ergebnis der Frage 2 und dessen rechte Seite über den Parameter 1 gebildet wird.

$$Frage1 < Parameter1 \tag{16}$$

Die zu verwendenden Komparatoren sind von den darunterliegenden Ergebnissen abhängig. Ist das Ergebnis binär oder in Textform, kann nur auf Gleich- oder Ungleichheit geprüft werden, handelt es sich hingegen um eine Zahl als Ergebnis, können auch größer oder kleiner Operatoren angewandt werden.

```
1 $q->getCompare()
2   ->isEquals(2);
```

Code 18: Vergleich des Ergebnisses

Eingesetzt wird der Komparator, wie in Code-Block 18 veranschaulicht, über getCompare(), was eine leeres Vergleichsobjekt im Kontext der aktuellen Frage generiert. Es stehen dann einige Funktionen zur Verfügung, wie zum Beispiel isEqual, isTrue, isFalse, isLager, isLargerOrEqual, isSmaller, isSmallerOrEqual, isInArray oder isNotEqual. Muss ein erweiterter Vergleich durchgeführt werden, kann dies mittels der 'is()' Methode, wie in Code-Block 19 dargestellt, geschrieben werden. Diese Methode verlangt als Parameter eine Funktion, welche den Vergleich des Ergebnisses durchführt und das gesetzte Ergebnis zurückliefert. Die beiden Beispiele führen den gleichen Vergleich durch und kommen zum selben Ergebnis. Dies ist darauf zurückzuführen, dass der Komparator standardmäßig beim Zutreffen der Bedingung *wahr* als Antwort zurückliefert. Zu beachten ist hierbei,

dass die Vergleichsfunktion wieder die Frage als Objekt zurückliefert und somit mittels Verkettung mehrere Operationen auf dem selben Objekt durchgeführt werden können.

```
1 $q->getCompare()
2 ->is(function($r){return $r === 2}, "preparing");
3 ->isEquals(2, "in stock")
4 ->else("out of stock");
```

Code 19: Vergleich mittels Funktion

Dadurch kann bei einem Komparator das Ergebnis geändert werden und aus einer binären Antwort ein Text generiert werden. Somit kann eine binäre Frage zum Vergleichen von offenen Fragen verwendet und der Komparator als Transformator eingesetzt werden.
Der Vorteil bei der Unterscheidung zwischen *on*() und *getCompare*() ist, dass die Frage selbst bereits auf Ergebnisse von einer Berechnung reagieren kann, wie es bei Entscheidungsfragen notwendig ist und darüber hinaus eine übergeordnete Frage auf das bereits geänderte Ergebnis reagieren und in ein neues Format oder Ergebnis transformieren kann.

### 5.4.2. Aktionen

Einen weiteren Fragetyp stellen die Aktionen dar. Durch dise Frage wird der reine Entscheidungsbaum gebrochen und über eine Rekursion und die Änderung des Parameter-Objektes das Erstellen von dynamischen Fragen in der Form von Netzwerken ermöglicht. Ein solcher Parameter kann zum Beispiel ein Datum sein, welches je nach Anforderung für die selbe Frage geändert werden kann. Eine Aktion soll in diesem Zusammenhang auf das Eintreten eines Ergebnisses reagieren und die Parameter für die Fragestellung verändern.

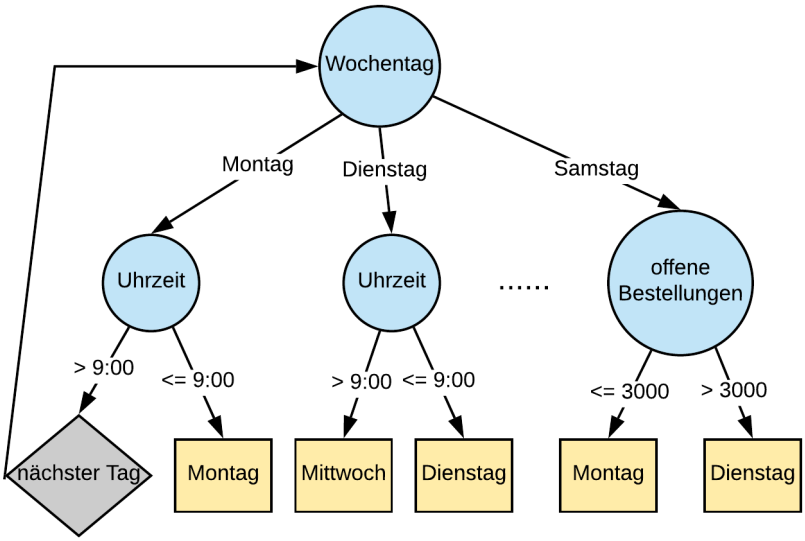


Abbildung 17: Aktion bei Frage, eigene Abbildung

In der Abbildung 17 wird für die Entscheidung, wenn es Montags nach 9 Uhr ist, die Aktion gesetzt, dass das Datum der nächste Tag ist. Wie in der Frage ersichtlich, muss für die Aktion der Fortsetzungspfad definiert werden. Durch die Rekursion ergibt sich, dass eine Endlosschleife generiert wird. In diesem Zusammenhang muss unbedingt die maximale Ausführungstiefe berücksichtigt werden, damit die Berechnung terminiert.
Wie der Code-Block 20 zeigt, wird die Aktion um die nächste Ebene des Baumes gestülpt und vorab ausgeführt. Der Parameter des Aktions-Konstruktors kann auch $\$this$ sein und es wird dann die aktuelle Frage noch einmal mit den geänderten Parametern evaluiert.

```
1 public function init(){
2   $q->on(false, new NextDateAction(new Question()));
3 }
```

Code 20: Aktion bei false

In der *process*() Methode der Aktion, wie in Code-Block 21 veranschaulicht, kann direkt auf das gesetzte Parameter-Objekt zugegriffen werden. Nach dem Aufruf der *process*() Methode wird immer die Frage, welche im Konstruktor definiert wurde, initialisiert und ausgeführt.

```
1 /**
```

```
2   * Action process
3   **/
4  public function process(){
5    $this->getParameter()->setDate(
6      $this->getParameter()->getDate() + 1
7    );
8  }
```

Code 21: Datum neu setzen

Alle bis zu diesem Zeitpunkt evaluierten Fragen dürfen nicht mehr in Betracht gezogen werden, da sich die Grundlage der Entscheidung geändert haben kann. Ergibt das Schema des Baumes eine Schleife, muss dadurch jede Frage noch einmal evaluiert werden. Nach 100 Ebenen wird die Ausführung automatisch gestoppt und das wahrscheinlichste Ergebnis für die aktuellen Parameter zurückgeliefert. Hierdurch ist eine Endlosschleife unmöglich.

## 5.5. Terminierung

Die wichtigen Faktoren für die Berechnung eines Baumes sind die Laufzeit, die Tiefe, die Vollständigkeit und die Glaubwürdigkeit der Berechnung. Die Laufzeit kann über den Parameter eingeschränkt werden, sodass die Ausführung nach erreichen dieser Zeit abgebrochen und ein temporäres Ergebnis, welches auf jedem Knoten mit einer bestimmten Glaubwürdigkeit liegt, zurückgegeben wird.

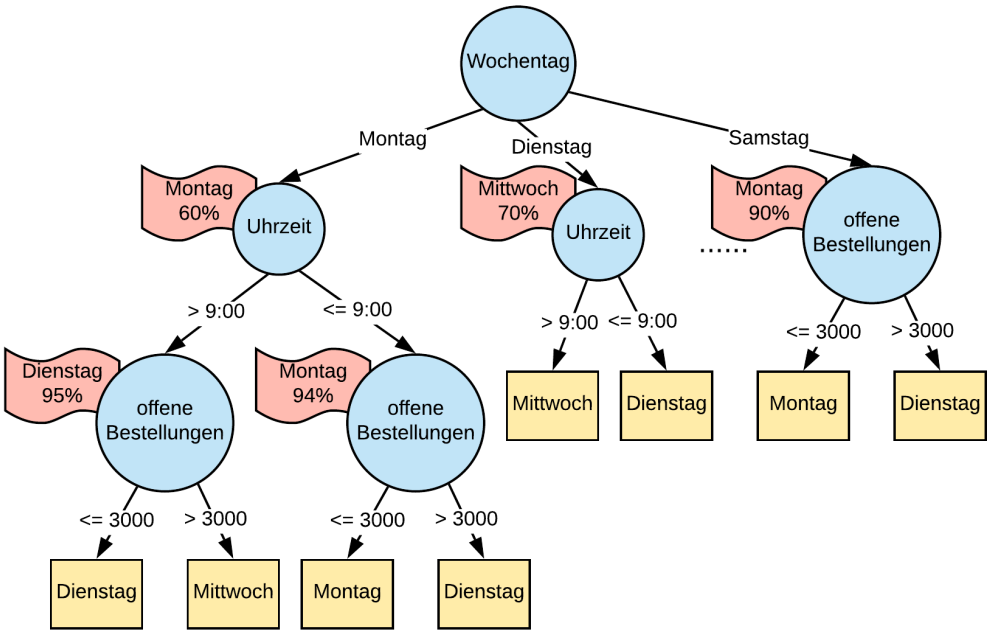


Abbildung 18: Entscheidungsmodell temporäres Ergebnis, eigene Abbildung

Neben der Laufzeit ist die Glaubwürdigkeit maßgeblich ausschlaggebend für das Terminieren einer Berechnung. Dies wurde in Abschnitt 5.4.1 bereits durch die Einführung der oberen Schranke betrachtet und führt zu einem frühzeitigen Abbruch der Berechnung, sobald der Entscheidungsbaum ein Ergebnis mit ausreichender Glaubwürdigkeit zurückliefert. Hierdurch kann vorab auf das temporäre Ergebnis zurückgegriffen werden, wenn die Glaubwürdigkeit von diesem bereits ausreichend ist. In Abbildung 18 ist das temporäre Ergebnis als rote Fahne neben einer Entscheidung dargestellt. Würde zum Beispiel eine Glaubwürdigkeit von $\alpha >= 90\%$ ausreichen und der Wochentag Samstag sein, kann sofort, nach der Evaluierung der ersten Ebene, Montag als Ergebnis zurück gegeben werden.
Darüber hinaus ermöglicht ein Einschränken der Tiefe des Baumes ein rasches Abfragen eines Ergebnisses. Geht man davon aus, dass der abgebildete Baum mit $max(d) = 1$ initialisiert wurde, muss die Berechnung nach dem ersten Schritt abgebrochen und das temporäre Ergebnis mit der entsprechenden Glaubwürdigkeit zurückgegeben werden.
Als dritter Terminierungsfaktor kann auch noch das Erreichen der Vollständigkeit = 1 gesehen werden. Dies ist darauf zurückzuführen, dass keine weiteren Ebenen und Zweige im Baum vorliegen und alle Ergebnisse beziehungsweise ein einzelnes Endergebnis für

die Evaluierung bereits vorliegen. Bei einem reinen Entscheidungsbaum ohne vereinende Fragen ist die Vollständigkeit durch ein Blatt im Baum erreicht. Bei vereinenden Fragen kann die Ausführung frühzeitig durch das Erreichen der Schranke in einer Ebene abgebrochen werden. Entscheidet jedoch eine darüber liegende Frage, dass die Glaubwürdigkeit nicht ausreicht, können die weiteren Fragen noch evaluiert werden.

## 5.6. Loader

Der Loader stellt den Lademechanismus des Modells dar und verhindert das mehrmalige Laden der Daten durch unterschiedliche Fragen beziehungsweise Ebenen. Es werden für diesen Zweck einzelne Quellen registriert und durch einen eindeutigen Schlüssel jederzeit referenziert. Für die Quelle kann ein Filter definiert werden, falls nur eine partielle Abfrage der Daten benötigt wird. Die Frage registriert beim Initialisieren die benötigten Quellen und Filter. Ist die Quelle bereits vorhanden, wird nur der Filter angepasst und überprüft, ob eine Neuberechnung notwendig ist. Im besten Fall wurden die Daten in einer vorherigen Ebene bereits gebraucht und müssen somit gar nicht neu geladen werden.

```
1  /**
2   * Initialize all sources
3   **/
4  public function init(){
5    Loader::instance()->add(new StockLoader());
6  }
7
8  /**
9   * Load data
10  **/
11 public function load(){
12   Loader::instance()->load($this->getParameter());
13 }
```

Code 22: Daten-Loader

Der Loader selbst ist, wie in Code-Block 22 dargestellt, ein Singleton[7]. Am Anfang des Lebenszyklus wird der Loader in der Initialisierungsmethode mit den aktuellen Parametern initialisiert und vor der eigentlichen Berechnung der Frage wird automatisch der Loader mit dem aktuellen Parameter-Objekt aufgerufen und die fehlenden Daten im Objekt ergänzt. Bei der Initialisierung wird bereits überprüft, ob das Objekt zum Laden der Daten bereits vorhanden ist und nur dann, wenn es nicht vorhanden ist, auch im Loader gesetzt.

```
1  /**
2  * Loader Method
```

[7]Ein Singleton ist eine Klasse von der nur eine Instanz existieren darf.

```
3  **/
4  public function process(){
5    $db_Product = new DB_Product();
6    $db_Product->setProduct_ID($this->getParameter()->getProduct_ID());
7    if($db_Product->quickfind()){
8      $this->getParameter()->set('Product_Stock', $db_Product->getStock());
9    }
10 }
11
12 /**
13 * Check Method
14 **/
15 public function check(){
16   return $this->getParameter()->isset('Product_Stock');
17 }
```

Code 23: Loader-Modul

Das Überprüfen des Parameter-Objektes wird von jeder Loader-Methode selbst in der *check*() Methode durchgeführt. Nur wenn diese Methode $false$ zurückliefert, wird auch *process*() ausgeführt und die Daten werden in das Parameter-Objekt gesetzt.
Der Loader überwacht darüber hinaus die Parameter-Objekte und führt ein Reload aus, falls sich die Daten der Parameter-Objekte durch Actions geändert haben. Hierfür muss jedes Modul wissen, welche Daten für die Abfrage wichtig sind und ob sich diese verändert haben. Dies muss immer über die Check-Methode implementiert werden. Jedem Modul müssen alle Parameter vorab bekannt sein, damit die Daten performant über gesammelte Abfragen bei externen System, wie zum Beispiel Datenbanken, nachgeladen werden können.

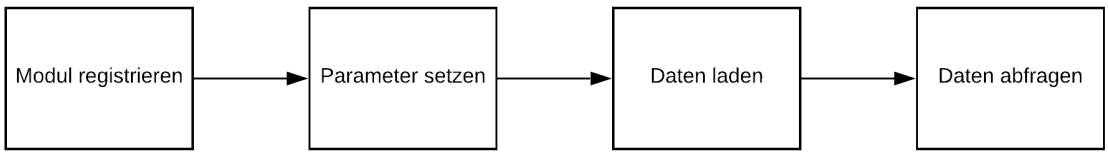


Abbildung 19: Loader, eigene Abbildung

Der Ablauf, wie in Abbildung 19 dargestellt, beginnt mit dem Registrieren der Module.

Anschließend werden die Parameter-Objekte gesetzt und die Daten für alle geladen. Erst wenn eine Frage die Daten benötigt, können diese vom Parameter-Objekt bezogen werden.

# 6. Vorhersage des Zustellungsdatums

Eines der zentralen Themen beim Einkaufen in einem Online-Shop ist das erwartete Zustellungsdatum. In Abbildung 2 ist dies auf der Detailseite eines Online-Shops dargestellt. Diese Berechnung beruht im Moment rein auf der Entscheidung der Produktverfügbarkeit, des nächsten Werktages, inklusive des aktuellen Werktages und der maximalen Anzahl an Bestellungen an einem Tag. Alle weiteren Informationen werden über ein externes System des jeweiligen Lieferdienstes bereitgestellt. Das Problem in diesem Zusammenhang ist jedoch, wenn der externe Service ausfällt, muss das Script mindestens eine Sekunde warten, um festzustellen, dass kein Ergebnis vorliegt.
Ziel der Verbesserung ist, dass kein Ergebnis von einem Drittanbieter geladen werden muss, sondern nur interne Vorhersagen und Abstrahierungen zu einem Zustellungsdatum führen.

| **Produkt** | **Anzahl** | **Produktart** | **Nächste Lieferung** | **Lieferdienst** | **Land** |
|---|---|---|---|---|---|
| PSF-001A | 1 | Physisch | 2019-05-10 | Post.at | Östereich |
| PSF-008Z | 1 | Physisch | 2019-05-16 | DHL | Deutschland |
| PSF-005R | 2 | Physisch | 2019-05-10 | Post.at | Österreich |
| PSF-001A | 1 | Physisch | 2019-05-10 | DHL | Deutschland |
| PSF-009 | 1 | Digital | * | E-Mail | Deutschland |

Tabelle 4: Vorhersagedaten

In der Tabelle 4 sind die einzelnen Features, wie zum Beispiel die Produktart oder das Lieferland, zu sehen. Jede Zeile repräsentiert ein Produkt in der jeweiligen Detailseite oder in der Bestellung.

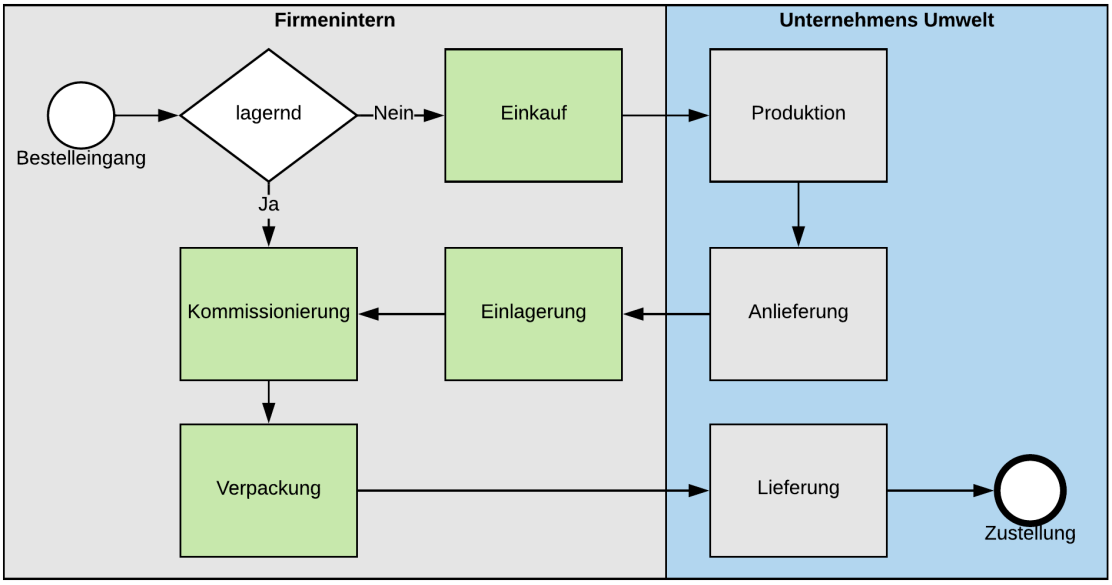


Abbildung 20: Bestellprozess, eigene Abbildung

In Abbildung 20 ist der vereinfachte Prozess vom Bestelleingang bis zur Zustellung abgebildet. In Grau sind hierbei jene Schritte gekennzeichnet, die nicht direkt unternehmensintern durchgeführt werden können und deswegen nur schwer zu beeinflussen sind. Hierzu gehören die Produktion, Anlieferung und Auslieferung an den Endkunden. Durch diese drei Prozesse werden klar die Schnittstellen zwischen den Unternehmen definiert. Die Schnittstellen sind auf der einen Seite in der Supply Chain vorgelagert, weg vom Produzenten oder Lieferanten in die Richtung des Unternehmens, das den Bestellprozess abwickelt und auf der anderen Seite in Richtung der Kundin beziehungsweise des Kunden, was die Auslieferung der Ware betrifft.

## 6.1. Datenevaluierung

Auf der einen Seite bildet die Grundlage der Daten die Stammdatenverwaltung des ERP-Systems. Zu diesen Werten zählt zum Beispiel die nächste Lieferung oder wie der Produkttyp ist. Bei der nächsten Lieferung handelt es sich in diesem Fall um eine Eingangslieferung, was eine Nachbestellung der Ware beim Lieferanten bedeutet. Darüber hinaus entscheidet die Produktart, ob ein Produkt überhaupt auf Lager liegt oder direkt, wie es zum Beispiel bei digitalen Produkten der Fall ist, per Mail versendet wird.
Auf der anderen Seite werden dynamische Werte, wie zum Beispiel das Lieferland, durch

die Kundin beziehungsweise den Kunden definiert. Diese Daten müssen zur Laufzeit von außen in das System eingesetzt werden und definieren die Grundlage für die Frage: “Was ist das zu erwartende Zustelldatum per {Lieferdienst} für {Anzahl} Stück vom Produkt {Artikel-Nr} nach {Land}.“ Alle weiteren Daten werden von dem Entscheidungsbaum ergänzt und dem Parameter-Objekt angefügt. Diese Daten bestehen aus der aktuellen Uhrzeit, dem verfügbaren Lagerstand, der nächsten Nachlieferung und vielen Features mehr. Diese Daten werden im Loader einmalig für alle Entscheidungen geladen und können somit wiederverwendet werden.
Die zentrale Frage in dem Entscheidungsbaum ist, ob das Produkt an dem gewünschten Datum versendet werden kann. Diese Frage hängt primär von der Warenverfügbarkeit, dem Wochentag und der aktuellen Uhrzeit ab. Darüber hinaus gibt es noch weitere Einflüsse wie die Gesamtanzahl der Bestellungen für einen Lieferdienst an dem betreffenden Tag. In weiterer Folge muss auch die Lieferzeit des jeweiligen Lieferdienstes für das Zielland berücksichtigt werden und basierend auf einem Startdatum ein richtiges Zustelldatum definiert werden.

## 6.2. Entscheidungsbaum

Auf Basis der bereits erlangten Informationen und Rückschlüsse können folgende manuelle Einzelfragen und deren spezielle Loader aufgestellt werden:

- Welcher Wochentag ist an dem angegebenen Datum?
- Ist das angegebene Datum ein Arbeitstag in dem betrachteten Land?
- Wie viele Stück sind verfügbar?
- Wie viele Bestellungen sind für diesen Tag für den jeweiligen Lieferdienst bereits vorgemerkt?
- Was ist die aktuelle Woche im Jahr?

Darüber hinaus kann das Modell von selbst alle Features des Parameter-Objektes verwenden, um die Entscheidung noch weiter zu verfeinern. Als Ergebnis kann aber nicht direkt das gewünschte Datum gesucht werden, es muss stattdessen ein Referenzdatum gesetzt und die Lieferzeit ab diesem berechnet werden. Das Datum wird direkt im Parameter-Objekt gesetzt und in allen manuellen Entscheidungen oder Aktionen berücksichtigt.

## 6.3. Implementierung

Die Implementierung basiert auf der Grundlage des Entscheidungsbaumes aus Kapitel 5 und berücksichtigt die Erkenntnisse der grundlegenden Datenstruktur. Die Daten für das Parameter-Objekt kommen auf der einen Seite beim Lernen aus den bisherigen Bestellungen, welche in der Datenbank gespeichert wurden und auf der anderen Seite aus den aufbereiteten Daten für die Ausgabe im Frontend. Diese Daten sind in einem Bean [8] gespeichert und werden für den Warenkorb, die Listen- und Detailseiten direkt aus der Datenbank geladen. Diese Daten müssen zwingend für die Abfrage der Lieferzeit verwendet werden und der Loader darf keine weiteren Informationen mehr nachladen. Darüber hinaus müssen die manuellen Fragen aus dem vorherigen Abschnitt implementiert werden, um anschließend über den Lernprozess ein automatisches Ergebnis zu erlangen.

### 6.3.1. Lernen

Die Daten für das Lernen werden direkt aus der Datenbank geladen und anschließend die Parameter-Objekte damit befüllt. Aus der Datenbank werden folgende Daten geladen:

- Bestelldatum
- Uhrzeit
- Wochentag
- Stück
- Artikel-Nummer
- Verfügbare Stück
- Eintreffen der nächsten Lieferung
- Lieferdienst
- Erwartetes Zustelldatum
- Offene Bestellungen
- Lieferland
- Postleitzahl

[8] Aus Java inspirierte Datenobjekte, welche die Arrays von PHP vollständig ersetzten.

Da jedoch zum Zeitpunkt des Lernens noch keine Aufzeichnung über den verfügbaren Lagerstand zum Bestellabschluss und die nächsten Lieferungen vorhanden sind, wird der Entscheidungsbaum in zwei Hälften aufgeteilt. Der erste Teil ist der firmeninterne, welcher vom Bestelleingang bis zur Auslieferung die Zeit widerspiegelt. Darauf aufbauend bestimmt der zweite Baum die benötigte Auslieferungszeit. Die Uhrzeit und der Wochentag können direkt aus dem Bestelldatum extrahiert und dem Parameter-Objekt angefügt werden. Das erwartete Zustelldatum vom Lieferdienst ist in diesem Zusammenhang bereits gespeichert und kann für diese Abfrage verwendet werden. Die offenen Bestellungen ergeben sich aus jenen Bestellungen, die vor der zu betrachtenden Bestellung getätigt wurden, aber noch nicht verpackt sind. Diese Daten werden insbesondere auf den jeweiligen Lieferdienst eingeschränkt, da es aufgrund der unterschiedlichen Cutoff-Zeiten [9] zu unterschiedlichen Ergebnissen kommen kann.

### 6.3.2. Daten

In Code-Block 24 ist die Abfrage der Daten aus der MySQL Datenbank ersichtlich. Die wichtigen Daten sind hierbei in zwei Tabellen gespeichert, auf der einen Seite stellt die ShopAccountJournal Tabelle alle Daten für die Belege, wie zum Beispiel Rechnung oder Lieferschein, zur Verfügung. Auf der anderen Seite muss auf die erweiterte Tabelle ShopAccountJournal_Data zurückgegriffen werden, da hier das erwartete Zustelldatum im JSON Format gespeichert ist. Dieses Zustelldatum wird zum Zeitpunkt der Belegerstellung vom jeweiligen Lieferdienst abgefragt.

```
1 SELECT SAJ.ShopDeliveryService_Code as DeliveryService,
2   SAJ.Country_Code_Journal as Country,
3   SAJ.ShopAccountJournal_ZIPCode as ZipCode,
4   SAJ.Table_Create as ReferenceDate,
5   SAJD.ShopAccountJournalData_Data->>"$.expectedArrivalRange.from" as
        ArrivalFrom,
6   SAJD.ShopAccountJournalData_Data->>"$.expectedArrivalRange.to" as
        ArrivalTo
7 FROM ShopAccountJournal as SAJ
8 JOIN ShopAccountJournalData SAJD on SAJ.ShopAccountJournal_ID =
    SAJD.ShopAccountJournal_ID
```

[9] Dies ist der letztmögliche Zeitpunkt, um ein Paket zu bestellen, damit es noch am selben Tag dem Lieferdienst übergeben werden kann.

```
9 WHERE SAJ.ShopAccountJournalType_Code = 'delivery'
```

Code 24: SQL Lieferzeit-Abfrage

Die resultierenden Daten sind in Tabelle 5 zu sehen und dienen als Grundlage für den zweiten Entscheidungsbaum. Sehr gut sind hierbei schon die unterschiedlichen Formate für die Postleitzahl zu sehen. Dies kann speziell bei alphanumerischen Werten zu einem Problem beim Clustering, was ein Zusammenführen von Entscheidungssträngen mit gleichem Ergebnis ist, führen.

| **Dienst** | **Land** | **Postleitzahl** | **Datum** | **Zustellung von** | **Zustellung bis** |
|---|---|---|---|---|---|
| dhl | DE | 01069 | 2018-04-27 | 2018-04-28 | 2018-04-30 |
| ups | GB | L69 3GJ | 2018-04-27 | 2018-05-02 | 2018-05-02 |
| ups | FR | 91480 | 2018-04-27 | 2018-05-02 | 2018-05-02 |
| P | AT | 3053 | 2018-04-27 | 2018-04-30 | 2018-04-30 |
| P | AT | 2700 | 2018-04-27 | 2018-04-28 | 2018-04-28 |
| dpdpl | PL | 10-409 | 2018-04-27 | 2018-05-04 | 2018-05-07 |
| dhl | DE | 85609 | 2018-04-27 | 2018-04-28 | 2018-04-30 |

Tabelle 5: Die Daten für das Lernen der Zustellungsvorhersage

Der zweite Teil der Entscheidung betrifft die Zeit, bis das Paket das Lager verlässt und wird vermehrt durch firmeninterne Einflüsse definiert. Hierbei stehen die Entscheidungen ob ein Produkt lagernd ist und an welchem Wochentag die Bestellung durchgeführt wird, im Mittelpunkt.

### 6.3.3. Implementierung

Die Implementierung des Baumes erfordert durch die generische Funktionalität des abstrakten Parents sehr wenige Zeilen. Einzig die *load*() Methode musste überschrieben werden, da weitere Informationen aus dem Datum extrahiert werden mussten. Dies ist in Code-Block 25 in Zeile fünf und sechs bei der Verwendung der Entscheidungen für den Wochentag und das Monat dargestellt. In Zeile sieben ist ein Datenpreprocessor für die Postleitzahl implementiert. Bei den verschiedenen Ländern besteht das Problem, dass oft alphanumerische Postleitzahlen verwendet werden, dies jedoch für den Vergleich nicht optimal ist. Der Grund dafür ist, dass niemals alle Postleitzahlen mit einem gültigen Wert im Entscheidungsbaum vorhanden sein werden, wenn aber ein Clustering anhand

der Postleitzahl durchgeführt wird, kann es vorkommen, dass kein Ergebnis gefunden wird. Der Source-Code für die Entscheidungsfindung ist im Appendix A.9 zu finden.

```
1  class DeliveryTimeTree extends Tree
2  {
3    public function load()
4    {
5      $this->use(new IsWeekdayDecission());
6      $this->use(new MonthOfYearDecission());
7      $this->use(new IntegerPartOfZipCode());
8      $this->ignore("date");
9      $this->ignore("zipcode");
10     parent::load();
11   }
12 }
```

Code 25: Intialisierung des Entscheidungsbaum

Die Daten aus Tabelle 5 werden im CSV Format geliefert und müssen zum Teil noch in ein richtiges Format übersetzt werden. Das Ergebnis wird aus der Datenbank als reines Datum geliefert und in der Berechnung aber als Tage seit Erstellung des Beleges, wie in Zeile 7 und 8 in Code-Block 26 ersichtlich, verwendet. Alle weiteren Daten werden direkt in das Parameter-Objekt gesetzt und danach eine Aufteilung auf ein Validierungs- ($vArray$) und Trainigsset ($pArray$) durchgeführt.

```
1  $pArray = [];
2  $vArray = [];
3  if (($handle = fopen("Arrival.csv", "r")) !== FALSE) {
4    while (($data = fgetcsv($handle)) !== FALSE) {
5      if($data[0] != 'veloblitz') continue;
6      $earlier = new DateTime($data[3]);
7      $from = (new DateTime($data[4]))->diff($earlier)->format("%a");
8      $to = (new DateTime($data[5]))->diff($earlier)->format("%a");
9      $v = (new DeliveryParameter())
10       ->setService($data[0])
11       ->setCountry($data[1])
12       ->setZipCode($data[2])
13       ->setDate($data[3])
14       ->setResult([$from, $to]);
15     if(rand(0,8) == 2){
```

```
16        $vArray[] = $v
17      } else {
18        $pArray[] = $v
19      }
20    }
21    fclose($handle);
22  }
23  $tree = new DeliveryTimeTree();
24  $tree->learn($pArray);
25  print($tree->getDescription());
```

Code 26: Definition der Parameter

Das resultierende Parameter-Objekt nach dem Initialisieren ist im Code-Block 27 ersichtlich. Herauszuheben sind hierbei die zwei Parameter, welche mit *use* beginnen. Diese sind über die manuellen Entscheidungen hinzugekommen und können deshalb bei der automatischen Erstellung des Entscheidungsbaumes berücksichtigt werden. Hierbei muss aber sichergestellt werden, dass diese Entscheidungen nicht zeitabhängig[10] sind, sollte dies der Fall sein, muss der Parameter gespeichert werden. Die Reihenfolge der manuellen Fragen ist hierbei fix über die Zahl nach dem # festgelegt. Das Ergebnis wird in diesem Fall nur beim Lernen angezeigt, da es von außen in das Parameter-Objekt gesetzt wurde. Im Parameter-Objekt bleiben auch die per *ignore*() ausgeschlossenen Features erhalten, sie werden nur im Feature-Selection Schritt ausgeschlossen.

```
1  {
2    "values":{
3      "service":"veloblitz",
4      "country":"AT",
5      "zipcode":"8010",
6      "date":"2018-12-13",
7      "use#0":"Donnerstag",
8      "use#1":"Dezember"
9      "use#2":"8010",
10   },
11   "result":["0","0"]
```

[10]Zeitabhängig bedeutet, dass sich über die Zeit für das selbe Parameter-Objekt das Ergebnis der manuellen Entscheidung ändert.

```
12 }
```

Code 27: Ein Parameter-Objekt

Für das relativ kleine Testset mit Veloblitz[11] bei nur rund 60 Test- und 5 Validierungs-Sets konnte ein gutes Ergebnis erzielt werden. Die Lieferzeit für diesen Lieferdienst wurde manuell im System eingetragen und für Montag jeweils ein Tag mehr gerechnet. An allen weiteren Tagen konnte immer am gleichen Tag zugestellt werden. Dies bildet der Baum auch schön ab. Wobei in diesem Zusammenhang zu sagen ist, dass die Postleitzahl eigentlich nicht ausschlaggebend ist für einen validen Baum. Wie im Code-Block 28 in der Zeile 12 ersichtlich, haben alle Ergebnisse vom Validierungs-Set mit jenen des Entscheidungsbaumes übereingestimmt.

```
1  (zipcode <= 8035)
2    (Monat == Dezember)
3      (Wochentag == Donnerstag) = ["0","0"]
4      (Wochentag == Dienstag) = ["0","0"]
5      (Wochentag == Montag) = ["1","1"]
6    (else) = ["0","0"]
7  (zipcode > 8035)
8    (Wochentag == Montag)
9      (Monat == Dezember) = ["1","1"]
10     (Monat == Jänner) = ["0","0"]
11   (else) = ["0","0"]
12 Error Rate: 0% (0 Fehler / 5.)
```

Code 28: Resultierender Baum (Veloblitz)

Im Vergleich dazu wurde von den selben Daten nur der Lieferdienst DHL für das Lieferland Niederlande herausgerechnet und dabei auf die Verwendung der Postleitzahl verzichtet. Aufgrund der Vorgabe des Lieferdienstes und Landes fallen diese zwei Features automatisch weg, da keinerlei Information darin enthalten ist. Aus diesem Grund bleiben nur noch der Wochentag und das Monat für die Entscheidung der Lieferzeit übrig. Das relativ kleine Trainigs-Set mit 391 Zeilen ergibt sich daraus, dass der Lieferdienst erst seit kurzer Zeit für den Versand in die Niederlande eingesetzt wird.

```
1 (Wochentag == Donnerstag) = ["4","5"]
2 (Wochentag == Freitag) = ["4","5"]
3 (Wochentag == Montag) = ["2","3"]
```

[11] Veloblitz ist ein Fahrradkurrier im Großraum Graz, welcher die Ware noch am selben Tag zustellt.

```
4 (Wochentag == Dienstag) = ["2","3"]
5 (Wochentag == Mittwoch)
6   (Monat == März) = ["5","6"]
7   (else) = ["2","5"]
8 Error Rate: 19.05% (8 Fehler von 42 Stichproben), 391 Trainingsset)
```

Code 29: Baum ohne Postleitzahl (DHL-NL)

In dem Beispiel aus Code-Block 29 lässt sich gut erkennen, dass Pakete am Donnerstag und Freitag jeweils um zwei Tage länger benötigen, als an den anderen Tagen. Am Mittwoch hingegen kann es durchaus möglich sein, dass Pakete bereits am Freitag zugestellt werden, ansonsten geht der Algorithmus davon aus, dass diese am Montag zugestellt werden. Knapp 20 Prozent Fehlerrate ist ein relativ hoher Wert und bedeutet, dass jeder achte Kunde beziehungsweise jede achte Kundin eine falsche Information angezeigt bekommt. Eine mögliche Erklärung für diese Verhalten sind fehlende oder zu wenige Informationen in den Parameter-Objekten. Bei fehlenden Informationen kann es sich zum Beispiel um die Feiertage in dem besagten Zeitraum handeln.

# 7. Diskussion und Ausblick

Die Implementierung des Standard-Entscheidungsbaumes ist durch die Objektorientierung sehr einfach durchzuführen. PHP bietet durch die Magic-Methods, wie zum Beispiel __call[12], gute Möglichkeiten, um generischen Code zu programmieren.
Darüber hinaus sind bei der Implementierung des Entscheidungsbaumes noch einige Fallstricke aufgefallen, was die Qualität der zu lernenden Entscheidungen und deren Grundlagen betrifft. Auch der Begriff Entscheidungsbaum ist durch die Einführung der Aktion nicht mehr zutreffend. Die zweite Frage, bezugnehmend auf die internen Abläufe des Unternehmens, wurde aufgrund von fehlenden Daten für das Anlernen des Baumes nicht implementiert. Es war nicht möglich zu eruieren ob und wie viele Stück von jedem Produkt zum Bestellabschluss vorhanden waren. Im Vergleich zu bestehenden Algorithmen wie C4.5 und CART kann gesagt werden, dass C4.5 einen sehr ähnlichen Baum produziert wie der hier entworfene Algorithmus. Es lassen sich auch bei C4.5 verschiede Arten von Entscheidungen kombinieren. Die Entscheidungsfindung basiert jedoch bei C4.5, wie in Abschnitt 4.2.3 beschrieben, auf der Entropie und dem Informationsgehalt. Der im Zuge dieser Arbeit entworfene Algorithmus funktioniert aber nach dem Prinzip, dass die Tiefe des Baumes minimiert wird. Dies wird dadurch sichergestellt, dass immer jenes Feature ausgewählt, wird mit der minimalen Anzahl an unterschiedlichen Ergebnissen und Werten des zu betrachdenden Features. Der Vorteil von diesem Ansatz ist, dass für eine breite Masse an Anfragen schnell ein Ergebnis gefunden werden kann und erst bei Eckfällen eine tiefere Ebene des Baumes erreicht wird. Darüber hinaus findet der Algorithmus nicht zwangsläufig die global beste Entscheidung, dies ist darauf zurückzuführen, dass kein Subbaum rekursiv ausgetauscht wird, wenn in einer Ebene erkannt wird, dass eine Entscheidung in den Ebenen davor nicht optimal war. Um den resultierenden Baum zu vereinfachen, wurden verschiedene Pruning-Technologien entworfen und implementiert. Das aus dem C4.5 bekannte Binärsplitting wird bei Zahlen automatisch angewandt. Darüber hinaus werden Entscheidungen mit nur einem Ergebnis automatisch als Endknoten und somit als Ergebnis definiert. Eine Besonderheit bietet der Else-Zweig, welcher die unterschiedlichen Ergebnispfade als normale Klassifizierungsentscheidungen zurückgeliefert und nur die Klassen, welche über 80 Prozent der gleichen Antworten beinhalten, zusammenfasst. Weiters kann beim Überschreiten einer gewissen Wahrscheinlichkeit das Ergebnis der Unterentscheidungen als Endergebnis zusammen-

[12] Call fängt jeden Aufruf einer Methode auf dem Objekt ab. Es können somit auch nicht definierte Methoden zu einem Standardverhalten führen.

gefasst und somit gestrichen werden. Der Algorithmus erlaubt auch, das mehrfache Evaluieren von einem Feature, dies ist vor allem für den numerischen Binärbaum, aufgrund der größer/kleiner Vergleichem, hilfreich jedoch bei Klassifizierungen nicht notwendig.

## 7.1. Kurzfristige Änderungen

Kurzfristige Änderungen können in dem Baum, sofern diese vorab eingelernt wurden, ideal bedient werden. Die Entscheidung kann zu jeder Zeit mit dem selben Modell neu berechnet werden und durch die sich verändernden Parametern zu einem anderen Ergebnis führen. Ein Beispiel hierfür ist die Abhängigkeit von der Cutoff-Zeit des Lieferdienstes, wodurch kurz vor Erreichen dieser Zeit eine andere Zustellzeit angegeben wird, als danach. Somit kann davon ausgegangen werden, dass beim Vorhandensein der Entscheidungen auch kurzfristige Änderungen mit einer bestimmten Unschärfe berücksichtigt werden können.

## 7.2. NoSQL

Als Datenspeicher für die anzulernenden Entscheidungen und die berechneten Modelle eignet sich NoSQL durch die Dokumentenstruktur bestens. Der Grund hierfür ist, dass keinerlei Relationen benötigt werden und das Objekt des Baumes direkt gespeichert werden kann. Das Parameter-Objekt zum Beispiel kann nach der Evaluierung mit dem erwarteten Ergebnis in einer Datenbank, für den nächsten Anlernprozess gespeichert werden. Es muss somit nur noch das eingetroffene Ergebnis gespeichert oder abgefragt und die Liste mit den neuen Parametern an das Modell zum Anlernen übergeben werden. Dieses Modell ist wiederum selbst ein Dokument und kann durch die Verschachtlung von Dokumenten auch in der Datenbank abgelegt werden.

## 7.3. Vorhersagen und deren Genauigkeiten

Durch das manuelle Anlegen von Parameter-Objekten ist es möglich, Vorhersagen für zukünftige Ereignisse zu treffen und diese mit der entsprechenden Genauigkeit durchzuführen. Die Genauigkeit selbst wird durch das berechnete Modell und die maximale Tiefe des Baumes bestimmt. Es ist in diesem Zusammenhang möglich, dass ein Modell mit vielen Ebenen berechnet wird, aber durch das Minimieren der Genauigkeit nur wenige Ebenen evaluiert werden müssen. Fazit ist, dass der Baum mit jeder Evaluierung, was einer Ebene gleichkommt, an Genauigkeit gewinnt und die Rechenzeit dadurch aber

steigt.

## 7.4. Fehlende Entscheidungen

Beim Lernen des Algorithmus müssen viele Daten zur Verfügung stehen. Ist dies nicht der Fall, kann der Algorithmus keine ausreichende Abstraktion der Entscheidungen berechnen und die Daten des Validation-Sets können nicht richtig zugeordnet werden. Um dies zu demonstrieren, kann im Datengenerator die Anzahl der Parameter-Objekte auf 100 Datensätze gestellt werden, womit durch das Aufteilen in Test- und Validation-Set nur noch 90 Parameter-Objekte zum Anlernen zur Verfügung stehen. Das erwünschte Ergebnis ist in Code-Block 16 zu sehen. Der Algorithmus hat jedoch den Entscheidungsbaum in Code-Block 30 entworfen, welcher durch viele unterschiedliche Klassifizierungen und eine hohe Fehlerrate von 10% nicht optimal ist. Dies bedeutet, dass die Entscheidungsfindung langsam und nur zu 90% richtig ist.

```
1  (type == mail) = 1
2  (type == product)
3    (IsProductInStock == in stock)
4      (stock == 5) = 0
5      (stock == 1) = 0
6      (stock == 4) = 0
7      (stock == 2) = 1
8      (stock == 3)
9        (ordercount == 0) = 0
10       (ordercount == 3) = 0
11       (ordercount == 2) = 1
12     (stock == 6)
13       (ordercount == 5) = 0
14       (ordercount == 0) = 0
15       (ordercount == 2) = 1
16     (stock == 8)
17       (ordercount == 2) = 1
18       (else) = 0
19     (stock == 7)
20       (ordercount == 0) = 0
21       (ordercount == 5) = 1
22       (else) = 0
23     (stock == 9)
```

```
24        (ordercount == 0) = 0
25        (ordercount == 2) = 1
26        (else) = 0
27    (IsProductInStock == out of stock)
28      (ordercount == 2) = 1
29      (else) = 0
```

Code 30: Klassifizierungsfehler

Bei wenigen Daten kann der mathematische Teiler aus Abschnitt 5.3.2 hilfreich sein und eine Abstraktion der Daten durch größer oder kleiner Vergleiche erlauben.

## 7.5. Widersprüche im Testset

Sind alle Entscheidungen im Testset klar definiert, kann ein homogener Baum mit 100 Prozent Glaubwürdigkeit angelernt werden. Ist jedoch ein Widerspruch in den Parameter-Objekten enthalten, muss nach dem Abarbeiten aller Parametern das Ergebnis mit der höchsten Wahrscheinlichkeit und dessen Glaubwürdigkeit übernommen werden. Diese Widersprüche können durch zu wenig Features beim Anlernen entstehen, sodass nicht alle Einflüsse der Entscheidung berücksichtigt werden. Stellt zum Beispiel ein Lieferdienst generell in Deutschland binnen 24 Stunde die Pakete zu und gibt es aber eine Gemeinde, in der die Zustellung mindestens 48 Stunden dauert, muss zumindest die Postleitzahl berücksichtigt werden, um ein valides Ergebnis zu erlangen. Darüber hinaus muss der Algorithmus die Analogie zwischen der Postleitzahl und den 48 Stunden erkennen. Dies bedeutet, dass unbekannte Daten auch nicht gelernt werden können und somit das Testset unbedingt groß genug sein muss. Im konkreten Beispiel soll für jede Postleitzahl in Deutschland zumindest ein Eintrag vorhanden sein.

```
1  $pArray = [];
2  for($i = 0; $i < 1000; $i++){
3    $result = false;
4    $r = rand(0,10);
5    if($r == 2 && rand(0,3)) $result = true;
6
7    $t = "product";
8    if($b == 4){
9      $result = true;
10     $t = "mail";
11   }
```

```
12    if($i % 10 > 6 && $r == 5) $result = true;
13    $pArray[] = (new MyParameter())->setStock($i % 10)
14    ->setOrderCount($r)
15    ->setType($t)
16    ->setResult($result);
17  }
```

Code 31: Generator für nicht eindeutigen Baum

Der Generator aus Code-Block 12 wurde, wie in Code-Block 31 ersichtlich, in Zeile 5 um die Bedingung $rand(0, 3)$ erweitert. Hierdurch ist der eindeutige Baum aus Code-Block 16 nicht mehr sichergestellt und es resultiert bei $ordercount = 2$ ein nicht vollständig definierter Zustand. Im konkreten Fall erzeugt der Generator ein Testset mit einer Fehlerrate von $1,6\%$. Dies widerspiegelt die Ausgabe des Baumes beim Code-Block 32 in Zeile 4 und 10. Es führt dazu, dass zum Beispiel für die Parameterkombination "Type ist Produkt, Lagerstatus = out of stock, 2 Bestellungen, Lagerstand = 1" zwei verschiedene Ergebnisse vorliegen. Wobei aber durch die Wahrscheinlichkeit von $rand(0, 3)$ eher 1 überwiegt[13].

```
1  (type == mail) = 1
2  (type == product)
3    (IsProductInStock == out of stock)
4      (ordercount == 2)
5        (stock == 1) = 1
6        (stock == 0) = 1
7      (else) = 0
8    (IsProductInStock == in stock)
9      (ordercount == 0) = 0
10     (ordercount == 2)
11       (stock == 4) = 1
12       (stock == 8) = 1
13       (stock == 3) = 1
14       (stock == 7) = 1
15       (stock == 2) = 1
16       (stock == 9) = 1
17       (stock == 5) = 1
18       (stock == 6) = 1
```

[13]PHP wandelt alle Integer $x > 0$ in den Wahrheitswert *true* um. Somit ergibt sich eine 25 prozentige Wahrscheinlichkeit, dass der berechnete Wert $false$ ist.

```
19      (ordercount == 5)
20        (stock <= 6) = 0
21        (stock > 6) = 1
22      (else) = 0
```

Code 32: Nicht eindeutiger Baum

Senkt man die Zufallsschranke auf $rand(0, 1)$, ergibt sich eine 50 prozentige Chance, dass der Wert wahr ist und der Algorithmus wählt, je nachdem welches Ergebnis überwiegt, das wahrscheinlichere aus. Dies führt, wie in Code-Block 33 in Zeile 5 beziehungsweise 6 dargestellt, zu der Unterscheidung auf Basis des Lagerstandes. Die Fehlerrate liegt in diesem Fall mit 1000 gelernten Samples bei $1,01$ Prozent. Wird dieses Ergebnis mit dem aus Code-Block 32 verglichen, fällt sofort die erweiterte mathematische Funktion bei "type = product, in stock, ordercount = 2" auf. Dies liegt daran, dass der Algorithmus einen Vergleichswert erkennt und deshalb die Daten beginnt zusammenzufassen. Im vorherigen Beispiel hätte der Algorithmus erkennen können, dass alle Werte der darunter liegenden Ebenen gleich sind und somit der weitere Vergleich nicht notwendig ist.

```
1  (type == mail) = 1
2  (type == product)
3    (IsProductInStock == out of stock)
4      (ordercount == 2)
5        (stock == 0) = 0
6        (stock == 1) = 1
7      (else) = 0
8    (IsProductInStock == in stock)
9      (ordercount == 0) = 0
10     (ordercount == 2)
11       (stock <= 8) = 0
12       (stock > 8) = 1
13     (ordercount == 5)
14       (stock <= 6) = 0
15       (stock > 6) = 1
16     (else) = 0
```

Code 33: Nicht eindeutiger Baum II

Zusammengefasst kann gesagt werden, dass die richtige Auswahl der Features und deren Eindeutigkeit im Bezug auf das Ergebnis für die richtige Berechnung des Baumes wichtig sind. Darüber hinaus muss zwingend ein Validierungs-Set verwendet werden, um die

Gültigkeit des Baumes darzulegen. Ein Prozent klingt nicht viel, bedeutet aber, dass von hundert Kundinnen und Kunden eine beziehungsweise einer ein falsches Ergebnis angezeigt bekommt. Bei der Neuberechnung sollte dieser Wert berücksichtigt werden und das neue Modell, falls der Fehler im Vergleich zum bestehenden Modell am Validierungs-Set größer ist, einfach verworfen wird.

## 7.6. Baum vs. Netzwerk

Die ursprüngliche Idee eines Entscheidungsbaumes konnte nicht bis zum Ende standhalten, da durch das Einführen der Aktionen eine Schleife und hierdurch eine netzwerkartige Struktur, wie in Abbildung 17 zu sehen ist, möglich wird. Prinzipiell kann aber jede Aktion auch als neuer Baum gesehen werden, es besteht im Grunde kein Zusammenhang zwischen den beiden Bäumen und es könnte auch der gleiche sein. Wichtig ist nur, dass alle bisher geladenen Daten, sofern sie von dem geänderten Feld abhängig sind, die Gültigkeit verlieren.
Darüber hinaus wird das Prinzip des Entscheidungsbaumes weiter durch die vereinenden Fragen, wie in Abbildung 14 ersichtlich, gebrochen. Es ist hierbei nicht mehr möglich das Ergebnis eines einzelnen Stranges beziehungsweise Blattes abzufassen, sondern das Ergebnis muss mindestens in x[14] Strängen zurückgeliefert werden, damit es gültig ist. Der Vorteil von dieser Entscheidung ist, dass auch eine Liste an Unterentscheidungen abgearbeitet werden kann, wie es bei der Entscheidung der Verfügbarkeit des gesamten Warenkorbes der Fall ist. Diese Frage in Kombination mit einer Aktion erlaubt es ein Datum zu finden, an dem der gesamte Warenkorb verfügbar ist, ohne weitere Ebenen oder Schleifen im Entscheidungsbaum der Warenverfügbarkeit zu definieren.

[14] x wird durch die obere Schranke definiert.

# Literatur


[Andrej Krenker, 2011] Andrej Krenker, Janez Bester, A. K. (2011). *Artificial Neural Networks*. IntechOpen, Rijeka.

[Apache, 2019a] Apache (2019a). Apache solr ref guide. `https://lucene.apache.org/solr/guide/6_6/index.html`. Abgerufen: 20.04.2019.

[Apache, 2019b] Apache (2019b). Solr features. `http://lucene.apache.org/solr/features.html`. Abgerufen: 20.04.2019.

[Barros et al., 2015] Barros, R., de Carvalho, A., and Freitas, A. (2015). *Automatic Design of Decision-Tree Induction Algorithms*. SpringerBriefs in Computer Science. Springer International Publishing.

[Braspenning et al., 1995] Braspenning, P. J., Thuijsman, F., and Weijters, A. J. M. M., editors (1995). *Artificial Neural Networks: An Introduction to ANN Theory and Practice*. Springer-Verlag, Berlin, Heidelberg.

[Chauvin and Rumelhart, 2013] Chauvin, Y. and Rumelhart, D. (2013). *Backpropagation: Theory, Architectures, and Applications*. Developments in Connectionist Theory Series. Taylor & Francis.

[Cloete and Zurada, 2000] Cloete, I. and Zurada, J. (2000). *Knowledge-based Neurocomputing*. Mit Press. MIT Press.

[de Ville and Neville, 2013] de Ville, B. and Neville, P. (2013). *Decision Trees for Analytics Using SAS Enterprise Miner*. SAS Institute.

[Dean and Ghemawat, 2004] Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. Technical report.

[Gibbons, 1985] Gibbons, A. (1985). *Algorithmic Graph Theory*. Cambridge University Press.

[Group, 2019] Group, P. (2019). Was kann php? `https://www.php.net/manual/de/intro-whatcando.php`. Abgerufen: 10.04.2019.

[Hartmunt Stadtler, 2004] Hartmunt Stadtler, C. K. (2004). *Supply Chain Management and Advanced Planning: Concepts, Models, Software and Case Studies*. Springer.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.

[Hope et al., 2018] Hope, T., Resheff, Y., Lieder, I., and Lotze, T. (2018). *Einführung in TensorFlow: Deep-Learning-Systeme programmieren, trainieren, skalieren und deployen*. Animals. O'Reilly.

[Jacob et al., 2008] Jacob, O., Doeffinger, d., Foerster, V., Hammermann, D., Hawig, J., Jacob, O., Kaminski, G., Lutz, F., Miller, M., Passenheim, O., et al. (2008). *ERP Value: Signifikante Vorteile mit ERP-Systemen*. Xpert.press. Springer Berlin Heidelberg.

[Kozak, 2018] Kozak, J. (2018). *Decision Tree and Ensemble Learning Based on Ant Colony Optimization*. Studies in Computational Intelligence. Springer International Publishing.

[Loeckx et al., 2013] Loeckx, J., Mehlhorn, K., and Wilhelm, R. (2013). *Grundlagen der Programmiersprachen*. Leitfäden und Monographien der Informatik. Vieweg+Teubner Verlag.

[Lutz et al., 2007] Lutz, M., Ascher, D., and Gherman, D. (2007). *Einführung in Python:*. O'Reilly.

[M. Oded, 2014] M. Oded, R. L. (2014). *Data Mining With Decision Trees: Theory And Applications (2nd Edition)*. Series In Machine Perception And Artificial Intelligence. World Scientific Publishing Company.

[Martin Hesseler, 2007] Martin Hesseler, M. G. (2007). *Basiswissen ERP-Systeme*. W3L AG.

[Meier, 2017] Meier, A. (2017). *Werkzeuge der digitalen Wirtschaft: Big Data, NoSQL & Co.: Eine Einführung in relationale und nicht-relationale Datenbanken*. essentials. Springer Fachmedien Wiesbaden.

[microtech GmbH, 2017] microtech GmbH (2017). Was bedeutet erp. `https://www.microtech.de/erp-wiki/erp`. Abgerufen: 14.04.2019.

[Niceshops, 2019] Niceshops (2019). Ecco verde - produkt detail seite. `https://www.ecco-verde.at/martina-gebhardt/martina-gebhardt-eye-care`. Abgerufen: 13.04.2019.

[Oracle, 2019a] Oracle (2019a). Mysql. https://www.oracle.com/technetwork/database/mysql/index.html. Abgerufen: 12.04.2019.

[Oracle, 2019b] Oracle (2019b). Mysql 8.0 reference manual. https://dev.mysql.com/doc/refman/8.0/en/. Abgerufen: 20.04.2019.

[Rojas, 2013] Rojas, R. (2013). *Theorie der neuronalen Netze: Eine systematische Einführung*. Springer-Lehrbuch. Springer Berlin Heidelberg.

[S N Sivanandam, 2006] S N Sivanandam, S. Sumathi, S. N. D. (2006). *Introduction to Neural Networks using Matlab 6.0*. Tata McGraw-Hill.

[Sadalage and Fowler, 2012] Sadalage, P. and Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Pearson Education.

[Sauer, 2002] Sauer, H. (2002). *Relationale Datenbanken - Theorie und Praxis*. Addison-Wesley.

[Schildgen, 2016] Schildgen, J. (2016). *MongoDB kompakt: Was Sie über die NoSQL-Dokumentendatenbank wissen müssen*. Datenbanken kompakt. Books on Demand.

[Shai Shalev-Shwartz, 2014] Shai Shalev-Shwartz, S. B.-D. (2014). *Understanding machine learning : from theory to algorithms*. Cambridge University Press.

[Theis, 2011] Theis, T. (2011). *Einstieg in Python*. Galileo computing. Galileo Press.

[Tyler, 2017] Tyler, J. (2017). How to install zts enabled php7 and the pthreads module. https://www.rapidspike.com/blog/php7-pthreads/. Abgerufen: 12.04.2019.

[Unterstein and Matthiessen, 2012] Unterstein, M. and Matthiessen, G. (2012). *Relationale Datenbanken und SQL in Theorie und Praxis*. eXamen.press. Springer Berlin Heidelberg.

[Vaish, 2013] Vaish, G. (2013). *Getting Started with Nosql*. Packt Publishing.

[Von der Lippe, 2018] Von der Lippe, P. (2018). *Deskriptive Statistik*. Walter de Gruyter GmbH & Co KG.

# Appendices

## A. Code

### A.1. AbstractQuestion

```
1  /**
2  * Class Question
3  * @method AbstractQuestion calculate()
4  */
5  abstract class AbstractQuestion {
6
7    protected $reliability  = 0;
8
9    protected $description = [];
10
11   protected $result = null;
12
13   protected $functions = [];
14   protected $members = [];
15
16   /**
17   * @var null|Parameter
18   */
19   protected $parameter = null;
20
21   /**
22   * @return null
23   */
24   public function getParameter()
25   {
26     return $this->parameter;
27   }
28
29   /**
30   * @param $text
31   * @return $this
```

```
32      */
33      public function addDescription($text){
34        $this->description[] = $text;
35        return $this;
36      }
37
38      /**
39      * @return string
40      */
41      public function getDescription($i = 0){
42        return str_repeat ( " ", $i*2).implode("\n".str_repeat ( " ", $i*2),
              $this->description);
43      }
44
45      /**
46      * @param Parameter $parameter
47      * @return $this
48      */
49      public function setParameter(Parameter $parameter)
50      {
51        $this->parameter = $parameter;
52        return $this;
53      }
54
55      /**
56      * @var Compare[]
57      */
58      protected $compare = [];
59
60      /**
61      * @var Action[]
62      */
63      protected $action = [];
64
65      /**
66      * Question constructor.
67      * @param Parameter|null $parameter
```

```
68      */
69      public function __construct(Parameter $parameter = null)
70      {
71        $this->parameter = $parameter;
72        $this->load();
73      }
74
75      protected function load(){
76
77      }
78
79      /**
80      * @param $name
81      * @param $arguments
82      * @return mixed
83      * @throws Exception
84      */
85      public function __call($name, $arguments = null)
86      {
87        if(substr($name,0,3) == "set" && strlen($name) > 3 && is_array(
              $arguments) && count($arguments)) {
88          if(count($arguments) > 1){
89            throw new Exception("Do not set more attributes on one member");
90          }
91          $name = strtolower(substr($name,3));
92          $arg = reset($arguments);
93          if(  is_object($arg) && ($arg instanceof Closure)){
94            $this->functions[$name] = $arg;
95          } else {
96            $this->members[$name] = $arg;
97          }
98          return $this;
99        } else if (substr($name, 0,3) == "get" && strlen($name)){
100         $name = strtolower(substr($name,3));
101         return $this->members[$name];
102       } else if($name == "calculate") {
103         $this->process();
```

```
104         $this->processCompare();
105         return $this;
106       } else if(method_exists($this,$name)) {
107         return call_user_method_array($name,$this,$arguments);
108       } else if(isset($this->functions[$name])){
109         return call_user_func_array($this->functions[$name], $arguments);
110       }
111       throw new Exception("Property ".$name." not found!");
112     }
113
114     /**
115     * Process all Compares
116     * stop on first result
117     */
118     protected function processCompare(){
119       if(count($this->compare)) {
120         $result = false;
121         foreach ($this->compare as $compare) {
122           [$found,$result] = $compare->compare();
123           if($found){
124             $this->reliability = 1;
125             break;
126           }
127         }
128         $this->result = $result;
129       }
130     }
131
132     /**
133     *
134     */
135     public function init(){
136       $this->description = [];
137     }
138
139     /**
140     * @return mixed
```

```
141     */
142     protected abstract function process();
143 
144     /**
145     * @return Compare
146     */
147     public function getCompare(){
148       $compare = new Compare($this);
149       $this->compare[] = $compare;
150       return $compare;
151     }
152     /**
153     * @return Action
154     */
155     public function getAction(){
156       $action = new Action($this);
157       $this->action[] = $action;
158       return $action;
159     }
160 
161     public function getResult(){
162       return $this->result;
163     }
164 
165     public function getReliability(){
166       return $this->reliability;
167     }
168   }
```

## A.2. Action

```
1 /**
2 * Class Action
3 * @method Compare isEqual($c,callable $f)
4 */
5 class Action
6 {
7 
```

```
8   /**
9   * @var AbstractQuestion
10  */
11  protected $question;
12  protected $else;
13  protected $members = [];
14  protected $functions = [];
15
16  public function __construct($question)
17  {
18    $this->question = $question;
19  }
20
21  /**
22  * @return AbstractQuestion
23  */
24  public function q(){
25    return $this->question;
26  }
27
28
29  /**
30  * @return $this
31  */
32  public function process(){
33    foreach ($this->members as  $val) {
34      $t = false;
35      switch ($val['name']) {
36        case "isequal":
37        $t = $this->q()->getResult() === $val["compare"];
38        break;
39      }
40      if ($t) {
41        $val["result"]($this->q()->getParameter());
42        break;
43      }
44    }
```

```
45      return $this;
46    }
47
48    /**
49    * @param $name
50    * @param $arguments
51    * @return mixed
52    * @throws Exception
53    */
54    public function __call($name, $arguments = null)
55    {
56      if(method_exists($this,$name)) {
57        return call_user_method_array($name,$this,$arguments);
58      } else if(is_array($arguments) && count($arguments)) {
59        if(count($arguments) > 2){
60          throw new Exception("Do not set more attributes on one member");
61        }
62        $this->addAction($name,$arguments);
63        return $this;
64      }
65      throw new Exception("Property ".$name." not found!");
66    }
67
68    /**
69    * @param $name
70    * @param $arguments
71    */
72    protected function addAction($name,$arguments){
73      $this->members[] = ["name" => strtolower($name), "compare" =>
          $arguments[0], "result" => $arguments[1]];
74    }
75  }
```

## A.3. Compare

```
1  /**
2  * Class Compare
3  * @method Compare isEqual($c,$r = true)
```

```
4  * @method Compare is(callable $c)
5  * @method Compare else($c)
6  */
7  class Compare {
8    /**
9    * @var AbstractQuestion
10   */
11   protected $question;
12   protected $else;
13   protected $members = [];
14   protected $functions = [];
15
16   public function __construct($question)
17   {
18     $this->question = $question;
19   }
20
21   /**
22   * @return AbstractQuestion
23   */
24   public function q(){
25     return $this->question;
26   }
27
28   /**
29   * @return array
30   */
31   public function compare(){
32     $result = false;
33     $t = false;
34     foreach ($this->functions as $f){
35       $result = $f($this->question->getResult(), $this->question->
             getParameter());
36       if($result){
37         $t = true;
38       }
39     }
```

```
40      if(!$t) {
41        foreach ($this->members as  $val) {
42          $t = false;
43          switch ($val['name']) {
44            case "isequal":
45            $t = $this->question->getResult() === $val["compare"];
46            break;
47          }
48          if ($t) {
49            $result = $val["result"];
50            break;
51          }
52        }
53      }
54      if(!$t && !is_null($this->else)){
55        $t = true;
56        $result = $this->else;
57      }
58      return [$t, $result];
59    }
60
61    /**
62    * @param $name
63    * @param $arguments
64    * @return mixed
65    * @throws Exception
66    */
67    public function __call($name, $arguments = null)
68    {
69      if(method_exists($this,$name)) {
70        return call_user_method_array($name,$this,$arguments);
71      } else if(is_array($arguments) && count($arguments)) {
72        if($name == "is" &&  is_object($arguments[0]) && ($arguments[0]
              instanceof Closure)) {
73          $this->functions[] = $arguments[0];
74        } elseif($name == "else") {
75          $this->else = $arguments[0];
```

```
76        } else {
77          if(count($arguments) > 2){
78            throw new Exception("Do not set more attributes on one member");
79          }
80          $this->addCompare($name,$arguments);
81        }
82        return $this;
83      }
84      throw new Exception("Property ".$name." not found!");
85    }
86
87    /**
88    * @param $name
89    * @param $arguments
90    */
91    protected function addCompare($name,$arguments){
92      $this->members[] = ["name" => strtolower($name), "compare" =>
          $arguments[0], "result" => isset($arguments[1])?$arguments[1]:true
          ];
93    }
94
95
96  }
```

## A.4. CummulateQuestion

```
1
2  /**
3  * Class CummulateQuestion
4  */
5  class CummulateQuestion extends AbstractQuestion{
6
7    /**
8    * @var float
9    */
10   protected $upperBound = 1;
11
12   /**
```

```
13    * @var null|float
14    */
15    protected $lowerBound = null;
16
17
18    /**
19    * @var AbstractQuestion[]
20    */
21    protected $questions = [];
22
23    /**
24    *
25    */
26    protected function process()
27    {
28      $r = 0;
29      foreach ($this->questions as $question){
30        $question->calculate();
31        $r += $question->getReliability();
32      }
33      if(count($this->questions)){
34        $this->reliability = $r / count($this->questions);
35      }
36    }
37
38    /**
39    * @return bool|null
40    */
41    public function getResult()
42    {
43
44      $r = 0;
45      foreach ($this->questions as $question){
46        $r += $question->getResult() ? 1 : 0;
47      }
48      $result = null;
49      if(count($this->questions)){
```

```
50        $result = $r / count($this->questions) >= 1;
51      }
52      return $result;
53    }
54
55
56    /**
57    * @param AbstractQuestion $question
58    * @return AbstractQuestion
59    */
60    public function addQuestion(AbstractQuestion $question){
61      if(is_null($question->getParameter())) {
62        $question->setParameter($this->getParameter());
63      }
64      $this->questions[] = $question;
65      return $question;
66    }
67  }
```

## A.5. Decission

```
1  /**
2  * Class Decission
3  */
4  class Decission extends AbstractQuestion{
5
6    /**
7    * @var array
8    */
9    protected $debug = [];
10
11   protected $on = [];
12
13   protected $default = null;
14
15   protected $question = null;
16
17   protected $name = null;
```

```
18
19      /**
20      * @return null
21      */
22      public function getName()
23      {
24        return $this->name;
25      }
26
27      /**
28      * @param null $name
29      * @return $this
30      */
31      public function setName($name)
32      {
33        $this->name = $name;
34        return $this;
35      }
36
37
38      /**
39      * @param null $question
40      * @return $this
41      */
42      public function setQuestion($question)
43      {
44        $this->question = $question;
45        return $this;
46      }
47
48      /**
49      * @var null|float
50      */
51      protected $lowerBound = null;
52
53      /**
54      *
```

```
55    */
56    protected function processCompare()
57    {
58      parent::processCompare();
59
60      foreach ($this->action as $action) {
61        $action->process();
62      }
63
64      $result = $found = null;
65      foreach ($this->on as $on) {
66        if(is_callable($on['c']) && $on['c']($this->result) || $on['c'] ===
              $this->result){
67          $found = true;
68          $result = $this->processOn($on);
69          break;
70        }
71      }
72      if(!is_null($this->default) && !$found){
73        $found = true;
74        $result = $this->processOn($this->default);
75      }
76      if($found) {
77        $this->result = $result;
78      }
79    }
80
81    /**
82    * @param $arrOn
83    * @return mixed
84    */
85    private function processOn(array $arrOn){
86      $result = null;
87      if($arrOn['f'] instanceof AbstractQuestion) {
88        $result = $arrOn['f']->setParameter($this->getParameter())->
              calculate()->getResult();
89      } elseif(is_callable($arrOn['f'])) {
```

```
90        $result = $arrOn['f']($this->getParameter());
91      } else {
92        $result = $arrOn['f'];
93      }
94      return $result;
95    }
96
97
98    /**
99    * @param $c
100   * @param $f
101   * @param $d
102   * @return $this
103   */
104   public function on($c,$f, $d = ""){
105     $this->on[] = ["c" => $c, "f" => $f, "d" => $d];
106     return $this;
107   }
108
109   /**
110   * @param $f
111   * @return $this
112   */
113   public function default($f, $d = ""){
114     $this->default = ["f" => $f, "d" => $d];
115     return $this;
116   }
117
118   /**
119   * @return mixed|void
120   */
121   protected function process(){
122     if($this->question instanceof  AbstractQuestion){
123       $this->question->setParameter($this->getParameter())->init();
124       $this->result = $this->question->calculate()->getResult();
125     } else {
126       $this->result = $this->getParameter();
```

```
127     }
128   }
129
130   public function getDescription($i = 0)
131   {
132     $desc = parent::getDescription($i)."\n";
133     foreach ($this->on as $on) {
134       if(strlen($on["d"])) {
135         $desc .= str_repeat(" ", $i * 2) . $on["d"];
136       }
137       if($on['f'] instanceof AbstractQuestion) {
138         $desc .= $on['f']->getDescription($i+1);
139       }elseif(is_string($on['f'])){
140         $desc .=  " = ".$on['f']."\n";
141       }elseif(is_array($on['f'])){
142         $desc .=  " = ".json_encode($on['f'])."\n";
143       }
144     }
145     if(strlen($this->default["d"])) {
146       $desc .= str_repeat ( " ", $i*2).$this->default["d"];
147     }
148     if($this->default["f"] instanceof  AbstractQuestion){
149       $desc .= $this->default["f"]->getDescription($i+1);
150     }elseif(is_string($this->default["f"]) && strlen($this->default["f"]))
        {
151       $desc .=  " = ". $this->default["f"]."\n";
152     }elseif(is_array($this->default["f"])){
153       $desc .=  " = ". json_encode($this->default["f"])."\n";
154     }
155     return $desc;
156   }
157 }
```

## A.6. Loader

```
1 /**
2 * Class Loader
3 */
```

```php
class Loader
{
  /**
  * @var LoaderModule[]
  */
  private $modules = [];

  /**
  * @var Loader
  */
  private static $loader = null;

  /**
  * @param Parameter $parameter
  * @return $this
  */
  public function load(Parameter $parameter){
    foreach ($this->modules as $module) {
      $module->process();
      $module->load($parameter);
    }
    return $this;
  }

  /**
  * @param LoaderModule $loader
  * @return $this
  */
  public function add(LoaderModule $loader){
    if(!isset($this->modules[$loader->getCode()])){
      $this->modules[$loader->getCode()] = $loader;
    }
    return $this;
  }

  /**
  * @return Loader
```

```
41    */
42    public static function instance(){
43      if(is_null(static::$loader)){
44        static::$loader  = new Loader();
45      }
46      return static::$loader;
47    }
48
49  }
```

## A.7. LoaderModule

```
1   /**
2   * Class LoaderModule
3   */
4   abstract class LoaderModule
5   {
6     abstract public function getCode();
7     abstract public function calculate();
8     abstract public function changed();
9
10    /**
11    * @var Parameter[] $parameter
12    */
13    private $parameter = [];
14
15    /**
16    * @param Parameter $parameter
17    * @return $this
18    */
19    public function addParameter(Parameter $parameter){
20      $this->parameter[] = $parameter;
21      return $this;
22    }
23
24    /**
25    * @return $this
26    */
```

```
27    public function process(){
28      if($this->changed()){
29        $this->calculate();
30      }
31      return $this;
32    }
33
34    /**
35    * @param Parameter $parameter
36    */
37    public function load(Parameter $parameter){
38
39    }
40  }
```

## A.8. Parameter

```
1   /**
2   * Class Parameter
3   */
4   class Parameter implements JsonSerializable {
5
6     /**
7     * @var array
8     */
9     private $arrParameter = [];
10
11    private $result = null;
12
13    /**
14    * @return null
15    */
16    public function getResult()
17    {
18      return $this->result;
19    }
20
21    /**
```

```
22    * @param null $result
23    * @return $this
24    */
25    public function setResult($result)
26    {
27      $this->result = $result;
28      return $this;
29    }
30
31    /**
32    * @param $name
33    * @param $arguments
34    * @return mixed
35    * @throws Exception
36    */
37    public function __call($name, $arguments = null)
38    {
39      if (substr($name, 0, 3) == "set" && strlen($name) > 3 && is_array(
          $arguments) && count($arguments)) {
40        if (count($arguments) > 1) {
41          throw new Exception("Do not set more attributes on one member");
42        }
43        $name = strtolower(substr($name, 3));
44        $this->arrParameter[$name] = $arguments[0];
45        return $this;
46      } else if (substr($name, 0, 3) == "get" && strlen($name)) {
47        $name = strtolower(substr($name, 3));
48        return $this->arrParameter[$name];
49      }
50    }
51
52    /**
53    * @param $name
54    * @return mixed
55    */
56    public function __get($name)
57    {
```

```
58      return $this->get($name);
59    }
60
61
62    /**
63    * @param $name
64    * @return mixed
65    */
66    public function get($name)
67    {
68      $result = null;
69      $name = strtolower($name);
70      if(isset($this->arrParameter[$name])){
71        $result = $this->arrParameter[$name];
72      }
73      return $result;
74    }
75
76    public function set($name, $value){
77      $name = strtolower($name);
78      $this->arrParameter[$name] = $value;
79      return $this;
80    }
81
82    /**
83    * @param $name
84    * @param $value
85    * @return $this
86    */
87    public function __set($name, $value)
88    {
89      return $this->set($name,$value);
90    }
91
92
93    /**
94    * @return array
```

```
95    */
96    public function toArray(){
97      return $this->arrParameter;
98    }
99
100   public function jsonSerialize()
101   {
102     return ['values' => $this->arrParameter, 'result' => $this->g\dfrac{}{
          Nenner}etResult()];
103   }
104 }
```

## A.9. Tree

```
1  /**
2  * Class Tree.
3  */
4  abstract class Tree extends Decission{
5
6    public $model = [];
7
8    protected $ignore = [];
9
10   protected $depth = 0;
11
12   /**
13   * @return array
14   */
15   public function getIgnore(): array
16   {
17     return $this->ignore;
18   }
19
20   /**
21   * @param array $ignore
22   * @return $this
23   *
24   */
```

```
25    public function setIgnore(array $ignore)
26    {
27      $this->ignore = $ignore;
28      return $this;
29    }
30
31    /**
32    * @param string $ignore
33    * @return $this
34    */
35    public function ignore(string $ignore)
36    {
37      $this->ignore[] = $ignore;
38      return $this;
39    }
40
41
42
43    /**
44    * @return string
45    */
46    public function getModel(): string
47    {
48      return json_encode($this->model);
49    }
50
51    /**
52    * @param string $model
53    * @return $this
54    */
55    public function setModel(string $model){
56      $this->model = json_decode($model,true);
57      return $this;
58    }
59
60    /**
61    * @var Decission[]
```

```
62    */
63    protected $use = [];
64
65    /**
66    * @param $decission
67    * @return $this
68    */
69    public function use($decission){
70      $this->use[] = $decission;
71      return $this;
72    }
73
74    /**
75    * @param array $array
76    * @return null
77    */
78    function array_key_first(array $array)
79    {
80      return $array ? array_keys($array)[0] : null;
81    }
82
83    /**
84    * @param Parameter[] $parameterList
85    */
86    public function learn(array $parameterList){
87      foreach($this->use as $dKey => $decission) {
88        $decission->init();
89      }
90      $this->model = $this->buildTree($parameterList,$this->getIgnore());
91    }
92
93    /**
94    * @param Parameter[] $parameterList
95    * @param $ignoreParameter
96    * @param $visitedParameter
97    * @return array
98    */
```

```
99  protected function calculateCurrentSplit($parameterList,$ignoreParameter
        , $visitedParameter){
100   foreach($this->use as $dKey => $decission){
101     foreach($parameterList as $pKey=> $parameter){
102       $parameterList[$pKey]->set("use#" . $dKey, $decission->
              setParameter($parameter)->calculate()->getResult());
103     }
104   }
105   $splitting = [];
106   $count = [];
107   $countHolder = [];
108   $ignore = [];
109   foreach ($visitedParameter as $val) {
110     if(!isset($ignore[md5($val)])){
111       $ignore[md5($val)] = 2;
112     }
113     $ignore[md5($val)]++;
114   }
115   foreach($parameterList as $pKey=> $parameter){
116     foreach($parameter->toArray() as $key => $value){
117       $v = 1;
118       if(in_array($key, $ignoreParameter)) continue;
119       if(isset($ignore[md5($key)])) $v = 1/((1 + $ignore[md5($key)]));
120       if(!isset($splitting[$key][$value][md5(print_r($parameter->
              getResult(), true))])){
121         $splitting[$key][$value][md5(print_r($parameter->getResult(),
              true))]['result'] = $parameter->getResult();
122         $splitting[$key][$value][md5(print_r($parameter->getResult(),
              true))]['keys'] = [];
123         ;
124       }
125       if(!isset($count[$key])){
126         $count[$key] = 0;
127       }
128       if(!isset($countHolder[$key][md5($value.print_r($parameter->
              getResult(),true))])){
129         $countHolder[$key][md5($value.print_r($parameter->getResult(),
```

```
            true))] = true;
130           $count[$key] += $v;
131         }
132         $splitting[$key][$value][md5(print_r($parameter->getResult(), true
              ))]['keys'][] = $pKey;
133       }
134     }
135     $key = null;
136     asort($count);
137     foreach ($count as $k => $count){
138       $key = explode('-',$k)[0];
139       if(count($splitting[$key]) > 1){
140         break;
141       }
142     }
143     if(is_null($key) || !isset($splitting[$key])){
144       return [null,[]];
145     }
146     return [$key, $splitting[$key]];
147   }
148
149
150   /**
151   * @param Parameter[] $parameterList
152   * @param array $ignoreParameter
153   * @param array $visitedParameter
154   * @return array|null|mixed
155   */
156   protected function buildTree(array $parameterList,$ignoreParameter = [],
          $visitedParameter= []){
157     if(!count($parameterList)) return null;
158     [$key, $split] = $this->calculateCurrentSplit($parameterList,
          $ignoreParameter,$visitedParameter);
159     if(count($split) == 0){
160       //Not unique result choose the one with highest probability
161       [$key, $split] = $this->calculateCurrentSplit($parameterList, $this
            ->getIgnore(), []);
```

```
162       $resultCount = [];
163       $resultArr = [];
164       foreach($split as $value => $result){
165         foreach ($result as $r => $v){
166           if(!isset($resultArr[$r])){
167             $resultCount[$r] = 0;
168             $resultArr[$r] = $v['result'];
169           }
170           $resultCount[$r] += count($v);
171         }
172       }
173       return $resultArr[array_search(max($resultCount), $resultCount)];
174     }
175     $resultModel = null;
176     $resultModel = ["key" =>$key];
177     $visitedParameter[] = $key;
178
179     foreach($this->calculateMerge($split) as $result){
180       if($result["end"]){
181         if($result["type"] == 'replace'){
182           $resultModel = $result['result'];
183         } else if($result["method"] == 'default'){
184           $resultModel[$result['method']] = $result['result'];
185         } else {
186           $resultModel[$result['method']][$result['value']] = $result['
                result'];
187         }
188       } else {
189         if($result["method"] == 'default') {
190           $resultModel[$result['method']] = $this->buildTree($this->
                getParameterListWithValue($parameterList, $key, $result['
                value'],$result['method']),$ignoreParameter,
                $visitedParameter);
191         } else {
192           $resultModel[$result['method']][$result['value']] = $this->
                buildTree($this->getParameterListWithValue($parameterList,
                $key, $result['value'],$result['method']),$ignoreParameter,
```

```
            $visitedParameter);
193       }
194     }
195   }
196   return $resultModel;
197 }
198
199 /**
200 * @param $split
201 * @return array
202 */
203 protected function calculateMerge($split){
204   $modelValueCount = [];
205   $type = "string";
206   $result = [];
207   $modelElse = [];
208   foreach ($split as $modelValue => $model){
209     foreach(array_keys($model) as $key){
210       if (!isset($modelValueCount[$key])) {
211         $modelValueCount[$key] = [];
212       }
213       $modelValueCount[$key][] = $modelValue;
214     }
215     if(is_integer($modelValue)){
216       $type = "integer";
217     }
218     $mKey = null;
219     $countKey = 0;
220     $sumKeys = 0;
221     foreach ($model as $sKey => $v){
222       $c = count($v['keys']);
223       $sumKeys += $c;
224       if($c > $countKey){
225         $countKey = $c;
226         $mKey = $sKey;
227       }
228     }
```

```
229       if(!is_null($mKey)) {
230         $probability = $countKey / $sumKeys;
231         $result[$modelValue]['result'] = $model[$mKey]['result'];
232         $result[$modelValue]['probability'] = $probability;
233         $result[$modelValue]['end'] = false;
234         if($probability > 0.9){
235           $result[$modelValue]['end'] = true;
236         }
237         $modelKey = [$model[$mKey]['result']];
238         if($probability < 0.8){
239           $modelKey[] = $probability;
240         }
241         if(!isset($modelElse[md5(print_r($modelKey,true))])){
242           $modelElse[md5(print_r($modelKey, true))] = [];
243         }
244         $modelElse[md5(print_r($modelKey, true))][] = $modelValue;
245       }
246       $result[$modelValue]['value'] = $modelValue;
247     }
248
249
250     $max = 0;
251     $maxKey = null;
252     $sum = 0;
253     foreach ($modelElse as $ckey => $values) {
254       $sum += count($values);
255       if (count($values) > $max) {
256         $max = count($values);
257         $maxKey = $ckey;
258       }
259     }
260     if(count($modelElse) == 1) {
261       $result = [[
262       'end' => true,
263       'type' => 'replace',
264       'result' => reset($result)['result'],
265       ]];
```

```
266     }else if ($max / $sum > 0.9) {
267       $result = [[
268       'end' => true,
269       'type' => 'replace',
270       'result' => reset($result)['result'],
271       ]];
272     }else if ($max / $sum > 0.8 && $max > 1) {
273       $model = null;
274       $valueExc = [];
275       foreach ($modelElse as $k => $value) {
276         foreach($value as $v){
277           if ($k == $maxKey) {
278             $model = $result[$v];
279             unset($result[$v]);
280           } else {
281             $valueExc[] = $v;
282             $result[$v]['type'] = $type;
283             $result[$v]['method'] = 'equals';
284           }
285         }
286       }
287       $model['method'] = 'default';
288       $model['type'] = $type;
289       $model['value'] = $valueExc;
290       $result['default'] = $model;
291     }else if($type == "integer") {
292       uasort($modelValueCount, function ($a, $b) {
293         return count($a) < count($b);
294       });
295       $i = 0;
296       $middle = [];
297       $minMiddle = $maxMiddle = null;
298       foreach ($modelValueCount as $r => $valueArray) {
299         if(is_null($minMiddle)){
300           $minMiddle = min($valueArray);
301           $maxMiddle = max($valueArray);
302         } elseif($maxMiddle > max($valueArray)){
```

```
303             $maxMiddle = max($valueArray);
304           } elseif($minMiddle < min($valueArray)){
305             $minMiddle =  min($valueArray);
306           }
307           $middle[] = array_sum($valueArray) / count($valueArray);
308           if (++$i > 1) break;
309         }
310         $border = ($maxMiddle + $minMiddle) /2;# array_sum($middle) / count(
                $middle);
311         if(count($middle) < 2 || $maxMiddle == $minMiddle){
312           $resultCount = [];
313           foreach($split as $value => $r1){
314             foreach ($r1 as $r => $v){
315               if(!isset($resultArr[$r])){
316                 $resultCount[$r] = 0;
317                 $resultArr[$r] = $v['result'];
318               }
319               $resultCount[$r] += count($v);
320             }
321           }
322           $result = [];
323           $result[] = [
324           'end' => true,
325           'type' => 'replace',
326           'result' => $resultArr[array_search(max($resultCount),
                $resultCount)],
327           ];
328         } else {
329           $result = [];
330           $result[] = [
331           'end' => false,
332           'value' => $border,
333           'type' => 'integer',
334           'method' => 'greater',
335           ];
336           $result[] = [
337           'end' => false,
```

```
338         'value' => $border,
339         'type' => 'integer',
340         'method' => 'less',
341         ];
342       }
343     } else {
344       foreach ($split as $value => $r) {
345         $result[$value]['method'] = 'equals';
346         $result[$value]['type'] = $type;
347         $result[$value]['end'] = false;
348         if (count($r) == 1) {
349           $result[$value]['result'] = reset($r)['result'];
350           $result[$value]['end'] = true;
351         }
352       }
353     }
354     return $result;
355   }
356
357   /**
358   * @param Parameter[] $paramList
359   * @param Decission|string $key
360   * @param $value
361   * @param string $operation
362   * @return array
363   */
364   protected function getParameterListWithValue($paramList, $key, $value,
        $operation = "equals"){
365     $cParamList = [];
366     foreach ($paramList as $param){
367       switch ($operation) {
368         case "equals":
369         if ($param->get($key) == $value) {
370           $cParamList[] = $param;
371         }
372         break;
373         case "less":
```

```
374         if ($param->get($key) <= $value) {
375           $cParamList[] = $param;
376         }
377         break;
378         case "greater":
379         if ($param->get($key) >= $value) {
380           $cParamList[] = $param;
381         }
382         break;
383         case "default":
384         if (!in_array($param->get($key),  $value)) {
385           $cParamList[] = $param;
386         }
387         break;
388       }
389     }
390     return $cParamList;
391   }
392
393   /**
394   * @param $model
395   * @param Decission|null $parent
396   * @return Decission
397   */
398   protected function buildModel($model, Decission $parent = null){
399     $return = $parent;
400     $this->handleUseQuestion($parent,$model);
401     if(isset($model["key"])){
402       if(isset($model["less"] )) {
403         foreach ($model["less"] as $result => $submodel) {
404           $decission = new Decission();
405           $this->handleUseQuestion($decission,$submodel);
406           [$function, $description] = $this->
                getCompareFunctionAndDescription($model,$result, "<=");
407           $parent->on(
408           $function,
409           $this->buildModel($submodel, $decission),
```

```
410           $description
411           );
412         }
413       }
414       if(isset($model["greater"] )) {
415         foreach ($model["greater"] as $result => $submodel) {
416           $decission = new Decission();
417           $this->handleUseQuestion($decission,$submodel);
418           [$function, $description] = $this->
                  getCompareFunctionAndDescription($model,$result, ">");
419           $parent->on(
420           $function,
421           $this->buildModel($submodel, $decission),
422           $description
423           );
424         }
425       }
426
427       if(isset($model["equals"] )) {
428         foreach($model["equals"] as $result => $submodel){
429           $decission = new Decission();
430           $this->handleUseQuestion($decission,$submodel);
431           [$function, $description] = $this->
                  getCompareFunctionAndDescription($model,$result);
432           $parent->on(
433           $function,
434           $this->buildModel($submodel, $decission),
435           $description
436           );
437         }
438       }
439       if(isset($model["default"] )) {
440         $decission = new Decission();
441         $this->handleUseQuestion($decission,$model);
442         $parent->default(
443         $this->buildModel($model["default"], $decission),
444         "(else)"
```

```
445         );
446       }
447     } else {
448       //$parent->addDescription("   ".$model);
449       $return = $model;
450     }
451     return $return;
452 
453   }
454 
455   /**
456   * @param $model
457   * @param $result
458   * @param string $type
459   * @return array
460   */
461   protected function getCompareFunctionAndDescription($model,$result,
        $type = "=="){
462     $key = $model['key'];
463     if(substr($key,0,4) == "use#") {
464       $q = $this->use[intval(substr($model["key"], 4))];
465       if (strlen($q->getName())) {
466         $key = $q->getName();
467       }
468       switch ($type){
469         case "==":
470         $f = function ($r) use ($result) {
471           return $r === $result;
472         };
473         break;
474         case "<=":
475         $f = function ($r) use ($result) {
476           return $r <= $result;
477         };
478         break;
479         case ">":
480         $f = function ($r) use ($result) {
```

```
481             return $r > $result;
482           };
483           break;
484         }
485
486       } else {
487         switch ($type){
488           case "==":
489           $f = function (Parameter $r) use ($key, $result) {
490             return $r->get($key) === $result;
491           };
492           break;
493           case "<=":
494           $f = function (Parameter $r) use ($key, $result) {
495             return $r->get($key) <= $result;
496           };
497           break;
498           case ">":
499           $f = function (Parameter $r) use ($key, $result) {
500             return $r->get($key) > $result;
501           };
502           break;
503         }
504       }
505       $d ="(" . $key . " ".$type." " . $result . ")";
506       return [$f,$d];
507     }
508
509     /**
510     * @param $decission
511     * @param $submodel
512     */
513     protected function handleUseQuestion($decission,$submodel){
514       if(is_array($submodel) && isset($submodel['key']) && substr($submodel[
          "key"],0,4) == "use#"){
515         $q = $this->use[intval(substr($submodel["key"],4))];
516         $decission->setQuestion($q);
```

```
517     }
518   }
519
520   /**
521   *
522   */
523   public function init()
524   {
525     parent::init();
526     $this->buildModel($this->model, $this);
527   }
528 }
```

## A.10. DeliveryParameter

```
1  /**
2  * Class IsWeekdayDecission
3  * @method DeliveryParameter getParameter()
4  */
5  class IsWeekdayDecission extends Decission
6  {
7
8    protected function load()
9    {
10     $this->setName("Wochentag");
11     parent::load();
12   }
13
14   public function init()
15   {
16     parent::init();
17     $this->getCompare()->isEqual(1, "Montag")
18     ->isEqual(2, "Dienstag")
19     ->isEqual(3, "Mittwoch")
20     ->isEqual(4, "Donnerstag")
21     ->isEqual(5, "Freitag")
22     ->isEqual(6, "Samstag")
23     ->isEqual(7, "Sonntag");
```

```
24      $this->getAction()->isEqual("Mittwoch", function(DeliveryParameter $p)
            {
25        $p->setOffset($p->getOffset()+1);
26      });
27    }
28
29    public function process()
30    {
31      $this->result = ((intval((new DateTime($this->getParameter()->getDate
            ()))->format('N')) + $this->getParameter()->getOffset())% 7) + 1;
32    }
33  }
```

## A.11. MonthOfYearDecission

```
1   /**
2   * Class MonthOfYearDecission
3   * @method DeliveryParameter getParameter()
4   */
5   class MonthOfYearDecission extends Decission
6   {
7
8     protected function load()
9     {
10      $this->setName("Monat");
11      parent::load();
12    }
13
14    public function init()
15    {
16      parent::init();
17      $this->getCompare()->isEqual(1, "Jänner")
18      ->isEqual(2, "Februar")
19      ->isEqual(3, "März")
20      ->isEqual(4, "April")
21      ->isEqual(5, "Mai")
22      ->isEqual(6, "Juni")
23      ->isEqual(7, "Juli")
```

```
24      ->isEqual(8, "August")
25      ->isEqual(9, "September")
26      ->isEqual(10, "Oktober")
27      ->isEqual(11, "November")
28      ->isEqual(12, "Dezember");
29    }
30
31    public function process()
32    {
33      $this->result = intval((new DateTime($this->getParameter()->getDate())
            )->format('m'));
34    }
35  }
```

## A.12. IntegerPartOfZipCode

```
1   /**
2   * Class IntegerPartOfZipCode
3   * @method DeliveryParameter getParameter()
4   */
5   class IntegerPartOfZipCode extends Decission
6   {
7
8     protected function load()
9     {
10      $this->setName("ZipCode");
11      parent::load();
12    }
13
14    public function process()
15    {
16      $this->result = intval(preg_replace('/[^0-9]/', '', $this->
            getParameter()->getZipCode()));
17    }
18  }
```

## A.13. ArrivalTree

```
1   /**
```

```
2  * Class ArrivalTree
3  * @method DeliveryParameter getParameter()
4  */
5  class ArrivalTree extends Tree
6  {
7    public function load()
8    {
9      $this->use(new IsWeekdayDecission());
10     #$this->use(new IntegerPartOfZipCode());
11     #$this->use(new MonthOfYearDecission());
12     $this->ignore("date");
13     $this->ignore("zipcode");
14     $this->ignore("offset");
15     parent::load();
16   }
17 }
```

## A.14. DeliveryParameter

```
1  /**
2  * Class DeliveryParameter
3  * @method $this setService(int $x)
4  * @method $this setDate(int $x)
5  * @method $this setFrom(string $x)
6  * @method $this setTo(string $x)
7  * @method $this setZipCode(string $x)
8  * @method $this setOffset(string $x)
9  * @method $this setCountry(string $x)
10 * @method getService()
11 * @method getDate()
12 * @method getFrom()
13 * @method getOffset()
14 * @method getTo()
15 * @method getZipCode()
16 * @method getCountry()
17 */
18 class DeliveryParameter extends Parameter
19 {
```

```
20
21 }
```

## A.15. run.php

```
1
2  $pArray = [];
3  $vArray = [];
4  if (($handle = fopen("Arrival.csv", "r")) !== FALSE) {
5    while (($data = fgetcsv($handle)) !== FALSE) {
6      if($data[0] != 'P') continue;
7      $earlier = new DateTime($data[3]);
8      $lFrom = new DateTime($data[4]);
9      $lTo = new DateTime($data[5]);
10     $from = $lFrom->diff($earlier)->format("%a");
11     $to = $lTo->diff($earlier)->format("%a");
12     if(rand(0,8) == 2 ){
13       $vArray[] = (new DeliveryParameter())
14       ->setService($data[0])
15       ->setCountry($data[1])
16       ->setZipCode($data[2])
17       ->setDate($data[3])
18       ->setOffset(0)
19       ->setResult([$from, $to]);
20     } else {
21       $pArray[] = (new DeliveryParameter())
22       ->setService($data[0])
23       ->setCountry($data[1])
24       ->setZipCode($data[2])
25       ->setDate($data[3])
26       ->setOffset(0)
27       ->setResult([$from, $to]);
28     }
29   }
30   fclose($handle);
31 }
32 $test1 = new ArrivalTree();
33 $test1->learn($pArray);
```

```
34
35
36 $test = new ArrivalTree();
37 $test->setModel($test1->getModel());
38 $test->init();
39 $error = 0;
40 $notFound = 0;
41 foreach($vArray as $p) {
42   $test->setParameter($p)->calculate();
43   if($test->getResult() !=  $p->getResult()){
44     #print("Result: " . json_encode($test->getResult()) . " vs " .
         json_encode($p->getResult()) . "\n");
45     $error +=  1;
46   }
47   if(is_null($test->getResult())){
48     $notFound++;
49   }
50 }
51 print($test->getDescription());
52 print "Error Rate: ".(($error/count($vArray)) * 100)."% ($error Fehler / "
     .count($vArray)." / $notFound nicht gefunden) overall ".count($pArray)
     .")\n";
```