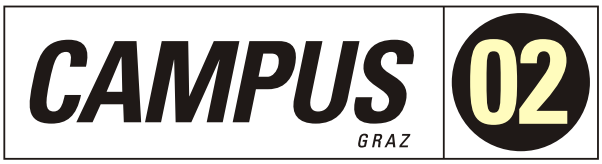
Masterarbeit

Entwicklung eines Vorhersagemodells des Arbeitsaufwandes in agilen Softwareprojekten mit Hilfe von Data- und Text Mining

ausgeführt am



Studiengang
Informationstechnologien und Wirtschaftsinformatik

Von: Dominik Leuzinger BSc MSc
Pers. Kennz. 1610320022

Graz, am 10. Dezember 2017

.....
Dominik Leuzinger BSc MSc

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....
Dominik Leuzinger BSc MSc

Danksagung

Während meiner Zeit am CAMPUS 02, sowie bei der Erstellung dieser Arbeit unterstützten mich viele Leute, denen ich hiermit meine Wertschätzung und meinen Dank ausspreche.

Zuerst gebührt mein Dank Herrn *DI Michael Neuwersch*, der meine Arbeit betreute und begutachtete und mir zu jeder Zeit mit Rat und konstruktiver Kritik zur Seite stand.

Ebenfalls möchte ich mich bei Herrn *DI Gerhard Fliess* bedanken, der mir die Datenbasis für die Durchführung der Fallstudie bereitstellte. Mit dieser Unterstützung wurde diese Arbeit überhaupt erst möglich. Vielen Dank dafür!

Ein herzliches Dankeschön gilt auch Frau *Maria Berghammer* für das Korrekturlesen und die aufmunternden Worte. Sie steuerte einen wesentlichen Beitrag zur Qualität dieser Arbeit bei.

Von ganzem Herzen möchte ich mich bei meiner gesamten Familie für den allzeit gebotenen Rückhalt bedanken. Ganz besonders möchte ich mich bei meiner Lebensgefährtin *Daniela Gröstlinger* und meinen beiden Töchtern *Marie Sophie* und *Klara Luisa* bedanken, die mich immer wieder motivierten und liebevoll unterstützten.

Dominik Leuzinger BSc MSc

Graz, am 10. Dezember 2017

Kurzfassung

Die Softwareentwicklung von Produkten und Lösungen orientiert sich zunehmend an agilen Vorgehensweisen, welche erhebliche Vorteile in der Kundenorientierung und Schnelligkeit in der Auslieferung der Lösungen mit sich bringt. In diesem Zusammenhang besteht auch die Herausforderung der Schätzung von User Stories, die aus Sicht des Benutzers, formuliert werden. Obwohl sich etwaige Methoden und Techniken zur Schätzung von User Stories etabliert haben (z.B. Point-Metriken), liegt dabei ein wesentlicher Nachteil in der subjektiven und relativen Bewertung des Aufwandes bzw. der Komplexität durch das Team durch welche eine Verzerrung der Schätzung entstehen kann. Ebenso beanspruchen Schätzpraktiken in agilen Vorgehensweisen, durch deren interaktiven Charakter, einen erheblichen Zeitaufwand.

Das Ziel dieser Arbeit bestand darin, zu prüfen, ob die Anwendung bzw. der Einsatz von Machine Learning den Schätzprozess in agilen Vorgehensweisen unterstützen kann. Der erste Teil der Arbeit bereitet dazu einen Einblick in die Welt der agilen Vorgehensweisen mit deren gängigsten Schätzverfahren. Nach einer kurzen Einführung in die Grundlagen des Data Minings bzw. Machine Learnings, wurden mögliche Lösungsansätzen zur Verbesserung der Schätzungen durch Machine Learning Verfahren untersucht. Die Untersuchung beinhaltet die Auswahl und Erläuterung eines Data Mining Vorgehensmodells. Dabei wurde der *Cross-Industry Standard Process for Data Mining (CRISP-DM)* als Referenzprozess für die Abhandlung der gesamten Arbeit gewählt. Auf Basis dieses Prozessmodells wurden einige relevante Techniken zur Textklassifizierung von User Stories abgehandelt. Demgemäß reichte die Untersuchung von der Phase *Business Understanding* bis hin zur *Evaluation* Phase eines Machine Learning Models. Basierend auf drei ausgewählten Machine Learning Verfahren (Naïve Bayes, Random Forest und Multilayer Perceptron) wurde eine Fallstudie durchgeführt. Diese Fallstudie umfasste die Datenvorverarbeitung, sowie die Modellierung und Implementierung eines Machine Learning Models auf Basis eines Real-life Datensets in der Programmiersprache *R* (Data Discovery und statistische Evaluation) und *Python* (Modellierung und Implementierung). Der letzte Teil der Fallstudie bestand in der Evaluation des trainierten Modells. Ziel der Fallstudie war es, zu prüfen, ob das Modell in der Lage ist, Story Points auf Basis von User Stories genauer zu schätzen, als durch ein zufälliges Raten bzw. Verteilen von Story Points. Dazu wurde eine ausgewählte Metrik (z.B. Accuracy) für jedes der drei Klassifizierungs-Verfahren, einem Dummy-Klassifizierer gegenübergestellt, welcher zufällig gewählte Story Points aus dem Datenset vorhersagte. Dies wurde auf Basis einer mehrstufigen Kreuzvalidierung (Cross Validation) durchgeführt um eine möglichst hohe Konfidenz der Vorhersagen zu erzielen. Anschließend wurde ein nicht-parametrischer statistischer Test (Wilcoxon Test) auf die erzeugten Vorhersagen (Metrik) angewendet, um festzustellen, ob sich

die Mittelwerte tatsächlich voneinander unterscheiden.

Die Ergebnisse der vorliegenden Studie zeigen, dass die Schätzung von Story Points durch die Anwendung von Machine Learning, das zufällige Raten erheblich übertreffen kann (unter den Voraussetzungen und Rahmenbedingungen der Fallstudie). Somit lässt sich sagen, dass Techniken des Machine Learnings für den vorliegenden Datensatz verwendet werden können, um den Schätzprozess zu beschleunigen, indem eine initiale Schätzung aller User Stories (z.B. aus einem Backlog) aus einem Machine Learning Modell bereitgestellt wird. Darüber hinaus bieten die Ergebnisse aus dem Modell, bis zu einem gewissen Grad einen unvoreingenommenen Blick auf Story-Point-Schätzungen, als Grundlage für Diskussionen innerhalb des Teams. Hierbei ist jedoch anzumerken, dass die Validität der Ergebnisse, auf den speziellen Voraussetzungen und Rahmenbedingungen der Fallstudie beschränkt sind. Daher sollten die Ergebnisse dieser Arbeit auf einer breiteren (Anzahl der Datensätze), teamübergreifenden und projektübergreifenden Datenbasis verifiziert werden. Dennoch stellt die vorliegende Arbeit einen ersten Schritt zur Nutzung des maschinellen Lernens für die Story-Point-Schätzung dar.

Abstract

Software development of products and solutions is increasingly moving towards agile development methodologies due to the huge advantages they offer in terms of customer orientation and the speed of deliverable solutions. In this context, the challenge of estimating the effort and time of user stories recorded from a user's perspective must be addressed. Although some techniques for estimating user stories (e.g. point-based metrics) have been established, the issue of relative assessment from team are quite biased and often consumes a considerable amount of time.

The objective of this thesis was to determine if machine learning techniques could be deployed to enhance the estimation process within agile software development methodologies. The first part of the thesis provides an understanding of agile development and its estimation techniques. After offering a brief overview of the basic concepts of machine learning, the possible ways in which machine learning techniques may improve estimations are examined. The investigation part consists of the selection and description of the data mining process. Here, the *Cross-Industry Standard Process for Data Mining (CRISP-DM)* was chosen as the main framework for this thesis. Based on that process model, some relevant techniques for conducting text classification of written user stories are explained. Thus, the investigation ranges from *Business Understanding* to the *Evaluation* phase of a machine learning model. Based on three selected machine learning techniques (Naïve Bayes, Random Forest and Multilayer Perceptron), a case study was conducted. The case study comprised the preprocessing

of a data set from a real-life application, as well as the modeling and implementation of a machine learning model in R (Data discovery and statistical evaluation) and Python (Modeling and implementation). The last part of the case study was the evaluation of the trained model. The aim of the case study was to prove the ability of the model to estimate story points based on user stories more accurately than random guessing. Therefore, the performance of each of the three selected Algorithms was compared with a Dummy classifier that predicted random story points for each user story in the data set. This was done by using a multi-step cross validation method to achieve a relatively high confidence for predictions. Subsequently, a non-parametric statistical test (i.e. the Wilcoxon-test) was applied to check whether their population means differ from each other.

The results of the present study indicate that story point estimation through machine learning techniques can considerably outperform random guessing with respect to the set-up of the conducted case study. This suggests that for the real-life data set at hand, machine learning techniques can be used to accelerate the estimation process by providing an initial set of estimations from a machine learning model. Moreover, to a certain degree, the method provides an unbiased view on story point estimations as a basis for discussions within the team. Here, it is important to note that the validity of the results is limited to the setting used in the case study. Therefore, the results in this thesis should be verified on a broader foundation of cross-team and cross-project settings. However, the present thesis represents a first step towards the use of machine learning for story point estimation.

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangssituation

Die Ermittlung des Arbeitsaufwandes spielt in Softwareprojekten eine entscheidende Rolle und ist ein wichtiger Bestandteil der Projektsteuerung. So hat der Arbeitsaufwand direkten Einfluss auf die Projektdauer, das zur Umsetzung benötigte Personal und die Projektkosten (Pilz, 2011). Der Aufwand kann in der Softwareentwicklung üblicherweise lediglich geschätzt bzw. sich über methodisch/algorithmische Ansätze angenähert werden. Dies impliziert, dass der geschätzte Aufwand sich in den seltensten Fällen mit dem tatsächlichen Aufwand decken wird. Demnach kommen in der Praxis unterschiedliche Schätzverfahren zum Einsatz mit dem Ziel, eine möglichst gute Approximation zur Wirklichkeit zu erreichen. Die Zuverlässigkeit der Vorhersage hängt vom jeweiligen Schätzverfahren ab. Jedoch lässt sich verallgemeinern, dass die Zuverlässigkeit von existierenden Schätzverfahren bis heute als gering einzustufen ist. (Feyhl, 2013; Pilz, 2011). Laut dem *Standish Group 2015 Chaos Report* (2015) wurden 52% aller Softwareprojekte zwar abgeschlossen, jedoch waren dabei Abweichungen zum Umfang oder zur initialen Zeit- und Kostenplanung zu verzeichnen.

Grundsätzlich können Schätzverfahren in Modell-basierte und in expertenbasierte Methoden unterteilt werden (Menzies, Chen, Hihn & Lum, 2006). Bei den modellbasierten werden algorithmische als auch nicht-algorithmische Verfahren auf bereits vorhandenen Daten aus vergangenen Projekten angewandt, um Vorhersagen für neue Projekte zu generieren, wohingegen die expertenbasierten Methoden menschliche Erfahrung als Vorhersagemaßstab verwenden (Sharma & Fotedar, 2014).

Techniken des Data Minings bzw. des Machine Learnings halten bereits Einzug in die Domäne der Softwareentwicklung, mit dem Ziel, genauere Vorhersagen zu erzeuge (Sharma & Fotedar, 2014). Die Anwendbarkeit von Data Mining in der Projektplanung, scheint nach wie vor große Herausforderungen mit sich zu bringen. So stellt unter anderem auch der Umgang mit unstrukturierten textuellen Daten ein noch sehr junges Forschungsgebiet dar. Dies ist insbesonders für das Requirements Engineering von großer Bedeutung, da hier der Großteil der Anforderungen[1] in textuellen Beschreibungen vorliegen (Palmeira, Chaves, Cavalcante & Favero, 2012). Aus den Informationen der Anforderung wird üblicherweise die Bedarfsplanung für die Umsetzung

[1] Stellvertretend für die Begriffe „User Story“, „Task“, „Requirements“, „Issue“.

gegründet. So liegt es nahe, dass die Anforderung der Quelle der Aufwandsplanung entspricht, die durch die Interpretation von implizitem als auch explizitem Wissen zu einer Aufwandsschätzung führen kann.

Hingegen würden diese Informationen von einer Maschine bzw. einem maschinellen Lernverfahren extrahiert und aufbereitet werden und folglich in einer automatisierten Aufwandsschätzung münden, wäre dies von großer Bedeutung für die Planung von Software - vorausgesetzt die Schätzung der Maschine ist genauer als jene, die durch manuelle Expertenarbeit bzw. aufwendige analytische Modelle zustande kommen würden.

Informationen aus der Anforderung können jedoch so, in der bestehenden Form des Textes, nicht direkt durch Machine Learning Modelle verarbeitet werden. Es bedarf hier der Anwendung von unterschiedlichen Techniken des Text Minings und Domänenwissens, um diese Herausforderung bewältigen zu können. Die vorliegende Arbeit sollte der Realisierung der automatisierten Vorhersage des Aufwandes aus Anforderungen ein Stück weit Rechnung tragen. Es stellt sich die Frage, wie Informationen aus Anforderungen extrahiert werden können und mit einem geeignetem Machine Learning Verfahren die Vorhersagegenauigkeit des Projektaufwands verbessert werden kann.

1.2 Aufgabenstellung

Die primäre Aufgabenstellung dieser Arbeit besteht darin, ein Machine Learning Modell zu entwickeln, welches als Prototyp für die Vorhersage des Aufwandes in agilen Softwareprojekten eingesetzt werden kann. Dabei dient der Titel und die Beschreibung einer Anforderung als Hauptquelle für das Vorhersagemodell. Die Entwicklung des Modells erfolgt auf Basis von historischen Daten aus agilen Projekten des Unternehmens XiTrust, was als Fallstudie im Rahmen der Arbeit durchgeführt wird. Die Aufgaben, die im Rahmen dieser Arbeit zu machen sind, sollten folgende Fragen beantworten:

- WARUM die Unterstützung bzw. Anwendung von maschinellem Lernen in Bezug auf die Aufwandsschätzung in Softwareprojekten von großer Bedeutung ist?
- WAS gemacht und berücksichtigt werden soll, um maschinelles Lernen für die Vorhersage einzusetzen?
- WIE der Prozess der Entwicklung eines Modells aussieht?
- WO die Grenzen des Modells liegen?

Eine detaillierte Ausführung der Aufgaben bzw. der Zielsetzung und Forschungsfrage dieser Arbeit wird im nachfolgenden Abschnitt erläutert.

1.3 Zielsetzung der Arbeit

Das Ziel der Arbeit ist es, ein Machine Learning Modell zu entwickeln, welche genauere, d.h. Vorhersagen zu machen, die näher an den tatsächlich verbuchten Aufwänden liegt. Da es bereits empirisch gut dokumentierte Versuche gibt, bei welcher die unterschiedlichsten Machine Learning Verfahren auf unterschiedliche Datensets aus der Softwareentwicklung Anwendung fanden, gilt es, jene Verfahren zu identifizieren, welche auf unser Datenset aus der unternehmerischen Praxis am geeignetsten sind. Der Schwerpunkt der Arbeit bzw. die größte Herausforderung liegt jedoch in der Extraktion von Parametern (sog. „Features“) aus unstrukturiertem Text. Extrahierte Informationen in Form von Feature-Vektoren können so mit bzw. durch einem Lern-Modell trainiert werden. Es wird angenommen, dass die extrahierten Features aus der textuellen Beschreibung einer Anforderung zu einer genaueren Vorhersage des Aufwandes führt, als wenn diese durch Zufall oder gar durch einen Menschen getroffen wurden. Letzteres wird im Rahmen dieser Arbeit nicht beleuchtet, jedoch ist es notwendig festzustellen, ob das konstruierte Modell überhaupt sinnvoll anwendbar und plausibel hinsichtlich der Vorhersageergebnisse ist.

Aus der Beschreibung von oben ergibt sich folgende **Forschungsfrage**, die im Rahmen der Arbeit beantwortet werden soll:

1. ***Wie kann eine Vorhersage durch ein Mining Modell zur Aufwandsschätzung in agilen Softwareprojekten aus einer textuellen Anforderungsbeschreibung getroffen werden, um die Schätzgenauigkeit zu erhöhen?***

   Im Zusammenhang mit dieser Forschungsfrage lassen sich **3 Ziele** ableiten:

   a) Evaluierung und Auswahl von Machine Learning Algorithmen zur Anwendung auf das vorhandene Datenset

   b) Anwendung von Text Mining und NLP Verfahren zur Extraktion von Features aus der Anforderungsbeschreibung im vorhandenen Datenset

   c) Implementierung und Validierung eines ausführbaren Mining-Modells auf Basis des vorhandenen Datensets zur Vorhersage des Arbeitsaufwandes

1.4 Vorgehen und Methodik

Zur Beantwortung der Forschungsfrage sollten geeignete Machine Learning Verfahren aus der Literatur identifiziert werden und als Input für die Fallstudie der Arbeit dienen. Im empirischen Teil der Arbeit erfolgt die Umsetzung und Validierung des/der Modell(e) mit einem Data Mining Framework (Beschreibung siehe Kapitel 6). Die Ergebnisse der Validierung werden einer Plausibilitätsprüfung unterzogen. Ein Modell gilt dann als plausibel und für das vorhandene Datenset einsetzbar wenn, die Vorhersagen durch das Modell genauer sind als jene, die durch zufälliges Schätzen zustande gekommen sind (Choetkiertikul et al., 2016).

Demnach ist auf dieser Basis ein Plausibilitätscheck durchzuführen. Bei diesem sollten die konstruierten Modelle den Ergebnissen eines *Random Guessings*, gegenübergestellt werden. Die genauere Beschreibung des Plausibilitätscheck bzw. dessen Operationalisierung erfolgt in Kapitel 6 dieser Arbeit.

Mit der Einführung des Plausibilitätscheck lassen sich von der Forschungsfrage abgeleitet eine Arbeitshypothese (H1) mit deren Nullhypothese (H0) aufstellen, die es zu falsifizieren gilt:

H0: Durch Feature Extraktion aus textuellen Anforderungsbeschreibungen kann die Schätzgenauigkeit des Aufwandes durch ein Mining Modell, im Vergleich zu einem „Random Guessing" des Aufwandes, NICHT erhöht werden.

H1: Durch Feature Extraktion aus textuellen Anforderungsbeschreibungen kann die Schätzgenauigkeit des Aufwandes durch ein Mining Modell, im Vergleich zu einem „Random Guessing" des Aufwandes, erhöht werden.

Nachfolgend wird der Aufbau der Arbeit genauer erläutert.

1.5 Aufbau der Arbeit

Zu Beginn der vorliegenden Arbeit erfolgt eine Einführung in das Thema des Software-Projektmanagements bzw. dessen Herausforderungen in der Praxis (*Kapitel 2*). Dabei wird im Speziellen auf agile Vorgehensweisen mit deren Aufwandsschätzungs-Praktiken eingegangen, um ein tieferes Verständnis für die Bearbeitung der Fallstudie zu erlangen. In *Kapitel 3* werden die wichtigsten Grundlagen des Data Minings bzw. Machine Learnings erläutert. Darauf aufbauend wird in *Kapitel 4*, das Klassifizierungs-Problem näher beleuchtet und dabei auf spezifischen Charakteristika der Modellierung und Evaluierung eingegangen. Das Feature Engineering von unstrukturiertem Text stellt einen zentralen und entscheidenden Teil dar, welcher aufgrund des Umfanges

und Komplexität separat in *Kapitel 5* abgehandelt wird. Die Fallstudie in *Kapitel 6* stellt den empirischen Teil der Arbeit dar. Dabei wird das erworbene Wissen aus den theoretischen Ausführungen der Arbeit als Grundlage genutzt und mit einem prototypischen Modell versucht, die Forschungsfrage anhand eines Datensets aus der unternehmerischen Praxis zu beantworten.

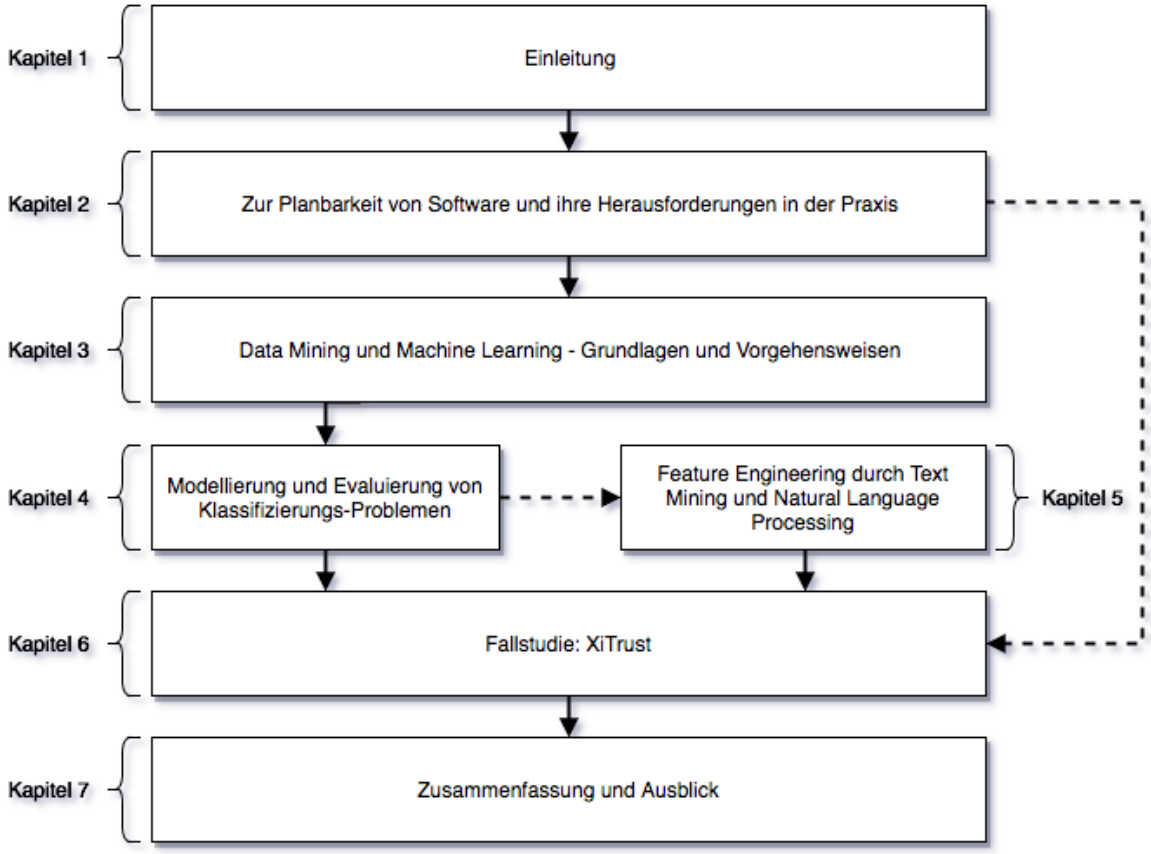


Abbildung 1.1: Aufbau der Arbeit

Im abschließende *Kapitel 7* werden die gewonnenen Erkenntnisse zusammengefasst und kritisch gewürdigt. Dazu gehört ebenso die Ausführung von Anknüpfungspunkten für zukünftige Forschungsarbeiten. In Abb. 1.1 ist nochmals der soeben beschriebene Aufbau grafisch dargestellt.

2 Zur Planbarkeit von Software und ihre Herausforderungen in der Praxis

2.1 Die Herausforderungen der Softwareentwicklung

Die Entwicklung von Software hat, damals wie heute, ein übergeordnetes Ziel: Die Entwicklung eines lauffähigen Systems, welches die Erfüllung bzw. Unterstüzung ein oder mehrerer bestimmter *Geschäftsziele* ermöglichen soll! Dieser Grundsatz gilt als das oberste Gebot eines gesamtheitlichen Verständnisses von Softwareentwicklung. Es werden aus den Geschäftszielen *Anforderungen* abgeleitet, die ein System erfüllen soll. Aus den Anforderungen wird ein *System* entwickelt, das diese Anforderungen umsetzt (Bleek & Wolf, 2011). So einfach wie das klingen mag, so herausfordernd sind die *Transformationsprozesse* (Softwareentwicklung i.e.S.) zwischen den einzelnen Teilschritten Geschäftsziel-Anforderungen-System. Das zu erstellende System wird als *Artefakt*[1] abgebildet und kann auch auf Basis der Anforderungen direkt getestet werden. Als Beispiel dafür kann der *Systemtest* genannt werden, welcher im wesentlichen die *funktionalen Anforderungen*[2], als auch die *nicht-funktionalen Anforderungen*[3] auf Erfüllung testen soll.

Für den Transformationsprozess von den Geschäftszielen zur Anforderung gibt es derzeit laut Bleek und Wolf (2011) keine direkte Möglichkeit. Es kann jedoch durch den Einsatz des Systems im Feld überprüft werden, ob die Geschäftsziele erfüllt werden können. Als Beispiel dafür kann der *Abnahmetest*, oder auch der *Akzeptanztest*[4], genannt werden. Beide Begriffe werden oft synonym verwendet. Dabei handelt es sich um die Überprüfung von *Akzeptanzkriterien*, die durch den Kunden zu Beginn des Prozesses artikuliert werden. Demnach können diese Akzeptanz- bzw. Abnahmekriterien als eine Sammlung von Geschäftsregeln für die Verwirklichung der Geschäftsziele des Auftraggebers gesehen werden, welche er auf Basis des Systemeinsatzes bei Erfül-

[1] Ein für den Menschen verständliches und als solches wahrnehmbares Produkt oder Zwischenergebnis aus der Softwareentwicklung

[2] Beschreiben die geforderten Funktionalitäten an ein System, d.h. was soll das System leisten?

[3] Beschreiben die Qualität mit der die Anforderungen umgesetzt werden sollen

[4] Kriterien die aus der Sicht des Kunden/Users erfüllt sein müssen, damit eine Story als erledigt gilt

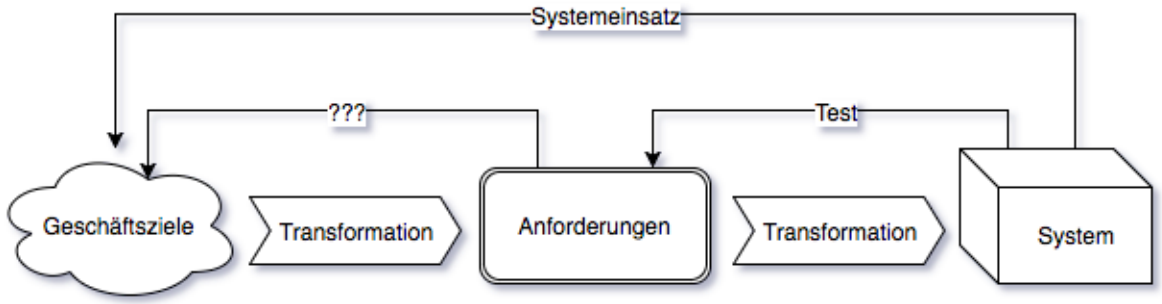


Abbildung 2.1: Geschäftsziel - Anforderungen - System (Bleek & Wolf, 2011)

lung bestätigt (Wirdemann, 2017). Abbildung 2.1 soll den Zusammenhang nochmals verdeutlichen.

Selbstverständlich ist dieses grundsätzliche Verständnis und dieser Ablauf eines Softwareentwicklungsprozesses im weiteren Sinn wie oben dargelegt nicht so trivial. Es ist geprägt von einer Reihe von Herausforderungen, die sich wie folgt darstellen lassen:

Die Geschäftsziele eines Unternehmen stehen unter dem Einfluss sich laufend ändernden *Rahmenbedingung* (z.B. Konjunktur, Wettbewerb, etc.), an denen Unternehmungen laufend gefordert sind sich anzupassen (Diederichs, 2005). So steht dies im Wettbewerb zur Anforderung, schnell qualitativ hochwertige Software auszuliefern.

Wurde die IT ursprünglich als Treiber für *Kosteneinsparungen* durch Prozessverbesserungen, so ist sie heute selbst zum Objekt von Kosteneinsparungsprogrammen in Unternehmen geworden. Die Entwicklung und der Betrieb von IT Systemen werden zunehmend hohem Kostendruck ausgesetzt. (Diederichs, 2005)

Weiters sind *IT-Landschaften* in Unternehmen wie wir Sie heute vorfinden, meist nicht durch einen Big Bang entstanden, sondern haben sich großteils evolutionär entwickelt. Das heißt, dass Softwarekomponenten, die vor Jahren entwickelt worden waren um – großteils auch heute noch – geschäftsrelevante Funktionen auszuführen, ständig gewartet und weiterentwickelt werden müssen.(Diederichs, 2005)

Eine weitere Herausforderung und ein stetig zunehmender Faktor ist die *Heterogenität* von IT Systemen (Diederichs, 2005). Dabei werden IT Systeme bzw. deren Komponeneten immer stärker vernetzt, um ein Geschäftsziel zu realisieren. Insbesondere haben sich in den letzten Jahren neue Architektur-Prinzipien etabliert, die auf einer „losen" Kopplung unterschiedlicher Teilsysteme basieren - wie beispielsweise bei *Microservices*[5]. Damit nimmt insgesamt der Komplexitätsgrad beim Design, Entwicklung und Betrieb von IT-Applikationen zu (Newman, 2015).

[5] Architekturmuster des Software Engineerings bei der umfangreiche Software-Anwendungen aus kleinen, voneinander unabhängigen Teilprogrammen via sprachenunabhängigen Schnittstellen kommunizieren

Mit den oben angeführten und größten Herausforderungen der Softwareentwicklung im weiteren Sinn, nimmt die Komplexität des Systems bzw. der Systeme bereits überhand, so dass die Entwicklung solch komplexer Systeme für den Menschen kaum noch überschaubar sind. Dies wirkt sich bereits in der Anforderungsermittlung aus, wo zu Beginn einer Entwicklung die *Stakeholder* selbst oft im unklaren sind, welche Anforderungen an ein System gestellt werden sollen, um das Geschäftsziel zu unterstützen oder gar zu realisieren (Pohl & Rupp, 2015). Bis hin zur Entwicklung und Betrieb der Systeme, wo ein erheblicher Teil von Projekten nicht einmal abgeschlossen wurden, geschweige denn im Budget- und Zeitrahmen gehalten werden konnten. Wie bereits in der Einleitung dieser Arbeit erwähnt, wurden im Jahr 2015 lediglich 52 Prozent der Projekte abgeschlossen, zudem mit einer Abweichung zum initial geplanten Zeit- und Kostenbudget (*Standish Group 2015 Chaos Report*, 2015).

Diese bis heute noch eher chaotischen Zustände hinsichtlich der Planung in der Softeareentwicklung geben Anlass dazu, sich verstärkt mit Lösungen zur „Bekämpfung“ der Komplexität zu beschäftigen. Selbstverständlich waren die Experten nicht untätig und brachten eine Vielzahl an Lösungsansätzen hervor. So wurde beispielsweise die Hochsprachenprogrammierung etabliert, integrierte Entwicklungsumgebungen geschaffen und eine Vielzahl an Projektmanagement-Methoden entwickelt (Brooks, 1987). Eine allgemeine Rezeptur zur Auflösung der *Komplexität* und *Volatilität* in Softwareprojekten blieb bis heute aus (Diederichs, 2005).

Im weiteren Verlauf werden Projektmanagement-Methoden in der Softwareentwicklung – insbesondere die der agilen Planungsmethoden – beleuchtet, welche vorwiegend den Herausforderungen der oben genannten Volatilität Rechnung tragen soll.

2.2 Vorgehensmodelle in der Softwareentwicklung

Dieser Abschnitt widmet sich der Betrachtung von Vorgehensmodellen in bzw. für die Softwareentwicklung, mit Schwerpunkt auf *agile Modelle* und deren grundsätzliches Vorgehen. Dabei wird die Methode *Scrum* näher erläutert, um ein Verständnis der Methode und dessen Begrifflichkeiten im Kontext der Forschungsfrage dieser Arbeit zu erlangen. Zudem gilt es, existierende Aufwandsschätzmethoden zu beleuchten und speziell auf jene von Scrum näher einzugehen.

2.2.1 Zu den Vorgehensmodellen in der Softwareentwicklung

Um IT-Projekte zielgerichtet abwickeln zu können, bedarf es Vorgehensmodelle, welche einen standardisierten Ablauf aufweisen. Der standardisierte Ablauf sollte sicher stellen, dass keine Teilschritte ausgelassen werden (Berg, Knott & Sandhaus, 2014). Im

Kontext des *eisernen Dreiecks* (oft auch *goldenes-*, oder *magisches Dreieck* genannt) des Software-Projektmanagements gesprochen: Möchte man in der Lage sein, die Dimensionen *Anforderung*, *Kosten* und *Zeit* als übergeordnete Zieldimensionen zu verfolgen, so ist es notwendig einen einheitlichen Ablauf hinsichtlich der Entwicklungsschritte, aber auch der Planungs- und Kontrollschritte einzuhalten. Vorgehensmodelle bilden genau diesen Rahmen, um sich bei der Entwicklung von Software nach einem praxiserprobten Ablauf zu richten. Grundsätzlich lassen sich Vorgehensmodelle in *phasenorientierte* - und in *agile Vorgehensmodelle* unterteilen (Berg et al., 2014). Auf die Unterschiede wird im folgenden näher eingegangen.

2.2.1.1 Phasenorientierte Vorgehensmodelle

Bei phasenorierten – oftmals auch *„klassische"* Vorgehensmodellen genannt – handelt es sich um vormals schwergewichtige Prozessmodelle. Diese wurden bereits in den frühen 70er Jahren aus anderen ingenieurswissenschaftlichen Disziplinen auf die Softwareentwickung übertragen, um auch dort ein standardisiertes Vorgehen für die Entwicklung von Software in Unternehmen zu etablieren (Weiss & Tan, 2004).

Grundsätzlich findet bei diesen phasenorientierten Modellen die Unterteilung der gesamten Systementwicklung in *Projektphasen* statt, welche zeitlich bestimmt und durch Meilensteine voneinander abgegrenzt sind. Im Laufe der Jahre wurde eine Vielzahl an unterschiedlichen Vorgehensmodellen entwickelt, welche die schwergewichtigen Prozessmodelle durch *Tailoring*[6] auf Prozesse handhabaren Niveaus verjüngte und diese dadurch praxistauglich wurden. Ebenso wurden Modelle evolutionär weiterentwickelt und erprobt, um dabei den Schwächen bzw. Nachteilen des zu dem Zeitpunkt bestehenden Referenzmodells entgegenzuhalten. So sind bis heute die phasenorientierten Modelle noch am weit verbreitesten in der unternehmerischen Praxis (Berg et al., 2014).

Zu den wohl bekanntesten und zugleich ursprünglichsten Vertretern der klassischen Modelle gehören das *Wasserfallmodell* und das *V-Modell*. Beim Wasserfallmodell handelt es sich um Vorgehensmodell, das bereits in den 1970er als Referenzmodell für die Abwicklung von großen Softwareprojekte ins Leben gerufen worden war (Royce, 1987). Dem Modell liegt zu Grunde, dass, wie der Name schon sagt, die einzelnen Projektphasen nacheinander von oben nach unten abgearbeitet werden und die Ergebnisse jeder Phase als Eingabe der nächsten dient. Abbildung 2.2 veranschaulicht das Prinzip des Wasserfallmodells. Es wird dabei nicht näher auf die einzelnen Phasen eingegangen, zumal dieses Modell in Literatur und Praxis unterschiedlich ausgeprägt und eingeteilt wird, so dass keine zwei identischen Beschreibungen des Modells existieren.

[6] Anpassung eines Vorgehensmodells oder vorgegebener Methoden an den tatsächlichen Anforderungen im individuellen Unternehmenskontext

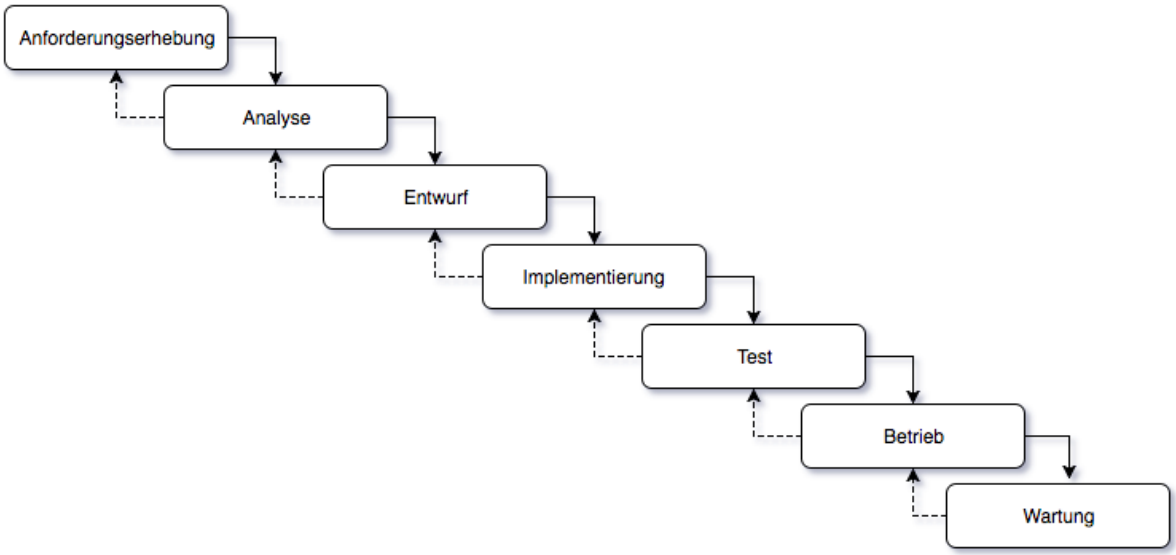


Abbildung 2.2: Typische Phasen des Wasserfall-Modells (Royce, 1987)

Diese einzelnen Phasen des Modells werden in einem Entwicklungsprojekt nacheinander und meist nur einmalig durchlaufen (Berg et al., 2014). Dadurch wird vermieden, dass Aufgaben nicht mehrmals in unterschiedlichen Phasen wiederholt werden und damit eine theoretisch maximale Effizienz erreicht wird (Moll et al., 2004). Diese Vorgangsweise birgt aber auch etliche Risiken. Eines der größten Missstände verursacht die Tatsache, dass Anforderungen, welche Stakeholder an ein System stellen und zu einem bestimmten Zeitpunkt artikulieren, sich grundsätzlich im Laufe der Zeit verändern bzw. mit dem System „wachsen'. So kommt es im speziellen bei großen Projekten, die sich über einen langen Zeitraum erstrecken, dass die Anforderungsphase nie wirklich abgeschlossen sein kann, bevor die Umsetzung beginnt (Moll et al., 2004). Ein weiterer bedeutender Nachteil ist u.a., dass kaum Möglichkeit besteht, auf die Schätzung des Aufwandes und in weiterer Folge der Kosten zu reagieren. Das Wasserfallmodell läßt in erweiterten Versionen Rücksprünge auf die davorliegenden Phase zu, jedoch lässt sich diese Möglichkeit auch als erfolglos enttarnen, wenn zur Behebung des eigentlichen Fehlers zwei bis drei Rücksprünge oder gar bis zur Anforderungsanalyse zurückgekehrt werden muss – was dann meist nur mit enormen Aufwand verbunden ist. Demnach werden Risikofaktoren im Projekt kaum Beachtung geschenkt und keine effizienten Korrekturprozesse dafür vorgesehen (Brandstäter, 2013). Zudem hat sich der hohe Dokumentationsaufwand nach jeder Phase und die mangelnde Partizipationsfähigkeit der Stakeholder/User im Prozess als nachteilig herausgestellt (Weiss & Tan, 2004).

Man hat schon sehr bald erkannt, dass es für die Entwicklung von Software einer iterativen Vorgehensweise bedarf und so wurden Modelle wie beispielsweise das *Spiralmodell* entwickelt (Moll et al., 2004). Ebenso eine Weiterentwicklung des Wasserfallmodell stellt das *V-Modell* dar, welches um Qualitätssicherungsmaßnahmen erweitert wurde (Berg et al., 2014). All diese Modelle stoßen zunehmend an die Grenzen wenn

es darum geht, in einem sich dynamisch veränderndem Marktumfeld nutzen-stiftende und funktionierende IT-Applikationen in kurzen zeitlichen Abständen auszuliefern.

Die Ausführungen oben sollten keineswegs die klassischen Modelle totsagen. Es sollte lediglich aufgezeigt werden, dass diese Ansätze es mit den Herausforderungen, die unter anderem in Abschnitt 2.1 beschrieben worden sind, großteils nicht mehr Schritt halten können. So haben die phasenorientierten Vorgehensmodell durchaus deren Berechtigung wenn *vorhersagbare Anforderungen*, *Stabilität des Prozesses* und *hohe Sicherheit* die Rahmendedingungen und Ziele eines Entwicklungsvorhabens sind. Zudem sind sie zur Steuerung *komplexer Strukturen* mit *weniger Experten* gut geeignet (Boehm & Turner, 2009).

In den 90er Jahren haben sich dazu agile Vorgehensweisen entwickelt. An dieser Stelle wurde bewusst das Wort „Vorgehensweisen“ anstatt „Vorgehensmodelle“ verwendet, da es sich dabei um *Richtlinien* oder *Regelwerke* handelt wie Softwareprojekte abgewickelt werden, nicht wie Prozesse im klassischen Sinn gesehen und gehandhabt werden (Bleek & Wolf, 2011; Berg et al., 2014).

Nachfolgend werden die agilen Methoden, insbesondere Scrum, ausführlicher behandelt.

2.2.1.2 Agile Vorgehensweisen

Wie zuvor erläutert, haben sich aus den Herausforderungen andere Herangehensweisen, ja gar neue Wertvorstellungen und Sichtweisen entwickelt. Bereits in den 90er Jahren hatten eine Vielzahl von Experten agile Methoden für das Management von Softwareprojekten kreiert und praktiziert. Eine davon ist Scrum, auf welche später noch genauer eingegangen wird. Vorerst ist aber zu sagen, dass alle agilen Vorgehensweisen einem bestimmten *Wertesystem* zugrunde liegen, auf dem die *agilen Prinzipien* und *agilen Methoden* aufbauen. Nach Bleek und Wolf (2011) sind dabei wesentliche Wert-Grundlagen:

- *Kommunikation* - Direkt, offen und ehrlich
- *Einfachheit* - Technisch, organisatorisch und methodisch
- *Feedback* - Rückkopplungen sind erwünscht
- *Mut* - Sich trauen, andere Wege zu gehen
- *Respekt* - Voraussetzung für offene, ehrliche und angst-freie Kommunikation

Alle agilen Methoden haben ein übergeordnetes Ziel, das heißt, schnell, flexibel Software in hoher Qualität auszuliefern. Im Jahr 2001 kam es zu einem Treffen von Experten bzw. Vertretern von unterschiedlichen agilen Verfahren die ein *Agiles Manifest* (in Englisch: *Agile Manifesto*) formulierte (*Manifesto for Agile Software Development*, 2001):

„Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

- *Individuen und Interaktionen stehen über Prozessen und Werkzeugen*
- *Funktionierende Software steht über einer umfassenden Dokumentation*
- *Zusammenarbeit mit dem Kunden steht über der Vertragsverhandlung*
- *Reagieren auf Veränderung steht über dem Befolgen eines Plans*

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein."

Weiters einigten sich die Vertreter auf 12 Leitprinzipien des agilen Software-Projektmanagement dienen sollen. An dieser Stelle wird auf das *Manifesto for Agile Software Development* (2001) verwiesen, sowie auf weiterführende Literatur von Cohn (2009) und Leffingwell (2010) zur Vertiefung der Prinzipien.

Doch worin unterscheiden sich agile Vorgehensweisen und klassische Ansätze denn nun genau? Wie bereits erwähnt wurde, gibt es ein Manifest und Prinzipien die dem agilen Vorgehen zugrunde liegen. Weiters gibt es agile Techniken und Methoden die im agilen Projektmanagement Anwendung finden. Letztere unterscheiden sich zwar in der Methodik von klassischen Ansätzen, sind jedoch nicht die wesentlichen Unterschiede der zwei Entwicklungsparadigmen. Es gibt eine Vielzahl von Unterschieden, welche Preußig (2015) ausführlich beschreibt und zusammenfasst. Auf einige wenige Punkte jedoch sollte näher eingegangen werden, welche als Vorbereitung und zum besseren Verständnis von Scrum dienen soll und zugleich Gültigkeit für alle agilen Projektmanagement-Methoden hat.

Eines der wichtigsten Ziele des agilen Projektmanagements ist es, dass die Stakeholder nicht nur am Anfang bzw. zum Projektstart eingebunden werden, sondern während des gesamten Projektverlaufs Einfluss auf die Anforderungen nehmen können und aktiv ins Projektgeschehen eingebunden werden (Cohn, 2009; Leffingwell, 2010). Es werden zyklisch Zwischenergebnisse den Stakeholdern ausgeliefert und präsentiert, anstatt sich alleine auf das Endergebnis zu fokussieren – wie es bei klassischen Ansätzen grundsätzlich der Fall ist. Dies wiederum bezogen auf das eiserne Dreieck des Projektmanagements (Anforderung, Kosten, Zeit), so stehen alle Dimension in Abhängigkeit zueinander (Preußig, 2015). Wird beispielsweise eines der Ziele geändert, so wirkt sich das direkt auf die anderen zwei Dimensionen aus. Dies könnte

bedeuten, dass, wenn eine Funktionalität oder Anforderung mehr umgesetzt werden soll, steigen entweder die Kosten bei Fixierung des Termins oder der Termin wird verschoben bei Fxierung der vereinbarten Kosten. Bei klassischen Ansätzen wird als oberstes Ziel üblicherweise die Anforderung die es umzusetzen gilt fixiert und die anderen Dimensionen danach gestellt. Der agile Ansatz stellt diese Idee auf den Kopf: Es werden meist die Anforderungen als Variable in diesem Dreieck behandelt (siehe Abb. 2.3). Kosten und Zeit werden fixiert. Das heißt, anstatt einen Termin für die Zwischenpräsentation zu verschieben, wird in Abstimmung mit dem Stakeholder der Umfang der Ergebnise für den betreffendne Zyklus reduziert (Preußig, 2015).

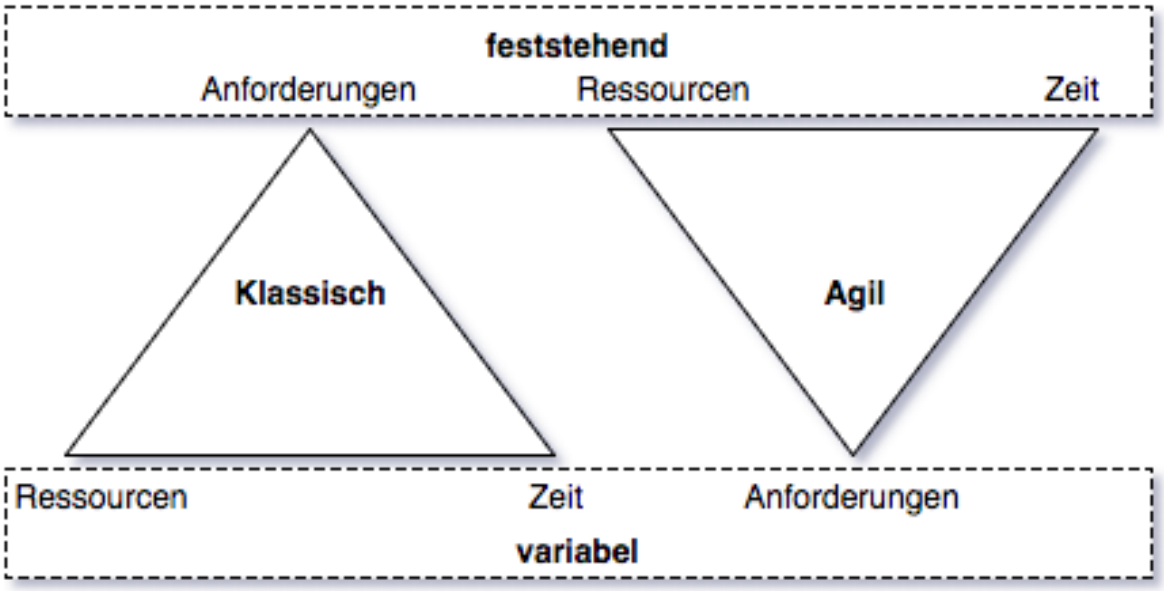


Abbildung 2.3: Gegenüberstellung der Ziele von klassischen vs. agilen Ansätzen

Ein Vorteil der sich daraus ergibt ist, dass sich die Kundenwünsche über den Projektverlauf nicht anhäufen, sondern durch diesen quasi *Priorisierungs-Mechanismus*[7] die tatsächlichen Wünsche der Stakeholder umgesetzt werden und damit unnötige Entwicklungsarbeit vermieden wird (Preußig, 2015). Damit wird in *zyklisch* und relativ *kurzen Abständen* ein tatsächlicher *Mehrwert* für die Stakeholder realisiert (ebd.). Agile Vorgehensweisen sind demzufolge *Wert-getrieben* im Vergleich zu klassischen Ansätzen, welche sich durch eine *Plan-getriebene* Vorgehensweise auszeichnen (Leffingwell, 2010).

Zusammenfassend kann gesagt werden, dass agile Methoden den Ansatz verfolgen, die *Komplexität* und *Unsicherheiten* im Zusammenspiel Geschäftsziel-Anforderung-System zu reduzieren. Dies wird erreicht durch *inkrementell-iteratives Vorgehen* (kleine Releases in kurzen Abständen), einen *kooperativen Projektrahmen* (Stakeholder und Auftragsnehmer arbeiten eng und während des gesamten Projektverlaufs zusammen),

[7] Was brauche ich als Stakeholder sofort bzw. was möchte ich sofort sehen/probieren um es mit meinen Zielen und Vorstellungnen abzugleichen?

einfache Methoden (leicht zu verstehen und einfach zu erlernen) und die Fähigkeit im Team zu etablieren, *adaptiv* auf (Ver-)Änderungen eingehen zu können.

Nachfolgend werden die grundlegenden Konzepte der agilen Projektmanagement-Methode Scrum erläutert.

2.2.2 Scrum - Ein Überblick

In diesem Abschnitt wird ein Überblick über Scrum gegeben. Das Vertraut machen mit den Schlüsselprinzipien und Konzepten, aber insbesondere mit dem Ablauf der Scrum Vorgehensweise (*Scrum Flow*) gilt für die vorliegende Arbeit als essentiell, da die Untersuchung in der Fallstudie im Rahmenwerk von Scrum eingebettet ist. Folglich trägt dieser Abschnitt wesentlich zur Beurteilung der Ergebnisse der Arbeit bei, indem Beurteilungskompetenz darüber erlangt wird, ob das Vorhersagemodell überhaupt den Ansprüchen des Rahmenwerks gerecht werden kann. Falls dies der Fall ist, gilt es zu beurteilen, wo und wie die Ergebnisse im Scrum Flow genutzt werden können.

Scrum ist als *Rahmenwerk* für agile Softwareentwicklung zu verstehen und bietet ein Regelwerk und konkrete Methoden um agile Softwareentwicklung zu praktizieren (Schwaber, 2004). Scrum ist die weit verbreitetste agile Methode, gefolgt von *Extreme Programming* (Berg et al., 2014). Unternehmen berichten aus der Praxis, dass *Produktivitätssteigerung* durch erfolgreich eingeführtes Scrum zu verzeichnen war (Gloger, 2014; Leffingwell, 2010). Der Erfolg von Scrum bzw. dessen Produktivitätssteigung lässt sich laut Gloger (2014) folgendermaßen beschreiben:

- Es arbeiten *cross-funktionale Teams* zusammen und stimmen sich direkt und sofort ab.
- Das Team kann sich auf *eine Aufgabe* bzw. ein Projekt konzentrieren.
- Die *Verwaltung* und *Dokumentation* kann auf ein Minimum reduziert werden
- Klar definierte *Verantwortlichkeiten*
- *Stetige Transparenz* der Ergebnisse, so dass sofort eingegriffen werden kann, wenn etwas nicht erwartungsgemäß abläuft oder unerwartet eintritt.

Im nächsten Schritt wird genauer auf die Prinzipien von Scrum eingegangen.

2.2.2.1 Das Team

Ein Scrum-Team besteht in der Regel aus sieben Personen. Davon ist eine Person der *Scrum Master* und ein anderer der *Product Owner*. Der Rest des Teams sind Entwickler - das *Entwicklungsteam*. Nach Wirdemann (2017) und Gloger (2014) lassen sich die Rollen folgendermaßen beschreiben:

Das Entwicklungsteam arbeitet *autonom*, d.h. sie organisieren die Aufgaben selbst. Sie tauschen Wissen untereinander aus und helfen sich gegenseitig und stehen im ständigen, direkten Kontakt zueinander. Sie sind verantwortlich für die Lieferung des Produktes bzw. der *Produktinkremente*. Das Team steuert selbst den Umfang den sie bewältigen können und stehen damit auch für Ergebnis und Qualität gerade.

Der Scrum Master schafft und organisiert die *Rahmenbedingungen* für das Entwicklungsteam. Er ist die organisatorische Ansprechperson des Entwicklungsteams. Der Scrum Master sorgt dafür, dass das Scrum Regelwerk eingehalten wird und schützt das Entwicklungsteam vor Einflüssen von außen, die fürs Team hinderlich sind.

Der Product Owner ist jene Person, die das Entwicklungsteam aus *fachlicher Sicht* steuert und die Produktentwicklung *strategisch* führt. Er entscheidet über die fachliche Umsetzung bezogen auf den Inhalt bzw. Funktionalität, als auch die zeitliche und budget-gerechte Umsetzung. Der Product Owner hat zudem noch die Rolle der fachlichen Schnittstelle zum Auftraggeber – also dem Kunden und zu den Users, den Anwendern des Produktes.

2.2.2.2 Der Ablauf in Kürze

Aussgangsbasis für die Entwicklung ist eine *Vision*. Sie dient als übergeordnete Vision, als strategisches Herzstück des Produktes. Die Umsetzung der Vision in das Produkt erfolgt über die Formulierung von Eigenschaften aus der Perspektive der User. Diese Verschriftlichung der Anforderung werden *User Stories* genannt. Um die User Stories werden dann auch Aggregationen formuliert, die mehrere User Stories zusammenfassen (*Epics*). Ebenso wird in die andere Richtung verfeinert, nämlich User Stories in tatsächliche *Tasks* aufgeteilt, wenn es zur Umsetzung kommt. (Wirdemann, 2017; Cohn, 2009; Leffingwell, 2010)

Was im Detail hinter einer User Story steckt, wird später noch genauer ausgeführt.

So dienen User Stories dem Auftraggeber bzw. User, wertschaffende Eigenschaften, die das Produkt aufweisen sollen, zu formulieren, welche dann vom Product Owner in einem *Product Backlog* festgehalten werden. Das Product Backlog dient, wie eine Liste, als Übersicht und Priorisierungswerkzeug für den Product Owner. So werden User

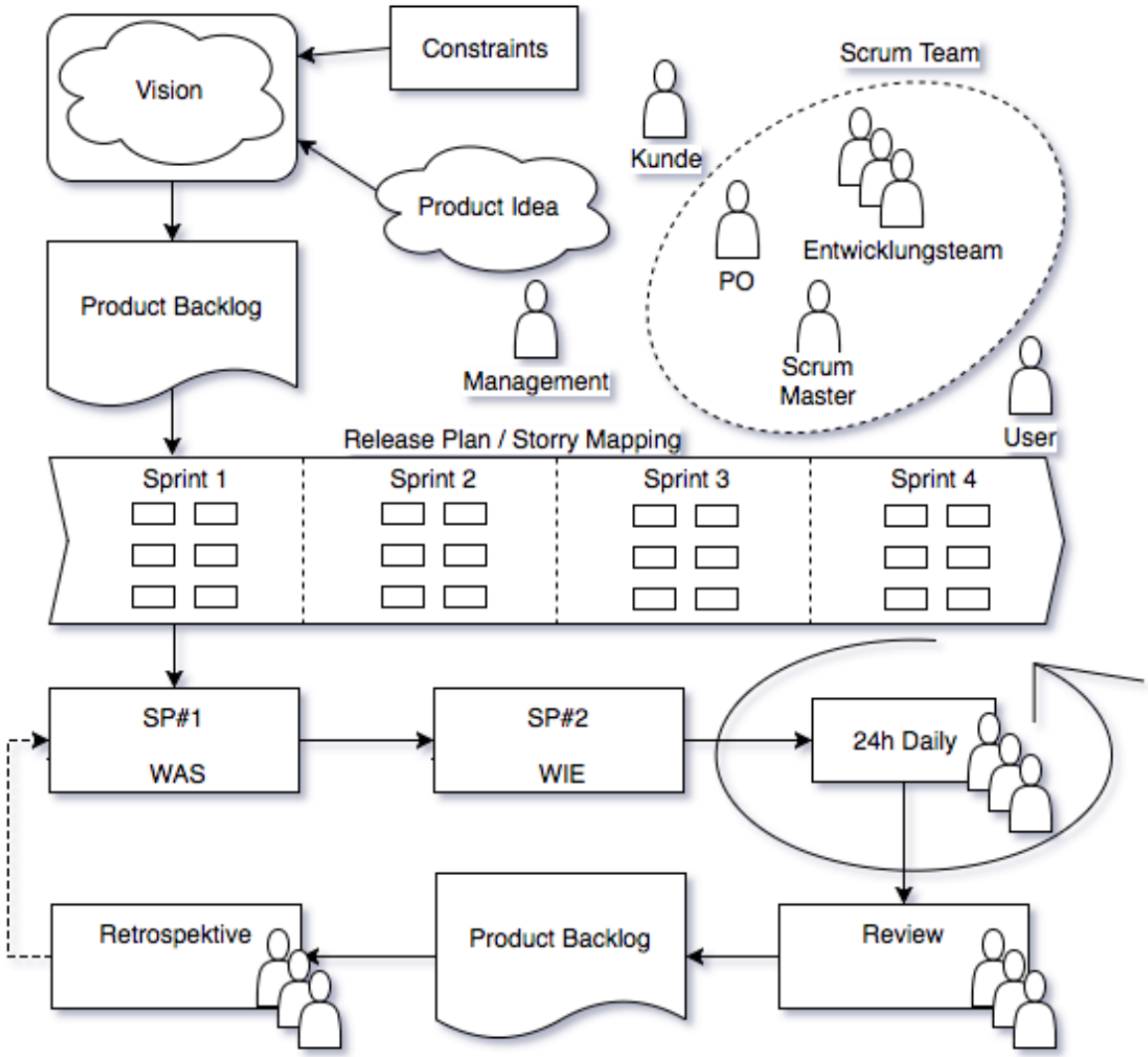


Abbildung 2.4: Der Basic Scrum Flow (Gloger, 2014)

Stories hinsichtlich vom Team in einem *Estimation Meeting* geschätzt und von dort in *Sprints* eingesteuert, welche wiederum in ein *Releaseplan* eingebettet werden. Für die Übertragung von User Stories in einen Sprint – also in die operative Umsetzungsphase – gibt es ein erstes *Sprint Planning*, in dem gemeinsam im Team festgelegt wird, welche User Stories in dem bevorstehenden Sprint umgesetzt werden sollen. Diese Stories werden dann in ein *Sprint Backlog* übertragen und dienen als quasi Arbeitsplan für das Team im Sprint. Danach erfolgt das zweite Sprint Planning, wo das Team gemeinsam mit dem Scrum Master Details klärt, wie die User Stories schlussendlich umgesetzt werden sollen (Applikation, Architektur, etc.). (Wirdemann, 2017; Cohn, 2009; Gloger, 2014)

Nun geht es in den Sprint. Der Sprint ist ein bis zu vier Wochen langes Intervall, wo genau das zuvor festgelegte Sprint Backlog vom Entwicklungsteam umgesetzt wird. Am Ende des Sprints steht bereits ein Teilprodukt, das *Product Increment* zur Auslieferung

an den Kunden bereit. Gleich anschließend wird ein *Sprint Review* durchgeführt, wo das Team die umgesetzten Stories dem Kunden bzw. den Usern präsentiert. Auf Basis des Reviews wird das Product Backlog aktualisiert, d.h. Stories werden neu geschätzt und ggf. neu priorisiert. Demnach wird auch der Releaseplan aktualisiert. Die *Sprint Retrospective* dient als Lessons Learned für das Team. Es werden Arbeitsprozesse, Kommunikation, etc. analysiert und Verbesserungsvorschläge für das nächste Sprint Planning in einem *Impediment Backlog* festgehalten. (Wirdemann, 2017; Cohn, 2009; Gloger, 2014).

Der gesamte Ablauf wird nochmal übersichtlich in Abbildung 2.4 veranschaulicht.

2.2.2.3 Die User Story

Nun zu den bereits erwähnten User Stories. Diese sind nicht als verschriftlichte Anforderung zu sehen, die dem Entwicklungsteam als Vorlage zur Umsetzung dient. Als User Story wird in Scrum – und auch anderen agilen Methoden – eine grobe Beschreibung eines *(Teil-)Geschäftswertes* für den Kunden bzw. aus dessen Perspektive verstanden (Leffingwell, 2010; Cohn, 2009). Die User Story wird gemeinsam, meist der Product Owner mit dem AUftraggeber bzw. User, verschriftlicht. Das Rahmenwerk Scrum sieht auch vor, die User Stories auf Kärtchen und in handschriftlicher Form festzuhalten, damit es eben auch den Charakter einer groben Beschreibung hat. Zudem muss die Beschreibung auch auf einem einzigen Kärtchen Platz haben. Gerade zu Beginn des Prozesses werden Kärtchen verwendet, jedoch im weiteren Projektverlauf ist es eher üblich, diese in digitaler Form, wie z.B. in einem Tabellenblatt oder auch in einer eigenst dafür vorgesehenen Software, festzuhalten. Nichts desto trotz – ob Kärtchen oder digital – User Stories sind per Definition ein Mittel der *Kommunikation* und nicht der Dokumentation (Leffingwell, 2010). Dies soll bedeuten, dass User Stories dazu anregen sollen, diese Stories im Team zu diskutieren, zu hinterfragen und die Wertigkeit für den Kunden/User zu verstehen. Leffingwell (2010, p. 103) bezeichnet es auch als: *„[...] a promise for the conversation of the intent."*

Eine weitere und wichtige Eigenschaft von User Stories sind *Akzeptanzkriterien*. Dabei handelt es sich um Kriterien, die aus der Sicht des Kunden/Users erfüllt sein müssen, damit die Story als erledigt gilt. Dazu werden *akzeptanztests* in Form von Testfällen oder funktionale Tests i.d.R. vom Product Owner bzw. dem Kunden/User erstellt und durchgeführt. So werden die Akzeptanzkriterien in gleicher Weise und simultan zu der User Story erstellt und bedingen sich gegenseitig. Bei der Arbeit mit Kärtchen werden diese Kriterien auf der Rückseite festgehalten. (Leffingwell, 2010; Gloger, 2014)

Auf diese Weise „wachsen" die Akzeptanzkriterien mit den User Stories. User Stories und Akzeptanzkriterien werden laufend angepasst, wenn sich die Bedürfnis- und Informationslage der Kunden im Zeitverlauf ändert (Gloger, 2014). Viele Diskussionen zu User Stories zwischen dem Team und dem Kunden/User finden so auch oft erst

während bzw. kurz vor der Implementierung statt (ebd.). Damit wird erst bei tatsächlicher Umsetzung durch das Entwicklungsteam über Details zur User Story im Team und mit dem Auftraggebern gesprochen. Dies ist mitunter wohl der radikalste Unterschied im Vergleich zu den klassischen Vorgehensmodellen, wo diese just-in-time Anforderungsdefinition nicht vorgesehen ist.

Nun bereits erahnt werden, wie simpel Qualität eigentlich in agilen Softwareprojekten sichergestellt wird: Durch die im Umfang eher bescheidenen, jedoch für den Kunden wertvollen Beschreibung wird eine Art Informationsknappheit erzeugt, welches aktives und stetiges aufeinander Zugehen erfordert, um Informationen anzureichern und dabei genau festgelegt ist, wann es startet und wann es erledigt ist.

Nun betrachtet man noch in kurzen Zügen die „Sprache" einer User Story. Im Scrum Rahmenwerk gibt es so etwas wie eine *Satzschablone*, die beschreibt, wie eine User Story aufgebaut sein soll. Es gibt dabei leicht unterschiedliche Versionen, welche teilweise andere Bezeichnungen für ein und dasselbe verwenden. Im Grunde jedoch ist die Satzsemantik immer die gleiche und lässt sich wie folgt auszeichnen:

„Als <*User/Rolle*> möchte ich/kann ich <*Aktivität*>, so dass ich <*Geschäftswert/Nutzen*>."

Diese Satzschablone ist im Grunde genommen selbstsprechend und damit einfach zu verstehen und auch einfach in der Anwendung. Ein fiktives Beispiel für eine User Story mit Hilfe dieser Satzschablone ist:

„Als *Buchhalter* möchte ich *jeden Tag am Morgen eine Aufstellung der verbuchten Zahlungseingänge des Vortags als Email gesendet bekommen*, so dass ich *Mahnungen rechtzeitig versenden kann*."

Diese Art der Fomulierung beinhaltet zu beiden Teilen den Problem- und Lösungsraum. Des weiteren stellt sie den User in den Mittlepunkt der Betrachtung und verhilft dem Team so gut es geht und über den gesamten Projektverlauf hinweg die Brille des Kunden bzw. des Users zu tragen und damit idealerweise ein System zu entwicklen, dass seine Wünsche und Bedürfnisse erfüllt. (Leffingwell, 2010)

Es werden im Zuge der Fallstudie, User Stories aus dem Echtbetrieb eines Unternehmens als zentrales Artefakt zur Vorhersage des Aufwandes behandelt. So ist es uns wichtig, die Bedeutung und Eigenheiten bzw. Unterschiede von Anforderungsbeschreibungen im agilen vs. klassischen Kontext kurz zu betrachten und die Basis davon verstehen zu geben. In Bezug auf die weiteren Ausführungen in dieser Arbeit – insbesondere für die Fallstudie – ist festzuhalten, dass User Stories zwar auf einer Kunden/User-zentrierten Verschriftlichung gründen, jedoch nicht mit vollem Umfang an Informationen (angereichert über den Zeitverlauf) ausgestattet sind. Viele Informationen werden – wie eben in Scrum vorgesehen – im direkten und situationsbedingten Austausch fließen und dabei grundsätzlich wenig verschriftlicht, und damit wenig

dokumentiert wird. Es gilt im Rahmen der Fallstudie dann zu prüfen, welche Quellen der Informationen bzw. Daten es noch zu den User Stories gibt.

In diesem Abschnitt wurde ein Versuch gestartet, die fundamentalen Unterschiede von agilen zu klassischen Vorgehensmodellen in der Softwareentwicklung überblicksartig zu skizzieren. Weiters wurde bei den agilen Vorgehensweisen auf das Rahmenwerk von Scrum näher eingegangen. Ebenso wurde der Ablauf und die Anforderungsdefinition im Kontext von Scrum beleuchtet, um das notwendige Kontext- und Prozessverständnis für die Bearbeitung der Fallstudie in dieser Arbeit zu erlangen. Nun wird im nächsten Abschnitt näher auf die Aufwandsschätzung in agilen Softwareprojekten eingegangen.

2.3 Schätzmethoden und deren Anwendung in Softwareprojekten

Den Arbeitsaufwand in einem Softwareprojekt vor Projektbeginn möglichst genau und zuverlässig zu schätzen ist wohl eines der essentiellsten Teilschritte, aber auch der herausfordernste in einem Softwareentwicklungs-Projekt. Ob in klassischen- oder in agilen Projektmanagement Methoden, hier haben sie eines gemeinsam: Die Zuverlässigkeit der Schätzung wirkt sich direkt auf die wirtschaftlichen und zeitlichen Dimensionen des Projektes, und damit indirekt auf die Kundenzufriedenheit aus.

Dieser Abschnitt ist den existierenden und praktizierten Schätzverfahren in der Softwareentwicklung gewidmet. Im speziellen gilt es, auf die agilen Schätzpraktiken und Methoden etwas näher einzugehen. Da das Ziel dieser Arbeit darin besteht, ein Vorhersagemodell auf Basis von Data- und Text Mining Verfahren zu entwickeln, beschäftigt sich dieser Abschnitt mit der Frage: Wie und was wird in agilen Projekten eigentlich geschätzt? Demnach wird der Prozess des Schätzens und die Methoden dazu näher beleuchtet.

2.3.1 Zuverlässigkeit von Schätzmethoden

Das Ziel einer Aufwandsschätzung in Software Projekten ist, den benötigten *personellen Aufwand*, die anfallenden *Kosten* oder die *Projektdauer*, vor Projektstart möglichts dem tatsächlichen Aufwand (retrospektiv) nach Ende des Projektes zu antizipieren (Pilz, 2011). Weiters ist die Schätzung ein wichtiger Maßtab um den *Projektfortschritt* messbar zu machen (Sneed, 2014). Die Praxis hat gezeigt, dass Aufwandsschätzungen kaum mit dem tatsächlichen Aufwand übereinstimmen (Pilz, 2011). Die Übereinstimmung von geschätztem Aufwand zu dem tatsächlich angefallenen Aufwand variiert zwar von

den eingesetzten Schätzmethoden, insgesamt ist jedoch die *Zuverlässigkeit* existierender Methoden als gering bis mäßig einzustufen (ebd.). Die Zuverläßigkeit einer Schätzmethode gibt die Wahrscheinlichkeit an, wie nahe das Schätzergebnis am tatsächlichen Aufwand liegt. Das Ziel bzw. einer der wichtigsten Anforderung an eine Schätzmethode – neben den damit verbundenen Kosten – ist es, eine so hohe Zuverlässigkeit zu erreichen, um sie in der Praxis effektiv anwenden zu können. Mit einer zu geringen Zuverlässigkeit einer Methode leidet oft die Anwendung in der Praxis, d.h. sie wird aufgrund mangelnder Akzeptanz kaum bis gar nicht angewendet (Wieczorek, 2001). Eine Definition von beispielsweise „geringe-" oder „mäßige Zuverlässigkeit" ist an dieser Stelle nicht möglich, da sich diese Maßstäbe an vielerlei unternehmensspezifischen- und projektspezifischen Gegebenheiten richtet. Aus der Literaturrecherche ergibt sich der Eindruck, dass sich bis heute noch kein Schätzverfahren großflächig in Theorie und Praxis durchringen konnte, welches universell[8] einsetzbar ist. Dies liegt vermutlich daran, dass jede Methode seine Vor- als auch Nachteile hat. Dazu ist anzunehmen, dass Schätzmethoden unterschiedlichen Unternehmens- und Projekt-Randbedingungen ausgesetzt sind, in denen sie gut funktionieren – oder eben weniger gut. Ein möglicher Schluss daraus ist, dass der Erfolg, sprich die Anwendbarkeit und Akzeptanz eines Aufwandsschätzverfahren sich nur durch eine Kombination von mehreren Verfahren durchsetzen lässt. Des weiteren ist es unabdingbar, Erfahrungswerte von Personen im Projekt in die Schätzung miteinzubauen und periodisch getaktete Schätzungen zu vollziehen (Pilz, 2011).

Für die Entwicklung des Vorhersagemodells im Rahmen dieser Arbeit gilt es daher, nicht dem Anspruch eines perfekten Modells hinsichtlich Zuverlässigkeit gerecht zu werden, da dies aufgrund der oben genannten Erkenntnisse einer Illusion gleichkommen würde. Selbstverständlich sollte das Modell eine im Rahmen der Fallstudie definierte Zuverlässigkeit erreichen, um eine Daseinsberechtigung zu erlangen, doch dazu sollte auch darauf geachtet werden, wie sich das Vorhersagemodell in den Prozess einfügen lässt und wie es möglicherweise mit anderen, bestehenden Verfahren im Unternehmen optimal eingesetzt werden kann.

2.3.2 allgemeine Schätzverfahren

Aufwandschätzmethoden in der Softwareentwicklung wurden bereits in den 1950 Jahren entwickelt und damit versucht, den Entwicklungsaufwand von Software vorherzusagen. Damals wie heute hatten Software-Ingenieure mit der Zuverlässigkeit von Methoden zu kämpfen (Sharma & Fotedar, 2014). Das Unterschätzen des Aufwandes führte zu Termin- und Budget Überschreitungen, wohingegen eine Unterschätzung einen negativen Einfluss auf die Konkurenzfähigkeit des Unternehmens hat (Choetkiertikul et al., 2016). So wurde bereits früh begonnen, sich mit der Aufwandsschätzung im praktischen- aber auch im wissenschaftlichen Kontext auseinanderzusetzen.

[8] unabhängig von Projektgröße, Projekttyp, eingesetzte Werkzeuge, und Projektpersonal

Grundsätzlich lassen sich Aufwandsschätzmethoden in *Experten-basierte* Methoden und in *Modell-basierte* Methoden unterscheiden (Choetkiertikul et al., 2016; Menzies et al., 2006). Im Folgenden wird nun konkret auf die Unterscheidungen und deren Methoden näher eingegangen.

2.3.2.1 Experten-basierte Methoden

Experten-basierte Methoden gründen auf der Idee, sich Meinungen von *Experten* zunutze zu machen, um den Aufwand eines bevorstehenden Projektes zu schätzen. Dabei bilden alleine die *Intuition* und *Erfahrung* der Experten die Grundlage für die Schätzung (Dumke, Schmietendorf, Seufert & Wille, 2014). Dieser Ansatz kann auch methodisch unterstützt werden. Ein Beispiel dafür ist die *Delphi-Methode*. Bei der Delphi-Methode werden Meinungen anonym und unabhängig von einem Experten durch eine Befragung erhoben. Nach der Zusammenfassung der ersten Befragung kann auf diese aufbauend ein weiterer Experte befragt werden. Beim Durchlauf dieses mehrstufigen Prozedere werden die Schätzungen von Experten mit jeder Befragung erweitert und verfeinert – so lange, bis sich die Meinungen nicht mehr signifikant voneinander unterscheiden. Das Endergebnis ist eine Zusammenfassung, welche die Aussagen der Experten und die Bandbreite der Meinungen darlegt. Zur weiteren Vertiefung dieser Methode wird auf weiterführende Literatur zur Methode verwiesen.

Die Fehlerrate bei Schätzungen mit Hilfe von Gruppenergebnissen erwies sich lt. Hummel (2011), um rund 50 Prozent geringer als bei Einzelschätzungen durch Experten. Die Fehlerrate ist jedoch stark abhängig von der *Erfahrung* der jeweiligen Personen und der *Qualität* der Erfahrungsdaten (Dumke et al., 2014). Damit wird bei Expertenschätzungen empfohlen, Gruppenschätzungen gegenüber Einzelschätzungen zu bevorzugen und sich an diesen zu orientieren. Dabei ist natürlich darauf zu achten, dass Experten auch deren Bezeichnung und Legitimation gerecht werden, d.h. das vorhandenen Wissen zum zu entwickelnden System mitbringen und bestenfalls auch bei der Umsetzung der Aufgaben die Verantwortung tragen (Hummel, 2011).

Für größere (Teil-)Projekte und Aufgaben ist es oftmals erforderlich, diese in kleinere Teile zu zerlegen, um sie mit einer entsprechenden Zuverlässigkeit schätzen zu können. Dabei kann beispielsweise die *Zwei-Punkt-Schätzung* für Abhilfe sorgen. Dabei wird zu jeder Aufgabe eine optimistische Schätzung und eine pessimistische Schätzung abgegeben. Aus diesen zwei Szenarien lässt sich ein Mittelwert bzw. ein Erwartungswert bilden, welcher als Schätzwert für die Planung angenommen werden kann.(Hummel, 2011)

Es werden zu einem späteren Zeitpunkt noch auf Schätztechniken im Scrum Prozess, wo sich nahezu alle Aspekte bzw. abgeleitete Versionen der oben angeführten Methoden wiederfinden, eingehender behandelt.

2.3.2.2 Modell-basierte Methoden

Hierbei handelt es sich um Methoden, die auf einem *formalisierten*, *strukturierten* und meist *parametrischen* Ansatz gründen. Ein Vertreter, der sehr häufig in der Praxis vorkommt und leicht anzuwenden ist, ist das *Analogieverfahren*. Dabei werden Entwicklungsaufwände aus vergangenen Projekten herangezogen, die eine bestimmte Ähnlichkeit aufweisen, um eine Schätzung für das neue Projekt vorzunehmen. Beispielhaft könnte eine analogie-basierte Schätzung folgendermaßen vonstatten gehen: Historische Projekte werden nach deren *Ähnlichkeit* zum durchführenden Projekt ausgewählt und in kleinere *Systemteile* zerlegt, so dass diese besser schätzbar werden. Danach erfolgt die Festlegung der Größeneinheit je Systemteil. So kann direkt der Aufwand in Personenmonate angegeben werden, als auch die *Lines of Code (LOC)*, *Datenpunkte*, etc. (Hummel, 2011). Danach erfolgt die Ableitung des Größenumfangs für das neue Projekt. Bei diesem Verfahren wird natürlich vorausgesetzt, dass Vergleichsdaten von ähnlichen Projekten vorhanden sind und die echten Aufwände von historischen Vergleichsprojekten herangezogen werden und nicht deren damalige Schätzwerte.

Weiters existieren *parametrische Modelle* bei denen der Entwicklungsaufwand für ein Projekt berechnet wird. Zugrunde liegen mathematischen Formeln mit Parametern, die zuvor festgelegt werden. Parametrische Verfahren sind meist empirischer Natur, d.h. diese Modelle stützen sich auf statistischen Erfahrungswerten aus einer Vielzahl von abgeschlossenen Projekten aus der Vergangenheit. Eines der bekanntesten und viel zitierten Methoden ist jenes aus der Familie der *Constructive Cost Model (COCOMO)*-Verfahren. Wenn man heute von COCOMO spricht, meint man meist das COCOMO II., welches aus dem COCOMO 81 bzw. COCOMO I. weiterentwickelt wurde. Beim COCOMO geht es im Prinzip um die Abschätzung der Kosten verschiedener Aspekte und Teilbereiche der Softwareentwicklung und Wartung (Dumke et al., 2014). Es ist ein *algorithmisches* Verfahren, welches Anweisungen zur Berechnung (siehe Gleichung (2.1)) des Projektaufwandes beinhaltet.

$$Aufwand = Typ * \left[\frac{Größe}{Produktivität}\right]^{SE} * (EF * QF) \tag{2.1}$$

Der Typ besagt dabei, um welchen Systemtypen[9] es sich handelt. DIe *Größe* wird angegeben in LOC bzw. in *Kilo Lines of Code (KLOC)* bzw. in *Point-Metriken*, auf welche später noch eingegangen wird. *Produktivität* ist eine Umrechnung der Point-Metriken pro Personenmonate. Der *Skalierungsexponent (SE)* ist der Mittelwert des Wiederverwendungsgrades. Der *Einflussfaktor (EF)* wie z.B. die Umgebungsreife oder Zielarchitektur. Der *Qualitätsfaktor (QF)* bezieht die weichen Faktoren, wie Teamkohäsion und Prozessreife, mit ins Modell ein. (Hummel, 2011)

[9]Standalone, Integrated, Distributed und Embedded Realtime

Wie schon aus der Beschreibung der Komponenten des COCOMO II. zu erahnen, ist die Anwendung in der Praxis nicht ganz trivial und sehr aufwändig. Zum einen ist die Ermittlung der Koeffizienten mit hohem Aufwand verbunden, und zum anderen ist es notwendig, eine ausreichend große Stichprobe für die Berechnung zu haben (Dumke et al., 2014). Demzufolge fällt es kleineren Unternehmen schwer, dieses Verfahren zielführend anzuwenden. Zudem wird von Experten kritisch angemerkt, ob LOC noch ein geeigneter Größenmaßstab für den Umfang und Komplexität von Softwareanwendungen sind (Dumke et al., 2014; Trendowicz & Jeffery, 2014).

Weiterer modell-basierte Ansätze haben sich vor allem mit der Etablierung der agilen Methoden in den letzten Jahrzehnten herausgebildet: Das Schätzen mit Hilfe von abstrakten Konstrukten. Die Rede ist von *Point-Metrikbasierten Verfahren*. Dabei werden unterschiedliche Point-Metriken wie *Function Points*, *Data Points*, *Object Points*, *Use-Case Points*, etc. verwendet, um den Umfang eines Projektes zu schätzen. Die Bestimmung bzw. Schätzung von Points, basiert auf der Annahme, den *Umfang* und die *Komplexität* eines Projektes oder (Teil-)Aufgaben davon, als abstrakte Einheit auf einer ordinalen Skala zu verdichten zu können. Die *Komplexität* ergibt sich dabei aus der Berücksichtigung von qualitativen-, plattform-, und prozessbezogenen Anforderungen, welche sich anhand der Gleichung (2.2) beschreiben lassen (Hummel, 2011).

$$Aufwand = \frac{Umfang * Komplexität}{Produktivität} \tag{2.2}$$

Die *Produktivität* ist ein Maß dafür, wieviel Points ein Entwickler bzw. das Team in der Lage ist, in einem bestimmten Zeitraum zu bewältigen.

Ein Urvater dieses Ansatzes ist sicherlich die *Function-Point-Methode*. Function-Points sind das Produkt aus fünf definierten Daten-Komponenten[10] und 14 gewichteten Umweltfaktoren[11]. Es wird die funktionale Größe eines Systems bestimmt und kann als Basis für die Aufwandsschätzung dienen. Da eine detailiertere Beschreibung dieser Methode den Rahmen dieser Arbeit sprengen würde, wird an dieser Stelle auf weiterführende und bereits oben zitierte Literatur verwiesen.

Es existiert noch eine Vielzahl an Point-Metiken die aufgrund des Umfangs in dieser Arbeit nicht vollständig ausgeführt werden können. Es wird jedoch eine spezielle und für diese Arbeit wichtige Point-Metrik im nachfolgenden Abschnitt näher betrachtet: Die *Story-Points*.

Abschließend lässt sich feststellen, dass das Feld der Schätzmethoden sehr divers im Ansatz, in den Ausprägungen und der in der Komplexität erscheint. Einer Meta-Studie zufolge gibt es teils wiedersprüchliche Ergebnisse zur Performance von Experten-basierten und Modell-basierten Verfahren, als auch zu den Methoden innerhalb der Verfahren (Shepperd & MacDonell, 2012). Somit macht es keinen Sinn, im weiteren

[10] Eingaben, Ausgaben, Anfragen, Dateien und externe Benutzerschnittstellen
[11] Datenkommunikation, Performance, Wiederverwendbarkeit, etc.

Verlauf der Arbeit sich mit der Betrachtung von möglichen Vorhersage-Referenzmodellen zu beschäftigen.

2.3.3 Schätzen in Scrum

2.3.3.1 Von Story Points und Velocity

In agilen Vorgehensweisen wird durch die Bank auf das Schätzen in Aufwänden verzichtet. Das Scrum Rahmenwerk sieht ein Schätzverfahren vor, welches auf der *relativen* Schätzung von *User Stories* basiert. Dazu wird ein einheitenloses Maß verwendet: Die *Story-Points*. Story Points sind ein Maß für die übergeordnete Größe einer User Story und beschreiben mit nur einer Zahl die potentielle *Größe* einer Story. Mit Größe ist nicht nur der Umfang bzw. das *Volumen* einer Story gemeint, es sind *Verständnis*, *Komplexität* und die *Unsicherheit* einer Story mit einzubeziehen (Leffingwell, 2010). Ein Story Point vereint also all diese Eigenschaften zu einem nummerischen Wert, welcher relativ zu anderen Stories vergleichbar ist. Die Wahl des numerischen Wertes ist im Grunde genommen nicht vorgegeben, es gibt jedoch eine Empfehlung, um die Verhältnisse zueinander richtig bzw. besser einschätzen zu können. Cohn (2009) betont, es sei vorteilhaft eine nicht-lineare Skala zu verwenden. So bietet sich die *Fibonacci-Reihe* an, welche in der Praxis häufig eingesetzt wird. Demnach können Story Points den Wert 1, 2, 3, 5, 8, 13 – und so weiter – annehmen. Ist der relative Point-Abstand von 1, 2 und 3 noch linear, so vergrößert sich der Abstand überproportional zur Zahl davor ab 5 (also 8, 13, und so fort...). Der Grund dafür ist ein ganz einfacher, welcher sich in der Praxis gut bewährt hat: Größere Point-Stories sind grundsätzlich mit einer höheren Unsicherheit behaftet als kleinere in Bezug auf die Schätzung (Cohn, 2009). Sieht man nun jede Zahl symbolisch als Eimer, wo Stories hineingekippt werden können, so wird man sich entscheiden müssen, in welchen Eimer eine Story gekippt werden soll, falls diese einen Wert dazwischen annehmen würde. Eine hypothetische 11-Point-Story würde demzufolge eher in die 13 gekippt werden, als in die 8.

Story Points werden im *kollektiv* geschätzt und nicht von einzelnen Personen. Obwohl es gute Gründe dafür gibt – und auch Erfolgsnachweise – Schätzungen einer Aufgabe derjenigen Person zu überlassen, welche die Aufgabe dann auch umsetzt, so wird beim agilen Vorgehen immer vom ganzen Team geschätzt. Das hat im speziellen zwei Gründe: Zum einen ist in agilen Projekten zum Zeitpunkt der Schätzung meist noch nicht festgelegt, wer die Story tatsächlich umsetzten wird. Trotz Spezialisten im Team ist nicht sichergestellt, dass diese Person in einem Sprint verfügbar sein wird – aus welchen Gründen auch immer. Somit sollte jeder der Teammitglieder bei der Schätzung mitwirken. Zweitens, bringen Teammitglieder unterschiedlichen Background mit und schätzen damit Stories möglicherweise unterschiedlich voneinander. Da bei der Umsetzung im Prinzip jeder zum Zug kommen könnte, ist eine sozusagen gemittelte bzw. eine gemeinsam-verbindliche Schätzung vorteilhaft. (Gloger, 2014; Cohn, 2009)

Relativ sind auch die Schätzung des Teams im Vergleich zu einem anderen Team. Daher kann eine 2-Point-Story eines Teams für ein anderes Team 3, oder gar 5 Points sein. Somit ist das Schätzergebnis selbst, unabhängig von der Aufgabe, eine Variable des schätzenden Teams. Schätzergebnisse sind damit, auch wenn die gleiche oder ähnliche Aufgabe vor liegen würde, nicht auf andere Scrum-Teams übertragbar (Cohn, 2009). Auch die Übertragbarkeit der Schätz-Skala eines Teams auf unterschiedliche Projekte ist fragwürdig (Sneed, 2014). Weiters sind Story-Point-Schätzungen nicht so weiters zu *dekompositionieren*. Das bedeutet, wenn eine Story oder ein Epic in weitere Stories zerlegt wird, so muss die Summe der Schätzungen dieser nicht gleich die der ursprünglichen, größeren Story oder Epic sein (Cohn, 2009). Dasselbe gilt auch für Tasks, deren Schätzung nicht gleich die Summe der Story ergeben muss (ebd.).

In der Praxis läßt sich beobachten, dass viele Teams dazu neigen, speziell bei den ersten paar Schätzungen nach dem Einführen dieser Schätzmetrik, die *Story-Point-Denkweis*e und die dazugehörigen *Referenzgrößen* nicht vorhanden sind (Sneed, 2014). Teammitglieder aus der klassischen Schule der Softwareentwicklung haben sich auf die Aufwandsschätzung (oftmals in Stunden) eingeschossen und haben mit dieser Art der Schätzung oft Schwierigkeiten (Gloger, 2014). Aber warum den eigentlich nicht Aufwände schätzen? Wie bereits erwähnt, müssten die Aufgaben bereits in einer sehr frühen Phase des Projektes zugeteilt werden, d.h. man weiß, wer die Story umsetzen wird und wie genau das geschehen soll. Dies ist zwar möglich, aber sehr zeitintensiv und mit einem hohen Unsicherheitsfaktor behaftet. Stories können sich im Projektverlauf noch mehrere Male ändern und damit sind unvorhersehbare Aufgaben vorprogrammiert. Außerdem sagt der Aufwand nichts über eine *Durchlaufzeit* aus, d.h. die Zeitspanne an dem die Story begonnen und dann tatsächlich umgesetzt ist. Nichts desto trotz hat die Aufwandsschätzung bzw. der Aufwandsvergleich durchaus seine Berechtigung. Gloger (2014) erwähnt dazu, dass die Aufwandsbetrachtung für das gesamte Projekt durchaus hilfreich und sinnvoll sein kann. So kann der kumulierte Aufwand für ein abgeschlossenes Projekt aus der Vergangenheit als Referenz für ein anstehendes herangezogen werden (Gloger, 2014), um z.B. einen Kostenvoranschlag für den potentiellen Auftraggeber zu entwerfen oder eine Roadmap-Planung durchzuführen. Für die Fallstudie in dieser Arbeit kann an dieser Stelle die Aufwandsbetrachtung aus dem soeben genannten Grund nicht ganz ausgeschlossen werden.

Nun sagen aber Story Points auch nicht unbedingt etwas über die Durchlaufzeit einer Story aus. Dazu sieht das Scrum Rahmenwerk eine Maßzahl für die *Geschwindigkeit* vor: die *Velocity*. Sie gibt an, wie viele Story Points in einem einzigen Sprint abgearbeitet werden können (Cohn, 2009). Diese Summe an umgesetzte Story Points können somit als Maßzahl für den nächsten Sprint verwendet werden. Auch wenn bei einem weiteren Sprint die Summe der umgesetzten Story Points etwas wenig oder auch etwas mehr ergibt, so wird sich über ein paar Iterationen eine durchschnittliche Maßzahl ergeben, die als *Team-Velocity* zu sehen ist. Daher, wenn auch die Team-Velocity je Sprint etwas unterschiedlich ausfällt, also eine *Varianz* vorhanden ist, so wird sich diese über mehrere Sprints ausmitteln. Mit der geplanten Anzahl an Sprints, welche zeitlich immer gleich lange sind, kann nun die voraussichtliche Dauer bzw. Durch-

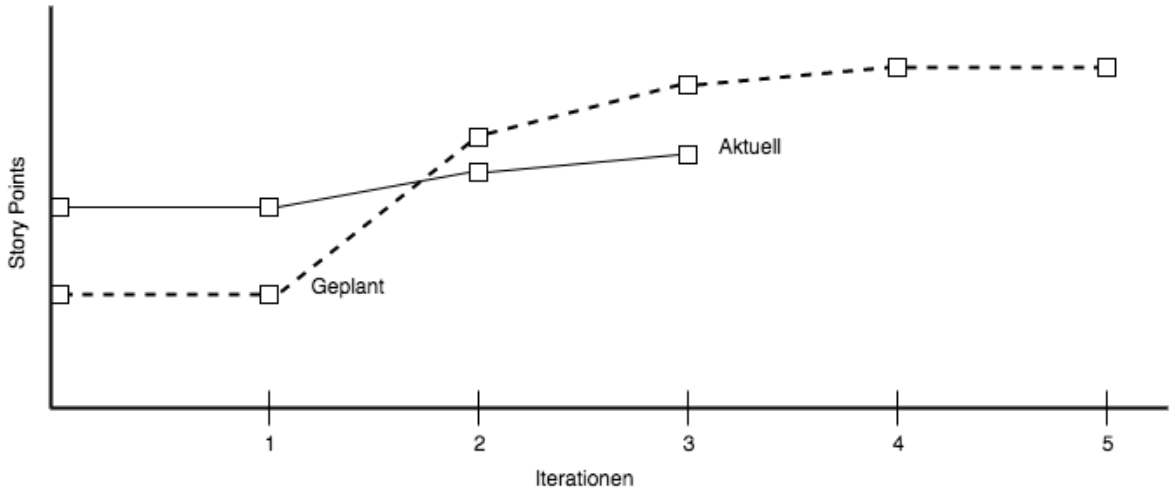


Abbildung 2.5: Geplannte und tatsächliche Veocity nach den ersten 3 Iterationen (Cohn, 2004)

laufzeit des Projektes ermittelt werden. Andersrum betrachtet, kann auch frühzeitig eine Korrektur der initialen Schätzung schon nach den ersten Iterationen erfolgen. EIn hypothetisches Beispiel: Ein Projekt wird mit 150 Story Points bei Projektstart geschätzt und das Team glaubt, eine Velocity von 30 Points per Sprint zu erreichen. Dies bedeutet, das dass Projekt mit fünf Iterationen erledigt wäre. Angenommen nach den ersten drei Sprints stellt sich eine Velocity von lediglich 25 ein, so würde das Projekt anstatt fünf Iterationen sich auf sechs verlängern. Damit wäre eine mögliche Verzögerung des Projektes schon sehr bald antizipiert. Eine Möglichkeit, die Velocity des Teams darzustellen, bietet das *Velocity-Chart* (siehe Abbildung2.5). Es ist eine einfache, aber übersichtliche Darstellung der geplanten bzw. geschätzten vs. der tatsächlichen Story Points per Sprint. An dieser Stelle sei jedoch gesagt, dass es unterschiedliche Varianten davon gibt, d.h. beispielsweise anstatt die fortlaufende Anzahl an Iterationen, die tatsächliche Zeitdauer zu verwenden, etc..

Insgesamt ist das Schätzen in Story Points ein einfaches und reliables Verfahren, mit welcher sich eine hohe Vorhersagbarkeit erreichen lässt (Leffingwell, 2010). Durch die Vereinheitlichung und Vergleichbarkeit der Größe eines Projektes bzw. Produktes, auf einer einzigen Werteskala, und der zeitlichen Fixierung der Iterationen (Sprints), lässt sich auch die Durchlaufzeit eines Projektes, ja sogar auch eines Features äußerst einfach und zuverlässig ermitteln. Zuletzt sei gesagt, dass das Scrum-Rahmenwerk noch weitere *Visualisierungs- und Prozessteuerungstechniken* (z.B. Parking-Lot-Chart, Burndown-Chart, etc.) beinhaltet, auf die jedoch in weiterer Folge nicht weiter eingegangen werden kann. Zur Vertiefung dazu wird auf Leffingwell (2010) und Cohn (2009) verwiesen.

2.3.3.2 Der Scrum Schätzprozess

Zuvor wurde das Schätzen in Story Points und die Bestimmung der Velocity bei Projekten etwas näher erläutert. Was jedoch noch fehlt, ist die Kenntnis darüber, wie und wo die Schätzung in einem Scrum-Prozess stattfindet.

So sei nochmals Abbildung 2.4 aus Abschnitt 2.2.2.2 in Erinnerung gerufen. Dabei wurde der grundsätzlichen Ablauf, der Scrum Flow, kennengelernt. Im Product Backlog sind alle User Stories enthalten, welche die gesamte Leistung oder besser gesagt, den Wert eines Produktes, definieren. Ziel zu Beginn ist es nun, das gesamte Backlog zu schätzen. Dabei bittet der Product Owner das gesamte Team, alle User Stories im Rahmen eines *Estimation Meetings* zu schätzen.

Damit dies nicht willkürlich und unkontrolliert abläuft, gibt es auch dafür Methoden im Scrum-Rahmenwerk. Einer der bekanntesten Methoden ist der *Planning Poker* nach Cohn (2009). Es wird dabei eine Art Kartenspiel, mit der Fibonacci-Reihe bedruckten Karten, gespielt. Jedes Teammitglied bekommt ein Deck Karten. Der Product Owner stellt nun die User Stories vor und beantwortet Verständnisfragen des Teams. Danach wird die Schätzung ausgerufen und jedes Teammitglied hält seinen Schätzwert hoch, so dass ihn der Product Owner sehen kann. Sind alle hochgehaltenen Karten gleich bzw. weisen nur vereinzelt kleine Abweichungen zwischen den Teilnehmern auf, so ist die Schätzung im Grunde genommen getan. Der Product Owner notiert nun das kollektive Schätzergebnis der Stories. Dies geht so lange, bis alle Stories mit einer initialen Schätzung versehen sind. Gibt es während des Spiels große Unterschiede im Wert der Karten, so ist das ok. Denn nun gibt es vermutlich etwas zu klären. Dabei ist vorgesehen, dass derjenige mit dem niedrigsten Wert und jener mit dem höchsten deren Erklärung bzw. Begründung für die Schätzung erläutern. In dieser Situation gilt es, nicht zu klären wer im Recht ist, sondern eine Diskussion zu fördern und Informationen auszutauschen. Nach der Darlegung der Teilnehmer wird die Schätzung wiederholt.

So kann in kurzer Zeit ein ganzes Backlog geschätzt werden, welches dann unter der Berücksichtigung der Velocity in einen *Releaseplan* übertragen werden kann. Neben dem eigentlichen Schätzergebnis bringt diese Methode auch den Vorteil mit sich, dass alle Mitglieder des Teams eine bessere Vorstellung davon haben, was entwickelt werden soll (Gloger, 2014).

Im nächsten Schritt erfolgt die Planung auf *taktischer Ebene*. Es werden die vom Team geschätzten und vom Product Owner hoch priorisierten User Stories in ein Sprint Backlog übertragen. In einem *Sprint Planning 1* werden die priorisierten Stories nochmals vom Product Owner vorgestellt und das Team gibt eine Schätzung darüber ab, was sie in dem bevorstehenden Sprint glauben zu schaffen. Im *Sprint Planning 2* wird darüber diskutiert, wie genau das Team die Stories umsetzen werden.

Während des Sprints wird empfohlen, mindestens zweimal ein Estimation Meeting einzuberufen. Ziel dieser Workshops ist es, dem Product Owner die Möglichkeit zu geben, das Product Backlog zu überarbeiten, neu schätzen zu lassen bzw. Schätzung anzupassen und damit auch den Releaseplan zu aktualisieren (Gloger, 2014).

Nach Ende jedes Sprints erfolgt ein Review, bei dem das Product-Increment dem Auftraggeber präsentiert wird. Alleine aus dem Feedback des Kunden und dem Team kommt es meist zu Abänderungen noch offener Stories bzw. überhaupt neue Stories hinzu. Es wird das Product Backlog aktualisiert und es Bedarf daher wieder eines Estimation Meetings zur Aktualisierung bzw. Erstschätzung. Die Ergebnisse aus den Sprints werden in der Retrospektive analysiert und fließen in die nächste Sprintplanung mit ein.

Zusammenfassend lässt sich sagen, dass die vorausschauende Planung und Schätzung ein wichtiger Bestandteil im Prozess darstellt. Es wird nicht nur einmal die Größe des zu entwickelnden Artefakts geschätzt – nämlich zu Beginn des Projektes – sondern vor jedem einzelnen Sprint und sogar bei Bedarf noch mehrmals innerhalb des Sprints in den Estimation Meetings. Es ist jedoch gut möglich, dass in der Praxis abgeänderte bzw. modifizierte Vorgehensweisen zum beschriebenen Schätzungsverfahren existieren. Die Ausführungen bis an diese Stelle der Arbeit bezog sich lediglich auf theoretischen Definitionen und die in der Literatur vorgefundenen Erfahrungsberichte und Best Practices.

In diesem Kapitel wurde versucht, das Verständnis für die Problematik in der Softwareentwicklung und deren Lösungsansätze zu schärfen. Die Betrachtung der agilen Vorgehensweisen mit den Schlüsselprinzipien und Prozessen bildet die Grundlage und das notwendige Domänenverständnis für die Entwicklung des Vorhersagemodells in dieser Arbeit.

Das nächste Kapitel widmet sich den Grundlagen des Data Minings und dessen Ansätzen zur Lösung unterschiedlicher Problemstellungen.

3 Data Mining und Machine Learning - Grundlagen und Vorgehensweise

In diesem Kapitel werden die notwendigen Grundlagen für die Entwicklung eines Vorhersagemodells des Aufwandes im Kontext agiler Vorgehensweisen in der Softwareentwicklung erläutert. Dabei wird ein kurzen Überblick bzw. ein allgemeines Verständnis von Data Mining zu verstehen gegeben. Weiters sind dem verwandte Begrifflichkeiten des Data Minings, wie dem Machine Learning, zu beschreiben und zueinander abzugrenzen. Es wird der Prozess des Data Minings sowie dessen einzelnen Schritte und die Auswahl von Verfahren eingegangen. An dieser Stelle ist vorab gleich wegzunehmen, dass aufgrund des zu Verfügung stehenden Umfangs dieser Arbeit, ausschließlich auf die wichtigsten Konzepte und Verfahren des Data Minings bzw. des Machine Learnings eingegangen werden kann, bzw. welche zum Verständnis und zur Umsetzung der Anforderungen in der Fallstudie beitragen. An bestimmten Stellen wo ergänzende Information zum besseren und vertiefenden Verständnis beitragen kann, wird auf einschlägige Literatur verwiesen.

3.1 Data Mining - Ein Überblick

Wir Leben in einer Zeit, in der wir mit Daten überflutet werden. Die technologischen Entwicklungen in den letzten zwei Jahrzenten schritt dermaßen rasch voran, so dass der physischen Speicherung von Daten und den Zugangstechnologien[1] kaum noch Grenzen gesetzt sind. Intelligente Alltagsgegenstände werden zu persönlichen Datensammlern und unsere persönlichen Gewohnheiten in fast allen Lebenslagen in einer Datenbank persistiert. Der Zuwachs an Daten ist enorm. Schätzungen zufolge, verdoppelt sich die Menge an weltweit gespeicherten Daten alle 20 Monate (Witten & Frank, 2005). Daten sind sogesehen keine Mangelware im 21 Jahrhundert. Die *Informations- und Wissensextraktion* aus gesammelten Daten hingegen ist ein Unterfangen, welches Wissenschaft und Praxis schon seit Anbeginn des digitalen Zeitalters beschäftigt. Das Ziel dabei war stets, die Entdeckung von Strukturen und Muster in Daten, um Entscheidungsprozesse unterschiedlicher Art zu unterstützen. Der Begriff *Data Mining* wurde in die Welt gesetzt, um genau dieses Ziel als eigene Disziplin zu etablieren.

[1] logische und physischen Datenmodelle sowie deren Abfrage und Verarbeitung nahezu in Echtzeit

Allgemein formuliert bedeutet Data Mining also, durch die Datenanalyse von bereits gespeicherten Daten und mit Unterstützung von Computern, Probleme zu lösen. Im hoch-komplexen, kunden-zentrierten und service-orientierten Innovations- und Wirtschaftsgeschehen ist dies eine vielversprechende Möglichkeit sich Wettbewerbsvorteile zu verschaffen – ja sogar völlig neuartige Geschäftsmodelle zu ermöglichen (Witten & Frank, 2005). Einen relativ einfachen und großteils intuitiven Zugang zur Analyse dieser Daten wird durch die technologischen Entwicklungen von Analyse-Werkzeuge bereitet. Heute gibt es eine Vielzahl an Werkzeugen, die als kommerzielle Software bzw. auch Open Source zur Verfügung stehen. Diese Umstände – also Berge von Daten und Technologien, um diese zu analysieren – bringen nun das Thema Data Mining an die Front der praktischen Anwendung in Unternehmen. Ebenso ist auch die Anzahl an Publikationen im wissenschaftlichen Kontext in den letzten Jahren fast schon explodiert. Mit diesen Entwicklungen in der Praxis und den Erkenntnissen der Wissenschaft haben sich wiederum auch neue Disziplinen und Teilbereiche des Data Minings entwickelt bzw. davon abgeleitet, welche nachfolgend kurz erläutern werden und eine Abgrenzung und Definition der Begriffe vorgenommen wird.

3.1.1 Begriffsabgrenzungen und Definitionen

Der Begriff *Data Mining (DM)* wurde bereits kurz angesprochen. Witten und Frank (2005, p. 5) definiert DM wie folgt:

Data mining is defined as the process of discovering patterns in data. The process must be automatic or (more usually) semiautomatic. The patterns discovered must be meaningful in that they lead to some advantage, usually an economic advantage. The data is invariably present in substantial quantities."

DM ist demnach als Prozess der Datenanalyse zu verstehen, was auch die Definition nach Aggarwal (2015, p. 1) zum Ausdruck bringt:

„Data mining is the study of collecting, cleaning, processing, analyzing, and gaining useful insights from data."

DM umfasst einen gesamtheitlichen Prozess zur Extraktion von Wissen aus Daten. Es existiert eine Vielzahl an Referenzmodelle für die Vorgehensweise in DM-Projekten. Im nächten Abschnitt wird ein bestimmter Referenzprozess genauer betrachtet.

Nun haben sich aber auch Begriffe wie *Machine Learning (ML)*, *Artificial Intelligence (AI)*, und seit ein paar Jahren der Begriff *Deep Learning (DL)* wie ein Virus verbreitet. Diese genannten Begrifflichkeiten werden der Recherche nach oft verwechselt bzw. synonym für einander verwendet, obwohl unterschiedliche Bedeutungen und Herkünfte zu Grunde liegen. Nachfolgend wird überblicksartig auf die Begriffe eingegangen und eine Abgrenzung der Begriffe und Bedeutungen für diese Arbeit vorgenommen.

ML wird oft mit DM in Zusammenhang gebracht – und dies nicht ganz unberechtigt. Machine Learning entwickelte sich in der zweiten Hälfte des zwanzigsten Jahrhunderts als Teilgebiet der von AI. Beim ML geht es um die Entwicklung von selbstlernenden Algorithmen, welche in der Lage sind, aus historischen Daten Wissen und Einsichten zu generieren um damit Vorhersagen zu machen. Um dies noch genauer zu verdeutlichen, kann an dieser Stelle die Definition von Mitchell (1997, p. 2) wie folgt wiedergegeben werden:

„A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Damit ist offensichtlich, dass es sich um eine Art Technik bzw. Prozess handelt, welcher innerhalb eines Programms abläuft. Aus einer praktischen Perspektive bedeutet dies, dass anstatt durch den Menschen manuell abgeleitete Regeln aus Daten, eine Maschine[2] dazu benutzt wird, um Entscheidungen abzuleiten bzw. diese zu unterstützen. Dies bedeutet nicht, dass Maschinen immer in der Lage sind, den Menschen damit zu „schlagen" oder gar zu ersetzen. Selbstverständlich ist das menschliche Lernen bei weitem komplexer und vielschichtiger. Der große Vorteil von Maschinen in diesem Zusammenhang ist jedoch, dass diese eine viel gößere Kapazität zur Verarbeitung von Daten aufweisen und in der Lage sind, sukzessiv die Leistung einer Vorhersage – in der Form eines Modells – zu verbessern (Raschka, 2015). Der fast schon Modebegriff DL ist als spezielles Verfahren zu verstehen, bei dem ein neuronales Netz – wie beim menschlichen Gehirn – als Modell zugrunde liegt. Wie genau Maschinen lernen, wird zu einem späteren Zeitpunkt und konkret an ausgewählten Verfahren des ML und des DL näher erläutert. Die soeben ausgeführte Analogie von mensch-basiertem Lernen zu maschinell-basiertem Lernen führt eine Spur direkt zum Begriff Artificial Intelligence.

AI bildet dabei einen Überbegriff einer Domäne, welche zum Ziel hat, menschliche Intelligenz mit Computern nachzustellen. Da es an der Definition von „Intelligence" mangelt, ist auch die Definition von AI nicht ganz eindeutig (Russell & Norvig, 1995). Nach Luger und Stubblefield (1993) ist AI ein Teilbereich der Informatik und beschäftigt sich mit der Automatisierung von intelligentem Verhalten. AI ist sozusagen die übergeordnete Domäne in der ML ihren Ursprung hat – unter der Annahme oder Behauptung, dass ML als „intelligent'è einzustufen ist. AI ist demnach eine viel ältere Disziplin, was auch das bekannte Beispiel der *Turing-Test* darlegt. Dabei definierte Alan Turing im Jahr 1950 intelligentes Verhalten einer Maschine als die Fähigkeit einer solchen, menschlichem Verhalten in allen kognitiven Aufgaben gleich zu werden (Russell & Norvig, 1995).

Nun ist da noch die *Statistik* zu nennen. Teilbereiche der Statistik[3] werden of mit den oben genannten Bereichen erwähnt, in Verbindung gesetzt oder gar synonym

[2] Steht dabei synonym für ein Programm

[3] Frequentistische Statistik als auch die Bayesche Statistik

verwendet.[4]. Dazu ist auch wieder zu sagen, dass dies seine guten Gründe hat, obwohl die synonyme Verwendung oftmals für Verwirrung sorgt. Statistik ist zwar eine in sich definierte Domäne mit abgegrenztem und umfassendem Forschungsgebiet, jedoch bedienen sich viele andere Disziplinen der Methoden, Verfahren und Erkenntnissen der Statistik. So auch ML und Bereiche von AI. Eine tiefgehendere Analyse zur Unterscheidung ist an dieser Stelle nicht angebracht, da dies im Grunde genommen keine Relevanz für diese Arbeit hat.

Zusammenfassend ist zu sagen, dass DM ein Begriff im Kontext des Prozesses der Wissensgenerierung zu verstehen ist. ML ist im Vergleich dazu eine eigene Disziplin mit konkreten Verfahren zur rechnerunterstützten Entscheidungsfindung, an der man sich im Rahmen des DM bedient. AI dazu ist als Überbegriff der durch Computer automatisiert erzeugten Intelligenz zu verstehen, in der sich ML als Teilgebiet einordnen lässt. Im Prozess des DM und des ML bedient man sich statistischer Methoden und Verfahren um Erkenntnisse aus Daten zu generieren bzw. um sich stochastische Modelle zur Vorhersage zunutze zu machen.

Im weiteren Verlauf dieser Arbeit wird DM als Begriff des Prozesses verstanden, wohingegen ML als Disziplin der konkreten Anwendung von Algorithmen zur Problemlösung aufgefasst wird.

3.1.2 Kategorien des Machine Learnings

ML wird in den verschiedensten Bereichen eingesetzt, um dort unterschiedliche Probleme zu lösen. Es wird dabei nicht im Detail auf die Anwendungen eingegangen, jedoch ist zu erwähnen, dass lt. (Witten & Frank, 2005) folgende Anwendungen schon als gute Beispiele vorangegangen sind und häufig als Best Practice bezeichnet werden:

- Bilderarbeitung (Image Processing)
- Energie-Forecasting
- Kreditvergabe
- Medizin (Diagnose)
- Marketing und Sales

Ohne nun konkrete Beispiele dazu zu nennen – davon gibt es unzählige – hat sich in diesen Feldern ML bereits gut etabliert. Dies bedeutet aber nicht, dass es nicht andere

[4]So trifft man beispielsweise oft auf die Begriffe „Statistical Learning“

Anwendungsfelder gibt, in denen ML zum Einsatz kommt und bereits gute Ergebnisse liefert. Die Bereiche dehnen sich laufend aus und ML rückt, wie bereits oben erwähnt, an die Front vieler Anwendungsszenarien der unternehmerischen Praxis.

Anwendungsfelder sollten eine bestimmte *Problemstruktur* aufweisen, um die Anwendung von ML Verfahren überhaupt möglich zu machen . Die Verfahren des ML lassen sich grundsätzlich in drei Kategorien einteilen: Dem *Supervised Learning*, *Unsuperviesed Learning* und dem *Reinforcement Learning*.

Das Ziel des *Supervised Learnings* ist es, von einem Modell mit gelabelte Trainingsdaten zu lernen, um Vorhersagen über die Zukunft zu machen. Der Begriff „Supervised" steht damit für eine Auswahl an Datensätzen (Trainingsdaten), bei dem das Ergebnis der Vorhersage (Labels) schon feststeht und dem jeweiligen Datensatz zugeordnet ist. Ein typisches und sehr ausgereiftes Beispiel dafür ist der *Spam-Filter*. Dabei wird ein supervised ML Algorithmus auf eine ganze Reihe an Email-Inhalten (Corpus) trainiert. Diese „Trainings-Emails" besitzen ein Label, dass dieses Email als „Spam" oder „Ham"[5] markiert. Damit wird nun versucht, zukünftig eintreffende Emails in einer dieser beiden Kategorien – also Spam oder Ham – zu klassifizieren. Da die Labels in diesem Beispiel nur diskrete Labels – also keine kontinuierlichen Werte – aufweisen können, spricht man von einer *Klassifikations-Aufgabe*. Im Gegensatz dazu wäre das Label mit kontinuierlichen Werten versehen, d.h. die Labels können fortlaufende nummerische Werte annehmen, so spricht man von einer *Regression*. Abbildung 3.1 verdeutlicht nochmals die Grundstruktur des Supervised Learnings.

Zu einem späteren Zeitpunkt werden konkrete Verfahren dazu vorgestellt. Eine wichtige Anmerkung im Zusammenhang mit Supervised Learning ist, dass das Lernen des ML-Algorithmus auf der Richtigkeit und Zuverlässigkeit der Labels basiert (Brink, Richards & Fetherolf, 2017). Das soll heißen, wenn Labels für Datensätze erstellt werden, ist darauf zu achten, das diese möglichst der Wirklichkeit entspricht, bezogen auf das, was erreicht werden soll. In dem zuvor genannten Beispiel des Spam Filters würde das bedeuten, dass grundsätzlich der Mensch auf Basis seiner subjektiven Empfinden entscheidet, ob eine Email Spam oder Ham ist und damit die Vorlage für den Lernalgorithmus liefert. Dieses Beispiel ist dafür vielleicht nicht gerade prädestiniert zur Verdeutlichung der Problematik, die dabei aufkommen kann, aber wenn man sich, wie in dieser Arbeit der Fall ist, auf Schätzungen bezieht, so ist dieser Aspekt kritisch zu betrachten und ggf. Strategien für den Umgang mit vagen Labels zu entwickeln. Auf diese Thematik wird zu einem späteren Zeitpunkt noch Stellung genommen.

Bei einer Supervised Learning Problemstellung ist sozusagen der Output bzw. das Ergebnis eines Datensatzes in Form von Labels bereits bekannt. Beim *Unsupervised Learning* hingegen sind die Ergebnisse in den Trainingsdaten unbekannt bzw. existieren gar nicht. Dabei wird von einer *unbekannten Struktur* der Daten gesprochen (Raschka, 2015). Mit Unsupervised Learning Verfahren wird dahingehend versucht,

[5]Eine geläufige Umgangsbezeichnung für eine Email die gewünscht ist und damit nicht als Spam zu sehen ist

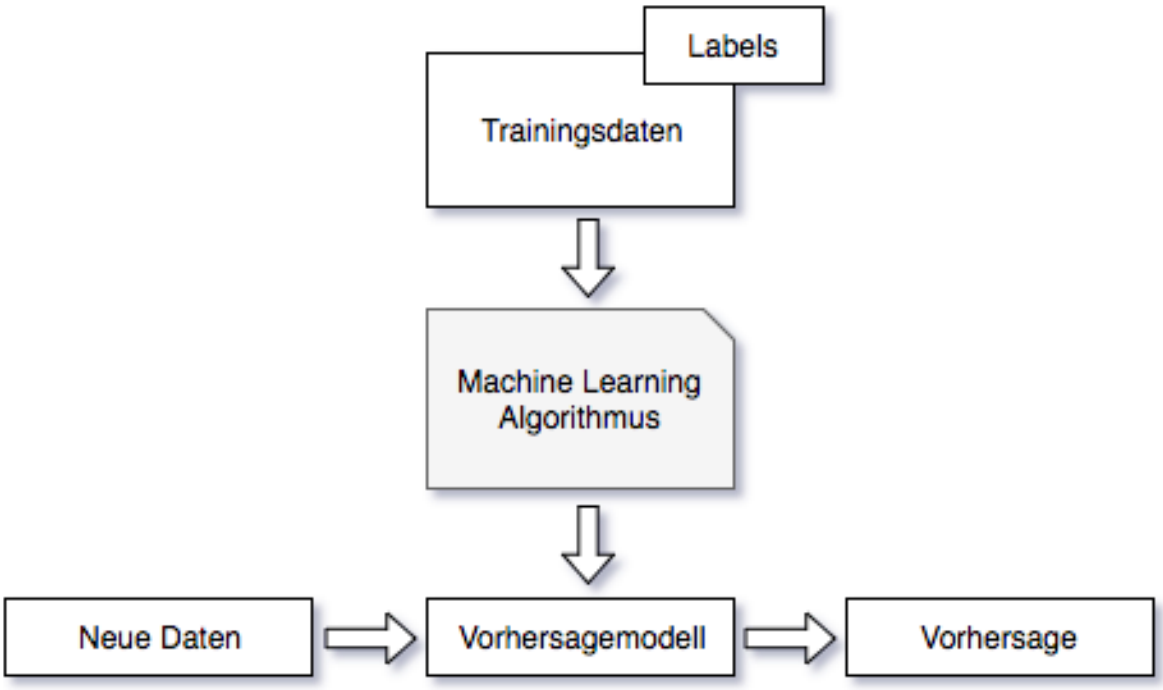


Abbildung 3.1: Struktur des Supervised Learnings (Raschka, 2015)

Strukturen in den Daten zu erforschen und daraus bedeutsames Wissen zu extrahieren – ohne etwas über den Ausgang oder der Ergebnisse zu wissen. Als Beispiel-Verfahren in Bezug auf Unsupervised Learning kann das *Clustering* genannt werden. Dabei wird versucht, Datensätze in Gruppen (Cluster) zu strukturieren, welche eine bedeutende und ähnliche Eigenschaften haben, ohne die Eigenschaften jedoch nennen zu können. Dieses Verfahren erlaubt es beispielsweise Unternehmen, deren Kundengruppen zu entdecken und nach bestimmten Interessen zu strukturieren, um diese in weiterer Folge gezielt mit Marketingmaßnahmen ansprechen zu können. Eine einfache Darstellung in Abbildung 3.2 zeigt, wie nicht-gelabelte Daten in drei unterschiedliche Gruppen, in Abhängigkeit von den Eigenschaften $x1$ und $x2$, strukturiert werden können. Zur Vertiefung von Unsupervised Learning Verfahren und den mathematischen Hintergrund verweisen wir auf Literatur von Celebi und Aydin (2016).

Eine weitere Kategorie des ML ist das *Reinforcement Learnin*g, dass zum Ziel hat, ein *System (Agenten)* zu entwickeln, welches in der Lage ist, mit dem Feedback der *Umwelt*, sich ständig zu verbessern. Dabei liegt die Voraussetzung zugrunde, dass die Umwelt ein sogenanntes *Reward*-Signal für den Lernalgorithmus bereithält und diesen dem Agenten mitteilt (Raschka, 2015). Damit kann das Reinforcement Learning ähnlich dem Supervised Learning gesehen werden, mit der Ausnahme, dass die Reward-Funktion nicht die Wahrheit über den tatsächlichen Ausgang darstellt, sondern dabei lediglich ein Maß berechnet wird, wie gut eine Aktion des Agenten bezogen auf die Reward-Funktion war. Damit kann ein Agent in Interaktion mit seiner Umwelt eine Reihe an Aktionen lernen und dabei die Reward-Funktion über einen *Trial-and-Error-*

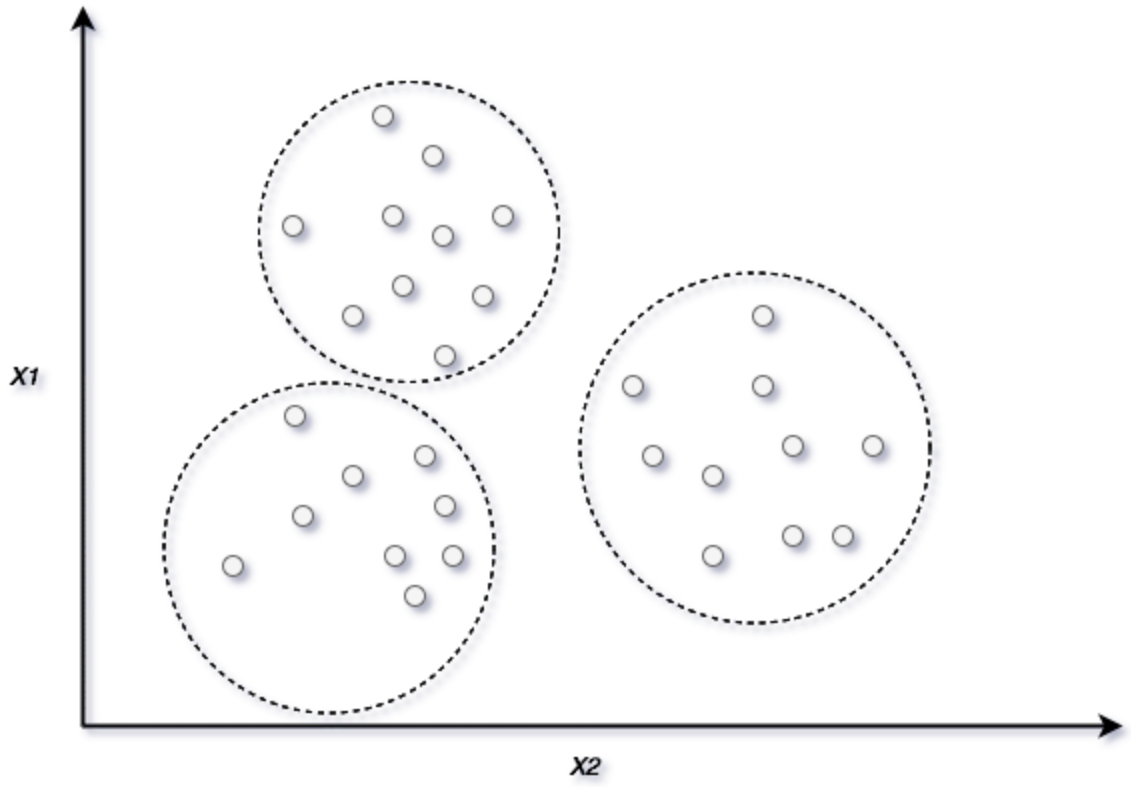


Abbildung 3.2: Clusteranalyse mit nicht-gelabelten Daten (Raschka, 2015)

Ansatz maximieren (siehe Abbildung 3.3. Ein bekanntes Beispiel dafür ist der *Schach Computer*. Dabei kann der Algorithmus auf Basis bestimmter Züge des Agenten und der jeweiligen Aufstellung des Feldes (Umwelt) entscheiden, wie der nächste Zug gesetzt wird. Der Reward kann dabei als „gewonnen“ oder „verloren“ am Ende des Spiels definiert werden. Sugiyama (2015) beschreibt ausführlich und detailliert die Verfahren des Reinforcement Learnings.

Es wurden nun die grundlegenden Lernstrukturen des ML erläutert. Darauf aufbauend wird zu einem späteren Zeitpunkt in Kapitel 4 eines dieser Lernstrukturen – dem Supervised Learning – im Detail betrachtet.

3.2 Ein Vorgehensmodell für Data Mining Projekte

Der folgenden Abschnitt hat zum Ziel, das Prozessverständnis des DM inklusive dessen Teilschritte näher zu betrachten. Damit wird ein theoretisches Grundgerüst zur Vorgehensweise im Rahmen der Fallstudie in dieser Arbeit erarbeitet.

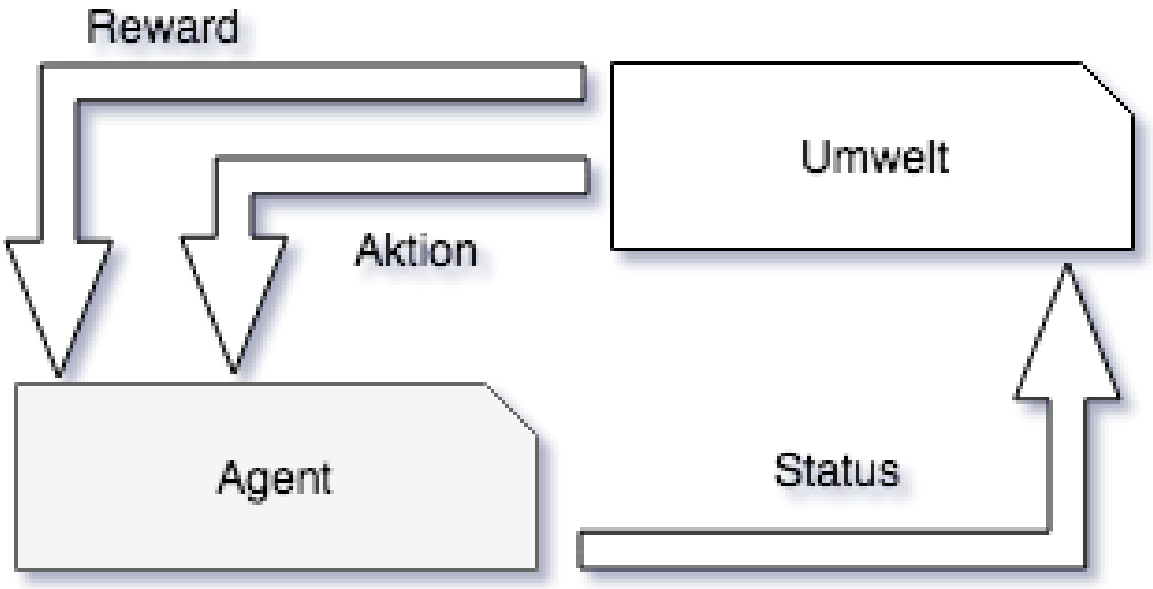


Abbildung 3.3: Grundlage des Reinforcement Learnings (Raschka, 2015)

Wie bereits in Kapitel 2 erwähnt, haben sich Vorgehensweisen und Prozesse zur Entwicklung von nutzenorientierter Software herausgebildet und über die Zeit evolutioniert. Dieses Wissen aus der Domäne des Software-Projektmanagements wurde in den neunziger Jahren auf DM-Projekte übertragen. So kam es, dass Piatetsky-Shapiro und Frawley (1991) erstmalig ein Prozessmodell für DM-Problemstellungen publizierten. Die Forscher heizten damit ein Thema an, welches in den darauffolgenden Jahren immer mehr in den Mittelpunkt des wissenschaftlichen Diskurses in Verbindung mit DM rückte. Dieses Prozessmodell wurde von Fayyad, Piatetsky-Shapiro und Smyth (1996) weiterentwickelt und unter dem Namen *Knowledge Discovery in Databases (KDD)* etabliert. Das KDD-Prozessmodell bietet eine Referenz zur Vorgehensweise für die Entwicklung eines DM-Projektes und beinhaltet neun Teilschritte, auf welche jedoch hier nicht im Detail eingegangen wird. Das KDD diente in den darauffolgenden Jahren als Referenzmodell bzw. Ausgangspunkt für die Entwicklung neuer Prozessmodelle im DM Umfeld. An dem KDD-Prozessmodell orientierten sich demnach eine Vielzahl an weiteren Modellen mit unterschielichen Schwerpunkten und Stoßrichtungen innerhalb der Domäne des DM. So sind an dieser Stelle stellvertretend *SEMMA*, das *5 A's* Modell, das Prozessmodell nach Cabena oder das Prozessmodell nach Annand & Buchner zu nennen (Marbán, Mariscal & Segovia, 2009).

Aus all den Ansätzen der oben genannten Modelle wurde von Forschern und Vertretern aus der unternehmerischen Praxis versucht, ein quasi Standard-Modell zu entwickeln, welches als einheitliche und konsistente Basis für DM-Projekte, quer über jede Industrie, einzusetzen ist. Demnach entwickelten und veröffenlichten Vertreter von Daimler Chrysler, SPSS und NCR im Jahr 1996 den sogenannten *Cross-Industry Standard Process for Data Mining (CRISP-DM)* (Shafique & Qaiser, 2014). Dieses Modell

wurde in den darauffolgenden Jahren verfeinert und ist heute als der de-facto Standard anzusehen, wenn es darum geht, DM Projekte in die Praxis umzusetzen. Aus diesem Grund wird nachfolgend das CRISP-DM näher erläutert.

Das CRISP-DM ist als ein hierarchisches Referenzmodell zu verstehen und bietet eine Orientierung in der Vorgehensweise in DM-Projekten. Es besteht aus den vier Abstraktionsebenen *phases*, *generic tasks*, *specialized tasks* und *process instances* (Chapman et al., 2000). Diese Hierarchie bietet zum einen, durch die generischen Aufgaben, eine Ebene, die als allgemeine Aufgaben verstanden werden kann, welche für alle möglichen DM-Problemstellungen geeignet ist. Erst unter dem Einfluss der Rahmenbedingungen (Context) werden konkretere Aufgaben daraus abgeleitet. Dies führt schlussendlich bis zur process instance, welche sich auf ein spezifisches Problem in einem DM Projekt bezieht. Dieser Zusammenhang ist in Abbildung 3.4 dargestellt. Allgemein gesprochen dienen demnach die Abstraktionsschichten dazu, eine Hierarchie in die Vorgehensweise einzuführen, um sich von der groben Problemstellung sukzessive bis zu den Detailtätigkeiten durchzuarbeiten und dabei stets eine Referenz bereitstellt. In diesem Zusammenhang gilt es zwischen dem CRISP-DM *Referenzmodell* und dem dazugehörigen *User Guide* zu unterscheiden. Das Referenzmodell bietet, wie eben dargelegt, lediglich eine Referenz zur Vorgehensweise, d.h. das Modell kann zweckmäßig erweitert oder angepasst werden und gibt Aufschluss darüber, *was* in einem DM Projekt zu machen ist. Der User Guide hingegen ist ein Sammelsurium an detaillierten Beschreibungen und Tipps *wie* ein DM Projekt abzuwickeln ist (Chapman et al., 2000).

Die Phasen des Modells gliedern sich in sechs Teilschritten:

- Business Understanding
- Data Understanding
- Data Preparation
- Modeling
- Evaluation
- Deployment

Aus Abbildung 3.5 ist zu erkennen, dass die Dastellung in Phasen zugleich die Beziehung der einzelnen Teilschritte offenbart und damit den idealtypischen Lebenszyklus eines DM Projektes abbildet (Wirth & Hipp, 1999). Jede Phase beinhaltet auch eine Referenz an generischen Aufgaben und deren Ergebnis-Artefakte, welche im *Anhang 1* vollständig angeführt sind. Im weiteren Verlauf der Arbeit wird an den oben genannten Phasen festgehalten. Die Inhalte der Phasen sind dabei als Orientierung für das nachfolgende Kapitel zu verstehen.

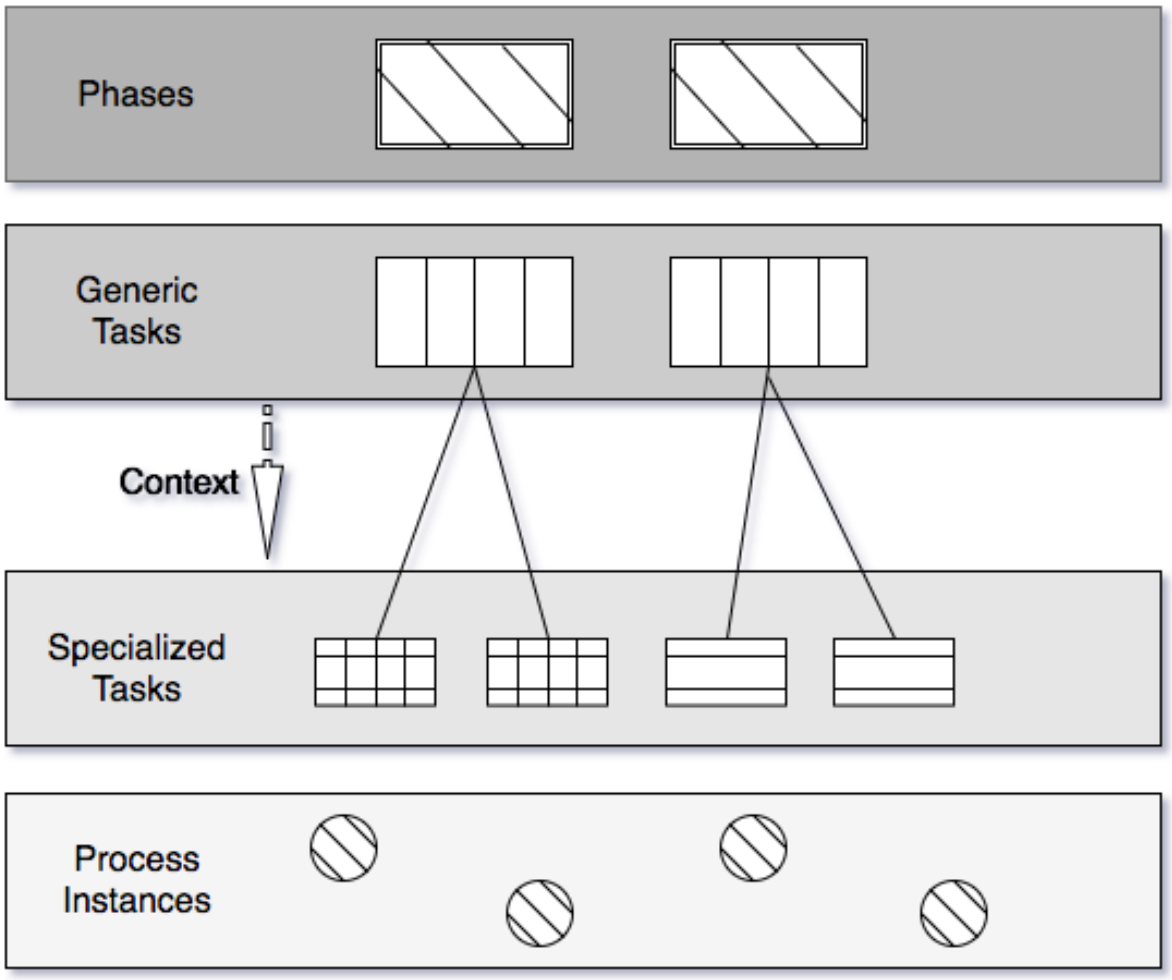


Abbildung 3.4: Die vier Ebenen des CRISP-DM Modells (Chapman et al., 2000)

3.2.1 Business Understanding

Business Understanding ist die initiale Phase und hat zum Ziel, das Projektvorhaben und dessen Anforderungen aus der Geschäftsperspektive zu betrachten und zu formulieren. Es sollte daher gleich zu Beginn eine Analyse durchgeführt werden, um möglichen Faktoren die das Projekt beeinflussen könnten, zu identifizieren (Brink et al., 2017). Eine möglichen Konsequenz bei Vernachlässigung dieser Phase könnte sein, dass nachgelagert viel Aufwand in die Erarbeitung der richtigen Antworten auf die falschen Fragen investiert wird (Chapman et al., 2000). Ein weitere wichtiger Schritt in diesem Zusammenhang ist die Erhebung der IST-Situation und die Beschreibung einer SOLL-Situation. Brink et al. (2017) empfehlen, dazu in der ersten Phase DM Projektes konkrete Anwendungsfälle abzuleiten und zu beschreiben. Dabei gibt es fünf Fragen, die es zu beantworten gilt:

1. Warum ist der Anwendungsfall wertvoll?

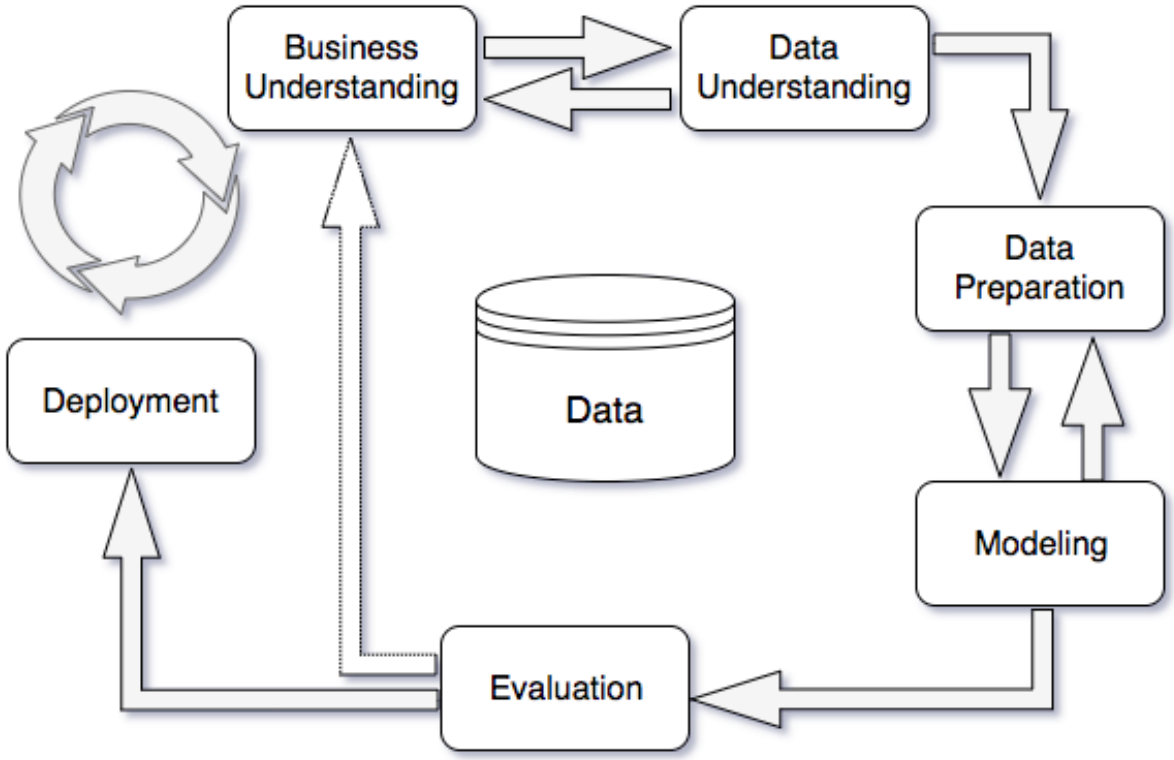


Abbildung 3.5: Cross-Industry Standard Process for Data Mining (CRISP-DM) (Wirth & Hipp, 1999)

2. Welche Art von Trainingsdaten werden benötigt?

3. Was wäre eine angemessene ML Modellierungsstrategie?

4. Welche Evaluierungskriterien sollten für die Vorhersage verwendet werden?

5. Sind die vorhandenen/verfügbaren Daten ausreichend für die Vorhersage dieses Anwendungsfalls?

Wie nun zu erahnen ist, sind die angeführten Fragen essentielle Richtungsweiser für den Projekterfolg, jedoch in dieser Phase teilweise noch kaum zu beantworten. Es sollte in dieser Phase zumindest die erste Frage beantwortet werden. Für die restlichen vier Fragen können und sollten Annahmen getroffen werden und im Verlauf der nächsten Phasen Stück für Stück nachgeschärft werden. Demnach sollten die soeben besprochenen Punkte schriftlich festgehalten werden und als Basis – als sozusagen vorläufiger Projektplan – für die nächste Phase dienen.

3.2.2 Data Understanding

Die Phase *Data Understanding* beginnt mit einer ersten Datensammlung und explorativen Datenanalyse. Dabei sollte ein Verständnis für die vorliegenden Daten erlangt werden und an dieser Stelle eine Qualitätsschätzung der Datenbasis vorzunehmen. Weiters sollten erste Einblicke und Hypothesen aufgestellt werden. Dazu dient üblicherweise die deskriptive Statistik, um etwaige Muster und Abhängigkeiten zu entdecken. Unterstützend können Diagramme und grafische Hilfsmittel eingesetzt werden, um Sachverhalte zu veranschaulichen. Es gilt hier lediglich ein Überblick über die Eigenschaften der vorhandenen Daten zu verschaffen und einen Abgleich zu ggf. noch benötigten Daten vorzunehmen. Eine Überschneidung mit der ersten Phase ist durchaus möglich und teilweise auch notwendig: Ohne Verständnis der Daten können in den meisten Fällen auch keine Ziele für das Projekt festgelegt werden. Ein letzter und wichtiger Punkt in dieser Phase ist die Sicherstellung der Datenqualität. Was bedeutet dies? In Datenaufzeichnungen aus der Praxis kommt es häufig vor, dass Daten nicht vollständig sind, sprich, manche Felder einfach ohne Inhalt sind. Insbesonders bei manuell, durch den Menschen erhobene und festgehaltene Daten kann es leicht zu Fehlaufzeichnungen kommen (Wirth & Hipp, 1999). Auch die inhaltliche Relevanz für den bestimmten Anwendungsfall sollte geprüft werden, ob bestimmte Daten überhaupt weiter betrachtet werden sollen bzw. schlussendlich dann in die Modellierung miteinfließen sollten (Chapman et al., 2000).

3.2.3 Data Preparation

Die Data Preparation Phase beinhaltet alle Aktivitäten rund um die Datentransformation und Aufbereitung zur weiteren Verarbeitung in einem DM-Modell. Diese Art von Aufgaben sind meist nicht einmalig in einem Projekt durchzuführen, sondern stellt eine wiederkehrende Aufgabe dar – losgelöst von spezieller bzw. bestimmter Reihenfolge der Verarbeitungsschritte (Wirth & Hipp, 1999). Zu den Aufgaben zählen beispielsweise Attribut- bzw. Feature Selektion[6], Daten Bereinigung, Feature Engineering[7] und die Transformation der Daten in ein bestimmtes Format, mit dem die Daten in einem Modell verarbeitbar werden. Diese Phase kann laut Aufzeichnungen aus der Praxis bis zu 80% des gesamten Zeitaufwandes eines Projektes in Anspruch nehmen (Brink et al., 2017). Neben der Tatsache, dass Daten aus der Praxis in ein bestimmtes Format gebracht werden müssen, so gilt diese Phase auch als besonders entscheidend für ein gutes Evaluierungsergebnis. Damit ist das Feature Engineering gemeint, welches sich als besonders geeignet und wichtig herausstellte, um die Performance (gemessen anhand der jeweiligen Metrik) des Vorhersagemodells zu steigern. Feature Engineering ist dabei so vielfältig und umfangreich, dass an dieser Stelle nicht

[6] Bezeichnet die Auswahl von Attributen (Spalten) in einem Datenset tabellarischer Form

[7] Bezeichnet die Aufgabe der Extraktion von zusätzlichen Attributen aus den bereits vorliegenden Datenset zur Verarbeitung im Modell, um dieses gemäß der ausgewählten Metriken zu verbessern

weiter darauf eingegangen werden kann. Dazu wird zu einem späteren Zeitpunkt in Kapitel 5 dem Feature Engineering in Zusammenhang mit der Verarbeitung von unstrukturiertem Text ein gesamtes Kapitel gewidmet.

3.2.4 Modeling

Die Modeling Phase ist jene Phase, in der die eigentliche Umsetzung in Form eines Modells erfolgt. Dabei können meist unterschiedliche Techniken für das ein und dasselbe Problem angewendet werden. Im Gegensatz dazu gibt es jedoch auch Problemstellungen, auf die nicht jede Technik anwendbar ist. Kurzum: Es muss eine Auswahl einer Modellierungsstrategie anhand der Problemstellung getroffen werden. In weiterer Folge gilt es auch, die „politischen Anforderungen" mit einfließen zu lassen (Chapman et al., 2000). Weiters ist diese Phase auch ganz eng an die vorherige Phase gekoppelt bzw. können sich diese überschneiden. So kommt es in der Praxis durchaus vor, dass sich z.B. die Repräsentation des Datensets als ungeeignet oder unzureichend für die Verarbeitung im Modell herausstellt oder neue Features bei der Modellierung entdeckt werden, welche die Überarbeitung der Datenrepräsentation erfordern (Witten & Frank, 2005). Abschließend in dieser Phase gilt es, das Modell anhand ausgewählter Metriken zu beurteilen, ob es schlussendlich den gesetzten Anforderungen entspricht (Chapman et al., 2000).

Da der Prozess der Modellierung und die darin einzusetzenden Verfahren eine genauere und umfangreichere Erläuterung bedarf, wird auf Kapitel 4 verwiesen, wo das Thema der Modellierung und Evaluation ausführlicher behandeln wird.

3.2.5 Evaluation

Die Evaluation Phase beinhaltet in erster Linie die Analyse der „Brauchbarkeit"des Modells bzw. die Auswahl von einem geeigneten Modell in Anbetracht der Geschäftsziele. Wo in der vorherigen Phase Modelle hinsichtlich Genauigkeit und Allgemeingültigkeit getestet wurden, so wird in dieser Phase getestet, ob es den spezifischen Geschäftsanforderungen tatsächlich nachkommt bzw. diese zu welchem Grad erfüllt (Wirth & Hipp, 1999). Grundsätzlich gilt hier, dass nicht einzig und alleine die Modelle wichtig sind, sondern auch die Erkenntnisse aus dem DM-Projekt welche die initiale Fragestellung ergänzen und/oder erweitern (ebd.). Gleichermaßen sind Erkenntnisse, die den DM Prozess selbst betreffen, zu analysieren und Verbesserungen für zukünftige Projekte daraus abzuleiten (Chapman et al., 2000). Zusammenfassend lässt sich sagen, dass in dieser Phase ein letzter fachlicher Check vor dem Deployment vorgenommen wird, um sicherzustellen, dass es den Anforderungen entspricht und alle Erkenntnisse – auch rund um die eigentliche Problemstellung – dazu darlegt.

3.2.6 Deployment

Nun gilt es, das fertige Modell in die Produktion überzuleiten, so dass es für den Kunden/Nutzer am Ende des Tages auch benutzbar wird. Dies kann sich auf der einen Seite auf eine ganz simple Aufgabe beschränken – z.B. dem einmaligen Erstellen eines Reports – auf der anderen Seite eine komplexe Anwendung sein, in dem das Modell eingebettet werden soll. In jedem Fall ist es aber wichtig, vorab zu wissen, wie das Modell in Einsatz gebracht werden soll. Somit ist ein Deployment Plan zu erstellen, welcher die einzelnen Schritte und wie diese durchzuführen sind im Detail beschreiben(Chapman et al., 2000). Ein wichtiger Bestandteil dieser Phase ist es auch, die infrastrukturelle Planung und Bereitstellung der Ressourcen sicherzustellen (ebd.). Dies ist stark abhängig von vielerlei Aspekten, wie z.B. die zu verarbeitende Datenströme, Latenz, Verfügbarkeit, etc. Diese Phase als solche kann sehr umfangreich im Sinne der Deployment-Strategie und deren Komplexität ausarten. Da diese Phase jedoch keinen wesentlichen Einfluss auf die Aufgabenstellung in dieser Arbeit hat – nämlich die Erstellung eines Modells und nicht dessen Deployment in ein produktives System – wird diese Phase im weiteren Verlauf der Arbeit inhaltlich nicht näher ausgeführt.

4 Modellierung und Evaluierung von Klassifizierungs-Problemen

In diesem Kapitel gehen wir detaillierter auf die Modellierung und Evaluierung von Klassifizierungsverfahren ein. Zu Beginn wird erläutert, was unter einem Klassifizierungs-Problem zu verstehen ist. Nachfolgend wird ein Workflow für die Erstellung eines ML-Modells vorgestellt. Die Abschnitte danach widmen sich, in selber Reihenfolge wie der Workflow, der genaueren Ausführung der einzelnen Schritte und Aufgaben mit Bezug zur Problemstellung in dieser Arbeit. Dabei wird auf mögliche Modellierungsstrategien eingegangen und ein damit theoretisches Fundament bzw. Vorgehensweise für die Fallstudie erarbeitet. Die Betonung liegt dabei auf „mögliche" Strategien, da eine vollständige Ausführung der Möglichkeiten im Rahmen dieser Arbeit schier unmöglich ist. Dies gilt auch für die Evaluierungsstrategien, wenngleich hier die Möglichkeiten nicht so vielfältig sind.

4.1 Klassifizierung durch Machine Learning

Allgemein gesprochen, stellt ein Klassifikationsproblem die Herausforderung an ein ML-Modell, die Struktur eines Datensets bzw. dessen *Klassen* zu „lernen". Was bedeutet das nun? Grundsätzlich muss ein Datenset zur Verarbeitung in einem ML-Verfahren in der Form einer Tabelle vorliegen – vergleichbar mit der Darstellung in einem Tabellenkalkulationsprogramm. Analog dazu werden die Spalten im Kontext des ML als *Features* oder auch *Attribute* bezeichnet, Zeilen hingegen als sogenannte *Instanzen* (Brink et al., 2017; Witten & Frank, 2005). Die sogenannten Klassen – häufig auch als *Class Labels* bezeichnet – identifizieren eine Instanz mit einer bestimmten semantischen Bedeutung im Kontext der Anwendung. Mit anderen Worten: Ein Class Label ist eine Eigenschaft, die von Interesse für die Anwendung ist bzw. nach denen das Datenset über eine Spalte gruppiert ist. Um dies zu veranschaulichen, wird auf einen Auszug aus einem beliebten und oft zitierten Datenset der gesunkenen Titanic, welches öffentlich und frei zugänglich ist (*Titanic: Machine Learning from Disaster*, 2012). Dieses Datenset wird ebenso in den nachfolgenden Abschnitten zur Veranschaulichung von Ausführungen unterstützend eingesetzt (siehe Tabelle 4.1).

In diesem Datenset handelt es sich um die Aufzeichnung von Passagierdaten der damalig verunglückten Titanic. Dieses Datenset ist mit einem Vermerk je Instanz

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S

Tabelle 4.1: Auszug aus dem Datenset „Titanic" (*Titanic: Machine Learning from Disaster*, 2012)

versehen, ob eine Person das Unglück überlebt hat („Survived" = 1) oder eben nicht („Survived = 0"). Aus Tabelle 4.1 ist ersichtlich, dass hierbei die Spalte „Survived" als Class Label dient und alle anderen Spalten als Features behandelt werden. Ohne auf die Bedeutung der Features im Beispiel näher einzugehen, liegt nun das Ziel eines ML-Modells darin, die Struktur der Features zu identifizieren, welche jeweils eine Klasse charakterisiert. Dies kann bei einer Hand voll Features und wenigen Instanzen durchaus durch die manuelle Sichtung eines Menschen bewältigt werden. Wenn es jedoch darum geht, mehrere Features und Instanzen, jenseits der 50 an der Zahl, einer bestimmten Klasse zuzuteilen, so ist ein ML-Klassifizierungsverfahren geradezu prädestiniert für diese Aufgabe. Der Algorithmus wird damit darauf „trainiert", die Struktur, bezogen auf die jeweilige Klasse, zu identifizieren und diese Struktur auf Datensets zu projezieren, welche ohne class labels ausgestattet sind. Dies würde für das Titanic-Beispiel bedeuten: Wie ist der Ausgang einer Person (=Survived (0,1)) abhängig von dessen Geschlecht, der gebuchten Kategorie, Alter, etc. In diesem Zusammenhang wird die Frage aufgeworfen: Welche Features sind denn nun relevant für die Vorhersage des Class Labels bzw. welche sollten in die Modellierung mit aufgenommen werden? Die Antwort auf diese Frage ist Teil der Modellierungsstrategie und wird in Abschnitt 4.4 entsprechend behandelt. Da es in dem Titanic-Beispiel nur zwei Ausgänge gibt – nämlich überlebt oder nicht überlebt – spricht man auch von einem *binären* Klassifikations-Problem. Es gibt folglich auch ein *Multiclass-Problem* bei dem es mehrere Labels innerhalb einer Klasse geben kann (Aggarwal, 2015). Diese Tatsache bzw. deren Unterscheidung ist jedoch nicht weiter von großer Bedeutung, da ein Großteil der Verfahren auch mit mehreren Klassen einzusetzen sind. Falls es im weiteren Verlauf der Arbeit unterschiedlicher Behandlung bedarf, wird dies entsprechend erläutert.

Warum wird nun von einem Klassifizierungs-Problem gesprochen? Das „Problem" liegt in der Eigenschaft eines ML-Modells, meist nicht alle Class Labels korrekt vorherzusagen und dies damit eine mögliche „Lücke" für bestimmte Anwendungsfälle darstellt. Damit hängt der Einsatz und die Legitimation des ML-Modells stark von den Rahmenbedingung und Anforderungen der Anwendung ab (Brink et al., 2017). Was heißt das konkret? Blicken wir doch mal in die Zukunft und stellen uns eine Anwendung für das autonome Fahren mit einem PKW auf der Straße vor. Wäre nun ein ML-Anwendungsfall jener, welcher das unmittelbare Spurwechseln von einem Auto

im Autobahnbetrieb vor einem Überholmanöver vorhersagen würde. Damit würde das überholende Fahrzeug beim Annähern des möglicherweise ausscherenden Fahrzeuges automatisch abbremsen bzw. die Geschwindigkeit reduzieren. Ohne nun zu wissen, ob dieser Anwendungsfall bereits in der Forschung und Entwicklung existiert oder je existieren wird, so ist anzunehmen, dass für die Genauigkeit der Vorhersage aufgrund der hohen Sicherheitsanforderungen ein kritisches und vermutlich hohes Maß anzusetzen ist. Hypothetisch gesprochen: Wäre die zu erreichende Genauigkeit für diesen Anwendungsfall bei 99% und der Algorithmus würde lediglich 97% erreichen, so erfüllt ML (zumindest diese Technologie) nicht die Kriterien und es sind alternative Technologien in Betracht zu ziehen. Selbstverständlich sind an dieser Stelle noch nicht die Metriken wie die soeben genannte Genauigkeit behandelt worden. Wie diese „Lücke" eines Modells zu bestimmen ist, wird an einem späteren Zeitpunkt in diesem Kapitel erläutert. Vielmehr sollte die vorangegangene Ausführung gezeigt haben, was Legitimation und Restriktionen hinsichtlich des Einsatzes von ML in der Praxis bedeuten kann.

Zusammenfassend kann gesagtwerden, dass die Klassifizierung in einem ML-Modell grundsätzlich immer in zwei Phasen durchgeführt wird. In der ersten Phase, der Trainingsphase, wird anhand von Trainingsdaten (mit Class Labels) eine Art mathematische Repräsentation der Struktur der Features zu den Class Labels erlernt. Selbstverständlich reichen dazu nicht drei Instanzen wie im Beispiel abgebildet. Es können theoretisch beliebig viele Instanzen sein, die verarbeitet werden können. Nach unten gibt es zwar keine klar definierte Grenze im wissenschaftlichen Kontext, jedoch spricht sich die Community von Scikit-learn, einem ML-Framework für die Programmiersprache Python, für mindestens 50 Instanzen als Daumenregel aus (Pedregosa et al., 2011). In Phase zwei wird das Modell auf Testdaten (ohne Class Labels) angewendet und damit die Class Labels durch das Modell versucht zu bestimmen. Da in der Trainingsphase die Class Labels vorhanden sind, und damit die Grundwahrheit bekannt ist, wird es auch in die Kategorie des Supervised Learnings eingestuft, welche in Kapitel 3 in den Grundzügen bereits beschrieben wurde.

4.2 Der Machine Learning Workflow

In diesem Abschnitt wird eine Vorgehensweise vorgestellt, welche einen idealtypischen Ablauf der Modellierung und Evaluierung beschreibt. Der Ablauf kann als Referenz bzw. Vorlage für den Bau von Daten-Pipelines verwendet werden. Data Pipelines sind Kontrollstrukturen in einem Programm, welche eine Abfolge von Transformationen automatisiert ausführt und so von der Eingabe bis zur Vorhersage durchläuft (Raschka, 2015). Der Ablauf, oft auch als Workflow bezeichnet, stellt sich in fünf Hauptkomponenten dar: *Data Preparation*, *Model Building*, *Model Evaluation*, *Model Optimization* und *Prediction* (Brink et al., 2017).

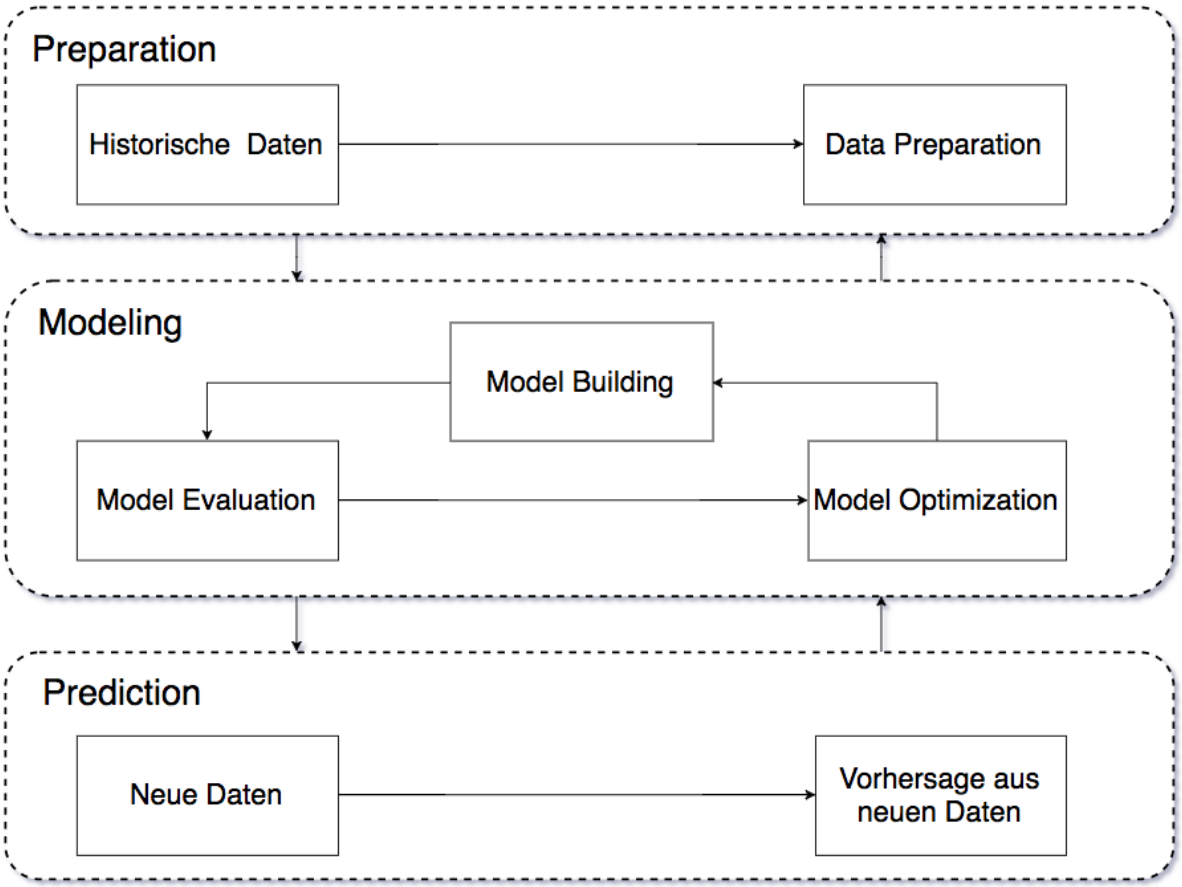


Abbildung 4.1: Der idealtypische ML-Workflow (Brink et al., 2017)

Der Workflow in Abbildung 4.1 beginnt mit der Data Preparation, also die Daten Vorbereitung. Dieser Teilschritt ist im Kontext eines bereits ausgewählten ML-Algorithmus zu verstehen und beinhaltet die konkrete Vorbereitung eines Datensets für eine konkrete Implementierung eines Modells. Diese Phase ist damit im Vergleich zur Phase Data Preparation im CRISP-DM als spezielle Daten Vorbereitung auf der Ebene der Implementierung zu verstehen. Das Model Building stellt die Implementierung selbst dar, d.h. die Umsetzung in einer ausgewählten Programmiersprache oder Werkzeug. Danach erfolgt die Model Evaluation. In diesem Arbeitsschritt erfolgt die Analyse der *Validität* und der *Reliabilität* des Modells, d.h. ist das Modell für den definierten Anwendungsfall geeignet? Wenn ja, welche Abschläge oder Nachteile müssten in Kauf genommen werden? Daran gekoppelt ist die Vorhersage von *neuen Daten*, d.h. Testdaten ohne Class Labels. Im letzten Schritt, der Modellierung und Evaluierung kann aus den Ergebnissen der Vorhersagen durch das Modell auf Testdaten Optimierungen vorgenommen werden. Model Optimization beinhaltet in diesem Zusammenhang in erster Linie die „innere" Optimierung von sogenannten *Tuning Parameters* des jeweiligen Modells.

In soeben beschriebener Reihenfolge werden in den nachfolgenden Abschnitten die Inhalte und mögliche Ausprägungen im Detail erläutert.

4.3 Die Datenrepräsentation in ML-Verfahren

Data Preparation ist hierbei als spezielle Vorbereitung für ein bestimmtes ML-Verfahren zu verstehen. So setzen bestimmte Algorithmen für die Verarbeitung der Daten bestimmte Eigenschaften voraus. Diese Eigenschaften beziehen sich insbesondere auf die*Vollständigkeit*, *Vergleichbarkeit* und *Interpretierbarkeit* der Daten für ein ausgewähltes ML-Verfahren. Nachfolgend werden konkrete Verfahren und Techniken vorgestellt, welche diese angeführten Eigenschaften adressieren.

4.3.1 Der Umgang mit nicht vorhandenen Werten

In diesem Abschnitt wird das Problem von fehlenden Daten und mögliche Lösungsstrategien dazu dargelegt. Damit ist die Eigenschaft der Vollständigkeit der Daten für ein bestimmtes Verfahren sicherzustellen. In der Praxis ist es nicht unüblich auf Datenquelle zu stoßen, welche Lücken – im Sinne von fehlenden Werten – beinhaltet. Dies ist grundsätzlich auf den Prozess der Datensammlung zurückzuführen, bei welchem aus den unterschiedlichsten Gründen Daten nicht aufgezeichnet werden. Die entscheidende Frage an dieser Stelle lautet: Welche Bedeutung haben bzw. hatten die fehlenden Werte im Rahmen der Datensammlung? Alleine die Tatsache, dass fehlende Werte im Datenset enthalten sind, kann bedeutende Information für das Verfahren darstellen (Witten & Frank, 2005). Es gilt dabei zu unterscheiden, ob das Fehlen der Daten eine mögliche Bedeutung im Kontext der zu beantworteten Frage hat oder ob die fehlenden Werte unbedeutend sind. Als Beispiel dafür kann dies anhand des Titanic-Datenset erläutert werden: Das Datenset weist fehlende Werte innerhalb des Features „Cabin" und „Age" auf. Die fehlenden Werte in „Cabin" könnte ein Indiz dafür sein, dass jene Passagiere einer niedrigeren sozialen Klasse angehörten und diese Annahme daher bedeutsam für die Vorhersage sein könnte. Hingegen könnte behauptet werden, die fehlenden Werte für „Age" waren einfach ein Defizit im Aufzeichnungsprozess und damit schlicht und ergreifend das Alter nicht erfasst wurde. Kurzum: fehlende Werte innerhalb „Cabin" sind bedeutsam, da sie durch die Tatsache, dass sie fehlen möglicherweise Information für unser Modell bereithält, wohingegen fehlende Werte in „Age" bedeutungslos hinsichtlich des potentiellen Informationsgehalts sind (Annahme). In beiden Fällen sollte unterschiedlich vorgegangen werden.

Für die *bedeutsamen* fehlenden Werte können Dummy-Werte wie z.B. „-1" eingesetzt werden oder wie im Fall des Beispiels für „Cabin" einen informativen Platzhalter für, wie z.B. „fehlt", einzuführen. Nun stellt sich die Frage, warum diese Unterscheidung

getroffen werden sollte? Einige Algorithmen können zwar fehlende Werte, ohne eine Transformation vorab vorzunehmen, verarbeiten. Würde man jedoch diese Unterscheidung nicht vornehmen, so würde man Gefahr laufen, die Performance des Modells massiv zu beeinträchtigen. Zurück zum Beispiel: Würde man für die unbedeutenden fehlenden Werte in „Age" eine „-1" einsetzen, so würde der Algorithmus diese Werte als metrische Werte behandeln und dies maßgeblich (bei verhältnismäßig vielen fehlenden Werten) die Abstandsmaße der Werte zueinander beeinflussen und somit das Ergebnis verfälschen. Das „Unbedeutend" bezieht sich in diesem Fall auf die semantische Bedeutung im Kontext der Fragestellung und nicht auf die Unbedeutsamkeit des Features „Age". Daher wird für fehlende Werte innerhalb von Features, die keine semantische Bedeutung haben – also unbedeutend sind – andere Techniken verwendet, welche man auch als *Imputation* bezeichnet, die wie folgt kurz beschrieben werden.

Die simpelste Strategie im Umgang mit fehlenden Werten in Datensets besteht darin, diese einfach wegzulassen bzw. die gesamte Instanz aus dem Datenset zu entfernen (Brink et al., 2017). Dies ist mit Abstand der einfachste, aber durchaus ein legitimer und praktikabler Weg. Zu Problemen kommt es nur meist dann, wenn zu viele Werte fehlen und diese entfernt werden müssen oder auch das Entfernen von Instanzen, welche bereits ein unterrepräsentiertes Class Labels aufweisen und die Anzahl dadurch weiter reduziert wird.

Eine weitere Strategie ist die Befüllung des fehlenden Wertes mit seinem Vorgänger-Wert, d.h. der Werte einer Instanz zuvor. Dabei wird von einer temporären Ordnung im Datenset ausgegangen, bei der sich von einer Instanz zur anderen der Werte in der betroffenen Spalte nicht ändert (Brink et al., 2017). Dabei ist anzunehmen, dass diese Annahme oft nicht der Realität entspricht. Diese Strategie kann jedoch durchaus attraktiv sein, da sie eine einfache Operation darstellt und für sehr große Datenmengen, bei welcher viele Imputation-Strategien nicht möglich sind, eine alternative Lösung bietet (Brink et al., 2017).

Eine präferierte Strategie ist, mehrere Werte für das Ersetzen des leeren Feldes heranzuziehen. So kann aus den Features beispielsweise der Mittelwert oder der Median verwendet werden, um die fehlenden Werte des jeweiligen Features zu füllen. Dies hat zwar den Vorteil, dass es den Mittelwert der tatsächlichen Werte darstellt und daher eher der Wahrheit entspricht, jedoch durch diese „Glättung" die Varianz in dem Feature zu „verdecken" bzw. zu verkleinern (Witten & Frank, 2005). Um diesem Problem entgegenzuwirken, kann auch ein ML-Verfahren selbst als Imputation-Technik dienen. Damit kann z.B. ein Regressionsmodell mit unabhängigen Variablen der Features abhängig die fehlenden Werte vorhergesagt werden (Soria, Martín-Guerrero, Martinez, Magdalena & Serrano, 2009). Dieses Verfahren ist zwar am repräsentativsten von all den vorangestellten Strategien und Techniken, jedoch hat es den entscheidenden Nachteil der Rechenintensivität bei großen Datenmengen, was sich in der Verarbeitungsdauer bemerkbar macht.

Zu guter Letzt ist zu sagen, dass grundsätzlich darauf zu achten ist, dass fehlende Werte erst gar nicht auftreten, da bei Verwendung von Dummy-Werten oder der Anwendung von Imputations-Strategien immer die Gefahr besteht, die „Realität" zumindest in den Daten zu verzerren und damit möglicherweise Scheinabhängigkeiten zu erzeugen (Witten & Frank, 2005; Soria et al., 2009). Falls es doch unumgänglich sein sollte, fehlende Werte zu ersetzen, so kann keine allgemeine Empfehlung oder Rezept für die Behandlung von fehlenden Werten ausgesprochen werden (Brink et al., 2017). Mit anderen Worten: Die Entscheidung darüber, welche Imputation-Strategie gewählt werden soll, ist individuell auf Basis der Bedeutung der fehlenden Variablen zu treffen.

4.3.2 Standardisierung und Normalisierung von Daten

In vielen Fällen werden verschiedene Features in einem Datenset auf unterschiedlichen Skalen repräsentiert. Dies hat zum Nachteil, dass diese Features nicht dirket zueinander vergleichbar sind. So sind beispielsweise im Titanic-Datenset die Features „Age" und „Pclass" auf unterschiedlichen Skalen-Niveaus repräsentiert. Angenommen ein ML-Verfahren berechnet die (euklidische) Distanz zwischen den beiden Features, so würde jenes Feature, welches die „größere" Skala repräsentiert (Age), deutlich dominieren. Um dieses Problem zu entschärfen, gibt es das Konzept der *Standardisierung* (Aggarwal, 2015). Die Standardisierung wird in Zusammenhang mit der Normalverteilung verstanden. Bei dieser Standardisierung werden metrische Features so transformiert, dass der Mittelwert $\mu = 0$ und die Standardabweichung $\sigma = 1$ beträgt. Betrachtet man den Fall das Feature j den Mittelwert μ_j und Standardabweichung σ_j aufweist, so kann jeder Wert i des j-ten Feature x_i^j gemäß der Gleichung (4.1) berechnet werden (Aggarwal, 2015).

$$z_i^j = \frac{x_i^j - \mu_j}{\sigma_j} \tag{4.1}$$

Ein Großteil der standardisierten Werte sind nun im Intervall $[-3, 3]$ der Normalverteilung angegeben. Damit ist nun das Maß des Features in Standardabweichungen angegeben und die Features miteinander vergleichbar.

Ein zweiter Vertreter dieser Art von Transformation ist die *Normalisierung*. Mit Normalisierung ist in diesem Zusammenhang nicht die Normalverteilung gemeint, sondern die Normalisierung im Sinne einer einheitlichen Skala – also einer normierten Skala. Ein bekannter Vertreter dafür ist das „Min-Max-Scaling" bei dem alle Werte eines Features in ein Intervall $[0, 1]$ transformiert werden (Aggarwal, 2015). Betrachtet man Gleichung (4.2), so stellen min_j und max_j jeweils die untere- und obere Grenze eines Features j dar und X_i^j jeden Wert and der i-ten Stelle.

$$y_i^j = \frac{x_i^j - min_j}{max_j - min_j} \tag{4.2}$$

An dieser Stelle ist zu sagen, dass diese Art von Transformation nicht die zu bevorzugende ist, wenn min_j und max_j Ausreißer sind (Aggarwal, 2015; Soria et al., 2009). Allerdings erfordern bestimmte ML-Verfahren, dass der Werte Bereich nicht negativ ist. In diesem Fall, eignet sich die beschriebene Normalisierung gut, wohingegen die Standardisierung hierbei versagen würde (da negative Werte vorkommen).

Die oben beschriebenen Methoden zur Standardisierung, im speziellen die der Normalisierung, sind nicht die einzigen, die es gibt. Sie demonstrieren lediglich die Situationen, Eigenschaften und Gebrauch von derartigen Methoden, um eine Modellierung überhaupt erst möglich zu machen bzw. die Performance des Modells zu verbessern.

4.3.3 Nicht-lineare Transformationen

In der Praxis entspricht die Verteilung von Werten innerhalb von Features in den seltensten Fällen einem bestimmten Verteilungsmuster. Insbesonders bei parametrischen Modellen, wie es z.B. bei der linearen Regression der Fall ist, gibt es die Voraussetzung für die Gültigkeit des Modells, dass die Residuen[1] normalverteilt sind (James, Witten, Hastie & Tibshirani, 2013). Demanch kann auch die Verteilung innerhalb von Features eine entscheidende Rolle für den Fit bzw. die Performance des Modells spielen. Werden die Features auf deren Verteilung untersucht, so kann es vorkommen, dass die Verteilung stark nach links oder nach rechts verschoben ist (Skew), d.h. überproportional viele Werte entweder am unteren Ende der Skala oder am oberen Ende sich häufen. Zudem können Ausreißer das ML-Model negativ beeinflussen (Aggarwal, 2015; James et al., 2013; Witten & Frank, 2005). Bei Ausreißern handelt es dabei um jene Werte, welche sich nicht in die erwartete Messreihe einfügen bzw. ein durch Quantile-definiertes Streumaß überschreiten (James et al., 2013). Um Verteilungen und Ausreißer zu analysieren, eignen sich grafische Darstellungen bzw. Diagramme sehr gut. So zum Beispiel gibt ein *Histogramm* des zu analysierenden Features Aufschluss über die Verteilung der Werte. Eine weitere und häufig eingesetzte Darstellung ist der *Quantil-Quantil-Plot (Q-Q-Plot)*. Bei dieser Darstellung werden die standardisierten Quantile (Aufgetragen auf de Abszisse) einer statistischen Variable (Aufgetragen auf der Ordinate) gegenübergestellt (siehe Abb. 4.2). Die detaillierte Ausführung dieser und weiterer explorativen Darstellungstechniken würde ein ganzes Kapitel füllen und daher den Umfang dieser Arbeit sprengen. An dieser Stelle wird auf weiterführende Literatur von James et al. (2013) und Brink et al. (2017) verwiesen.

Trotz der umfangreiche Thematik der statistischen Analyse und Transformation von Daten, sollten an dieser Stelle ein paar wenige und häufig eingesetzte nicht-lineare Transformations-Methoden erläutert werden. Zu diesen zählen unter anderem $logX_i^j$, $X_i^{j^2}$ oder $\sqrt{X}$ (James et al., 2013). Dabei wird jeder i-te Wert des Features j der soeben

[1] Auch als Fehlerterm bezeichnet und stellt die quadratischen Abstände der Regressionsgeraden für jeden „echten" i-ten Datenpunkt dar

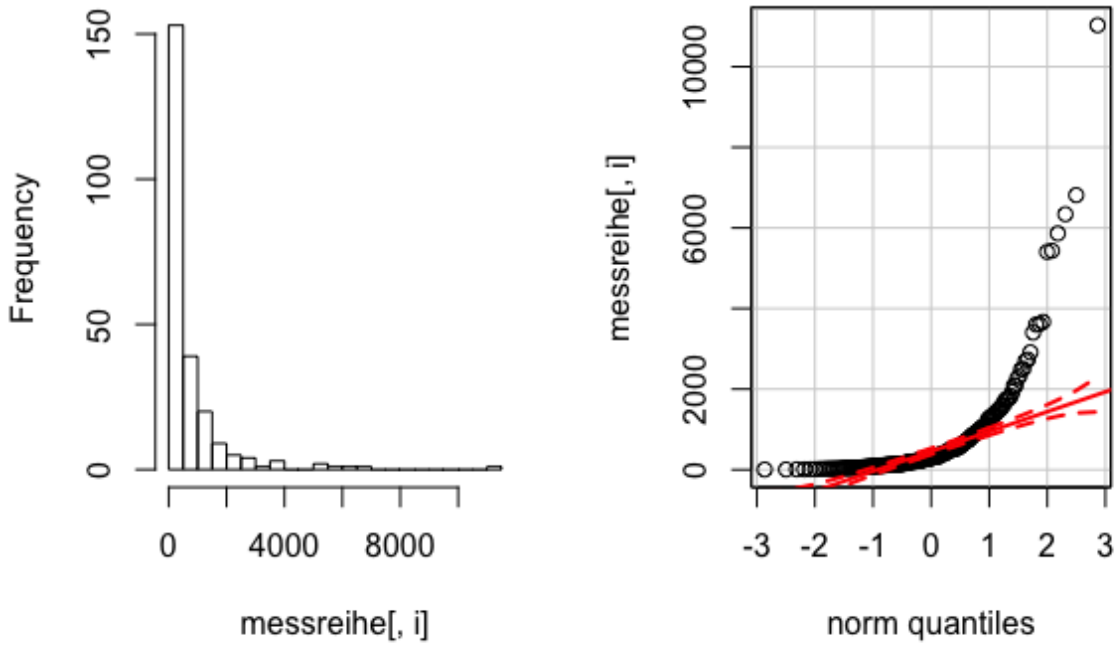


Abbildung 4.2: Histogramm (links) und ein Q-Q-Plot (rechts) einer Messreihe

angeführten mathematischen Operationen unterzogen. Zur Veranschaulichung dieser Transformationen wurde von einer Messreihe aus einem anonymen Datenset ein Histogramm und ein Q-Q-Plot erzeugt (siehe Abb. 4.2). Es ist dabei klar zu erkennen, dass die Verteilung der Messwerte sehr linkslastig ausfällt. Auch beim Q-Q-Plot ist zu erkennen, dass die Werte nach oben hin die erwartete Fortsetzung der Messreihe verlassen. Würde man nun dieses Feature in ein ML-Modell aufnehmen, so könnte dies zu einer Verschlechterung führen bzw. negativen Einfluss auf die Vorhersagegenauigkeit haben (James et al., 2013). Nun ist es z.B. möglich, diese Messreihe auf eine logarithmische Skala zu mappen, d.h. eine logarithmische Transformation ($logX$) der Werte vorzunehmen. Das Ergebnis nach der Transformation sieht, wie in Abb. 4.3 ersichtlich, schon sehr viel besser aus im Sinne einer Ähnlichkeit zur gaußschen Normalverteilung.

Dieses kleine und rudimentäre Beispiel zeigt , dass nicht-lineare Transformationen dazu verwendet werden können, um die Verteilung von Werten symmetrischer zu machen und den negativen Einfluss von Ausreißer in einem ML-Modell dadurch zu reduzieren.

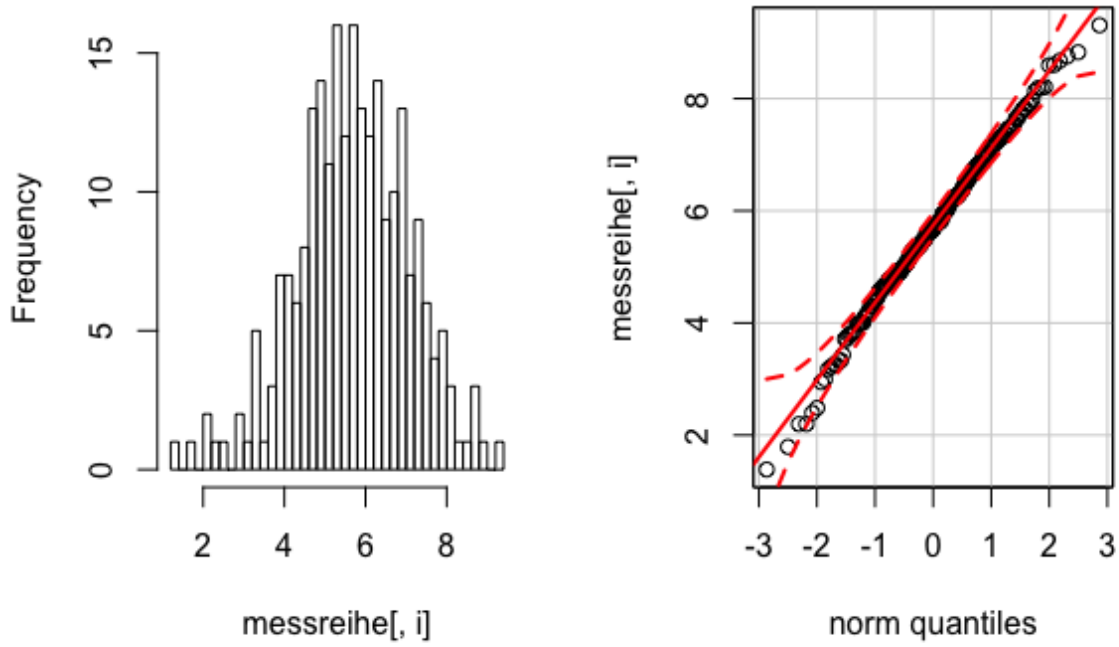


Abbildung 4.3: Histogramm (links) und ein Q-Q-Plot (rechts) der logarithmierten Messreihe

4.3.4 Transformation von Datentypen

In einer praktischen ML-Problemstellung werden Datenquellen verwendet, welche nur in den seltensten Fällen vom selben Datentyp sind. So auch im Titanic-Datenset, wo z.B. „Fare" vom Typ Float ist, hingegen „Sex" vom Typ Character ist. Dabei ist für das ML-Verfahren nicht der Datentyp als solcher zu bestimmen bzw. ein bestimmter Typ erforderlich. Vielmehr geht es um die Darstellung der „levels of measurement" (Witten & Frank, 2005, p.50), auf die nachfolgend näher eingegangen wird. Dabei ist zwischen *nominalem*- bzw. *ordinalem* Datenniveau, sowie hinsichtlich *Interval*- und *Ratio*-Niveau zu unterscheiden. Letztere zwei werden nicht näher erläutert, da diese in ML-Modellen nicht von großer Bedeutung sind (Witten & Frank, 2005). Ein nominales Datenniveau bedeutet, dass die Werte eine symbolische Unterscheidung zu den anderen Werten des Features aufweisen – ohne ordnende Struktur. Betrachtet man das Beispiel-Datenset, so ist exemplarisch das Feature „Sexäls Vertreter eines nominalen Datenniveau zu nennen. Zwischen „male" und „female" gibt es keine Ordnungsstruktur, d.h. es wäre falsch zu sagen $female > male$ oder umgekehrt. Trotzdem können die beiden Werte aufgrund deren Symbolik und Semantik klar voneinander unterschieden werden. Diese Art von Datenstruktur werden oft auch als *kategorische* Werte bezeichnet (Witten & Frank, 2005; Aggarwal, 2015; Brink et al., 2017).

Betrachtet man hingegen das Feature „Pclass", welche die gebuchte Klasse darstellt, in der ein Passagier damals reiste, so unterliegen die Werte offensichtlich einer Ordnungsstruktur, denn Klasse 1 ist bekanntlich „besser" als Klasse 2 und Klasse 3 ist „schlechter" als Klasse 2. Es handelt sich dabei um ein sogenanntes ordinales Daten-Niveau. Wie eben ausgeführt, lassen sich die Werte ordnen bzw. in eine Rangfolge bringen, jedoch sind dabei keine Abstände zwischen den Werten zu messen, was den Unterschied zu *metrischen* bzw. *kontinuierlichen* Datenniveaus ausmacht (Witten & Frank, 2005). Dieser Unterschied zwischen nominalem und ordinalem Niveau ist nicht immer eindeutig und so gilt es, stets eine Bestimmung diesbezüglich vorzunehmen. Aber warum? Was bedeutet dies nun im Kontext der Datenvorbereitung für die Modellerstellung?

Manche ML-Algorithmen können mit nominalen – also kategorischen Daten – umgehen, d.h. diese ohne manuelle Vorbereitung verarbeiten. Für ein Großteil der Algorithmen jedoch ist es notwendig, die kategorischen Features in für diese interpretierbare nummerischen Werte zu transformieren. Dieser Schritt, von kategorischen Features auf numerischen Features zu mappen, wird als *Binarisierung* bezeichnet (Aggarwal, 2015). Exemplarisch würde dass für das Titanic-Beispiel bedeuten, dass das Feature „Sex" für jede Ausprägung des Labels eine zusätzliche Spalte hinzugefügt wird, in der das zutreffende Label je Instanz mit „1" und die verbleibenden Spalten mit „0" versehen werden.

Sex	Sex:male	Sex:female
male	1	0
female	0	1
female	0	1
female	0	1
male	1	0

Tabelle 4.2: Binarisierung von kategorischen Features

Man nennt diese Art der Kodierung auch „One-hot encoding" (Müller & Guido, 2016). Diese Art der Kodierung hat den Vorteil, dass dem Algorithmus keine Ordnungsstruktur oder Rang vorgegeben wird. Es gibt auch die Möglichkeit, die Ausprägungen des Features einfach durch zu nummerieren, d.h. im Kontext des Beispiels „male" = 1, „female" = 2. Dies kann jedoch dazu führen, dass der Algorithmus diese als Rangfolge interpretiert, was real nicht der Fall ist.

Eine durchaus geläufige Situation bei der Datenvorbereitung ist es auch, numerische Wert zu kategorisieren. Dieser Vorgang wird auch als *Diskretisierung* bezeichnet (Aggarwal, 2015). Dies ist insbesondere der Fall, wenn bestimmte Intervalle auf einer nummerischen Skala von Interesse sind und diese in Klassen zusammengefasst werden können. Für das Titanic-Datenset könnten beispielsweise Intervalle innerhalb von „Age" gebildet werden und diese dann mit einer Bezeichnung oder einem nummerischen Wert versehen werden. Dies könnte folgendermaßen aussehen:

[0,14] = „Kind", [15,18] = „Jugend", [19,90] = „Erwachsen"

Des weiteren kann diese Methode auch zur Diskretisierung von Zeitreihen in Zeitsequenzen eingesetzt werden (Aggarwal, 2015). Bei dieser Art von Transformation ist zu bedenken, dass durch die Diskretisierung in der Regel Informationen „verloren" gehen (ebd.). Daher sollte eine Diskretisierung gut bedacht und begründet werden.

Eine weitere und wichtige Transformation ist an dieser Stelle noch ausständig: Die Transformation von Text in numerische Werte. Diese Transformation ist schlussendlich sehr bedeutend im Rahmen dieser Arbeit, da ja aus textuell beschriebenen User Stories Vorhersagen hinsichtlich Aufwand gemacht werden soll. Aus diesem Grund wird in Kapitel 5 darauf ausführlich eingegangen.

4.4 Modellierung von Klassifizierungs-Verfahren

In diesem Abschnitt wird eine genauere und spezifischere Betrachtung der Modellierung vorgenommen. Dies beginnt mit der Auswahl eines Klassifizierungsverfahren und dessen spezielle Datenvorbereitung. Die grundsätzliche Beschreibung des Funktionsprinzips ausgewählter Klassifizierungs-Algorithmen und dessen Eigenschaften steht im Zentrum dieses Abschnittes.

4.4.1 Die Auswahl von ML-Verfahren

Für die Auswahl eines ML-Verfahrens sind bestimmte Faktoren oder Eigenschaften zu berücksichtigen, von denen die wichtigsten nachfolgend umrissen werden.

4.4.1.1 Vorhersagegenauigkeit vs. Interpretierbarkeit

Wie bereits in Abschnitt 3.1.2 erwähnt, existieren unterschiedliche Lernansätze, mit denen verschiedene Problemstellungen adressiert werden. Die Auswahl von Lern-Algorithmen stellt dabei eine zentrale und entscheidende Aufgabe zu Beginn der Modellierung dar. Die schlechte Nachricht diesbezüglich lautet: Es gibt grundsätzlich kein „bestes" Modell für eine bestimmte Art von Problemstellung. Daher ist es nicht möglich, vorab eine Auswahl, wie in einem Katalog, zu treffen, wenn es sich beispielsweise um ein Klassifizierungs-Problem handelt. Jeder Klassifizierungs-Algorithmus hat zwar seine inherenten und spezifischen Eigenschaften, was jedoch nicht dazu veranlassen sollte, eine generelle Empfehlung bezogen auf eine konkrete Problemstellung abzugeben – und schon gar nicht zur Performance des Verfahrens (Raschka, 2015;

Brink et al., 2017). Nun die gute Nachricht: Es existiert eine Vielzahl an Algorithmen, welche sich durch diverse Frameworks für die unterschiedlichsten Programmiersprachen mit relativ geringem Aufwand implementieren und testen lassen. Daher ist es für die Auswahl essentiell, nicht nur ein einziges Verfahren zu betrachten, sondern mehrere Modelle miteinander zu vergleichen (Raschka, 2015). Dabei ist es selbstverständlich zulässig, Annahmen über die Klassifizierungs-Aufgabe und dessen Auswirkungen auf die Problemstellung zu treffen (ebd.). So zum Beispiel kann für die Problemstellung aus dem Titanic-Beispiel angenommen werden, dass aufgrund der binären Ausprägung des Class Labels und durch die die Datenstruktur der Features, eine *Logistische Regression* und ein *Decision Tree* auf die Daten anzuwenden und zu vergleichen sind. Das Hintergrundwissen über diese Annahme ist selbstverständlich an dieser Stelle der Arbeit noch nicht vorhanden. Hierbei sollte lediglich die Herangehensweise verdeutlicht werden.

Eine weitere wichtige Fragestellung drängt sich auf, wenn es um das *Ziel* oder den *Zweck* des ML-Einsatzes geht. Diesbezüglich existieren zwei Stoßrichtungen, die mit ML verfolgt werden können. Zum einen kann die oberste Priorität in der Zielsetzung sein, eine möglichst genau Vorhersage für neue Daten zu erzeugen. Dabei ist der Anspruch, möglichst genau die Realität wiederzugeben und mit den Vorhersagen Entscheidungen herbeizuführen oder die Implementierung in einem automatisierten Prozess vorzunehmen. Das Ziel, möglichst genauen Vorhersagen zu machen, ist der häufigste Anwendungsfall von ML in der Praxis (Brink et al., 2017). Als Beispiele dafür sind typischerweise Forecastings in den verschiedensten Bereichen, bis hin zur Entschlüsselung von handschriftlichen Ziffern oder Sprachaufzeichnungen zu nennen (ebd.). Betrachtet man das Titanic-Beispiel, so könnte ein definiertes Ziel sein, für einen neuen Passagier, anhand der ausgewählten Features, eine möglichst genaue Vorhersage über das hypothetische Überleben oder Ableben des Passagiers zu machen. Was in diesem Zusammenhang mit „genau" gemeint ist, wird zu einem späteren Zeitpunkt noch erläutert. Mit der Eigenschaft eines Modells einer hohen Vorhersagegenauigkeit, geht jedoch ein maßgeblicher Nachteil einher: Es kann meist nicht festgestellt werden, *wie* der Algorithmus zu seiner Vorhersage kommt. Mit anderen Worten: Der Algorithmus ist wie eine Blackbox zu sehen, welcher zwar in der Lage ist Vorhersagen mit hoher Genauigkeit zu generieren, jedoch die innere Struktur der Entscheidung nicht ersichtlich ist. Ein bedeutender Nachteil würde sich daher ergeben, wenn die Zielsetzung eines ML-Vorhabens in der *Inferenz*[2] liegt. Dabei steht nicht die Maximierung der Vorhersagegenauigkeit für neue Daten im Vordergrund, sondern vielmehr die *Interpretierbarkeit* der Zusammenhänge und Auswirkungen der Features auf die abhängige Variable (= Class Label). Im Sinne der Inferenz ist also ein Modell anzustreben, welches die Beziehungen, also die innere Struktur des Modells, offenbart um damit Fragen wie zum Beispiel

- Welche Features korrelieren am stärksten mit der abhängigen Variable?

[2]Schlussfolgerung eines Vorgangs, bei dem neue Erkenntnisse aus bereits bekannten Fakten durch die Anwendung von Regeln abgeleitet werden

- Stehen diese Beziehungen positiv oder negativ zueinander?
- Besteht eine lineare oder nicht-lineare Beziehung?

zu beantworten (James et al., 2013; Brink et al., 2017). Um es wieder im Kontext des Titanic-Beispiels zu betrachten, so könnte die Fragestellung beispielhaft lauten: Welches der Features ist am maßgebendsten am Überleben oder am Tod eines Passagiers beteiligt? Noch konkreter: Gibt es einen Zusammenhang zwischen dem Alter der Passagiere und deren Überlebensrate?

Bedauerlicherweise ist hier gleich vorwegzunehmen, dass beides, also eine hohe Vorhersagegenauigkeit und gleichzeitig ein bestmöglich durch den Menschen interpretierbares Modell zur Hand zu haben, nicht möglich ist (James et al., 2013; Witten & Frank, 2005). Dies bedeutet, dass das Erreichen einer möglichst hohen Vorhersagegenauigkeit immer zu Lasten der Interpretierbarkeit eines Modells geht und vice versa. Der Grund dafür liegt in der Eigenschaft eines Modells, von vornherein eine mathematische Repräsentation in Form einer Funktion anzunehmen oder das Modell sich generisch durch die Datenstruktur selbst erzeugt. Diese Eigenschaft deutet dabei direkt auf die Unterscheidung von *parametrischen-* zu *nicht-parametrische* Verfahren, welche nun kurz erläutert werden.

4.4.1.2 Parametrische Modelle vs. nicht-parametrische Modelle

Parametrische Modelle zeichnen sich dadurch aus, dass eine spezifische Funktion f zugrunde liegt. Auch wenn die einfache lineare Regression kein Verfahren zur Klassifikation ist, eignet sie sich aufgrund der einfachen mathematischen Beschreibung hervorragend zur Demonstration, was unter einem parametrischem Modell zu verstehen ist. Die einfache *lineare Regression* stellt sich wie folgt dar:

$$f(X) = \beta_0 + X_1 * \beta 1 + X_2 * \beta_2 + ... + X_n * \beta_n \tag{4.3}$$

In Gleichung (4.3) wird der Koeffizient β_0 als Intercept bezeichnet, welcher der erwartete Wert von $f(X)$ ist, wenn $X_{1...n} = 0$ ist. Die Koeffizienten $\beta_{1...n}$ bestimmen sozusagen die Steigung einer Geraden, die in den Ergebnisraum gelegt wird, so dass die Summe der quadratischen Abstände zu jedem Datenpunkt minimiert wird (James et al., 2013). Vereinfacht ausgedrückt werden für diese Problemstellung lediglich die Koeffizienten bestimmt und in Gleichung (4.3) eingesetzt. Die Funktion $f(X)$ selbst bleibt jedoch in dieser Repräsentation erhalten, d.h. die Parameter des Algorithmus sind fix gesetzt.

Im Gegensatz dazu stehen die *nicht-parametrischen Modelle*. Bei diesen Modellen wird *keine* strikte mathematische Formulierung vorausgesetzt, sondern die Anzahl der Parameter wächst mit den Features, Class Labels und deren Ausprägungen (Witten

& Frank, 2005). Damit wird keine Annahme über die Form der Funktion getroffen, sondern versucht, eine möglichst gute Approximation von f zu den Datenpunkten zu erreichen (James et al., 2013). Dies hat den Vorteil gegenüber parametrischen Modellen, dass dadurch ein größerer Bereich an funktionalen Formen genutzt werden kann, um dabei eine genauere Anpassung von f zu erreichen (James et al., 2013). Damit kann gesagt werden, dass im direkten Vergleich zu den parametrischen Modellen, die nicht-parametrische Modelle dazu tendieren, am ehesten die Realität widerzuspiegeln. Jedoch sind parametrische Modelle nicht ganz frei von Nachteilen. Da nicht-parametrische Modelle das Problem nicht auf ein paar wenige Parameter von f reduzieren, ist eine entsprechend große Menge an Daten von Nöten (bei weitem mehr als für parametrische Modelle), um eine möglichst genaue Schätzung für f zu bekommen (James et al., 2013). Ein weiterer Nachteil besteht darin, dass durch die möglichst genaue Approximation von f zu den Daten eine Überanpassung stattfinden kann (ebd.). Was vereinfacht gesagt dazu führt, dass die Trainingsdaten zu akribisch vom Modell gelernt werden und dabei die Schätzung für neue Daten, welche nicht genau den Mustern der Vergangenheit folgen (was meist der Fall ist), darunter leiden, sprich zu einer großen Abweichung führen. Diese Eigenschaft wird auch als *Overfitting* oder *Überanpassung* bezeichnet (Brink et al., 2017; Witten & Frank, 2005; James et al., 2013). In Abschnitt 4.5.1 wird darauf noch näher eingegangen und mögliche Strategien erläutert, um dies zu vermeiden.

Für die Auswahl ist abschließend festzuhalten, dass mehrere Verfahren auf Daten angewendet werden sollten, um deren Ergebnisse anhand bestimmter Metriken zu vergleichen. Weiters gilt es, die Problemstellung bzw. die Ziele klar zu definieren bzw. zu definieren, welche Fragen mit dem Einsatz von ML beantwortet werden sollten. Geht es dabei um Fragen zu Beziehungen und Abhängigkeiten von Features zu den Class Labels, so sind grundsätzlich parametrischen Modelle zu bevorzugen. Liegt die oberste Priorität in der Vorhersagegenauigkeit, so sind tendenziell nicht-parametrische Modelle zu präferieren. Wie auch in vielen Bereichen gilt hier: Ausnahmen bestätigen die Regel! Es existieren auch nicht-parametrische Modelle, welche eine „Einsicht" in die Entscheidungsstruktur eines Modells ermöglichen. Ein Beispiel dafür ist der Entscheidungsbaum, der zu einem späteren Zeitpunkt in einer umfassenderen Ausprägung beschrieben wird. Genauso gibt es parametrische Modelle, wie das Multilayer Perceptron, welches vergleichsweise akkurate Vorhersagen produziert und dabei – untypisch für parametrische Modelle – die Entscheidungsstruktur nicht nachvollziehbar ist. Nichtsdestotrotz gilt, abgesehen von ein paar wenigen Ausnahmen, der Trade-off zwischen Vorhersagegenauigkeit und Interpretierbarkeit bei der Auswahl eines Modells.

4.4.2 Funktionsprinzip von Klassifizierungs-Algorithmen

In Abschnitt 4.1 wurde bereits die grundsätzliche Problemstellung von Klassifizierungs-Problemen und deren Elemente erläutert. In diesem Abschnitt geht es nun darum,

das Funktionsprinzip von Klassifizierungs-Algorithmen zu erläutern, um das Verständnis und Wissen für die Implementierung in der Fallstudie zu erlangen. Die Herausforderung besteht jedoch darin, dass es kaum einen gemeinsamen Nenner unter den existierenden Algorithmen gibt, sodass wir gezwungen sind, eine Auswahl zu treffen. Es wurde ja zuvor in diesem Kapitel erläutert, dass es kein pauschal „bestes" Verfahren gibt und dadurch mehrere Modelle *getestet* und *evaluiert* werden sollten, um danach eine endgültige Auswahl treffen zu können. Es werden folglich drei Klassifizierungs-Algorithmen (Kurzform: Klassifizierer) ausgewählt und in den Grundzügen beschrieben. Nachstehend gilt es, die Auswahl kurz zu begründen. Dazu muss kurz vorgegriffen werden und ein paar wenige, aber wichtige Eigenschaften der Aufgabenstellung in der Fallstudie und jener der Algorithmen erläutert werden.

Die Aufgabenstellung lautet grob, dass anhand der textuellen Beschreibung der Anforderung, eine Schätzung des Aufwandes (in Story Points) durch das ML-Verfahren abgegeben werden soll. Das Class Label (Story Points) ist demnach ein numerischer Wert mit mehr als zwei Ausprägungen, nämlich „1", „2", „3", „5" und „8". Damit ergibt sich bereits ein Ausschluss für jene Algorithmen, die nur mit binären Class Labels arbeiten können. Es können also nur Algorithmen in die engere Auswahl kommen, welche *Multiclass Classification* unterstützen. Weiters gilt es, jene Klassifizierer zu identifizieren, die fehlende Werte und Sparse-Vektoren grundsätzlich verarbeiten können. Bedeutung und Einsatz von Sparse-Vektoren wird in Kapitel 5 noch genauer erläutert. Diese soeben genannten Einschränkungen stellen, wenn man so will, die K.o.-Kriterien dar.

Der nächste Schritt besteht darin, aus den zuvor sondierten Klassifizierer eine grobe Auswahl nach den bereits bekannten Eigenschaften und Erfahrungswerten zu treffen. Folglich wurde eine Auswahl getroffen, welche sich zusammengefasst durch folgende Eigenschaften der Verfahren begründen lässt:

1. Naïve Bayes

    a) Relativ einfaches Funktionsprinzip, welches in der Praxis gute Ergebnisse liefert (Witten & Frank, 2005)

    b) Besonders gut geeignet für Text-Klassifikations-Probleme (Aggarwal, 2015; Brink et al., 2017)

    c) Kann mit fehlenden Werten umgehen bzw. diese ohne Vorverarbeitung verwenden (Mitchell, 1997)

    d) Wird in Literatur und Praxis als Baseline-Estimator verstanden (Brink et al., 2017)

2. Random Forest

a) Relativ gute Vorhersage-Performance bei nicht-linearen Problemstellungen (Mitchell, 1997)

b) Kaum bis keine Datenvorverarbeitung notwendig (Brink et al., 2017)

c) Robust gegen Ausreißer und fehlende Werte (Brink et al., 2017; Mitchell, 1997)

d) Liefert interpretierbare Ergebnisse mit Möglichkeit zur Inferenz (Witten & Frank, 2005; Mitchell, 1997)

3. Neuronales Netz

a) Hohe Vorhersage-Performance bei komplexen Problem-Strukturen (Aggarwal, 2015; Brink et al., 2017)

b) Automatische Feature Extraktion (Patterson & Gibson, 2017)

c) Ready-to-use Implementierungen verfügbar

Nachfolgend werden die Funktionsprinzipien und Eigenschaften der soeben erwähnten Klassifizierer näher erläutert.

4.4.2.1 Multinomial Naïve Bayes

Naïve Bayes ist ein Verfahren, das nach dem berühmten englischen Philosophen Thomas Bayes aus dem 18. Jahrhundert benannt wurde. Die Teilbezeichnung „Bayes" verweist auf das *Bayes'sche Theorem* und „naïve" bezieht sich dabei auf die naïve Annahme, dass die Features, bezogen auf die Klasse, unabhängig voneinander sind (Mitchell, 1997; Witten & Frank, 2005). Dies bedeutet zugleich, dass die Annahme zugrunde liegt, dass alle Features den gleichen Einfluss auf die Klasse haben. Diese Annahmen sind auf den ersten Blick eine sehr simplifizierte Darstellung der Wirklichkeit. Doch trotz dieses naïven Ansatzes liefert dieses Verfahren erstaunlich gute Leistungen in der Praxis (Witten & Frank, 2005). Da dieses Verfahren auch nativ Sparse-Vektoren verarbeiten kann, eignet es sich gut für die Text-Klassifikation (Brink et al., 2017). Durch den naïven Ansatz, lässt sich dieser auch in ein paar wenigen Zeilen herleiten:

Wieder zurück zu dem Titanic-Beispiel: Hierbei sollte „Survived" durch die Wahrscheinlichkeit $p(C_s|x)$ mit s=0 (nicht überlebt) oder s=1 (überlebt) basierend auf dem Feature x der jeweiligen Instanz bestimmt werden. Dabei stellt sich das Bayes'sche Theorem folgendermaßen dar:

$$p(C_s|x) \sim p(C_s) * p(x|C_s) \tag{4.4}$$

$p(x|C_s)$ ist dabei als die *Verbundwahrscheinlichkeit* des Features x zu verstehen, wenn die Instanz aus der Klasse C_s stammt. Da dieses Modell, wie bereits erwähnt, auf der Annahme gründet, dass jedes Feature unabhängig voneinander ist, so können die Wahrscheinlichkeiten jedes Features, abhängig von der Klasse, einfach multipliziert werden:

$$\begin{aligned} p(C_s|x) &\sim p(C_s) * p(x_1|C_s) * p(x_2|C_s) * p(x_3|C_s) * ... * p(x_n|C_s) \\ &= p(C_s) \prod_{i}^{n} p(x_i|C_s) \end{aligned} \tag{4.5}$$

Da $p(C_s)$ die *Randverteilung* darstellt – also die übergeordnete Verteilung von Überlebenden zu Nicht-Überlebenden, welche man leicht aus den Daten erkennen kann, so gilt es nur $p(x_i|C_s)$ zu bestimmen. Dieser Ausdruck kann als die Wahrscheinlichkeit eines spezifischen Features für eine spezifisches Class Label verstanden werden (Brink et al., 2017). Auf das Beispiel umgelegt: Es ist anzunehmen, dass für Passagiere aus der 1. Klasse eine höhere Wahrscheinlichkeit besteht, als jene der 3. Klasse. Löst man die Berechnung der bedingten Wahrscheinlichkeit anhand einem Feature und einer Klassenausprägung (=Class Label) auf, so sieht dies folgendermaßen aus:

$$\begin{aligned} &\frac{Anzahl\ von\ x[PClass = 1]\ wenn\ C[Survived = 1]}{Anzahl\ C[Suvived = 1]} > \\ &\frac{Anzahl\ von\ x[PClass = 3]\ wenn\ C[Survived = 1]}{Anzahl\ C[Suvived = 1]} \end{aligned} \tag{4.6}$$

Wie aus Gleichung (4.6) ersichtlich, basiert die Berechnung im Grunde genommen lediglich auf Vorkommnisse bestimmter Ausprägungen, durch welche ein Wahrscheinlichkeitswert, durch die Multiplikation der Teilwahrscheinlichkeiten, je Class Label bestimmt wird. Vereinfacht gesprochen, können damit neue Daten anhand der gelernten Wahrscheinlichkeits-Struktur einem Class Label zugeordnet werden. Dieses Verfahren eignet sich im speziellen daher sehr gut für die Text-Klassifizierung, bei welcher in der einfachsten Form lediglich Wortvorkommnisse über alle Dokumente gezählt werden. Aus diesem Anwendungsfall lässt sich auch der Begriff „Multinomial" erklären, da die Verteilung von Wortvorkommnissen über mehrere Dokumente einer Klasse einer *Multinomial*- oder auch *Polynomial*-Verteilung nahe kommt (Witten & Frank, 2005; Brink et al., 2017). In diesem Fall wird $p(x_i|C_s)$ zu:

$$p(x_i|C_s) \sim \prod_{i} p_{s_i}^{x_i} \tag{4.7}$$

Dabei repräsentiert s_i den Anteil, wie oft ein Wort in einem der Class Labels vorkommt. x_i ist dabei das Feature einer Instanz, für die eine Vorhersage getroffen werden soll.

Es wurden nun die Grundlagen zu Naïve Bayes erläutert, wobei an dieser Stelle zu sagen ist, dass es in der Praxis unterschiedliche Implementierungen gibt, welche

die Beschreibung von oben noch erweitern oder diese in einer abgeänderten Form übernommen haben. Insgesamt lässt sich sagen, dass dieser Klassifizierer in der Praxis oft eingesetzt wird (z.B. Spam-Filtering) und einen guten Baseline-Estimator zum Vergleich mit anderen Verfahren darstellt.

4.4.2.2 Random Forest

Hinter dem Begriff *Random Forest* verbirgt sich ein Verfahren, bei dem mehrere *Entscheidungsbäume* (*Decision Trees*) dazu benutzt werden, um Vorhersagen für Class Labels zu erzeugen. Der Random Forest gehört zu den *Ensamble*-Methoden, welche als *Meta-Lernverfahren* zu verstehen sind. Meta-Lernverfahren haben zum Ziel, durch eine aufbauende oder ergänzende Anwendung eines odere mehrerer Verfahren bessere Vorhersagen zu erzielen, als mit nur einem einzigen Verfahren (Aggarwal, 2015). Eines der Ensamble-Methoden nennt sich *Bagging*. Der Random Forest ist dabei als eine Sonderform des Baggings zu sehen (Witten & Frank, 2005). Unter Bagging versteht man, eine durch das Zufallsprinzip getroffenen Auswahl von Trainings-Datensätzen zu verwenden, um mehrere Instanzen einer Vorhersage zu generieren (ebd.). Dabei werden die Vorhersagen von einzelnen Modellen bzgl. der Wahrscheinlichkeit, dass eine Instanz sich in einer bestimmten Klasse befindet, gemittelt (bei Regression) oder durch eine *Mehrheitsentscheidung* (bei Klassifikation) angenommen. Im konkreten Fall von Random Forests bedeutet dies, dass mittels Bagging durch zufällig ausgewählte Datensätze mehrere Entscheidungsbäume erzeugt werden. Es entstehen dabei sogenannte *Random Trees*, welche eben auf Basis eines Subsets der verfügbaren Features erzeugt werden (Witten & Frank, 2005). Versuche haben gezeigt, dass sich die Anwendung des Zufallsprinzip positiv auf das Ergebnis auswirkt (Mitchell, 1997; Witten & Frank, 2005). Wie schon erwähnt, liegt dem Random Forest das Konzept des Entscheidungsbaums zugrunde, dessen Funktionsprinzip in groben Zügen nun kurz erläutert wird.

Entscheidungsbäume sind sehr weit verbreitet, wenn es um Klassifizierungs-Probleme geht. Ein Entscheidungsbaum wird darauf trainiert, Erfahrungswerte aus den Trainingsdaten zu strukturieren. Bei diesem Vorgang entsteht der Entscheidungsbaum, welcher nach dem Aufbau bestimmten Regeln folgt und die Klassifikation durch eine Sortierung von der *Wurzel* (*Root*) bis zu den *Blattknoten* (*Leaf Nodes*) ermöglicht (Aggarwal, 2015; Mitchell, 1997). Um dies zu veranschaulichen wird ein oft zitiertes, einfaches Beispiel von Mitchell (1997) herangezogen. Es sollte klassifiziert werden, ob an einem bestimmten Tag, abhängig von den Wetterverhältnissen, ein Tennisspiel stattfindet oder nicht (In diesem Fall „Yes" oder „No"). Dazu wurde ein Entscheidungsbaum erzeugt (siehe Abbildung 4.4), bei dem jeder Knoten ein Test von einem bestimmten Attribut darstellt und jede Kante, absteigend vom Knoten, einen dazu korrespondierenden Wert darstellt. Dieser Vorgang wird für alle Sub-Bäume, von der Wurzel ausgehend, wiederholt. Die Reihenfolge der Features (von Root zu Leaf) kommt so zustande, indem in absteigender Reihenfolge anhand der Wichtigkeit sortiert wird. Die Wichtigkeit der Features wird anhand des *Informationsgewinns*

je Knoten berechnet (zuvor generisch als „Test" bezeichnet) (Witten & Frank, 2005). Für die Berechnung des Informationsgewinns sind beispielhaft die *Entropie* oder der *Gini-Koeffizient* zu nennen, welche in (Witten & Frank, 2005) und (Aggarwal, 2015) ausführlich beschrieben sind.

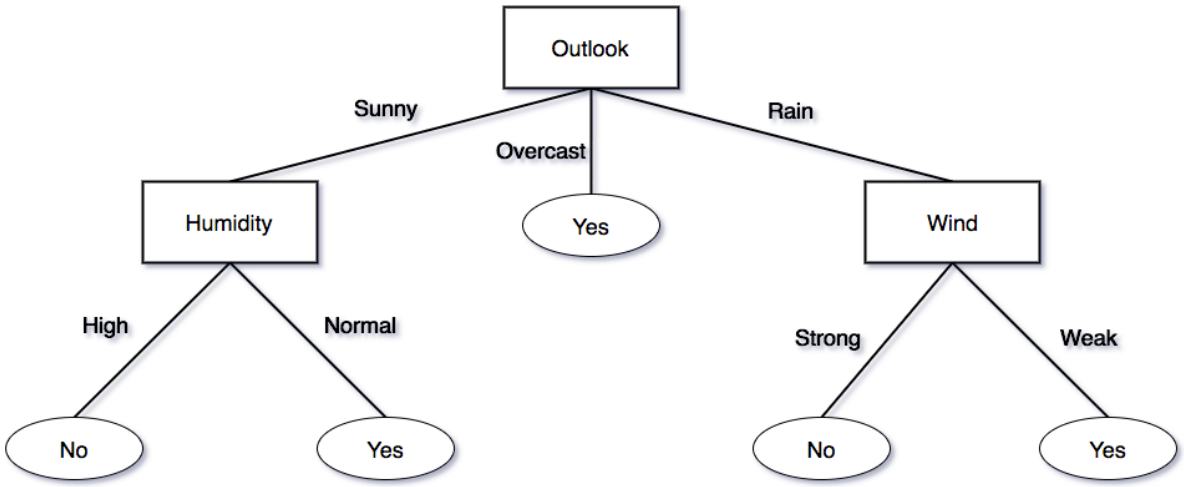


Abbildung 4.4: Entscheidungsbaum zur Klassifizierung von Tennisspielen an einem bestimmten Tag (Mitchell, 1997)

Nun zum Beispiel: Angenommen, es würden folgende Informationen für einen Tag vorliegen und es gilt vorherzusagen, ob das Tennisspiel stattfinden wird oder nicht:

$$(Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong)$$

Es würde nun der linke Ast abwärts sortiert werden und damit diese Instanz als negativ klassifiziert werden – also keine Spiel an diesem Tag (Tennisspiele = „No"). Dabei bildet der Entscheidungsbaum in seiner Gesamtbetrachtung eine ODER-Verknüpfung von UND-Verknüpfungen der Features. Dies bedeutet, dass von der Wurzel bis zum Blatt, jeder Ast eine ODER-Verknüpfung zum andern Ast darstellt und jeder Ast in sich eine UND-Verknüpfung bildet. Übertragen auf das Beispiel bedeutet dies konkret, dass mit folgenden Konstellationen der Features ein Spiel zustande kommen würde:

$$\begin{aligned}&(Outlook = Sunny \ \wedge \ Humidity = Normal)\\ &\vee (Outlook = Overcast)\\ &\vee (Outlook = Rain \ \wedge \ Wind = Weak)\end{aligned}$$

Die Tiefe des Baumes, wie auch dessen Komplexität, ist vor dem Aufbau nicht bestimmbar, sondern bestimmt sich erst im Zuge des Trainings mit Daten (Brink et al., 2017). Daher zählt dieses Verfahren zu den nicht-parametrischen Verfahren. Ein großer Nachteil der Entscheidungsbäume besteht darin, dass die oberste Ebene des Baumes einen überproportional großen Einfluss auf die Antwort hat (Mitchell, 1997). Wenn also neue Daten nicht exakt dieselbe Verteilung der Trainingsdaten aufweisen, so ist es sehr wahrscheinlich, dass das Generalisierungvermögen eines erzeugten Baumes darunter leidet (Brink et al., 2017). Und genau hier kommt der Random Forest ins Spiel.

Dieser mindert das Risiko einer zu starken Überanpassung an die Trainingsdaten durch das oben beschriebene Bagging.

Random Forests sind für deren Eigenschaft bekannt, eine relativ hohe Genauigkeit für nicht-lineare Problemstellungen zu erzeugen (Mitchell, 1997; Brink et al., 2017). Darüber hinaus sind Random Forests unempfänglich für nicht relevante Features und sehr robust gegen Rauschen in den Daten (fehlende Werte, fehlerhafte Labels, etc.) (Brink et al., 2017). Obwohl dieses Verfahren klar zu den nicht-parametrischen Verfahren zu zählen ist, kann das Modell im Sinne der Inferenz „eingesehen" werden, da viele Implementierungen es ermöglichen, die Feature-Wichtigkeit zu ermitteln (Pedregosa et al., 2011). All diese Eigenschaften machen den Random Forest zu einem sehr leistungsfähigen Klassifizierer, der wenig bis keine Datenvorverarbeitung benötigt. Ein möglicher Nachteil bei der Anwendung eines Random Forests könnte sich in der Laufzeit der Trainingsphasen mit großen Datenmengen ergeben (Mitchell, 1997). Diesem Nachteil kann jedoch mit *Parallelisierung* bzw. *Verteilung* des Rechenvorgangs auf mehrere Rechenkerne entgegengewirkt werden (Brink et al., 2017).

4.4.2.3 Multilayer Perceptron

Das *Künstliche Neuronale Netz (KNN)* ist die Basis aller *Deep Learning* Verfahren und damit auch der Ursprung einer Ausprägungsvariante: dem *Multilayer Perceptron (MLP)*. Deshalb wird kurz auf die Grundlagen neuronaler Netze eingegangen und im Anschluss das Funktionsprinzip eines Multilayer Perceptrons beschrieben. Es ist dabei gleich vorwegzunehmen, dass aufgrund der Komplexität und Ausprägungsvielfalt auf Details bzw. auf die vollständig mathematische Beschreibung verzichtet wird. Es kann im Rahmen dieser Arbeit lediglich ein Überblick bzw. ein Auszug über die Funktionsweise gegeben werden.

Die Erforschung von KNN hatte ihre Anfänge bereits im Jahr 1940, als Warren McCulloch and Walter Pitt als erstes beschrieben, wie ein KNN funktionieren könnte (Raschka, 2015). Diese Forschung wurde bis in die späten 50er Jahre fortgesetzt. Danach schwand das Interesse, als keine brauchbaren Methode zur Verfügung stand, um ein derartiges Netzwerk zu trainieren (Raschka, 2015). In den 80er Jahren brachten Wissenschaftler den *Backpropagation-Algorithmus* hervor, welcher theoretisch das Trainieren von neuronalen Netzen möglich bzw. effizienter machte und dabei sehr gute Ergebnisse lieferte (Mitchell, 1997; Aggarwal, 2015). In den vergangenen Dekaden – und bis heute anhaltend – gelang es Forschern, immer wieder große Durchbrüche in der Weiterentwicklung von neuronalen Netzen zu erzielen, welche wir heute dem Deep Learning zuschreiben.

Das Konzept des KNN gründet auf dem Funktionsmodell des menschlichen Gehirns, in welchem eine Informationsverarbeitung durch die Verbindung vieler Nervenzellen, den sogenannten *Neuronen*, ermöglicht werden (Patterson & Gibson, 2017). Ein KNN

besteht in seiner Grundform aus mehreren Neuronen, welche *Eingabewerte* empfangen, danach über eine *Aktivierungsfunktion* an das nächste Neuron weiterleiten bzw. ein Neuron einen *Ausgabewert* erzeugt. Letzteres ist die Konsequenz, dass es nur ein Neuron bzw. einen einzigen *Output Node* gibt. Diese Variante bildet eine Einheit, welche auch als *einfaches Perceptron* bezeichnet wird (siehe Abbildung 4.5(a)). Das einfache Perceptron lässt sich in dieser Form nur auf linear-seperierbare Daten anwenden bzw. ausschließlich für lineare Problemstellungen einsetzen (Witten & Frank, 2005). Um das Konzept des KNN auf nicht-lineare Problemstellungen zu erweitern, werden mehrere neuronale Schichten, sogenannte *Hidden Layers* in einem Netzwerk hierarchisch angeordnet, so dass jedes Neuron mit jedem nachfolgenden Neuron verbunden ist (siehe Abbildung 4.5(b)). Als Ergebnis erhält man das MLP, welches durch die gerichtete Informationsweitergabe (ohne Rückkopplung innerhalb des Netzwerkes) zu den *Feedforward*-Netzwerke zu zählen ist (Patterson & Gibson, 2017; Witten & Frank, 2005). An dieser Stelle ist zu sagen, dass noch weitere Ausprägungen von KNN existieren, welche jedoch in dieser Arbeit nicht weiter betrachtet werden.

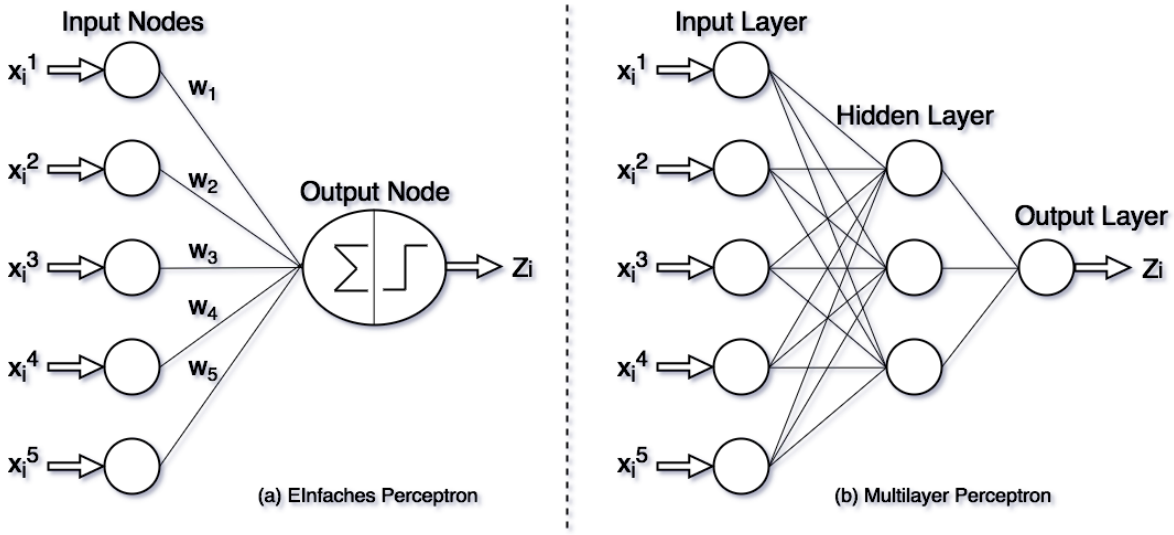


Abbildung 4.5: Der Aufbau eines (a) einfachen Perceptrons und (b) eines Multilayer Perceptrons (Aggarwal, 2015)

Nun zum Funktionsprinzip des MLP: Wie aus Abbildung 4.5(b) zu entnehmen ist, liegen in diesem Beispiel die *Eingabewerte* x_i^j an der ersten Schicht (*Input Layer*) an. Die Eingabewerte werden an den Verbindungskanten durch die Multiplikation mit deren *Gewichtung* w_i an alle Neuronen der ersten Schicht geleitet. Dort passiert, in jedem Neuron, die *Aktivierung*, für welche unterschiedliche mathematische Funktionen existieren. Eine häufig verwendete Aktivierungsfunktion is die *Sigmoid-Funktion*, die sich wie folgt beschreiben lässt:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.8}$$

Vor Aktivierung werden die Eingabewerte aller Vorgänger-Neuronen bzw. der Eingabewerte selbst mit der Gewichtung multipliziert und aufsummiert und der Akti-

vierungsfunktion übergeben. Damit wird in diesem speziellen Fall der gewichtete Eingabewerte der Gleichung (4.8) zugeführt. Zudem wird ein *Bias* β hinzugefügt, welcher als Eingabewert zur Normierung (Patterson & Gibson, 2017) und zur Vermeidung einer bestimmten symmetrischen Ausgangslage der Daten (Brink et al., 2017) dient. Dieser nimmt meist den Wert 1 an, wobei die Gewichtung den invertierten, ursprünglichen Schwellwert annimmt (Witten & Frank, 2005) – also -1 . Die Berechnung von Eingabewerten $x_i^1 ... x_i^d$ in einem Neuron nimmt daher folgende Gestalt an:

$$z_i = \sum_{j=1}^{d} w_j * \frac{1}{1 + e^{-x_i^j}} + \beta \tag{4.9}$$

Der berechnete Ausgabewert z_i ist nun nicht mehr das vorhergesagte finale Class Label im Intervall $[-1, +1]$, sondern der Input für den nächsten Hidden Layer. In einem einfachen Perceptron ist das tatsächliche Label – also die *Ground Truth* – bekannt und stellt daher ein Optimierungsproblem dar, welches durch die Berechnung des kleinsten quadratischen Abstandes gelöst werden kann (Aggarwal, 2015). Die Gewichtung wird dabei durch ein *Gradientenabstiegsverfahren*[3] angepasst. Nun sind in einem MLP keine Class Label bekannt nach der Berechnung des Input Layers, d.h. die Hidden Layer halten keine Informationen bereit, um direkt dort eine Optimierung anzustellen. Im Grunde genommen ist der Output z_i in einem Multilayer Perceptron also das Ergebnis einer mehrschichtigen Berechnung. Nun drängt sich die Frage auf, wie dann die Anpassung der Gewichte an den Kanten funktioniert?

Das Ziel des MLP ist es nun, die Gewichtung an jeder Kante so zu bestimmen bzw. zu trainieren, dass für jede Instanz der Ausgangswert z_i so nahe wie möglich an dem echten Label z_* ist. Diese Problemstellung kann durch *Backpropagation* gelöst werden. Dabei handelt es sich um einen Algorithmus, welcher für jede Instanz im Trainingsprozess durchlaufen wird. Der Algorithmus kann grundsätzlich in zwei Phasen gegliedert werden:

Die erste Phase, die *Feedforeward-Phase*, besteht simpel gesagt aus der Durchschleusung eines Wertes anhand der Gewichtungen an den Kanten (zu Beginn zufällig initialisiert) über alle Schichten hinweg bis hin zur Ausgabe – wie eben zuvor beschrieben. Der Ausgabewert z_i wird mit dem tatsächlichen Class Label verglichen um zu sehen, ob die Klassifizierung erfolgreich war, sprich eine Übereinstimmung der Vorhersage mit dem tatsächlichen Class Label vorhanden ist. Die auftretenden Differenz $y - y*$ wird auch als *Fehlerterm* oder *Error Rate* bezeichnet, die es im gesamten Prozess zu minimieren gilt (Patterson & Gibson, 2017).

Nun kommt die *Backpropagation-Phase*, in welcher auf Basis der abgeleiteten Error Rate eine rückwärts-gerichtete Fehlerminimierung gestartet wird und dabei die Gewichtung für jede eingehende Kante in einem Neuron rückgerechnet werden kann (Witten

[3]Ein numerisches Annäherungsverfahren mit dem Ziel, den kleinstmöglichen Gradienten einer Funktion zu finden, d.h. die Steigung der Tangente so klein wie möglich ist.

& Frank, 2005; Patterson & Gibson, 2017). Dabei bedient man sich wieder dem Gradientenabstiegsverfahren. Die Geschwindigkeit, mit welcher sich dieses Verfahren an ein lokales Minima in der Error Rate-Funktion annähert, kann durch die *Lernrate* (*Learning Rate*) zuvor parametrisiert werden (Patterson & Gibson, 2017). Der Algorithmus wird so lange ausgeführt, bis die Error Rate konvergiert bzw. ein vorher festgelegter Schwellwert unterschritten wird. Damit wird nun erreicht, dass die Gewichte an jeder Kante im Netwerk so angepasst werden, dass eine möglichst gute Annäherung der Vorhersagen an die Class Labels der Trainingsdaten als Ergebnis vorliegt. Wie bereits erwähnt, können wir eine mathematische Herleitung bzw. detailiertere Beschreibung zum Backpropagation-Algorithmus an dieser Stelle nicht im Detail ausführen und verweisen daher auf Patterson und Gibson (2017) und (Witten & Frank, 2005) zur Vertiefung des Wissens.

Zusammenfassend lässt sich sagen, dass KNN, insbesondere das MLP, ein Verfahren darstellt, welches sich äußerst gut für stark nicht-lineare Problemstellungen eignet (Brink et al., 2017). Das Verfahren ist in der Lage, komplexe Datenzusammenhänge zu klassifizieren, so wie es z.B. bei der Erkennung von Mustern in Bildern oder in Text der Fall ist. Das MLP zählt durch dessen mathematischer Beschreibung zu den parametrischen Verfahren, welches – analog zum Random Forest – zu den Ausnahmen hinsichtlich der Eigenschaften der jeweiligen Modell-Klasse (parametrisch, nicht-parametrisch) zu zählen ist. Dies ist im Fall des MLP darin begründet, da die üblicherweise gute Inferenz-Eigenschaft von parametrischen Modellen hier nicht gegeben ist. Durch die algorithmisch-iterative Bestimmung der Gewichte an den Kanten bleibt eine mögliche Erklärung, wie es zu dieser Bestimmung gekommen ist, im verborgenen. Man spricht in diesem Zusammenhang auch von einem *Blackbox-Verfahren* (Brink et al., 2017). Ein weiterer Nachteil von KNN liegt in der rechenintensiven Trainingsphase, d.h. der Zeitaufwand, den ein Modell benötigt, trainiert zu werden (Witten & Frank, 2005). Dies ist insbesondere beim *Hyper-Parameter-Tuning*[4] der Fall, wo das Training mit unterschiedlichen Parametern ausgeführt wird und dadurch immens lange Trainingszeiten entstehen können (Raschka, 2015).

Es wurden nun drei ML-Verfahren etwas näher beschrieben, welche als Grundlage für die Implementierung der Verfahren in der Fallstudie dient.

4.5 Performance Evaluation von Klassifizierungs-Verfahren

Das primäre Ziel eines ML-Verfahrens ist es, so genaue Vorhersagen wie nur möglich zu machen. Dies bedeutet konkret, dass das trainierte ML-Modell so trainiert sein

[4]Beschreibt den Prozess des Findens der „besten" Parameter eines Algorithmus, die vor der Trainingsphase einzustellen sind

sollte, dass es für neue Daten so verallgemeinernd Vorhersagen erzeugt, dass diese in Produktion eine hohe Qualität aufweisen. Was eine hohen Qualität in diesem Zusammenhang bedeutet, ist Gegenstand dieses Abschnitts. Zudem werden Methoden vorgestellt, um zufällige, als auch systematische Verzerrungs-Effekte in der Ermittlung der Modell Performance zu mindern.

4.5.1 Überanpassung und Modell Optimismus

Was im Zentrum der letzten Abschnitte stand, nämlich die Modellierung eines Verfahrens, welches eine möglichst genaue Vorhersage für neue Daten generieren soll, unterliegt einem kleinen Dilemma. Dieses besteht darin, dass die Intensität und Genauigkeit des Modells in der Trainingsphase, den Trainingsdaten derart angepasst werden, dass es eben möglichst genaue Vorhersagen dafür erzeugt. Mit anderen Worten: Das Modell studiert die Trainingsdaten so genau ein, mit der Annahme, dass neue Daten eine ähnliche Verteilung bzw. Muster aufweisen und diese dementsprechend genau vorhersagen kann. Dabei ist nicht schwer zu erkennen, dass dieser Annahme ein fundamentaler Irrglaube zugrunde liegt, denn neue Daten, bei denen Klassen vorhergesagt werden sollen, sind in den seltensten Fällen von exakt gleicher Verteilung. Dieses Phänomen wird als *Overfitting* oder *Überanpassung* bezeichnet (Witten & Frank, 2005). Nun besteht ein möglicher Ansatz, diese Überanpassung zu vermeiden, indem z.B. eine Vereinfachung des Modells herbeigeführt wird (James et al., 2013). Insbesondere nicht-parametrische Modelle neigen eher zu einem Überanpassungsverhalten auf Trainingsdaten, als parametrische Modelle (Brink et al., 2017). Es wäre also ein Lösungsansatz, ein nicht-parametrisches Modell durch ein parametrisches Verfahren zu ersetzen. Dies ist jedoch etwas zu kurz gegriffen, denn die relativ starre Repräsentation von parametrischen Modellen durch beispielsweise lineare Modelle führt bei komplexen Klassifikations-Problemem zu einer hohen Fehlerrate bei neuen Daten (wie bereits in diesem Kapitel beschrieben wurde). Diese Eigenschaft wird in diesem Zusammenhang auch als *Bias* bezeichnet (James et al., 2013). Bias bezieht sich dabei auf die Fehlerrate, die auf der fehlenden Flexibilität eines parametrischen Modells gründet, wenn dieses auf ein komplexes, reales Problem angewandt wird (ebd.). Im Gegensatz dazu steht die *Varianz* eines Modells. Hierbei ist die Varianz als die Veränderung der Fehlerrate zu verstehen, die aus dem Training eines Modells mit unterschiedlichen Traingsdaten resultiert (James et al., 2013). Da Traininigsdaten dazu verwendet werden, das ML-Verfahren zu trainieren, resultieren mit unterschiedlichen Traingsdaten, unterschiedliche Fehlerraten bezogen auf die Vorhersage der Klasse. Idealerweise sollte jedoch die Fehlerrate in Bezug auf die Class Lables nicht allzu sehr zwischen den unterschiedlichen Trainingsdaten schwanken (James et al., 2013). Wenn eine Methode also eine hohe Varianz aufweist, dann bewirken kleine Veränderungen in den Trainingsdaten große Veränderungen in der Vorhersage des Modells. Grundsätzlich weisen hier nicht-parametrische Modelle ein höheres Varianz-Verhalten auf, als weniger flexiblere Verfahren (Witten & Frank, 2005; Brink et al., 2017).

Die soeben genannten Umstände machen eine Abwägung notwendig, um diesbezüglich eine Entscheidung für die Modellierung und Evaluierung zu treffen. Die Evaluierung der Modell Performance kann bei einem Overfitting der Trainingsdaten zu einem irreführenden Optimismus hinsichtlich der Vorhersage für neue Daten führen (Brink et al., 2017). Versucht man demnach eine Überanpassung völlig zu vermeiden, so geht dies zugunsten eines größeren Bias auf Trainingsdaten, d.h. eine zu grobe Verallgemeinerung der Trainingsdaten findet statt, welche in einer mäßigen Vorhersagegenauigkeit mündet. Umgekehrt führt eine zu starke Anpassung zum selben Resultat, jedoch aufgrund der zuvor beschriebenen Ursache. Eine mögliche und häufig eingesetzte Strategie, eine Überanpassung zu vermeiden und trotzdem nicht allzu sehr zu verallgemeinern, ist das Durchführen einer *Cross-Validation*, die nun direkt im Anschluss erläutert wird. Zudem gilt es auch, geeignete Performance Metriken für eine vorliegende Problemstellung zu beleuchten.

4.5.2 Cross-Validation

Wie soeben im vorangegangenen Abschnitt beschrieben, kann es bei einer *Performance Evaluierung* unter ausschließlicher Verwendung der Trainingsdaten zu einem verzerrten Ergebnis führen, was die „echte" Performance von Vorhersagen eines Modells für neue Daten betrifft. Nun stellt sich die Frage, wie eine Performance Evaluierung durchgeführt werden soll, wenn nur ein beschränkte Anzahl an Trainingsdaten vorliegen? Nun, die einfachste Variante wäre, das gesamte Trainings-Datenset in ein Trainings-Subset und in ein Test-Subset aufzuteilen, wie bereits oben erwähnt wurde. Diese Art von Evaluierung wird auch als *Hold-Out Methode* bezeichnet (Witten & Frank, 2005). Dabei wird der Split meist so vorgenommen, dass mit rund zwei Drittel das Modell trainiert wird und mit einem Drittel getestet wird, d.h. die Class Labels für dieses Test-Subset vorhergesagt werden und anschließend auf Übereinstimmung verglichen wird. Jetzt kann es jedoch durchaus vorkommen, dass in einem Split des Trainings-Subsets, als auch im Test-Subset die Class Labels jeweils unter-, bzw. überrepräsentiert sind. Um dem entgegenzuwirken, sollte jede Klasse in beiden Subsets im selben Verhältnis repräsentiert sein, wie es für das gesamte Datenset der Fall ist. Dies kann durch eine randomisierte Auswahl unter der Berücksichtigung der Klassenverteilung im gesamten Datenset sichergestellt werden. Diese Prozedur wird auch als *Stratification* bezeichnet und stellt einen wesentlichen Faktor in der Evaluierung dar (Aggarwal, 2015; Witten & Frank, 2005). Obwohl diese Methode zwar grundsätzlich das Risiko einer Unterrepräsentation mindert, so ist sie doch nur bedingt wirksam, da sie ja nur die anteilsmäßige Verteilung der Class Labels innerhalb der Subsets sicherstellt. Es kann daher vorkommen, dass auch die relevanten Attribute – also die Features – ungleichmäßig in den Trainings- und Test-Subsets repräsentiert sind.

Eine fortgeschrittenere und praxis-erprobte Methode stellt die *k-Fold Cross-Validation* dar (Brink et al., 2017; James et al., 2013). Dabei wird das gesamte Datenset in k folds aufgeteilt. „Folds" sind dabei als Datensubsets zu verstehen, die aus dem gesamten

Datenset gesampelt werden. Auch hier kann eine *Stratified*-Prozedur angewandt werden, welche dann als *Stratified k-Fold Cross Validation* bezeichnet wird (Raschka, 2015). k steht dabei für die Anzahl der Teile – oder eben folds – die erzeugt werden. Wenn also ein Datenset aus 100 Instanzen besteht, dann werden bei $k = 10$, die 100 Instanzen in zehn Folds aufgeteilt, welche jedes für sich die Verteilung der Klassen aus dem gesamten Datenbestand wiederspiegeln (nur bei einer Stratified-Prozedur). In diesem konkreten Fall wird das Modell mit neun Folds trainiert und mit einem wird die Vorhersage durchgeführt (=Testset). Dies wird nun für jeden Fold $k-1$ mal wiederholt, sprich, jeder Fold wird einmal als Test-Fold verwendet. Danach wird über alle zehn Vorhersagen der Mittelwert gebildet. Die Anzahl für $k = 10$ hat sich in der Praxis durchgesetzt, bzw. besteht auch ein theoretischer Beweis dafür, dass diese Anzahl an Folds am repräsentativsten die Fehlerrate in Bezug auf neue Daten widerspiegelt (Witten & Frank, 2005). Es ist jedoch genauso legitim $k = 5$ oder $k = 20$ zu verwenden, wenn es dafür gute Gründe gibt (ebd.).

4.5.3 Performance Metriken

Im Abschnitt zuvor fielen einige Male die Begriffe *Genauigkeit* oder auch *Performance* im Zusammenhang mit der Vorhersage eines Modells. Nun gilt es, diese Begrifflichkeiten aufzulösen und Metriken zur Beurteilung der Modell Performance näher zu beschreiben. Performance kann im Kontext der Evaluation vielschichtige Ausprägungen aufweisen. Demnach gibt es nicht nur eine einzige Möglichkeit der Messung, sondern eine Vielzahl an Metriken. Da sich der Inhalt und Ziel dieser Arbeit auf Klassifizierungs-Probleme beschränkt, werden folglich auch nur jene Metriken betrachtet, die für eine Evaluation von Klassifizierungs-Aufgaben relevant sind.

Nach dem Training eines Modells mit Hilfe der zuvor beschriebenen Cross-Validation, gilt es nun, das Modell anhand dessen Performance zu bewerten. Eines der einfachsten Performance-Measures ist die *Accuracy*. Diese wird einfach dadurch bestimmt, indem die Anzahl der korrekten Klassifizierungen der Gesamtanzahl an Vorhersagen gegenüber gestellt wird. Die Accuracy ist also nichts weiter, als der Anteil der korrekt klassifizierten Instanzen im Test-Set. Um dies zu veranschaulichen, wird erneut das Titanic-Beispiel herangezogen. Dazu wurde ein Naïve Bayes mit den Trainingsdaten aus *Titanic: Machine Learning from Disaster* (2012) trainiert und gegen das ebenfalls dort bereitgestellte Testdaten-Set validiert. Auf die genaue Beschreibung dieser Modellierung wird verzichtet, da es lediglich zur Demonstration eines bestimmten Teilaspektes in der Evaluation dient. Die korrekte, aber auch nicht korrekte Klassifikation lässt sich grafisch sehr anschaulich durch die sogenannte *Confusion Matrix* in Tabelle 4.3 darstellen.

In Tabelle 4.3 ist nun ersichtlich, wie viele Instanzen der insgesamt 267 Instanzen aus dem Test-Set richtig und falsch klassifiziert wurden. Diese Darstellung ist so zu verstehen, dass 55 Überlebende $(Survived^* \wedge Survived)$ und 130 $(Died^* \wedge Died)$ richtig

		Predicted Outcome		
		Survived	Died	Total
Actual Labels	Survived*	55	49	104
	Died*	33	130	163
	Total	88	179	267

Tabelle 4.3: Darstellung des Klassifizierungs-Ergebnisses mit Hilfe einer Confusion Matrix

klassifiziert wurden, d.h. eine Übereinstimmung der vorhergesagten Labels mit der Ground Truth vorhanden ist. Bei den 55 die hinsichtlich der Überlebenden korrekt klassifiziert wurden, spricht man auch von einer *True Positive (TP) Klassifizierung*. Bei den 130, welche auch korrekt als Nicht-Überlebende klassifiziert wurden, spricht man von der *True Negative (TN) Klassifizierung*. Nun gibt es jedoch auch noch Missklassifikationen, die sich in dem Beispiel zeigen. $Died^* \wedge Survived$ sagt aus, dass 33 Instanzen (oder Personen) als überlebend vorhergesagt wurde, jedoch in Realität nicht überlebten. Dieser Klassifizierungs-Fehler wird auch als *False Positive (FP)* bezeichnet. Im Gegensatz dazu wurden bei $Survived^* \wedge Died$ ebenso 49 Personen der Tod vorhergesagt, obwohl diese eigentlich überlebten. Dieser Fehler wird als *False Negative (FN)* bezeichnet. Die zuvor erwähnte Accuracy lässt sich dabei bereits auf dieser Basis wie folgt errechnen:

$$Overall\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{55 + 130}{55 + 130 + 33 + 49} \approx 0.69 \tag{4.10}$$

Damit ergibt sich eine Accuracy von rund 69% für das gesamte Modell. Nun ist dies jedoch nicht die ganze Wahrheit, denn FP und FN haben auch eine Bedeutung im Sinne von Gütekriterien für die Robustheit und Stabilität eines Klassifizierers (Witten & Frank, 2005). Betrachtet man das Verhältnis der Vorhersage der Überlebenden des Modells zu jenen, die tatsächlich alle überlebten, so entschärft dies doch etwas die Accuracy des gesamten Modells. Denn aus insgesamt 104 Überlebenden konnten lediglich 55 als solche identifiziert werden. Dieses Performance-Gütekriterium wird auch *Recall* (Kubat, 2005) oder *Sensitivity* (Witten & Frank, 2005) bezeichnet. Damit ist die Eigenschaft eines Modells gemeint, ein positives Beispiel auch als solches zu identifizieren. Der Wert dafür lässt sich wie folgt berechnen:

$$Recall = \frac{TP}{TP + FN} = \frac{55}{55 + 49} \approx 0.53 \tag{4.11}$$

Damit sagt dieses Modell lediglich rund 53% der positiven Werte voraus (= Überlebende). Anders ausgedrückt: Dieses Modell ist in dem konkreten Beispiel kaum besser als ein Zufallsprozess in der Vorhersage der Überlebenden. Ersetzt man TP und FN in Gleichung (4.11) durch die Werte von FP und TN, so stellt dies die *Spezifität* (Recall für Klasse $Died^*$) dar (Kubat, 2005) und ist mit rund 80% deutlich besser. Daraus lässt sich grundsätzlich schlussfolgern, dass dieses Modell, mit den verwendeten Trainingsdaten, besser in der Vorhersage von Nicht-Überlebenden geeignet ist, als für die Vorhersage der Überlebenden. Dennoch liegt noch ein Schatten auf den der-

zeitigen Erkenntnissen zur Performance des Modells. Denn wenn es darum geht zu evaluieren, mit welcher Genauigkeit das Modell die jeweils Überlebenden zu den Nicht-Überlebeneden vorhersagt, so ist der Recall nicht ausreichend. Dieser gibt nur an, wieviel von den Überlebenden vorhergesagt werden. Ist jedoch die Genauigkeit von großer Bedeutung, sprich, das Verhältnis der TP zu allen als überlebend vorhergesagten Personen, so spricht man von *Precision* (Patterson & Gibson, 2017; Raschka, 2015) die sich wie folgt berechnet:

$$Precision = \frac{TP}{TP + FP} = \frac{55}{55 + 33} \approx 0.62 \tag{4.12}$$

Das Ergebnis aus Gleichung (4.12) ist demnach so zu interpretieren, dass der Klassifizierer in rund 62% der Fälle mit der Vorhersage von Überlebenden richtig lag. Dieselbe Berechnungslogik lässt sich auch auf die Klasse der Nicht-Überlebenden anwenden. Was aber bedeutet dies nun für die praktische Anwendung? Nun, es gibt Problemstellungen bzw. Anwendungsdomänen, da spielt die Precision eine größere Rolle als der Recall. So z.B. bei Recommender in E-Commerce Systemen, welche auf Basis der Logik „Kunden-die-X-gekauft-haben-kaufen-auch-Y" funktionieren, ist die Eigenschaft der Precision eine essentielle (Kubat, 2005). Hier möchte der Kunde schlussendlich das richtige Produkt empfohlen bekommen, denn ansonsten schwindet das Interesse des Kunden an der Anzeige und die Anzeige wird zukünftig möglicherweise ignoriert. Der Recall hat dabei kaum eine Bedeutung, denn es spielt nur eine untergeordnete Rolle, wenn nur ein kleiner Anteil an angebotenen Artikeln, die der Kunde wirklich bevorzugt, vorhergesagt wird. Im Gegensatz zu diesem Anwendungsfall steht die Diagnose in der Medizin. Im Fall der Diagnose lautet die Aufgabe, die richtige Erkennung von tatsächlich vorhandenen Krankheiten (Kubat, 2005). Eine Situation, bei der ein Patient, welcher unter X leidet und auch als X erkannt wurde, ist der anzustrebende Fall (TP). Ein zu vermeidendes Szenario wäre wohl, wenn der Patient an X leidet, aber nicht als solcher erkannt wird (FN). Einfach ausgedrückt: Man möchte hier möglichst viele „Leidende" oder „Kranke" identifizieren. Daher ist für diesen Anwendungsfall ein hoher Recall anzustreben (Kubat, 2005).

Oft stehen sich Precison und Recall mit gegensätzliche Ergebnissen gegenüber. Dabei können sich Situationen ergeben, in denen nicht immer sofort ersichtlich ist, welche Metrik herangezogen werden sollte bzw. welche eine größere Bedeutung für eine bestimmte Problemstellung hat. Um diese Abwägung zu quantifizieren, wurde der $F1$-Score eingeführt. Dieser bildet das harmonische Mittel von Recall und Precision in einem einzigen Wert ab (Patterson & Gibson, 2017; James et al., 2013). Der $F1$-Score lässt sich wie folgt berechnen:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.13}$$

Der $F1$-Score stellt damit eine Balance zwischen den zwei relevanten Metriken dar und bietet eine bessere Quantifizierung der Modell Performance (Aggarwal, 2015). Die fehlende Monotonie zwischen Recall und Precision lässt sich trotz des $F1$-Score nicht

vollständig abbilden, so dass eine Betrachtung aller soeben beschriebenen Metriken notwendig ist, um bezogen auf die konkrete Problemstellung, den Metriken unterschiedliche Gewichtungen im Rahmen der Evaluation beizumessen. Eine zusätzliche Möglichkeit im Sinne einer intuitiveren Bewertung der Modell Performance, ist die Verwendung von Diagrammen, wie beispielsweise die *Receiver Operating Characteristic (ROC)* oder das *Precision-Recall*-Diagramm. Letztere ist, wie der Name schon verrät, die grafische Gegenüberstellung von Precision und Recall. Die ROC-Kurve kommt insbesondere bei binären Klassifizierungs-Problemen (wie das TItanic-Beispiel) zur Anwendung (Witten & Frank, 2005). Durch die Gegenüberstellung der TP-Rate (Ordinate) zur FP-Rate (Abszisse) ist es möglich, sich ein Bild über die tatsächliche Performance des Modells zu machen. Grundsätzlich gilt dabei, dass je größer die Fläche unter der Kurve, desto besser das Modell (Aggarwal, 2015). Da in der Fallstudie im Rahmen der Arbeit mehrere Class Labels als Output erwartet werden (also keine binäre Klassifizierung, sondern Multiclass Output), wird nicht näher auf die ROC-Kurve und andere grafische Evaluationswerkzeuge eingegangen. Für weiterführende Literatur zum Thema wird auf die ausführlichen Werke von James et al. (2013) und Witten und Frank (2005) verwiesen.

Abschließend ist zu sagen, dass die Evaluation der Performance eines Modelles stark von den Anforderungen abhängig ist. Die *Eine* und allumfassende Metrik existiert demnach nicht, sondern es ist vielmehr die Betrachtung und Abwägung von Accuracy, Recall, Precision und $F1$-score.

In diesem Kapitel wurde nun das Grundgerüst für die Modellierung und Evaluierung der Fallstudie erläutert. Im nächsten Kapitel wird die Datenvorverarbeitung (Data Preparation) und Modellierung mit Theorie zum Feature Engineering aus unstrukturiertem Text weiter angereichert.

5 Feature Engineering durch Text Mining und Natural Language Processing

Um eine Klassifizierung von unstrukturierten Datenquellen, wie Texte bzw. ganze Dokumente zu ermöglichen, ist dieses Kapitel den Grundlagen des Feature Engineerings gewidmet. Dazu zählen Basis-Konzepte des Natural Language Processing's als auch jene des Text Mining's. Mit Abschluss dieses Kapitels liegen nun alle theoretischen Konzepte vor, die für die Erstellung und Evaluation eines ML-Modells im Zuge der Fallstudie benötigt werden.

5.1 Grundlagen des Feature Engineering's

Bis hierher wurde eine theoretische Grundlage zur Datenvorverarbeitung, über die Modellierung bis hin zur Evaluierung, bereitet. Dabei wurde stets von *Features* gesprochen, welche in den vorangegangenen Ausführungen in deren einfachster Form dargestellt wurden: Als Attribute im Rohformat. Der Begriff „Rohformat" bezieht sich in diesem Kontext auf einen Datensammlungs-Prozess, bei welchem Attribut-Ausprägungen und Werte aufgenommen wurden (manuell oder automatisiert) und sich auch unverändert in dieser Form im Datenset befinden. Diese Attribute sind zwar teilweise durch Transformationen aufbereitet worden, um diese in einem ML-Verfahren verarbeiten zu können, jedoch wurden dabei keinerlei Berechnungen, Aggregationen, Extraktionen etc. der bzw. aus den Features durchgeführt. Das Feature Engineering hingegen ist als exakt jener Prozess zu verstehen, bei dem eine *Auswahl*, und *Verbesserung* von Features aus Rohdaten mit Hilfe von mathematischen Transformationen, Extraktionsmethoden aus Text und Domänenwissen herbeigeführt ist (Brink et al., 2017). Der Wert des Feature Engineerings wird oftmals äquivalent zur Modellierung selbst gesehen, sprich, Feature Engineering ist mindestens genau so essentiell zur Erreichung von performante und qualitativ hochwertige Vorhersagen, wie die Auswahl und Implementierung des Lern-Algorithmus selbst (Combs, 2016). Doch warum ist das so?

Zum einen besteht grundsätzlich die Möglichkeit, den Lern-Algorithmus mit beliebig viel Daten zu speisen. Doch nicht jedes Feature hat eine Bedeutung für die Vorhersage der abhängigen Variablen (=Klasse). Und selbst wenn Features eine Bedeutung

im Kontext des ML haben sollten, so werden all diese bedeutenden Features nicht zwangsläufig benötigt, um eine bestimmte Konfidenz für die Vorhersage des Modells zu erreichen (Ramasubramanian & Sing, 2016). Zum anderen bedeutet eine steigende Anzahl an Features eine exponentielle Steigung der Rechenzeit des Lern-Algorithmus (ebd.). Mit Feature Engineering wird nun genau diesen Umständen entgegengewirkt. Es wird also versucht, Features in einem Datenset gezielt zu selektieren, ohne dabei Informationen der Daten zu verlieren, um als Ergebnis hochperformante Vorhersagen zu erzeugen.

Ein weiterer Grund für den Einsatz von Feature Engineering ist die Erzeugung von repräsentativeren Features hinsichtlich der vorherzusagenden Klasse. In diesem Zusammenhang wird versucht, weitere Features aus den „Rohfeatures" zu erzeugen, die eine größere Bedeutung in Hinblick auf die abhängige Variable aufweisen. Um wieder ein Beispiel aus dem Titanic-Datenset aufzugreifen: Es könnte hierbei z.B. die Information von akademischen Graden bzw. der Titel je Person aus dem Feature „Name" ein starkes Indiz dafür sein, ob jene Passagiere das Unglück überlebten oder nicht. Die Annahme dazu wäre, dass Personen mit akademischen Abschluss tendenziell mehr für ein Ticket bezahlt haben – da diese Personengruppe womöglich ein höheres Einkommen hat – als jene Personen ohne akademischen Grad und daraus eine höhere „Rettungs-Rate" für Akademiker resultiert. Um diese Informationen zu erzeugen, bedient man sich an Extraktionsmethoden, wie in diesem konkreten Fall der *Regular Expressions*.

Neben der Extraktion von Features aus unstrukturiertem Text (wie soeben im Beispiel angedeutet), besteht noch die Möglichkeit, ganze Textblöcke oder gar ganze Dokumente so zu transformieren, dass diese in einem ML-Verfahren verarbeitet werden können. Damit besteht Feature Engineering im Kontext von unstrukturiertem Text in erster Linie aus der *Informations-Extraktion* und *Transformation*. Dazu ist noch anzumerken, dass Feature Engineering keine exakte Disziplin ist – wie auch für das ML insgesamt – sondern vielmehr ein *explorativer*, *kreativer* Prozess ist, bei welchem man sich an statistischen Methoden, als auch dem Natural Language Processing's bedient.

Im nächten Abschnitt werden dazu grundlegende Konzepte des *Natural Language Processing's* und die des *Text Mining's* erläutert.

5.2 Text Mining und Natural Language Processing - Überblick und Begriffsabgrenzung

In den vergangenen Jahren hat die Speicherung von Text, insbesondere von digital generiertem Text signifikant zugenommen. Dies ist hauptsächlich dem Umstand geschuldet, dass die Kommunikation und Speicherung zunehmend, in allzu jedem

Lebens- und Wirtschaftsbereich, digitalisiert wurde. Die Digitalisierung bzw. Speicherung von Text steht uns allen heute im großen Ausmaß zur Verfügung und hält eine nahezu unausschöpfliche Quelle an Information bereit. Studien zufolge sind rund 80% aller Informationen, die heutzutage gespeichert werden, im Text-Format vorzufinden (Ramasubramanian & Sing, 2016). Die große Herausforderung in der Informationsgewinnung aus natürlichsprachigem Text, insbesondere aus unstrukturiertem Text, liegt in der Tatsache, dass Maschinen bzw. Computer (hier als abstrakte Einheit eines System mit EVA-Prinzip zu verstehen) keinen natürlichsprachigen Text als linguistisches Konzept (wie wir Menschen) „erkennen". Maschinen sind grundsätzlich lediglich in der Lage, Text als Symbolsprache zu identifizieren und zu verarbeiten. Daraus nun eine Semantik – also die Bedeutung und Zusammenhänge aus bzw. in natürlichsprachigem Text – zu identifizieren und in weiterer Folge zu verarbeiten, ist bis dato nur dem Menschen vorbehalten. Es wird jedoch versucht, Maschinen linguistische Fähigkeiten anzutrainieren, bzw. Ontologien[1] zu verwenden, mit denen es gelingt, Information aus natürlichsprachigem Text zu extrahieren (Kao & Poteet, 2007; Arellano, Carney & Austin, 2015). Aus diesem Anspruch heraus haben sich zwei grundlegende Forschungs-, bzw. Anwendungsdomänen entwickelt, auf die nun kurz eingegangen wird.

Die Verarbeitung und Informationsgewinnung aus Text im Allgemeinen lässt sich in die Domäne des *Text Mining's*, oder auch *Text Analytics* genannt, einordnen. Laut Kao und Poteet (2007, S. 1) lässt sich die Bedeutung von Text Mining in zwei Sätzen charakterisieren:

„Text Mining is the discovery and extraction of interesting, non-trivial knowledge from free or unstructured text. This encompasses everything from information retrieval (i.e., document or web site retrieval) to text classification and clustering, to (somewhat more recently) entity, relation, and event extraction."

Text Mining ist demnach, ähnlich dem Data Mining, als Zusammenfassung mehrerer Methoden und Verfahren zum Zweck der Informationsgewinnung aus unstrukturiertem Text zu verstehen. Dabei lässt sich erahnen, dass die Anwendungsfälle enorm vielfältig sind und durch die zunehmende Speicherung von Information in Text-Form laufend neue Anwendungsbereiche aufgeschlossen werden. Im Rahmen dieser Arbeit wird das Konzept der *Text Klassifikation* zu einem späteren Zeitpunkt näher beleuchtet.

Im Kontext der Informationsgewinnung aus unstrukturiertem, natürlichsprachlichem Text gibt es in Literatur und Fachkreisen kaum einen Weg, der an Natural Language Processing (NLP) vorbeiführt. Wie der Name bereits andeutet, geht es um die Verarbeitung von natürlichsprachlichem Text – ähnlich der Bedeutung des Text Mining's. Doch worin liegt nun der Unterschied? Auch wenn Text Mining und NLP eng miteinander verwoben sind und inhaltlich ineinandergreifen, so gibt es einen wesentlichen

[1] Bezeichnet eine Struktur zur Gewinnung von Wissen über eine bestimmte Domäne oder Bereich durch die Bereitstellung von Konzepten und Beziehungen dazwischen (Kao & Poteet, 2007)

Unterschied, welcher sich im Anspruch der semantischen Analyse von unstrukturiertem Text auszeichnet. Das Konzept des NLP kann folgendermaßen zusammengefasst werden:

„Natural Language Processing (NLP) is the attempt to extract a fuller meaning representation from free text. This can be put roughly as figuring out who did what to whom, when, where, how and why." (Kao & Poteet, 2007, S. 1)

Mit Hilfe von linguistischen Konzepten, wie z.B. dem Part-of-Speech (POS) tagging und grammatikalischen Strukturen, wird dabei versucht, Informationen über die Bedeutung eines vorliegendes Textes zu extrahieren (Sarkar, 2016). Dabei gibt es die unterschiedlichsten Techniken (POS wurde bereits genannt), welche großteils ein spezielles Vorwissen benötigen und zudem eine entsprechend hohe Komplexität und Umfang aufweisen, dass auf eine Ausführung in dieser Arbeit verzichtet wird. Nichtsdestotrotz wird nachfolgend auf ein paar grundlegende Konzepte des NLP eingegangen, welche für die Durchführung der Fallstudie benötigt werden bzw. sich als große Hilfe erweisen könnten.

Abschließend ist zu sagen, dass Text Mining als umfassenderes Konzept der Informationsgewinnung aus unstrukturiertem Text zu sehen ist. Dazu bedient man sich unter anderem an Techniken und Konzepten des NLP, um ggf. „mehr" Bedeutung aus einem Text zu erschließen. NLP ist dabei als eigenes Forschungsfeld etabliert und als Teil des ML zu verstehen. Text Mining beinhaltet nicht zwangsweise Konzepte des NLP, jedoch stellte sich heraus, dass NLP und dessen anhaltende Fortschritte in der Forschung hier gute Dienste erbringen.

5.3 Grundlagen und Konzepte zur Klassifizierung von Text

In diesem Abschnitt werden Grundlagen des NLP und jene des Text Minings, insbesondere die der Text Klassifikation, erläutert. Der Begriff „Grundlagen" ist dabei so zu verstehen, dass bestimmte Transformationen und Verarbeitungen von Text ohnehin zu machen sind, bevor diese einem Lern-Algorithmus zugeführt werden können. Zudem bedarf es für diese Grundlagen noch kaum Spezialwissen in der natürlichsprachigen Textverarbeitung, so dass diese noch im Umfang der Arbeit abgehandelt werden können. Um die Erläuterung und Demonstration der Konzepte nicht von Grund auf ausführen zu müssen, wird die Programmiersprache Python (Version 3.5.2) mit dem Natural Language Toolkit (NLTK) (Version 3.2.5) verwendet. Dieses Framework ermöglicht es, die folgenden Konzepte mit wenigen Zeilen Code zu demonstrieren. Das NLTK Framework steht frei und kostenlos zum Download zur Verfügung (*NLTK 3.2.5 documentation*, 2017). Zudem existiert eine ausführliche Online-Dokumentation darüber.

5.3.1 Tokenisation von Text

Ein relativ einfaches, aber wichtiges Konzept des NLP ist *Tokenisation* (*Tokenization*). Dabei handelt es sich um die Teilung eines Textes in dessen einzelne Wörter (=Tokens) (Chopra, Joshi & Mathur, 2016). So einfach dies klingen mag, bei genauerer Betrachtung stellt man schnell fest, dass sich diese Aufgabe als nicht so trivial entpuppt wie zuvor angenommen. Grundsätzlich können Tokens aus zusammenhängenden alphabetischen Sequenzen gebildet werden, doch wie sollten numerische Sequenzen aufgesplittet werden? Numerische Werte können dabei Vorzeichen, Dezimaltrennzeichen und möglicherweise auch mathematische Notationen beinhalten. Diese müssen durch Angabe von Regeln geparst werden (Witten & Frank, 2005). Zudem können auch alphanumerische Werte Probleme bereiten, wenn man bedenkt, dass die Wortbegrenzungen durch Leerzeichen, Trennzeichen oder auch durch Satzzeichen erfolgen kann. Insbesondere Punkte stellen hier eine große Herausforderung dar, da sie Bestandteil eines Wortes sein können (Abkürzungen, Titel, etc.), aber eben auch zur Satztrennung verwendet werden. Dieselbe Problematik gilt für Apostrophe und Bindestriche. Daher bedarf es einer expliziten Logik hinsichtlich der soeben beispielhaft genannten Fallunterscheidungen. Die Erklärung dieser Logik würde den Rahmen dieser Arbeit sprengen. Um dennoch das Konzept der Tokenisation in dieser Arbeit zu beschreiben und in weiterer Folge anzuwenden, wird das eingangs erwähnte NLTK verwendet. Diese Framework bringt bereits eine Reihe von korpus– und lexikalischen Ressourcen mit sich. Darunter auch verschiedene Tokenizer-Funktionen. Die Tokenisation als Implementierung stellt sich grundsätzlich so dar, dass ein Text als String eingelesen wird und einer Tokenizer-Funktion übergeben wird. Diese trennt den String in dessen einzelne Wörter auf (=Tokens) und gibt diese als Liste (In Python wird anstatt einem Array grundsätzlich das Konzept der Liste verwendet) zurück. In dieser Liste stellt nun jeder Eintrag ein Wort bzw. Token dar. In diesem Fall ist von einer *Wort-Tokenisierung* die Rede. Das Wort ist dabei die nützlichste Einheit, da ein Satz in seiner Bedeutung am besten durch jedes beinhaltete Wort charakterisiert ist (Witten & Frank, 2005). Um dies zu veranschaulichen ein kleines Beispiel:

```
1 import nltk
2 text = "Willkommen liebe Leser! Ich hoffe ihr findet diese Arbeit
     interessant. Sagt es weiter, wenn sie euch gefaellt."
3 from nltk.tokenize import word_tokenize
4 return print(word_tokenize(text))
5 ['Willkommen', 'liebe', 'Leser', '!', 'Ich', 'hoffe', 'ihr',
     'findet', 'diese', 'Arbeit', 'interessant', '.', 'Sagt', 'es',
     'weiter', ',', 'wenn', 'sie', 'euch', 'gefaellt', '.']
```

Listing 5.1: Wort-Tokenisierung mit Hilfe von Python und NLTK

In Listing 5.1 wurde demonstriert, wie ein Text nach der Tokenisierung aussieht. Dabei ist zu erkennen, dass auch Satzzeichen als Token enthalten sind. Für die Bedeutung bzw. Aussage des Satzes haben diese Satzzeichen grundsätzlich keine Relevanz (Sarkar, 2016). Ebenso sind im Beispiel Tokens wie z.B. „ich", „ihr", usw. vorhanden, die als

sogenannte *Stoppwörter* (*Stopwords*) bezeichnet werden. Was es damit auf sich hat wird nachfolgend erläutert.

5.3.2 Normalisierung von Text

Unter *Normalisierung* im Kontext von NLP versteht man die Vereinheitlichung des Textes, so dass nur noch Informationen aus dem Text vorhanden sind, die eine Bedeutung im Kontext der Problemstellung haben (Sarkar, 2016). Dabei kommen Konzepte zum Einsatz, wie z.B. die Eliminierung von Satzzeichen, das Entfernen von Stoppwörtern, die Konvertierung von Text in Kleinbuchstaben, das Erweitern von Abkürzungen, Kanonisierung von Text, etc. Es besteht eine Vielzahl an Techniken, von denen nachfolgend zwei genauer beschrieben werden.

Wie aus dem Ergebnis aus Listing 5.1 ersichtlich ist, sind auch bedeutungslose Satzzeichen (Punctuations) tokenisiert worden. Somit besteht der erste Schritt im Rahmen der Normalisierung, diese zu entfernen. Für diese Problemstellung bietet die `string` Bibliothek die Funktion `string.punctuation`, welche diese Aufgabe übernimmt. Nach der Ausführung ist die Liste mit Tokens frei von Satzzeichen.

Der nächste Schritt besteht in der Entfernung von Stoppwörtern. Dabei handelt es sich um Wörter, die sich in einem Text sehr oft wiederholen und keinen Beitrag zur Bedeutung des Satzes leisten (Chopra et al., 2016). Stoppwörter können als Implementierung in Listen oder Dictionaries geführt werden und können natürlich in jeder Sprache vorkommen. Eine universell gültige Liste an Stoppwörter gibt es grundsätzlich nicht, jedoch bietet NLTK hier wieder ein Repertoire an Stoppwörter in unterschiedlichen Sprachen, welche natürlich im Kontext einer bestimmten Domäne erweitert oder reduziert werden können(Sarkar, 2016).

Bevor die Bereinigung von Stoppwörter stattfindet, empfielt es sich, den gesamten Text in *lower case* oder in *upper case* zu konvertieren (Sarkar, 2016; Bird, Klein & Loper, 2009; Witten & Frank, 2005). Damit wird verhindert, dass bei einer Zählung von gleichen Wörtern im Korpus, z.B. „Sagt" und „sagt", nicht als zwei unterschiedliche Wörter gezählt werden. Diese Art von Konvertierung ist auch eine Normalisierungsform von Text. In Listing 5.2 werden die soeben beschriebenen Normalisierungs-Schritte aufeinanderfolgend implementiert. Dabei wird als Eingabe der tokeniserte Text `tok` (Zeile 3) aus Listing 5.1 verwendet. Als Ergebnis steht `cleanTokens` als Liste zur Verfügung.

Bis hierher wurden ein paar wenige, aber sehr essentielle Schritte, hin zu einem normalisierten Text, aufgezeigt. Diese Schritte sind deshalb so wichtig, da sie fast in jeder Datenvorverarbeitungs-Phase für ein ML-Verfahren aus einer unstrukturierten Textquelle durchzuführen sind.

```
1  import string
2  from nltk.corpus import stopwords
3  # Erstelle eine Liste nopunc aus tok bei der alle Satzzeichen
       entfernt sind
4  nopunc = [char for char in tok if char not in string.punctuation]
5  # Liste stopwords mit deutschen Stoppwoertern aus NLTK
6  stopwords = stopwords.words('german')
7  # Erstelle eine Liste, in welcher alle Zeichen in Kleinbuchstaben
       konvertiert werden und diese nicht in stopwords enthalten sind
8  cleanTokens = [char.lower() for char in nopunc if char not in
       stopwords]
9  print(cleanTokens)
10 ['willkommen', 'liebe', 'leser', 'ich', 'hoffe', 'findet',
       'arbeit', 'interessant', 'sagt', 'gefaellt']
```

Listing 5.2: Code Snippet zur Bereinigung von Satzzeichen und StoppwÃ¶rter

Weitere Schritte zur Erreichung von normalisiertem Text aus Sicht des Autors ist stark im Kontext der Anwendung bzw. der Domäne zu sehen. Dies bedeutet, eine weiterführende Normalisierung könnte, abhängig von den Anforderungen der Anwendung und der angestrebten Performance des ML-Verfahren, durchaus zielführend sein. Dazu empfehlen einige Autoren (Sarkar, 2016; Chopra et al., 2016; Bird et al., 2009; Brink et al., 2017; Arellano et al., 2015) das Lemmatisieren von Tokens, um einen noch höheren Normalisierungsgrad zu erreichen. Lemmatisation bzw. auch Stemming genannt, wird im folgenden Abschnitt näher beschrieben.

5.3.3 Lemmatisation und Stemming

In vielen Fällen ist es von Vorteil, verschiedene Variationen von einem Wort zu konsolidieren, da die grundsätzliche Bedeutung dabei oft gleich ist. Dabei spricht man von sogenannten Morphemen, welche als kleinste Einheit mit einer Bedeutung zu verstehen ist (Sarkar, 2016). Morpheme existieren in zwei Ausprägungen: dem Wortstamm und Affixe (Chopra et al., 2016). Affixe (Suffix, Präfix, etc.) sind gebundene Morpheme, welche einen Wortstamm lediglich erweitern, d.h. Affixe als solche können nur in Verbindung mit dem Wortstamm existieren. Der Wortstamm ist ein freies Morphem und existiert alleine in dessen Reinform. Demnach wird für das Feature Engineering aus unstrukturiertem Text der Wortstamm als Zielobjekt gesehen, welches die Bedeutung des Wortes trägt.

Der Prozess der Wortrückführung wird als *Stemming* oder *Lemmatisation* bezeichnet. Zur Umsetzung dieses durchaus komplexen Prozesses verhilft uns wieder das NLTK Package, mit dessen High-Level Stemming- und Lemmatisierungs-Funktionen. Ein bekannter und viel verwendeter Stemmer-Algorithmus ist der *Porter Stemmer* (Sarkar,

2016). Versucht man z.B. das englische Wort „happiness" auf seinen Stamm mittels der Funktion `stem()` aus einer Instanz von `nltk.PorterStemmer()`[2] zurückzuführen (Implementierung für Porter Stemmer nur für englischer Sprache in NLTK), so erhält man das Wort „happi". Wie an diesem Beispiel zu sehen ist, wurde dabei das Affix entfernt, jedoch ist „happi" so gesehen kein lexikografisch korrektes Wort, d.h. es ist nicht zwangsläufig so, dass der Wortstamm in Wörterbüchern zu finden ist.

Sucht man hingegen das *Stammwort*(=Lemma), so bezeichnet man diesen Prozess als Lemmatisation. Dabei wird ein Wort, analog zum Stemming, von Affixe befreit, jedoch nur dann, wenn dieses ein lexikografisch korrektes Wort ist, sprich, im Wörterbuch zu finden ist (Bird et al., 2009). Aus diesem Grund ist der Prozess der Lemmatisierung wesentlich langsamer als jener des Stemmings (Sarkar, 2016). Der NLTK Framwork benutzt zur Lemmatisierung das WordNet[3] Kontext und Part-of-Speech zur Rückführung auf das Lemma (Sarkar, 2016). Lemmatisiert man nun mit der Funktion `lemmatize()` aus einer Instanz von `nltk.WordNetLemmatizer()` das Wort „hapiness", so erhält man das Wort „happiness", da durch Lemmatisation kein passendes, lexikografisch korrektes Wort im WordNet Korpus gefunden werden konnte.

Des weiteren ist darauf zu achten, für welche Sprache die Implementierung von Stemmer und Lemmatisierer verwendet werden. Wie bereits erwähnt, bietet das NLTK Package keinen Lemmatisierer für die deutsche Sprache. Für Stemmer existiert der `SnowballStemmer()`, welcher, neben Englisch, noch weitere 13 Sprachen unterstützt (darunter auch Deutsch). Um diesen Abschnitt nun zu vervollständigen, wird das zuletzt verwendete Beispiel im Rahmen dieses Kapitels um das Stemming erweitert:

```
1 from nltk.stem.snowball import SnowballStemmer
2 sbstemmer = SnowballStemmer('german')
3 print([sbstemmer.stem(char) for char in cleanTokens])
4 ['willkomm', 'lieb', 'les', 'ich', 'hoff', 'findet', 'arbeit',
     'interessant', 'sagt', 'gefaellt']
```

Listing 5.3: Code Snippet zur Wortstamm RÃ¼ckfÃ¼hrung mittels SnowballStemmer aus dem NLTK Package

Wie nun aus Listing 5.3 ersichtlich ist, finden sich im Ergebnis (Zeile 4) Tokens, die auf den Wortstamm zurückgeführt wurden (z.B. „lieb") und manche nicht (z.B. „interessant"). Dies liegt daran, dass der Stemming-Algorithmus bei den Wörtern, die gleich geblieben sind, keinen Wortstamm finden konnte und damit das Eingabe-Wort unverändert ausgibt.

Ob man nun Stemming oder Lemmatisation verwendet, ist ganzheitlich abhängig von der Problemstellung bzw. dem Ziel der Informations-Extraktion und der vorliegenden Daten. Stemming ist als solches kein definierter Prozess und neben dem Porter

[2] Implementierung für Porter Stemmer nur für englischer Sprache in NLTK

[3] WordNet wurde an der Princeton University entwickelt und ist ein lexikalisch-semantisches Netz der englischen Sprache (*Wordnet: A lexical database for English*, 2017)

Stemmer- Algorithmus existiert noch eine Vielzahl an anderen Stemming-Algorithmen. Grundsätzlich ist der Porter Stemmer ein geeigneter Ansatz, um Text, z.B. zu indexieren und damit die Suche nach einem Wort zu unterstützen bzw. zu vereinfachen (Bird et al., 2009). Sucht man hingegen nach einer Liste an gültigen Lemmas aus einem Korpus, so empfiehlt sich die Lemmatisation (ebd.). Stemming verhilft grundsätzlich zur Verbessrung des Recall's, jedoch meist etwas zu Lasten der Precision (Aggarwal, 2015). Nichtsdestotrotz ist Stemming ein Konzept zur Erreichung eines besseren Normalisierungsgrades, was oft zu einer höheren Qualität des ML-Modells führt (Brink et al., 2017; Aggarwal, 2015).

In diesem Kapitel wurden die grundlegenden Konzepte des NLP vorgestellt. Die ausgeführten Techniken und Methoden und deren ausgeführte Reihenfolge verstehen sich lediglich als Empfehlung des Autors dieser Arbeit, welche je nach Problemstellung und Anforderung abweichen können. Damit steht nun eine Basis bereit, die zur Transformation der Tokens in eine numerische Repräsentation zur Verarbeitung in einem ML-Verfahren verwendet werden kann. Diese Transformation ist Teil des nächsten Abschnittes.

5.3.4 Das Bag-of-Words Modell

Das Bag-of-Words (BOW) Modell ist eines der einfachsten und dennoch leistungsfähigsten Methoden, wenn es um die Extraktion von Features aus Text geht (Sarkar, 2016). Zudem stellt es auch die Basis für die *Text Klassifizierung* durch ein ML-Verfahren dar. Ebenso baut ein Großteil fortgeschrittener Methoden (z.B. TF-IDF, Similarity Measures, etc.) im Bereich des Text Minings auf das BOW Modell auf, welche jedoch in dieser Arbeit nicht weiter ausgeführt werden. Aus diesen genannten Gründen wird folglich auch näher auf das BOW Modell eingegangen. In der Literatur herrscht Uneinigkeit darüber, ob das BOW Modell nun zu den NLP-Methoden zu zählen ist oder der Domäne des Text Mining's angehört. Aus Sicht des Autors liegt jedoch eine Tendenz vor, dass das Konzept des BOW –was im Grunde genommen lediglich aus eine Liste mit Tokens besteht– als Artefakt des NLP zu verstehen ist. Hingegen das BOW als Modellbegriff zur Informations-Extraktion durch ein ML-Verfahren, eher im Kontext des Text Mining's zu finden ist.

Durch die Bearbeitung und Transformation eines Textes, wie es in den vorangegangenen Kapitel beschrieben wurde, liegt nun bereits ein BOW vor, welcher aus mehreren normalisierten Tokens besteht. Nun besteht die Aufgabe darin, den BOW so aufzubereiten, dass dieser für ein ML-Verfahren verarbeitbar wird. Nachdem ML-Algorithmen großteils nur mit numerischen Werten arbeiten, muss der BOW in einen *Feature-Vektor* transformiert werden (Brink et al., 2017). Dies passiert durch einen Prozess, der als *Vektorisierung* (*Vectorization*) bezeichnet wird (ebd.). Dabei werden alle unterscheidbaren Tokens aus dem BOW nach deren Vorkommnisse gezählt und das Ergebnis je Feature (=Token) als Vektor gespeichert. Angenommen, es gäbe mehrere Texte, so könnte

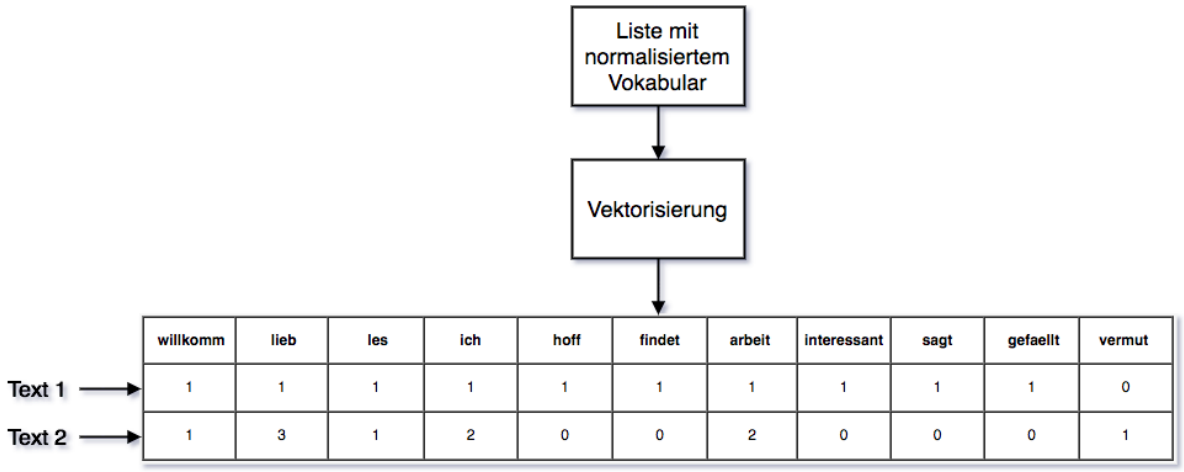


Abbildung 5.1: Prozess des BOW Modells anhand eines Beispiels

dieser auf die gleiche Weise bearbeitet, transformiert und vektorisiert werden, um damit einen Vektor-Raum (Vector space) aus gezählten Tokens zu erzeugen (Sarkar, 2016). Dabei repräsentiert jede Reihe bzw. Zeile im Vektor-Raum einen Text, Korpus oder gar ein ganzes Dokument. Spalten stellen dabei den jeweils zugehörigen Token dar. Um dies zu verdeutlichen, ist der Prozess der Vektorisierung in Abb. 5.1 anhand des Beispiel-BOW aus Listing 5.3 abgebildet. Die zweite Zeile ist das Ergebnis der Normalisierung und Vektorisierung des folgenden Quelltextes:

```
,,Willkommen liebe Leser! Ich vermute du liebst diese Arbeit. Ich
liebe diese Arbeit auch.''
```

Dieser Text weist Überschneidungen hinsichtlich der Wörter aus dem Text aus dem Beispiel in Abschnitt 5.3 auf. Durchläuft dieser Quelltext nun all die Schritte aus Abschnitt 5.3, so erhält man eine Liste mit folgenden Tokens:

```
['willkomm', 'lieb', 'les', 'ich', 'vermut', 'lieb', 'arbeit',
'ich', 'lieb', 'arbeit']
```

Wie aus Abb. 5.1 ersichtlich, werden alle unterscheidbaren Tokens der beiden Texte als Features geführt, d.h. jedes unterscheidbare Wort nimmt eine Spalte ein. Das bedeutet, dass für jedes neue, unterscheidbare Wort (zu den bereits bestehenden) aus einem Text oder Dokument, der Vektor-Raum um ein zusätzliches Feature erweitert wird. Nun kann es bei größeren Texten bzw. wortreichen Dokumenten durchaus vorkommen, dass hunderte bzw. tausende von Features existieren, wonach für die meisten Features über die Zeilen hinweg keine Einträge vorhanden sind. Dies führt meist dazu, dass die einzelnen Vektoren (Zeilen) großteils Nullwerte enthalten. Demnach werden diese Vektoren auch als *licht* oder *sparse* bezeichnet (Brink et al., 2017; Witten & Frank, 2005). Zur Repräsentation von BOW mit einem hohen *sparsity* – also überproportional viele

Null-Werte in einem Vektor oder Matrix – werden *Dense-Vektoren*, wie sie in Abb. 5.1 dargestellt sind, in sogenannte *Sparse-Vektoren* transformiert. Dazu bietet das *SciPy*[4] Framework für Python die notwendigen Funktionen an. Betrachtet man beispielsweise den Dense-Vektor

```
denseVec = array([0,1,1,0,0,0,0,0]),
```

welcher die Vorkommnisse der BOW-Features eines Textes repräsentiert, so besteht in diesem Fall der Text aus lediglich zwei Wörtern, was natürlich kein repräsentatives Beispiel ist, sondern der Einfachheit halber zur Erklärung an dieser Stelle dient. Wie nicht schwer zu erkennen ist, werden bei der Dense-Repräsentation auch die Nullwerte mit angeführt und auch gespeichert. Genau hier liegt der Nachteil dieser Form der Repräsentation bei großen bzw. lichten Vektoren. Die Nullwerte werden auch gespeichert und nehmen folglich verhältnismäßig viel Speicher ein. Die Sparse-Repräsentation hingegen würde für das oben genannte Beispiel folgendermaßen aussehen:

```
sparseVec = dict{(0,1):1,(0,2):1}
```

Zur Darstellung wurde für dieses Beispiel das Konzept eines assoziativen Feldes, das Dictionary (`dict`) in Python verwendet. Die zweier Tupel sind dabei die Indizes (Zeile, Spalte), die auf einen Wert im Vektor-Raum zeigen, die keine Nullwerte enthalten, sondern die Werte nach dem Doppelpunkt einnehmen. Das Konzept ist analog zu einem Key-Value Pair zu sehen, das man auch aus anderen Programmiersprachen und Dateiformaten kennt. Dabei ist zu sagen, dass Sparse-Repräsentationen in unterschiedlichen Ausprägungen in der Darstellung bzw. Implementierung existieren, auf die jedoch nicht näher eingegangen wird.

Wie bereits erwähnt, liegt der große Vorteil in der „leichtgewichtigen" Speicherung von BOW-Matrizen, da kein extra Speicher für Nullwerte gehalten werden muss (Witten & Frank, 2005). Ein möglicher Nachteil von Sparse-Repräsentationen besteht in deren Verarbeitbarkeit. Es existieren nur wenige ML-Algorithmen, die nativ mit Sparse-Daten umgehen können (Aggarwal, 2015; Brink et al., 2017). Dazu zählen z.B. der Naïve Bayes und der Random Forest. Das Multilayer Perceptron hat die Eigenschaft mit nicht-, oder niedrig-signifikanten Features umzugehen, ohne dabei die Accuracy zu vermindern (Patterson & Gibson, 2017). Diese Eigenschaften sind, unter anderem, auch als Rechtfertigung oder Untermauerung der Auswahl der Verfahren in Abschnitt 4.4 zu sehen.

In diesem Kapitel wurden die Grundlagen und Konzepte des Feature Engineerings erläutert, die für die Entwicklung eines ML-Modells im Rahmen der Fallstudie benötigt werden. An dieser Stelle ist zu sagen, dass der Umfang an Verfahren, Methoden und Techniken weit über denen in dieser Arbeit beschriebenen hinausgehen. Dies erstreckt sich von der Data Preparation Phase, über die Modellierung, bis hin zur Evaluation.

[4]Open-Source-Framework für das wissenschaftliches Rechnen und Visualisieren mit Python

Trotz diesem Umstand wurde versucht, einen kompakten und dennoch ausreichend detaillierten Einstieg ins Thema zu bereiten. Folglich wird das erworbenen Wissen im Rahmen der Fallstudie angewendet um die Beantwortung der Forschungsfrage zu ermöglichen.

6 Fallstudie: XiTrust

In diesem Kapitel werden das theoretische Wissen und die erlangten Erkenntnisse daraus auf eine echte Problemstellung anhand eines Datensets des Unternehmens XiTrust angewandt. Zum Ende des Kapitels wird mit dieser Fallstudie versucht, die Forschungsfrage mit den abgeleiteten Hypothesen zu beantworten.

6.1 Einleitung

6.1.1 Ausgangssituation

Die XiTrust ist ein Unternehmen aus Graz, welches sich als Anbieter für alle Dienstleistungen rund um die elektronische Signatur versteht. Für die Entwicklung der angebotenen Lösungen wird in der Softwareentwicklung auf die agile Vorgehensweise – genauer genommen auf Scrum – gesetzt. Demnach sind auch die in Abschnitt 2.2 beschriebenen Konzepte installiert, nach denen die Entwicklung abläuft. Damit existieren in prosa text beschriebene User Stories die in einem Software-Projektmanagement Tool gespeichert und verwaltet werden. Weiters sind zu jeder User Story, Story Points zugeordnet und gespeichert. Zudem sind für jede der abgeschlossenen Stories die tatsächlich gebuchten Stunden der beteiligten Personen im Team zugeordnet. Es existieren zwar noch weitere Metadaten zu einer User Story, welche aber für die Bearbeitung der Fallstudie irrelevant sind und daher nicht näher erläutert werden.

6.1.2 Ziel und Zweck

Der Entwicklungsleiter des Unternehmens ist nun interessiert an der Fragestellung, ob und wie denn nun die Story-Point Schätzungen mit Hilfe von computergestützten Verfahren verbessert werden können? Der Begriff „Verbesserung" wird dabei im Zusammenhang mit dem Schätzprozess gesehen, d.h. die computergestützte Schätzung liefert eine Basis, mit welcher die Schätzungen aus dem Planning Poker verglichen werden können und zur Diskussion anregen soll. Eine Substitution der Schätzung des

Teams durch eine Maschine ist nicht Ziel des Vorhabens und würde auch den Grundprinzipien der agilen Vorgehensweise widersprechen (*„Individuen und Interaktionen stehen über Prozessen und Werkzeugen"*). Das Ziel des zu entwickelnden Modells ist es daher, den Schätzungen des Scrum Teams anhand einer vorliegenden User Story so nahe wie möglich zu kommen und so eine initiale Schätzung für den Planning Poker vorzuhalten. Eine Verbesserung der Schätzgenauigkeit – also der Abgleich von Plan-Werten zum tatsächlichen Aufwand – ist nicht im Rahmen dieser Arbeit vorgesehen. Hier bedarf es einer Strategie zur Korrektur der Story Points in einer retrospektiven Analyse, welche sich jedoch durchaus als zukünftig weiterführender Forschungsbeitrag im Bezug auf diese Arbeit anbieten würde.

6.1.3 Ablauf der Fallstudie

Um die Beantwortung der in sec:Einleitung:ziele postulierten Forschungsfrage im Rahmen dieser Fallstudie durchzuführen, orientiert sich der Ablauf grundsätzlich am CRISP-DM. Da es sich jedoch um kein Projekt handelt, welches in Produktion eingesetzt wird, finden sich nur jene Prozessabschnitte mit deren Tätigkeiten wieder, die für ein prototypisches Modell vom Autor als relevant erachtet werden. Folglich ergeben sich vier große Blöcke, welche den grundsätzlichen Ablauf für die Bearbeitung der Fallstudie darstellen:

1. **Datenexploration**: Im Zentrum dieses Teilschrittes steht primär die Gewinnung von relevanten Informationen und Erkenntnissen hinsichtlich der Beziehungen der unabhängigen Variablen auf die abhängige Variable. Weiters steckt in diesem Prozessschritt bereits ein Teil des Feature Engineering's, da versucht wird, mittels statistischen Methoden und Werkzeugen bereits erste Features zu erzeugen und diese dann im Anschluss auf Abhängigkeiten zu anderen Variablen (insbesondere der abhängigen Variablen) zu analysieren. Für diese Phase existieren keine vordefinierten Prozesse oder Methoden. Vielmehr geht es darum, ein exploratives Vorgehen mit Hilfe von statistischen und grafischen Werkzeugen zur Gewinnung von Erkenntnissen heranzuziehen. Dieser Abschnitt kann im Kontext des CRISP-DM dem *Business Understanding* und dem *Data Understanding* zugeordnet werden.

2. **Datenvorbereitung und Transformation**: Besteht nun Klarheit über die vorhandenen Daten und deren Eigenschaften und Beziehungen, so besteht der nächste Schritt in der Vorbereitung des Datensets für die Modellierung. Dazu gehört die Bereinigung der Daten, Anpassung der Datentypen und die Transformation in ein BOW Modell. Diese Phase ist sehr eindeutig der Data Preparation Phase des CRISP-DM zuzuschreiben.

3. **Modellierung und Implementierung**: In dieser Phase erfolgt die Erarbeitung eines generischen Workflows, welcher in Form eines Modells abgebildet wird.

Aufbauend darauf erfolgt die Implementierung des Modells in Python. Dieser Abschnitt stellt sozusagen den Kern der Fallstudie dar und ist der Modeling Phase im CRISP-DM äquivalent.

4. **Evaluation und Ergebnisse**: In diesem Abschnitt erfolgt die Bewertung der Modell Performance bzw. dessen Qualität. Darin wurden Überlegungen zur Evaluationsstrategie und dessen Umsetzung bzw. Implementierung dargelegt. Nach statistischer Analyse der Ergebnisse wurden die wichtigsten Erkenntnisse zusammengefasst und erläutert. Als letzten Schritt gilt es, die Hypothesen zu falsifizieren und die Forschungsfrage zu beantworten.

Innerhalb der jede soeben beschriebenen Phase wurde versucht, die durchgeführten Schritte zu erläutern, um eine Nachvollziehbarkeit sicherzustellen. Dazu wurden z.B. auch relevanten Code Snippets im Anhang zur Verfügung gestellt.

6.1.4 Methoden und Tools

Für die Fallstudie wurde, wie bereits erwähnt, das CRISP-DM als Richtlinie für den Ablauf herangezogen. Innerhalb der Phasen dieses Prozessmodells gibt es, abhängig von der Problemstellung, geeignete Methoden, die bereits im theoretischen Teil dieser Arbeit ausführlich beschrieben wurden. Demnach wurde für die Fallstudie versucht, diese Methoden und Konzepte so gut wie möglich anzuwenden bzw. umzusetzen. Um genau diese Umsetzung zu ermöglichen, wurde sich an Open Source Projekten und die dadurch zu Verfügung stehenden Tools und Frameworks bedient, da es sonst nicht möglich gewesen wäre, die Forschungsfrage im vorgesehenen Umfang dieser Arbeit abzuhandeln bzw. zu beantworten.

Als eines der meist verwendeten Werkeuge in dieser Arbeit ist *Python* (V.3.5.2) zu nennen. Diese Programmiersprache ist bekannt für die umfangreichen Funktionen für das wissenschaftliche Rechnen (Scientific computing) und die dafür geschriebenen ML-Frameworks wie z.B. *scikit-learn* (V.0.19.0) oder *NLTK* (V.3.2.4), welche im Rahmen der Fallstudie hauptsächlich Anwendung fanden. Da in den Ausführungen der Fallstudie auf alle Eventualitäten und Details der Implementierung nicht eingegangen werden konnte, wird bereits an dieser Stelle auf die Online Dokumention verwiesen (*Documentation of scikit-learn 0.19.1*, 2017; *NLTK 3.2.5 documentation*, 2017). Weitere wichtige Bibliotheken zur Datenvorverarbeitung in Python wie *Pandas* (V.0.20.3), *Numpy* (V.1.13.1), *Scipy* (V.0.19.1) wurden ebenso eingebunden und verwendet. Als Entwicklungsumgebung für die Implementierungen in Python wurde das interaktive *Jupyter Notebook* (V.1.0.0), basierend auf einem *IPython-Kernel* (V.4.2.0), verwendet.

Weiters wurden die Datenanalysen und Visualisierungen (Diagramme und Plots) in der Phase Datenexploration mit der statistischen Scriptsprache *R* durchgeführt.

Hierbei wurde die Entwicklungsumgebung *RStudio* (V.1.0.136) verwendet, welche bereits eine Vielzahl an statistischen und grafischen built-in Funktionen beherbergt (*RStudio*, 2016). Zudem diente RStudio auch in der statistischen Analyse der Ergebnisse und damit der Operationalisierung der Hypothesen. Verwendete und für das Ergebnis relevante Funktionen wurden in den Ausführungen zwar erwähnt, jedoch sind dabei die Details in der Online Dokumentation nachzuschlagen (*RDocumentation*, 2017).

Obwohl für Python auch Statistik-Packages existieren, die eine ähnliche Funktionalität mitbringen, wie jene in R bzw. RStudio, und für R ebenso ML-Bibliotheken verfügbar sind, wurde die Verwendung beider, doch sehr unterschiedlichen Sprachen, im Sinne eines *Best-of-Breed* Ansatzes gesehen. R bzw. RStudio bringt dabei ein umfangreiches und relativ komfortabel zu verwendendes Statistik Repertoire mit einer weit verbreiteten Anwendung in Statistik-Fachkreisen und dort als die Sprache bzw. Tool der Wahl gesehen wird. Python hingegen glänzt als Multi-Purpose Programmiersprache in welcher eine Vielzahl an Angeboten an ML-Frameworks und Bibliotheken existieren. Zudem bietet die Sprache auch die Möglichkeit einer nativen Implementation der Modelle in Produktion, da Python auch in der Entwicklung von Web- und Desktopapplikationen verwendet wird bzw. verwendet werden kann.

6.2 Explorative Datenanalyse

Dem Unternehmen XiTrust ist es gelungen, die bereits erfassten und abgeschlossenen User Stories mit einem Großteil von dazugehörigen Metadaten aus dem Projektmanagement-Tool als Flatfile zu exportieren. Diese Daten kamen aus zwei unterschiedlichen Repositories bzw. zwei unterschiedlichen Produkten. Diese Daten wurden vom Autor zur Bearbeitung zusammengeführt. Um die Daten danach noch ggf. Filtern bzw. getrennt voneinander analysieren zu können, wurde eine zusätzliche Spalte „typ" mit den Ausprägungen „projekt" und „core" zum jeweiligen Datensatz vermerkt. Damit stellt das Datenset in Anhang 2 die Ausgangslage der Fallstudie dar. Die Inhalte der Spalten „Projekte.Aufgaben", „datei", und „text" wurden zur Erläuterung und Darstellung anonymisiert, da es sich hierbei um sensible Informationen der XiTrust handelt. Ebenso wird im weiteren Verlauf der Fallstudie Wert auf höchste Diskretion zum Schutz des Unternehmens XiTrust gelegt, was bedeutet, dass sensible Daten, welche zwar einen Einfluss auf das Ergebnis haben könnten, in dieser Arbeit nicht erläutert werden, sondern ggf. in anonymisierter Form ausgeführt werden oder umschrieben werden. Damit ist es trotz der Diskretion möglich, die Ausführungen und Ergebnisse nachvollziehbar und transparent zu halten.

Nun besteht der erste Schritt darin, eine Übersicht der Daten zu erlangen. Dies erfolgt zum einen durch eine deskriptive, statistische Zusammenfassung der einzelnen Spalten, zum anderen aus einer Analyse der Abhängigkeiten der Spalten zueinander – insbesondere auf jene der abhängigen Variable (*sp*). Doch zuvor noch eine kurze Erläuterung der Spalten und deren Bedeutung:

- **X**: Versionsnummer der jeweiligen User Story.
- **Projekte.Aufgaben**: Dabei handelt es sich um eine Kurzbezeichnung des Projektes, welches der Story zugeordnet werden kann (hier: anonymisiert).
- **sp**: Story Points (sp) die für die Umsetzung als letztes vor der Fertigstellung geschätzt worden war (= abhängige Variable)
- **Summe**: Summe der gebuchten Stunden des Entwicklungsteams für die die User Story nach Umsetzung.
- **datei**: Der Dateiname, unter welcher die User Story abgelegt wurde. Diente lediglich zum Einlesen des Textes der Datei in einen Data Frame (hier: anonymisiert).
- **text**: User Story als prosa Text (hier: anonymisiert)
- **typ**: Bezeichnet den Projekttyp bzw. welchem Hauptprodukt diese Story zuzuschreiben ist.

Wie bereits oben erwähnt wurde eine Zusammenfassung (Summary) des Datensets in R erstellt, welche in Anhang 3 zu finden ist. Von speziellem Interesse sind die Spalten mit numerischen Werten, für die es eine bedeutende Beschreibung gibt. Für *sp* lässt sich feststellen, dass diese von 1 bis 8 reichen, wobei es für die 8 nur zwei Datensätze gibt und für die 5 lediglich 4 aus insgesamt 102 Datensätzen. Wie mit diesen Datensätzen umzugehen ist, wird zu einem späteren Zeitpunkt erläutert.

Die Summe (*Summe*) der gebuchten Stunden reicht von unter einer Stunde bis hin zu rund 197 gebuchten Stunden je Story. Hier ist vorerst nichts weiter auffällig. Eine wichtige Erkenntnis im Zusammenhang *Summe* zu *sp* ergibt sich durch die Überprüfung der Abhängigkeit der beiden Variablen. Dabei gilt es zu überprüfen, ob *sp* in einer positiven Korrelation zum tatsächlich gebuchten Aufwand, also zur *Summe*, stehen. Dahinter verbirgt sich die Annahme, dass je höher die Schätzung in Story Points ausfiel, desto höher auch die dafür aufgewendete Zeit. Ein Boxplot in Anhang 4 zeigt durchaus einen linear verlaufenden Zusammenhang. Eine interessante, aber doch zu erwartende Beobachtung zeigt sich in den Quantilsgrenzen und den Ausreißern: Sie deuten möglicherweise auf eine Tendenz zur Überschätzung einer Story hin. Die Verteilung für sp=5 und sp=8 bestärkt wohl das Vorhaben, die insgesamt sechs Datensätze aus dem Datenset zu entfernen. Eine positive Korrelation (0.51)[1] zwischen *Summe* und *sp* untermauert die Annahme, dass die Schätzungen soweit eine bestimmte Plausibilität aufweisen und *sp* als Class Label in einem ML-Verfahren zu verwenden sind. Die *Summe* kann in diesem Kontext nicht verwendet werden, da es sich um eine *post-treatment* Variable handelt (McElreath, 2016). Auch aus einer Prozess-Perspektive gesehen, ist diese Spalte nicht als Feature geeignet, da die Werte

[1] Pearson-Korrelationskoeffizient

	X	Projekte.Aufgaben	sp	Summe	datei	text	typ	textlength
0	03.03.04	a09601bf47a1b3...	1	26.11	319f9d91a38073...	08b3581e9dd09b...	projekt	557
1	03.03.05	c3eec41b9a11fb...	3	65.86	79462438fdbbf4...	9fb499a68baf54...	projekt	2417
2	03.03.06	e53697f3eaa5cb...	1	16.67	cdb223e1cdb256...	220871dc8fc63c...	projekt	306
3	03.03.08	501085c04a3ef0...	1	21.69	217eb070e2ea6a...	1fd845be993bf5...	projekt	365
4	03.03.09	134a50696ca871...	1	4.81	2cd161e7bacf6f...	b5cc9e856c1ab8...	projekt	137

Tabelle 6.1: Auszug aus dem Datenset mit dem erzeugten Feature „textlength"

erst nach Abschluss der Story vorliegen und dadurch keine Information vorab für die Klassifizierung bereitgehalten wird.

Ein weiteres interessantes Ergebnis der explorativen Untersuchung ergab sich im Zusammenhang mit der *Textlänge* der User Story (*text*) und der Höhe bzw. Anzahl der Story Points. Es wurde die Textlänge (in Characters) jeder einzelnen User Story gezählt und in eine neue Spalte *textlength* gespeichert. Damit ist eine Überprüfung einer Korrelation zu *sp* möglich. Ein Boxplot im Anhang 5 verdeutlicht den Zusammenhang. Es scheint so, als ob die Textlänge stark positiv (0.66)[2] mit der Höhe der Story Points korreliert. Eine mögliche und plausible Interpretation wäre: Je höher die eingeschätzte Komplexität einer Story ausfiel, desto ausführlicher oder umfangreicher ist/war die jeweilige textuelle Beschreibung der User Story. Die Spalte *textlength* wird daher als Feature zur weiteren Verarbeitung aufgenommen (siehe Tabelle 6.1) da diese einen guten Diskriminator für die spätere Klassifizierung der *sp* abgeben kann. Demnach werden die Features *textlength*, *text* (bis hierher noch unberührt) und die abhängige Variable *sp* (=Class Label) als relevant für die weiteren Schritte erachtet. In Tabelle 6.1 findet sich ein Auszug aus dem Datenset mit dem zusätzlichen Feature *textlength*.

Die explorative Datenanalyse verhalf zu einem besseren Verständnis der Daten und auch einem eindringlicheren Verständnis über die Prozesse des Auftraggebers (in diesem Fall der XiTrust) und der Rahmenbedingungen eines Projektes. Zudem konnten bereits erste Features extrahiert werden in der Hoffnung, die Qualität der Vorhersageergebnisse dadurch zu verbessern. In weiterer Folge geht es an die Bearbeitung des Textes (*text*) und den notwendigen Transformationen zur Verarbeitung in einem ML-Verfahren.

[2]Pearson-Korrelationskoeffizient

6.3 Datenvorbereitung und Transformation

Dieser Abschnitt besteht aus zwei wesentlichen Aufgaben, und zwar a) die Transformation der Features in geeignete Datentypen und b.) die Textverarbeitung und Transformation des unstrukturierten Textes.

Da abgesehen von der User Story als Text, nur die Textlänge als relevantes Feature zu betrachten ist, so gilt es vorerst mal *textlength* auf Datentyp und Verteilung hin zu analysieren. Zur Übersicht wurde eine Auflistung über die bestehenden Datentypen je Spalte erzeugt (siehe Anhang 6). Das Feature *text* ist dabei als `object` deklariert. An dieser Stelle ist anzumerken, dass es in Python (Version 3.x.x) innerhalb von Pandas keinen Datentypen *String* oder *Character* gibt, sondern diese immer als `object` deklariert ist. Da für die User Story ohnehin zu einem späteren Zeitpunkt eine umfassendere Bearbeitung vorgesehen ist, wird als nächstes das Feature *textlength* betrachtet.

Wie aus Anhang 6 zu entnehmen ist, ist *textlength* als `int64` deklariert, was einem 64-bit Integer Datentyp entspricht. Die Textlänge kann grundsätzlich nur ganze Zahlen annehmen, daher ist `int64` passend und muss dahingehend nicht geändert werden. Sieht man sich die deskriptive Beschreibung von *textlength* in Anhang 3 etwas näher an, so ist die Differenz zwischen Median und Mean durchaus eine Beachtliche (209 Zeichen). Dies deutet auf eine starke Linksverschiebung der Werte hin. Um dies zu verdeutlichen, wurde ein Histogramm von *textlength* und den dazugehörigen Verteilungen der Quantile (Q-Q-Plot) erzeugt (siehe Anhang 7). Wie bereits in Abschnitt 4.3 erläutert wurde, kann eine logarithmische Transformation Abhilfe schaffen. Dies ist so zu verstehen, dass die Anwendung von `log()` in diesem konkreten Fall, die linksschiefe Verteilung in ein nahezu normal-verteiltes Muster transformiert (siehe Anhang 7). Die „Bereinigung" dieser A-Symmetrie des Features *textlength* kann sich später vorteilhaft auf die Qualität des Modells auswirken. Die logarithmische Transformation kann und wird im Zuge der Modellierung bzw. Implementierung berücksichtigt.

Im nächsten Schritt wird die Spalte *sp* näher betrachtet. Wie aus Anhang 6 zu entnehmen ist, ist *sp* als `int64` deklariert. Sollte diese Spalte nun als Class Label in scikit-learn verwendet werden, so ist eine entsprechende Encodierung erforderlich (Pedregosa et al., 2011). Vorher jedoch werden alle Instanzen mit sp=5 und sp=8 vollständig entfernt (gesamt 6 aus 102 Instanzen). Damit reduziert sich das Datenset auf 96 Instanzen mit sp von 1 bis 3. Der Grund für die Entfernung liegt in der geringen Anzahl (mit stark verschobener Verteilung) an Instanzen für sp=5 und sp=8. Diese könnten später ansonsten den Recall bzw. Precision der sp=1-3 negativ beeinflussen. Zur Encodierung existiert die Funktion `LabelEncoder()` aus dem scikit-learn Package mit welcher alle Klassenausprägungen (1, 2 und 3) zwischen 0 und (n-1)-Klassen encodiert werden. Dieses Mapping der Encodierung zu den ursprünglichen Werten ist jedoch etwas umständlich, da es damit zu einer optischen Verschiebung der Werte (sp=1 wird zu Class Label „0") kommt und dies im Sinne der Darstellung und Interpretation der Ergebnisse irreführend wirken könnte. Um eine Zuordnung der Labels als numerische

äquivalent darzustellen, d.h. sp = 1 entspricht Label „1", usw., transformiert die Funktion `inverse_transform()` wieder zurück in deren Ausgangswert bzw. werden die Label- Kategorien nun numerische gleich dem Ursprungswert dargestellt. Die Labels werden folglich in einen Array gespeichert und stehen zur Verwendung als Trainings- und Testlabels für die Modellierung und Implementierung zur Verfügung. Das Code Snippet der soeben beschriebenen Schritte zur Vorbereitung der Labels ist in Anhang 8 angeführt.

Als nächsten und letzten Schritt ist die Bearbeitung des Textes und dessen Transformation in ein BOW Modell. Unter „Bearbeitung" ist dabei von der Tokenisierung und Normalisierung des Textes die Rede und für „Transformation" in erster Linie die Vektorisierung gemeint ist. Nach einer Sichtung und Analyse der User Stories (text) nach deren am häufigst vorkommenden Vokabular wurde eine zusätzliche Stoppwort-Liste erstellt:

```
removalwords = ['story', 'beschreibung', 'titel', 'points', 'id',
'akzeptanzkriterien']
```

Diese Wörter sollten zusätzlich zu den aus dem NLTK Package entfernt werden, da die Wörter nahezu in jeder User Story vorkommen und daher keine Bedeutung für die Unterscheidung haben, sprich, keine geeigneten Diskriminatoren darstellen.

Weiters wurde aus der Sichtung festgestellt, dass innerhalb 4 User Stories keine Texte bzw. keine gültigen Zeichen vorhanden waren und daher unbrauchbar sind. Eine Vermutung für diesen Umstand ist ein fehlerhafter Export bzw. ein Bug im Export Tool, welches jedoch außerhalb des Kontextes dieser Arbeit liegt. Daher werden die vier Instanzen vollständig aus dem Datenset entfernt, was nun bedeutet, dass ab sofort insgesamt nur noch 92 Instanzen zur Verfügung stehen.

Ansonsten konnten keine besonderen Erkenntnisse aus der Zählung der Wörter aus dem gesamten Vokabular gewonnen werden. An dieser Stelle ist zu erwähnen, dass keine semantische Analyse des Textes durchgeführt wurde, da dies nicht im Umfang der Arbeit vorgesehen ist und dies zudem auch als nicht notwendig für die Beantwortung der Forschungsfrage erscheint.

Im nächsten Schritt wurde eine Funktion zur Bearbeitung und Normalisierung der User Stories erstellt (siehe Anhang 9), die sich grob aus folgenden Schritten zusammensetzt:

1. Entfernung aller Satzzeichen aus dem Text

2. Tokenisierung des Textes

3. Entfernung der Stoppwörter (mit Kleinschreibung) gemäß des Stoppword Dictionary aus dem NLTK Package

4. Entfernung der Stoppwörter (mit Kleinschreibung) aus `removalwords`

5. Stemming mittels SnowballStemmer aus NLTK

Nach der Vorbereitung eines BOW kann dieser in ein vektorisiertes Modell transformiert werden. Dies geschieht mit der Funktion `CountVectorizer()`, welche die Methode `.fit_transform()` implementiert. Diese Methode vereint gleich drei Schritte: a.) Die Bearbeitung des Textes durch die Übergabe der zuvor definierten Funktion `text_process`, b) Die Zählung der unterscheidbaren Tokens, und c.) die Vektorisierung bzw. Erstellung einer Sparse-Repräsentation. Das Code Snippet dazu ist in Anhang 10 zu finden. Damit bietet diese Funktion einen äußerst komfortablen Weg, einen Text über eine einzige Methode in ein BOW Modell zu transformieren. Transformiert man nun alle User Stories in ein BOW Modell, so erhält man eine Sparse-Matrix in der Form `(92, 1621)`. Diese ist so zu interpretieren, dass sich die Matrix aus 96 Zeilen (Anzahl der Instanzen) und 1636 Spalten (unterschiedliche Wörter bzw. Tokens) als Features zusammensetzt. Die *Sparsity* beträgt dabei $0.02(2\%)$, was bedeutet, dass aus allen Einträgen in der Matrix $(92 * 1621 = 149132)$, nur 2985 Einträge keine Nullwerte enthalten. Wie an diesem Fall zu sehen ist, kann eine Sparse-Repräsentation den Speicher des BOW Modells – im Vergleich zu einer Dense-Repräsentation – verhältnismäßig gering halten.

Nun stellt das BOW Modell, bestehend aus einer Vielzahl an Features (1621) als Input für die Implementierung bereit. Es wurde jedoch auch das Feature textlength erzeugt, welches ebenso als Feature Vektor mit in das Modell aufgenommen werden sollte. Wie dies zu bewerkstelligen ist und darauf ein Modell aufzubauen, ist Gegenstand des nächsten Abschnittes.

6.4 Modellierung und Implementierung

In dieser Phase besteht die Aufgabe, die vorbereiteten Features nun in ein Modell zu „gießen", um schlussendlich Vorhersageergebnisse der Story Points zu erhalten. Die Auswahl der ML-Verfahren wurde bereits in Abschnitt 4.4 getroffen, d.h. es wurde bereits abgehandelt, warum in weiterer Folge ein *Naïve Bayes*, *Random Forest* und ein *Multilayer Perceptron* als Klassifizierer zur Anwendung kommen. In Abschnitt 4.4 wurden ebenso die nun benötigten Grundlagen der Modellierung erläutert. Jetzt geht es darum, ein gesamtheitliches, generisches Modell zu erarbeiten, welches als Grundlage für die Implementierung in Python (scikit-learn) dient. Die Herausforderung dabei besteht im Aufbau der Funktionen bzw. einer Kontrollstruktur zur Ausführung der einzelnen Schritte. Auch wenn scikit-learn hier bereits viele nützliche Funktionen mitbringt, so ist darauf zu achten, dass diese auch den Anforderungen der Aufgabenstellung bzw. jener Anforderungen aus der Datenexploration gerecht werden.

Die ersten Schritte der Modellierung beginnen mit Überlegungen, die bereits getätigten Bereinigungen und Transformationen in eine Abfolge zu bringen, d.h. welche Bearbeitungsschritte vorzuziehen sind und welche nachgelagert werden können/müssen. Da es sich im vorliegenden Fall zum einen um eine Feature Matrix (BOW Modell aus *text*) und zum anderen um einen eindimensionalen numerischen Feature (*textlength*) handelt, so sind diese gesondert voneinander vorzubereiten. Danach besteht die Herausforderung, die Ergebnisse der Bearbeitung wieder zusammenzuführen, um diese einem Klassifizierer übergeben zu können.

Scikit-learn bietet hier die Möglichkeit, diese Schritte in sog. *Pipeline*s abzubilden. Das Konzept der Pipeline bringt den Vorteil mit sich, dass die einzelnen Bearbeitungsschritte in eine Art Batch zusammengefasst werden können und danach mit der Methode `.fit()` auszuführen bzw. zu trainieren. Pipelines lassen sich auch ineinander schachteln und bieten damit die Möglichkeit einer getrennten Bearbeitung von Features (Sub-Pipelines) innerhalb einer Pipeline. Dies zählt zwar bereits zu den etwas fortgeschritteneren Implementierungsmethoden, jedoch ist es für die vorliegende Problemstellung in dieser Fallstudie von großem Vorteil. Um dies zu verdeutlichen wurde zunächst ein generisches Modell (siehe Abb. 6.1) erstellt, auf welchem dann die Implementierung der ausgewählten Klassifizierer erfolgen kann.

Die Grundidee hinter dem Modell in Abb. 6.1 liegt in der getrennten Bearbeitung zweier Features, welche grundsätzlich unterschiedlich hinsichtlich dessen Bearbeitung zu behandeln sind. Dazu wird eine Pipeline instantiiert, in der alle Bearbeitungsschritte, inklusive die Übergabe an den Klassifizierer, von statten gehen. Innerhalb dieser Pipeline werden zwei weitere Pipelines instantiiert, in der zum einen, die Transformation der Spalte aus dem initialen Datenset (Tabelle 6.1) in ein BOW Modell (Feature Matrix) und zum anderen die Extraktion und Transformation von (*textlength*) durchgeführt wird. Durch den `FunctionTransformer()` besteht die Möglichkeit, eine Funktion von außerhalb der Pipeline aufzurufen und die Rückgabe in der Pipeline weiter zu verarbeiten. In der oberen Pipeline, wie auch in der unteren, dient diese Funktion an erster Stelle zur Extraktion der gewünschten Spalten (in diesem Fall *text* und *textlength*). Danach erfolgt in der oberen Pipeline die Normalisierung des Textes, Zählung der Tokens und der Vektorisierung mit Hilfe der Funktion `CountVectorizer()` (welche bereits zuvor erläutert wurde).

Als letzten Schritt der oben und unten abgebildeten Pipeline ist die Skalierung bzw. Normalisierung des Wertebereiches innerhalb der Sparse-Matrix (obere Pipeline) bzw. des Feature Vektors (untere Pipeline). Durch die Funktion `MaxAbsSclaler()` werden alle Werte so transformiert, dass der maximal absolute Wert 1 beträgt. Diese Skallierungsfunktion verschiebt bzw. zentriert die Daten nicht, so dass die Sparsity nicht zerstört wird (Pedregosa et al., 2011). Damit ist diese Skalierungsfunktion bestens für Sparse-Repräsentationen geeignet.

Die untere Pipeline beinhaltet in der Mitte noch einen zusätzlichen Schritt: die logarithmische Transformation. Wie bereits in Abschnitt 6.3 erläutert wurde, ist die

Verteilung des Features *textlength* stark verschoben. Dieser Verschiebung kann mit einer Logarithmischen Transformation entgegengewirkt werden. Dazu wird die Funktion `log10()` aus dem Numpy (np) Package verwendet. Da wir für das vorliegende Datenset sicherstellen können, dass keine Nullwerte in *textlength* enthalten sind, ist die Anwendung des Logarithmus (zur Basis 10) möglich und auch sinnvoll. Wäre dem nicht so, so wären entweder die Nullwerte zu bereinigen (z.B. Addition von 1 über alle Instanzen) oder die Funktion `log1p()` zu verwenden.

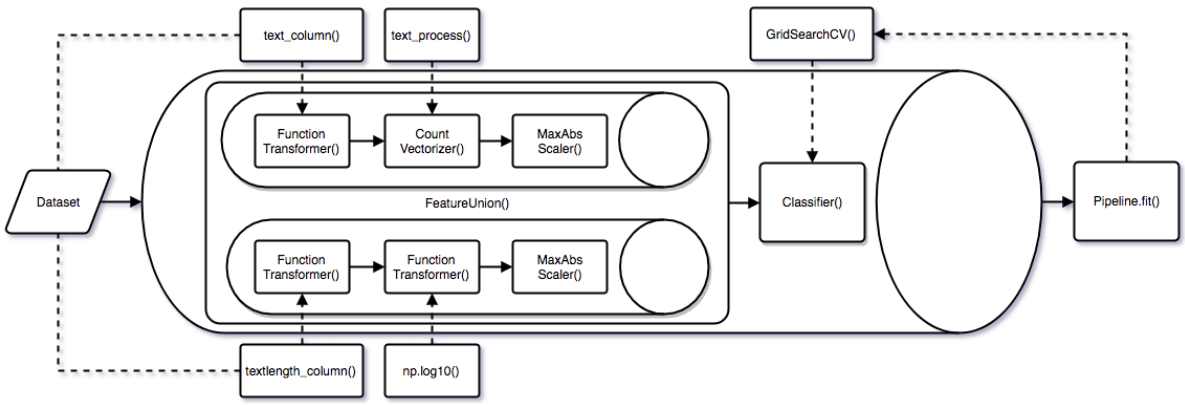


Abbildung 6.1: Generisches Modell zur Implementierung in Python mit scikit-learn

Diese beiden Pipelines werden nun innerhalb der Funktion `FeatureUnion()` implementiert. Diese ermöglicht die *Zusammenfassung* und *Gewichtung* der beinhalteten Features, welche dann dem Klassifizierer übergeben werden können. Die Bezeichnung `Classifier()` stellt im Modell eine quasi-Abstraktion dar. Die gesamte Pipeline wird dabei jeweils mit einem der drei `Classifier()` Objekte `MultinomialNB` `RandomForestClassifier()` und `MLPClassifier()` aus scikit-learn über die implementierte Methode `.fit()` trainiert. Dieser Methode sind in der Regel Parameter für den Klassifizierungsvorgang zu übergeben. Selbstverständlicherweise sind diese Parameter zwischen den genannten Klassifizierern sehr unterschiedlich. Obwohl für jedes `Classifier()` Objekt Default-Parameter existieren, ist die „richtige" Wahl der Parametereinstellungen in erster Linie von den Trainingsdaten abhängig und essenziell für Performance und Qualität des Lernverfahrens (Raschka, 2015).

Das Finden der „richtigen" Parametrisierung der `Classifier()` Objekte wird oftmals mit dem Begriff *Hyperparameter Tuning* gleichgesetzt (Raschka, 2015). Dabei sind *Hyperparameters* jene Parameter, welche die spezifische Anpassung des Algorithmus an das Datenset ermöglichen (Brink et al., 2017). Für die Durchführung der Optimierung gibt es grundsätzlich zwei Methoden: Die erste Methode besteht in einem Brute-Force-Ansatz, bei dem der Klassifizierer mit mehreren Möglichkeiten und Kombinationen der Parameter trainiert und evaluiert wird. Die Parameter-Kombination mit der besten Performance-Metrik (muss zuvor bestimmt werden) kann danach in das Modell übernommen werden. Der andere Ansatz gründet auf einem „zufälligen" Suchen der Parameter-Kombinationen. Dies bedeutet, dass die Parameter anhand einer bestimmten und vorgegebenen Verteilung einem Sampling unterworfen werden und dadurch

innerhalb einer bestimmten Verteilung nach Parameter gesucht wird. Mit diesem Ansatz wird nicht jede Kombination der Parameter ermittelt. Im Rahmen der Fallstudie entschied sich der Autor, den ersten Ansatz – also dem Brute-Force-Verfahren – zu verwenden. Um dieses Verfahren nicht von Grund auf zu implementieren, wurde eine Klasse aus dem scikit-learn Package verwendet: das `GridSearch CV()` Objekt. Dieses Objekt hat unter anderem die zwei Methoden `.fit()` und `.score()` implementiert, die es ermöglichen, einen Klassifizierer mit unterschiedlichen Kombinationen zu trainieren und zu bewerten. Zusätzlich wird über Cross Validation (CV) bereits ein Evaluationsverfahren mit den Daten erzeugt, dessen Ergebnis zur Beurteilung der Performance herangezogen werden kann. Da es sich bei diesem Ansatz auf das Finden der „besten" Kombination beschränkt, ist der Instanz von `GridSearchCV ()` ein `param_grid` zu übergeben. Bei diesem Grid handelt es sich lediglich um ein Dictionary, welches vorab mit den für den jeweiligen Klassifizierer verfügbaren Parameter und Ausprägungen manuell zu befüllt ist, aus welchen schlussendlich die Kombinationen erzeugt werden.

Zusammengefasst bedeutet dies, dass über eine Instanz von `GridSearchCV()`, abhängig vom gewählten Scoring-Modell und den vorliegenden Daten, ein Trainings- und Testset (im CV-Verfahren) erzeugt und trainiert wird und ein Ergebnis zur Auswahl der Parameter für das Modell bereitstellt wird. Da jedoch das Tuning grundsätzlich nur einmal zur Bestimmung des Parameter-Sets verwendet wird, ist es in Abb. 6.1 außerhalb der Pipeline dargestellt.

Um das Vorgehen und die Implementierung in diesem Abschnitt nochmals im Detail nachvollziehen zu können, wurden die ausgeführten Schritte von soeben als Code beispielhaft für das Multilayer Perceptron in Anhang 11 zur Verfügung gestellt. Im nächsten Abschnitt wird auf die Evaluation der Modelle eingegangen und dessen Ergebnisse erläutert.

6.5 Evaluation und Ergebnisse

An dieser Stelle gilt es nun, das soweit implementierte Modell auf dessen Qualität der Vorhersagekraft zu prüfen und zu validieren. Dies geschieht auf Basis ausgewählter Validationsverfahren und Performance Metriken, die gleich im Anschluss erläutert werden. Danach erfolgt die Ausführung der Operationalisierung der Hypothesen durch die Generierung von repräsentativen Stichproben aus dem Modell und die Durchführung von statistischer Tests.

6.5.1 Evaluation-Strategie und Ausführung

Der erste Schritt hin zu einem Evaluationsergebnis beginnt mit der Festlegung des Validationsverfahrens und den relevanten Performance Metriken. Welche Indikatoren bzw. Metriken dabei herangezogen werden, ist dabei stark abhängig vom Ziel der Klassifizierung und den vorliegenden Daten mit deren Eigenschaften. Laut Literaturrecherche und der theoretischen Abhandlung in Abschnitt 4.5 wurde das *k-Fold Cross-Validation*-Verfahren als Basis für die Durchführung aller nachfolgender Klassifizierungsvorgänge gewählt. Aufgrund der zur Verfügung stehenden 92 Instanzen, wurde k=5 gewählt (anstatt k=10 wie oftmals in der Literatur empfohlen). Der Grund dafür liegt zum einen an der relativ geringen Anzahl an Datensätzen und zum anderen an der zu erwartenden hohen Varianz bei k=10. Somit wurde zwar das untere Ende hinsichtlich der Anzahl der Splits für die Validation verwendet, jedoch ist dies, aus Sicht des Autors, aufgrund der Datenlage hinreichend akzeptabel im Rahmen dieser Arbeit.

Bei der Auswahl der Performance Metrik wurde aufgrund keiner besonderen Anforderungen an die Klassifizierung bzw. dessen Ergebnisse als maßgebende Kennzahl die *Accuracy* verwendet. Obwohl diese Metrik im Zusammenhang mit der Klassifizierung in der Literatur kontrovers diskutiert wird, so erscheint diese für die vorliegende Problemstellung am geeignetsten, da diese a.) einen generalisierenden Charakter der Qualität des Modells aufweist und b.) zudem vergleichsweise leicht zu interpretieren ist. Dies bedeutet nicht, dass ausschließlich die Accuracy zur Qualitätsbeurteilung gesehen wird (Recall, Precision und $F1$-Score werden auch betrachtet), jedoch wird die Accuracy im Rahmen der Fallstudie als Scoring-Metrik innerhalb der Validierung verwendet. Konkret bedeutet dies, dass diese Metrik für das Hyperparameter Tuning und als Referenz-Metrik für die statistischen Tests herangezogen wurde.

Wie bereits im vorhergehenden Abschnitt kurz erläutert, wurde das Modell mit Hilfe der Methode `GridSearchCV()` optimiert, sprich versucht, eine bessere Kombination des Parameter-Sets für die jeweiligen Klassifizierer zu finden, als deren Default-Parametrisierung. Die Verwendung des Wortes „bessereïmpliziert, dass folglich nicht die „beste"Kombination gefunden wurden, da nicht mit allen Parameter-Kombinationen trainiert wurde. Dies liegt daran, dass eine vollständig kombinatorische Validierung im Zuge des Hyperparameter Tunings zu rechenintensiv wäre. Demnach würden sich zu lange Laufzeiten (>6 Stunden) des Tunings ergeben, was jedoch in keiner Relation zu den zu erwartenden Verbesserungen hinsichtlich der Accuracy steht (dies gilt insbesondere für das Multilayer Perceptron). Es wurden daher einzelne Parameter der Modelle ausgewählt und dafür das zuvor beschriebene `param_grid` mit möglichen Ausprägungen befüllt. Dieses wurde zusammen mit `Scoring='Accuracy'` und `cv=5` der `GridSearchCV()` Instanz mitgegeben und mit dessen Methode `.fit()` trainiert (siehe Anhang 11). Dieser Vorgang wurde mit unterschiedlichen Ausprägungen mehrmals wiederholt und sich damit manuell Schritt für Schritt an Kombinationen an Ausprägungen herangetastet, bis die errechnete Accuracy keinen deutlichen Anstieg mehr verzeichnete. Die Verwendung des Parameters `random_ state=101`

diente dabei, um bei wiederholter Durchführung stets reproduzierbare Ergebnisse zu generieren. Damit wird ausgeschlossen, dass Veränderungen in den Ergebnissen nicht durch den Randomisierungs-Effekt beeinflusst werden, sondern ausschließlich auf die Parameter-Kombinationen zurückzuführen sind.

Die Ergebnisse des Tunings sind für die drei ausgewählten Klassifizierungsverfahren in Anhang 12 dargestellt. Dabei wurden die verfügbaren Parameter (*Param*) je Klassifizierer mit deren Ausprägung (*Wert*) gelistet. Der Zusatz „(T)" in der Spalte *Wert* steht für „Tuned" und deutet auf eine Abänderung des Default-Wertes durch das Hyperparameter Tuning. Im Umkehrschluss bedeutet dies, dass alle nicht mit „(T)" gekennzeichneten Werte, mit dem jeweiligem Default-Wert übergeben wurden. Erklärungen bzw. Details zu den Bedeutungen der Parameter sind der scikit-learn Online-Dokumentation zu entnehmen (*Documentation of scikit-learn 0.19.1*, 2017).

Weiters ist in der Spalte der Bezeichnung des Klassifizierers in eckigen Klammern die erzielte Accuracy ausgewiesen. Da diese Accuracy bereits in einem K-Fold Cross Validation Verfahren erzeugt wurde, deuten diese Werte auf einen relativ hohen Wert bzw. auf ein überraschend gutes Ergebnis hin (im Vergleich zur Annahme des Autors vor Ausführung). Das Multilayer Perceptron schneidet hierbei mit einer Accuracy von 96.73% am besten ab, dicht gefolgt vom Random Forest mit 94.56%. Der Multinomial Naïve Bayes liegt ebenso knapp dahinter mit einem Wert von 93.47%. Diese Werte sind nun so zu verstehen, dass z.B. das Multilayer Perceptron mit den übergebenen Parameter und den vorliegenden Daten in einem K-Fold CV-Durchlauf (k=5), rund 96% der Story Points (*sp*) korrekt vorhersagte. Die Parameter, wie in Anhang 12 angeführt, wurden für alle nachfolgenden Ausführungen übernommen.

Nun stellt sich die Frage, wie die Vorhersage im Detail aussieht, sprich, welche Ausprägungen des Class Labels *sp*, wie vorhergesagt wurden. Um diesen Einblick in die Ergebnisse zu ermöglichen, müssen im ersten Schritt Vorhersagen für jede Instanz erzeugt werden. Um dies auch in einem CV-Verfahren zu ermöglichen, wurde für jede Instanz die sich einmal im Test-Split (1 von k=5) befand, das Ergebnis der Vorhersage gespeichert. Dies bedeutet, um ein Datenset mit vollständigen Vorhersagen zu bekommen, muss jede Instanz mindestens einmal im Test-Split vorkommen. Dazu wurde auf die Funktion `cross_val_predict()` aus scikit-learn zurückgegriffen, welche genau diese Vorhersagen erzeugt und die Ergebnisse auf die Instanzen abbildet. Danach wurden mit diesen Ergebnissen über die Funktion `confusion_matrix()` und `classification_report()` eine Confusion Matrix und ein Klassifikation- Bericht erzeugt. Letzteres stellt im Grunde genommen eine Zusammenfassung der bereits in Abschnitt 4.5 ausführlich beschriebenen Performance-Metriken *Precision*, *Recall* und $F1$-*Score* dar.

In Anhang 13 wurde für jedes der drei Klassifizierungs-Verfahren eine Confusion Matrix erstellt. Daraus wird nun ersichtlich, wie sich die Vorhersagen der Story Points (Predicted Label) in Bezug auf die „echten" bzw. den gelabelten Story Points (True Label) verhalten. Grundsätzlich lässt sich festhalten, das der *Random Forest* für sp=1

und 2 alle korrekt klassifizierte. Lediglich bei sp=3 konnten 5 Instanzen nicht richtig klassifiziert werden. Der Naïve Bayes Algorithmus und das Multilayer Perceptron lieferten für sp=1 und 2 gleichwertige Ergebnisse, unterscheiden sich jedoch in der Vorhersage von sp=3. Hierbei schnitt das MLP etwas besser ab. Die Confusion Matrix zeigt eine allgemeine Klassifizierungsschwäche für sp=3, was unter Umständen daran liegt, da nur rund 21% der Instanzen mit sp=3 vorhanden sind.

Um die Ergebnisse nun in vollständiger Art und Weise zu quantifizieren bzw. in Kennzahlen auszudrücken, wurde ein Klassifikations-Report erstellt, welcher die Klassifizierungs-Ergebnisse für alle drei Klassenausprägungen zusammenfasst. Dazu wurden die zuvor erwähnten Performance-Metriken Recall, Precision und $F1$-Score errechnet und in eine tabellarische Darstellung übertragen (siehe Anhang 14). Wie aus der Confusion Matrix bereits ersichtlich war, leidet der Recall (0.75) in Bezug auf den Random Forest Classifier. Ohne nun im Detail jede Ergebnisse der drei Klassifizierer im Detail zu erläutern, so kann das Multilayer Perceptron als ausgewogenster Klassifizierer und damit als bevorzugter für diese Aufgabenstellung und Datenset gesehen werden, was sich auch im $F1$-Score (0.95) widerspiegelt.

6.5.2 Statistische Prüfung der Hypothesen

Nun dienen die Ergebnisse des Hyperparameter Tunings, der Confusion Matrix und des Klassifikations-Report in erster Linie dazu, einen Vergleich der trainierten Klassifizierer zu ermöglichen. In der Praxis würden die bereits zugrunde liegenden Ergebnisse ausreichen, um die Verwendung von diesem oder jenem Klassifizierers mit deren Eigenschaften auszuwählen und zu argumentieren. Im Rahmen dieser Fallstudie jedoch, sollte bewiesen werden, dass zumindest eines der trainierten Verfahren eine höhere Accuracy aufweist, als ein zufälliges Auswählen bzw. Aufteilen von vorhandenen Story Points im Datenset. Zu diesem Zweck wurde ein Dummy Klassifizierer trainiert, welcher unabhängig von den Trainingsdaten bzw. Features, alle Instanzen nach dem Zufallsprinzip labelt. Um dieses Experiment trotz des Zufallsprinzips etwas an den Gegebenheiten des Datensets anzupassen, wurden a.) nur jene Story Points verwendet, die auch tatsächlich im Datenset vorkommen und b.) die zufälligen Vorhersagen an die Klassenverteilung der Story Points angepasst. Dies bedeutet beispielsweise, dass sich auch für das Zufallsprinzip die Anzahl der Labels sp=1 im selben Verhältnis der tatsächlich vorkommenden sp=1 (44%) im Datenset widerspiegelt. Scikit-learn bietet dazu wieder ein passendes Objekt `DummyClassifier()` an, welches sich mit dem Parameter `'stratified'` instantiieren lässt, um die soeben beschriebenen Eigenschaften zu implementieren. Damit liegt nun ein Referenz-Klassifizierer vor, welcher ein *Random Guessing* auf dem Datenset ausführt.

Der nächste Schritt besteht darin eine Messgröße für den Unterschied in den Ergebnissen der Klassifizierer zu einem Random Guessing festzumachen. Dieser Unterschied wird auch als *Effektstärke* bezeichnet und stellt ein wichtiges Maß der statistisch,

praktischen Relevanz dar (Hedderich & Sachs, 2015). Der Effekt lässt sich auf unterschiedliche Art und Weise bestimmen, zu dem auch in der Literatur rund ein Dutzend unterschiedliche, teilweise kontroverse, Ansätze existieren und diskutiert werden. Für die vorliegende Problemstellung wurde gemäß der Eigenschaften[3] des vorliegenden Datensets die modifizierte Effektstärke nach Cohen (Cohen's d_s) gewählt (Hedges, 1981; Bühner & Ziegler, 2009). Diese Effektstärke ist relativ einfach zu berechnen, als auch einfach zu interpretieren und stellt sich wie folgt dar:

$$d_s = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{(s_1^2 + s_2^2)/2}} \tag{6.1}$$

Dabei sind $\bar{X}_1$ und $\bar{X}_2$ die jeweiligen Mittelwerte der Stichproben. s_1^2 und s_2^2 sind die Varianzen der beiden Stichproben. Für diese modifizierte Ausprägung des Cohen's d_s der gepoolte Varianz (auch Hedges g genannt) in Gleichung (6.1) sind unterschiedliche Varianzen in den Stichproben zulässig, jedoch muss die Anzahl der Stichproben in beiden Gruppen gleich sein (Bühner & Ziegler, 2009).

Als Grundlage für die Berechnung des Cohen's d_s wurde ein randomifiziertes Validierungs-Datenset erzeugt. Dazu wurde jeweils ein K-Fold CV-Vorgang (k=5) 10 mal hintereinander ausgeführt, so dass nach jedem Durchgang das Datenset durchgemischt wurde und, unter Berücksichtigung der Klassenverteilung der Story Points, die Splits für den nächsten Durchgang gemacht wurden. Damit wurde ein Stichprobenset mit jeweils 50 Accuracy-Werten je Klassifizierer erzeugt. Diesem Sampling-Vorgang liegt das Argument zugrunde, dass sich ohne dieses wiederholte splitten, zu wenig Evidenz für eine repräsentative Darstellung der Accuracy ergeben könnte, da der Split nur einmal auf dem Datenset erfolgen würde. In der Umsetzung des Samplings wurde die Funktion `RepeatedStratifiedKFold()` aus dem scikit-learn Package verwendet, welche genau die soeben beschriebene Vorgehensweise implementiert und dient nun als Basis zur Berechnung von d_s. Um nun einen Überblick über die Verteilung und deren wichtigsten Eigenschaften zu bekommen, wurde auf Basis der soeben gesampelten Werte, mit der Funktion `describe()` in R, eine deskriptive Statistik erzeugt (siehe Tabelle 6.2).

Mit den Eigenschaften *Mean* und *SD* aus Tabelle 6.2 wurden gemäß Gleichung (6.1) die Effektstärke d_s je Klassifizierer in R berechnet. Damit ergaben sich folgenden Effektstärke $d_s$je Klassifizierer (im Vergleich zu DummyClassifier()):

- MultinomialNB(): 6.07
- RandomForestClassifier(): 7.10
- MLPClassifier(): 7.12

[3]Gleiche Anzahl an Stichproben, normalverteilte Datenbasis und unterschiedlich große Varianzen

	DummyClassifier	MultinomialNB	RandomForsest	MLPClassifier
Mean	0.36	0.9	0.94	0.95
SD	0.11	0.06	0.04	0.05
Median	0.33	0.89	0.94	0.95
MAD	0.12	0.07	0.01	0.08
Min	0.11	0.72	0.83	0.83
Max	0.61	1	1	1
Range	0.5	0.28	0.17	0.17
Skew	0.2	-0.83	-0.4	-0.75
Kurtosis	-0.44	0.14	-0.01	-0.37
SE	0.02	0.01	0.01	0.01

Tabelle 6.2: Deskriptive Statistik des Validierungs-Datensets

Zur Orientierung über die Effektstärke ist folgendes zu sagen: Ein kleiner Effekt weist mindestens ein d_s von 0.20, ein mittlerer mindestens 0.50, und ein großer Effekt mindestens 0.80 auf (Hedderich & Sachs, 2015; Bühner & Ziegler, 2009).

Wie es nun scheint, liegen die Ergebnisse für die Effektstärke der trainierten Klassifizierer allesamt zwischen 6.07 und 7.12, was offensichtlich einen überaus großen Effekt im Vergleich zu einem Random Guessing darstellt.

Im nächsten Schritt stellt sich die Frage, wie viele Stichproben aus dem k-Fold CV Verfahren ermittelten Accuracy notwendig sind, um eine statistische Prüfung überhaupt „relevant" zu machen. Hierbei kommt die *Power* ins Spiel. Die Power oder auch *Power-Analyse*, ist in diesem Zusammnhang als statistischer Test zu verstehen, welcher Aufschluss über die Wahrscheinlichkeit gibt, dass die Nullhypothese (*H0*) korrekterweise verworfen werden kann, wenn die alternative Hypothese (H2) wahr ist (Haslwanter, 2016). In anderen Worten als Frage formuliert: Wie hoch ist die Wahrschenlichkeit einen realen Effekt zu messen? Mit diesem Test lässt sich die benötigte untere Anzahl an Stichproben berechnen, die mit einer bestimmten Wahrscheinlichkeit einen Effekt in gegebener Größe zum Ausdruck bringen. Dazu muss ein *Effekt*, ein *Signifikanz-Niveau* und die *Power als Wahrscheinlichkeit* bestimmt bzw. definiert werden. Im vorliegenden Fall wurde für die Berechnung der Power der Wert 0.95 verwendet, da der Anspruch seitens des Autors besteht, den Effekt mit einer 95%-igen Wahrscheinlichkeit entdecken zu wollen. Für das Signifikanz-Niveau wurde ein weit verbreiteter quasi-Standard von $\alpha = 0.05$ übernommen, was der maximal zulässigen Irrtumswahrscheinlicheit entspricht (Bühner & Ziegler, 2009). Zusammengefasst bedeutet dies nun, dass auf Basis des Effektes von 6.07 (kleinster Effekt), mit einem Signifikanz-Niveau von 5%, die benötigte Anzahl der Stichproben errechnet werden, um diesen Effekt mit einer Wahrscheinlichkeit von 95% in den Daten zu entdecken. Die Power Berechnung wurde ebenso in R mit der Funktion `pwr.t.test()` durchgeführt (siehe Anhang 15). Details zur Power Analyse bzw. dessen mathematischen Grundlagen sind dem Werk von Hedderich und Sachs (2015) zu entnehmen.

Die Ergebnisse der Power Berechnung weisen eine Stichprobengröße je Gruppe (Klassifizierer) von mindestens n = 2.24 – also 3 Stichproben – aus. Diese geringe Anzahl an Stichproben ist nicht weiter verwunderlich, da die Effektstärken aller Klassifizierer überproportional groß ausfielen. Mit diesem Ergebnis der Power Analyse sind die Voraussetzung hinsichtlich des Stichprobenumfangs für den bevorstehenden Gruppenvergleich allemal gegeben.

Auf Basis des Validierungs-Datensets ist nun abschließend ein sogenannter Null Hypothesis Significance Test (NHST) durchzuführen. Der Begriff NHST steht stellvertretend für ein Konglomerat an unterschiedlichen Signifikanztests, welchen eine Nullhypothese (*H0*) zu Grunde liegt, die es zu falsifizieren gilt . Dies impliziert, dass es eine alternative Hypothese (*H1*) geben muss, die im Gegenzug bewiesen werden soll. Zur Wiederholung sind an dieser Stelle *H0* und *H1* nochmals angeführt:

H0: Durch Feature Extraktion aus textuellen Anforderungsbeschreibungen kann die Schätzgenauigkeit des Aufwandes durch ein Mining Modell, im Vergleich zu einem „Random Guessing" des Aufwandes, NICHT erhöht werden.

H1: Durch Feature Extraktion aus textuellen Anforderungsbeschreibungen kann die Schätzgenauigkeit des Aufwandes durch ein Mining Modell, im Vergleich zu einem „Random Guessing" des Aufwandes, erhöht werden.

Im Verlauf der Arbeit konnte der Begriff „Aufwand" mit dem Schätzmaß der Story Points erarbeitet werden und ist an dieser Stelle stellvertretend dafür zu sehen.

Um *H0* nun zu falsifizieren, besteht der nächste Schritt in der Auswahl eines geeigneten NHST. Da sich die Daten der erzeugten Validierungs-Stichproben (annähernd) normalverteilt darstellen (siehe Tabelle 6.2), ist die Anwendung eines *t-Tests* grundsätzlich möglich. Ebenso sind die Stichproben der beiden Gruppen unabhängig voneinander, was ebenso ein wesentliches Kriterium für die Anwendungsvoraussetzung darstellt (Hedderich & Sachs, 2015). Der t-test wurde in R mit der Funktion `t.test()` durchgeführt (siehe Anhang 17). Trotz der zugrunde liegenden, annähernd normalverteilten Datenlage weisen die Daten des z.B. der MultinomialNB, aber auch des MLPClassifier einen erheblich negativen *Skew*-Wert (siehe Tabelle 6.2) auf, was darauf schließen lässt, dass in beiden Fällen eine linksschiefe Verteilung vorliegt und dahingehend eine zugrunde liegende Normalverteilung anzuzweifeln ist. Zur Veranschaulichung der Verteilungen der Accuracy Werte wurden Density-Plots erzeugt, welche in Anhang 16 abgebildet sind. Aus diesem Grund wurde ein weiterer NHST verwendet: der *Wilcoxon-Test* oder auch *Wilcoxon-Vorzeichen-Rang-Test*. Dieser Test zählt zu den nicht-parametrischen Signifikanztests und kann angewendet werden, wenn die Normalverteilung der zugrunde liegenden Daten nicht gegeben ist und die Stichproben voneinander unabhängig sind (Bühner & Ziegler, 2009). „Unabhängig" ist in diesem Zusammenhang so zu verstehen, dass sich die einzelnen Werte der Stichproben in den jeweiligen Gruppen nicht gegenseitig beeinflussen (ebd.). Der Wilcoxon-Test basiert

auf dem Konzept der Rangierung von Daten. Dabei werden nicht die Differenzenwerte der Stichproben errechnet, sondern die Werte in Ränge angeordnet. Somit ist für diesen Test einzig und alleine die Rangfolge ausschlaggebend und nicht die absoluten Differenzen zwischen den Werten (Bühner & Ziegler, 2009; Hedderich & Sachs, 2015). Um den Test durchzuführen, wurde in R die Funktion `wilcox.test()` verwendet, welcher ebenso in Anhang 17 ausgewiesen ist. Details und die mathematisch exakte Beschreibung des t-Tests sowie des Wilcoxon-Tests, sind den Werken von Hedderich und Sachs (2015) und Bühner und Ziegler (2009) zu entnehmen.

Die soeben genannten NHST's sind in der Berechnung grundverschieden, jedoch lassen sich die Ergebnisse bzw. die Signifikanz auf gleicher Weise interpretieren. Dazu wurde anhand dem MLPClassifier() das Code Snippet für den t-Test, als auch für den Wilcoxon-Test mit deren Ergebnissen im Anhang 17 angeführt. Für die beiden Tests wurde jeder Kassifizierer mit den Dummy Werten (Random Guessing) aus dem Validierungs-Datenset den jeweiligen Tests als Vektor übergeben.

Für die Ausführung ist vorab zu bedenken, dass die Arbeitshypothesen *H0* und *H1* nicht zu verwechseln sind mit jenen Hypothesen der NHST's. Für die Tests sind *H0* und *H1* so zu verstehen, dass diese die statistische Prüfung auf Lage der Mittelwerte durchführt. Somit gilt für *H0:* $\bar{X}_{class} = \bar{X}_{rand}$ und für *H1:* $\bar{X}_{class} \neq \bar{X}_{rand}$

Da geprüft werden soll, ob der Klassifizierer eine signifikant größere Accuracy (Mittelwert) aufweist, als ein Random Guessing, wurde auf die Alternative „greater" getestet, also $\bar{X}_{class} > \bar{X}_{rand}$. Dies bedeutet, dass aufgrund des offensichtlich großen Effektes der Klassifizierer nicht auf lediglich einen Unterschied getestet wurde (two.sided), sondern explizit darauf, dass der Mittelwert der Klassifizierer signifikant größer ist als jener des Random Guessings. Dieses Test-setting gilt ebenso für den Wilcoxon-Test.

Der *p-value* ergab für beide Tests den Wert $< 2.2e - 16$ im Beispiel aus Anhang 17. Dieser p-value ist nun folgendermaßen zu interpretieren: Unter der Annahme, dass *H0* wahr ist ($\bar{X}_{class} = \bar{X}_{rand}$), kann der beobachtete Effekt in $\ll 0.05\%(< 2.2 * 10^{-16})$ der Fälle durch Zufall erklärt werden. Die wichtigsten Ergebnisse aller Klassifizierer aus der Validierung wurden in Tabelle 6.3 nochmals tabellarisch zusammengefasst.

	t-Test	Wilcoxon	Effekt	Accuracy	SD
	[p-value]	[p-value]	[cohen d_s]	[Mean]	[SD]
MultinomialNB()	< 2.2e-16	< 2.2e-16	6.07	0.90	0.06
RandomForsestClassifier()	< 2.2e-16	< 2.2e-16	7.10	0.94	0.04
MLPClassifier()	< 2.2e-16	< 2.2e-16	7.12	0.95	0.05

Tabelle 6.3: Ergebnisse der Validierung für drei Klassifizierer auf Basis des Validierungs-Datensets

Daraus geht hervor, dass jeder Klassifizierer einen p-value $< 2.2e-16$ (unabhängig der angewandten Tests) aufweist und damit ein hochsignifikantes Ergebnis vorliegt. Dies

bedeutet, dass jeder auf das Validierungs-Datenset trainierte Klassifizierer signifikant „besser" performt, als ein zufälliges Raten von Story Points. Damit kann *H0* verworfen werden, was nun auch bedeutet, dass die Arbeitshypothese *H1* als vorläufig bestätigt anzunehmen ist. Da die Hypothesen *H0* und *H1* als Untersschiedshypothesen formuliert sind, kann demnach argumentiert werden, dass *H1* unter den Bedingungen der Fallstudie (Stichprobengröße, Verteilung, etc.) dieser Arbeit wahr ist.

Zusammenfassend kann gesagt werden, dass durch die gesetzten Maßnahmen in der Datenvorbereitung, Modellierung, Implementierung und Evaluation eine deutlich höhere Genauigkeit erzielt werden konnte, als durch zufälliges Schätzen der Story Points. Übertragen als Antwort auf die Forschungsfrage ist jedoch festzuhalten, dass die *Schätzgenauigkeit* unter der Bedingung der schwachen bzw. fehlenden Grundwahrheit der Labels (Story Points) nicht verbesserte werden konnte, sondern ausschließlich ein möglicher Weg zur Prozessverbesserung der Schätzverfahren im agilen Umfeld (z.B. Planning Poker) durch ML aufgezeigt werden konnte. Das Ergebnis der Fallstudie ist lediglich als Plausibilitätsüberprüfung zu verstehen und weist als solches noch nicht zwangsläufig einen praktischen Nutzen auf. Durch eine derart hohe Accuracy und der durchaus hohen Modellqualität insgesamt, ist davon auszugehen, dass ein Vergleich bzw. ein Plausibilitätscheck auf anspruchsvollerem Niveau mit höheren Anforderungen durchzuführen ist bzw. angemessen wäre, um einen Benchmark nahe der Grenzen solcher Modelle zu etablieren. Dennoch konnte durch die Fallstudie aufgezeigt werden, dass der Einsatz von ML in der Schätzung von Story Points ein großes Potential beherbergt und es sich lohnt, sich aufbauend auf dieser Arbeit, um weitere Forschungsarbeit zu bemühen.

7 Zusammenfassung und Ausblick

Die Softwareentwicklung als organisatorische Einheit mit dessen beteiligten Personen und Stakeholder sind zunehmend konfrontiert mit steigenden Anforderungen und Komplexität von Softwarelösungen. Um diesen Umständen gerecht zu werden, hielten in den vergangenen Jahren nach und nach agile Methoden und Vorgehensweisen Einzug in die Praxis. Das Arbeiten mit bzw. innerhalb von agilen Frameworks setzt ein hohes Maß an Flexibilität und Disziplin des Teams voraus, um dem Grundgedanken des agilen Manifests tatsächlich gerechnet zu werden. Darunter, unter anderem, auch der Umgang mit dem Schätzen des Umsetzungsaufwandes. Das Schätzen eines Entwicklungsvorhabens bzw. einer User Story in abstrakten Point-Metriken setzt eine bestimmte Erfahrung und Vertrautheit im Umgang mit High-Level Anforderungen und ein hohes Maß an Kontinuität in der Zusammenarbeit des Scrum-Teams voraus. Eine wesentliche Herausforderung ist dabei der interaktive Prozess des Schätzens innerhalb des Scrum-Teams, welcher zum Großteil von subjektiver Einschätzung auf Basis der Erfahrung der einzelnen Personen im Team erfolgt. Dabei ist anzunehmen, dass in vielen Fällen Schätzungen einem Bias unterliegen, durch welchen es zu ungewollten Verzerrungen der Story Points Schätzungen kommen kann. Um dieses Problem zu relativieren, kann der Einsatz von ML dazu verhelfen, eine initiale Schätzung dem Team bereitzustellen, welche als Basis für die Diskussion im Team dienen kann. Dazu wird eine Vielzahl an geschätzten User Stories aus der Vergangenheit genutzt, um eine initiale Schätzung durch eine ML-Modell dem Team bereitzustellen.

In dieser Arbeit konnte festgestellt werden, dass die Vorverarbeitung und Normalisierung von unstrukturiertem Text (User Story) und dessen Transformation in ein BOW Modell eine ausreichend gute Repräsentation darstellt, mit welchen durch die Verarbeitung in ausgewählten ML-Verfahren eine Accuracy von mindestens 90% erzielt werden konnte. Dies ist eine beachtliche Leistung, wenn man bedenkt, dass diese Ergebnisse aus einem wiederholten k-Fold CV Verfahren (k=5) ermittelt wurden. Demnach konnte die Nullhypothese *H0* mit einem hoch signifikanten p-value verworfen werden und damit die Arbeitshypothese *H1* indirekt bestätigt werden.

Nichtsdestotrotz liegt ein Schatten auf den Ergebnissen dieser Arbeit. So ist trotz des k-Fold Validierungsverfahrens nicht sichergestellt, dass Vorhersagen in der Zukunft an die ermittelte Accuracy aus der Fallstudie (unter der Berücksichtigung der Standardabweichung SD) anschließen können. Ein mögliche Verschlechterung der Accuracy Werte bei zukünftigen Schätzungen des ML-Modells kann sich a.) durch Veränderung des Schätzprozesses und dessen beteiligten Personen einstellen oder b.) in der Art und

Weise der Erstellung von User Stories gründen. Dies bedeutet zum einen, dass, wenn sich das Schätzverhalten des Scrum-Teams verändert, es nahe liegt, dass auch die Accuracy-Werte sich verschlechtern bzw. es zu großen Schwankungsbreiten kommen könnte. Zum anderen würde Annahmen zufolge auch das Verhalten in der Erstellung der User Stories zu einer Verschlechterung der Accuracy des Modells führen. So ist dies z.B. der Fall, wenn es mehrere Verfasser zu den User Stories gibt, welche unterschiedliche Formulierungen und ein unterschiedliches Vokabular verwenden. In beiden Fällen würde die Wirksamkeit bzw. praktische Relevanz des Modells eine Minderung erfahren. Um diesem Phänomen entgegenzuwirken, ist es unabdingbar, das Modell in bestimmten zeitlichen Abständen immer wieder anzupassen, sprich, neu zu trainieren, d.h. jede neue Vorhersage-Instanz in das Training mit aufzunehmen (in zeitlichen Abständen getaktet oder Online Learning[1]). Dazu empfiehlt der Autor, ein *aktives Monitoring* der gewählten Performance Metriken einzurichten, falls ein Modell in den produktiven Einsatz übergehen sollte.

Ein weiterer und wesentlicher Nachteil der Schätzung mit Story Points im Zusammenhang mit der Unterstützung durch ML besteht im Wahrheitsgehalt der Labels in den Trainingsinstanzen. Von einer *Ground Truth* kann nur gesprochen werden, wenn Labels jene Werte repräsentieren, welche auch im Kontext der Problemstellung tatsächlich als wahr bezeichnet werden können. Diese Anforderung hinkt etwas hinterher, wenn es um die Beimessung der Wahrheit von Story Points geht. Die Story Points in unserem Trainings-Datenset wurden zu Beginn des Sprint Plannings nach der Einschätzung des Teams definiert und danach ggf. nochmals nach dem Review im Backlog aktualisiert. Dies bedeutet, dass die gelabelten Story Points nur insofern der Wahrheit entsprechen, wie eben der Wissensstand des Teams zu Beginn bzw. während der Bearbeitung war, jedoch nicht nach Abschluss einer User Story. Es fehlt demnach die Information, ob nun eine Story, die initial mit „3" geschätzt wurde, nach Abschluss tatsächlich eine 3er Story war. Damit ist trotz der hohen Accuracy der trainierten Modelle in der Fallstudie die Vorhersagekraft lediglich auf Basis der Schätzungen der Teams aus der Vergangenheit limitiert. Um eine umfassendere und wahrheitsgetreuere Betrachtung zu erreichen, wäre eine retrospektive Korrektur der Story Points nach Abschluss jeder Story durchzuführen. Dies bedeutet, dass ein Strategie bzw. Logik entwickelt werden müsste, um eine derartige Korrektur durchführen zu können, was Gegenstand weiterer Forschungsaktivitäten sein könnte. Damit würde neben der Verbesserung des Schätzprozesses im Scrum Flow auch die die Grundlage zur Verbesserung der *Schätzgenauigkeit* bereitet.

Weiters ist anzumerken, dass die Ergebnisse der Fallstudie auf einem Datenset gründen, welches unter ganz bestimmten und unternehmensspezifischen Rahmenbedingungen entstanden ist. Alleine diese Tatsache schreit förmlich nach einer Prüfung auf breiterer Basis. So gilt es zu prüfen, ob die Ergebnisse auch bei *projekt- und teamübergreifenden* Settings haltbar sind. Zudem wäre es ebenso unabdingbar, eine größere Stichprobe als in der Fallstudie zu untersuchen.

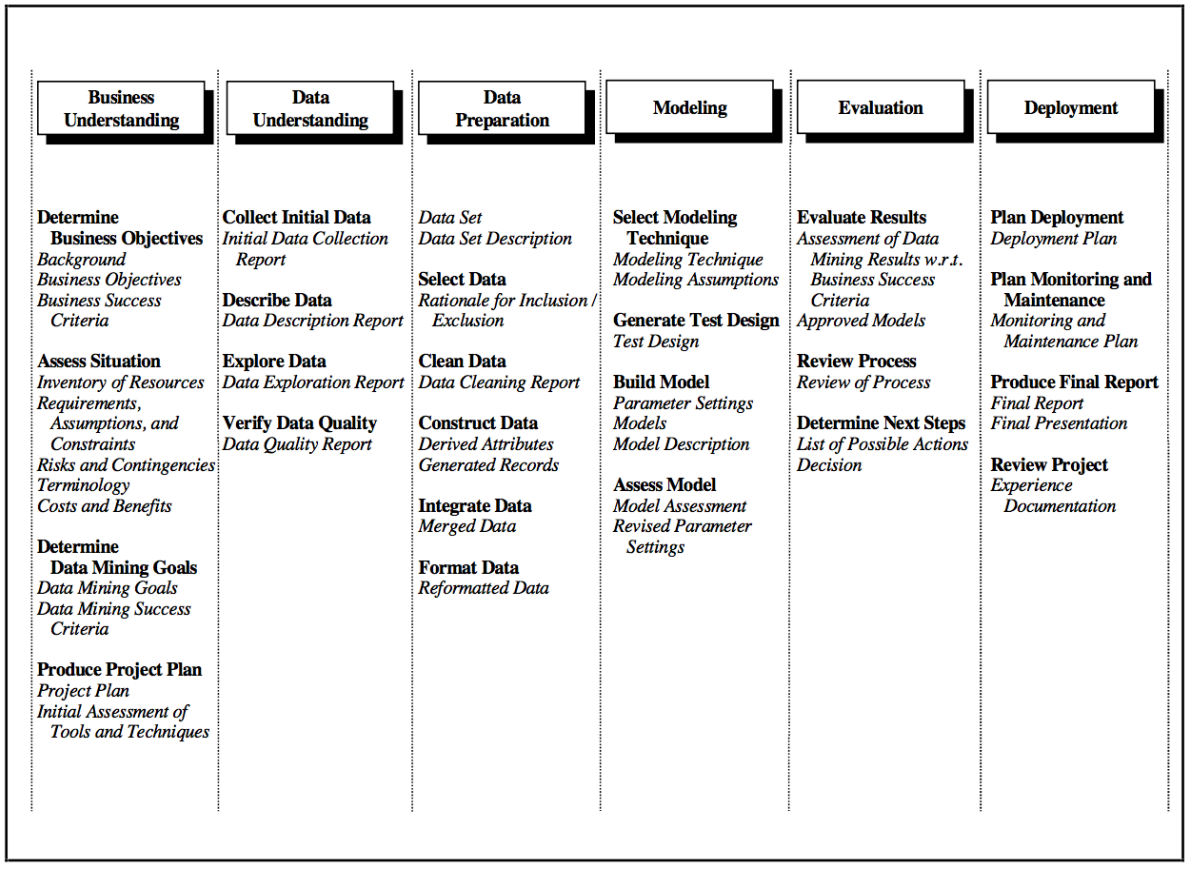
[1] Online Learning Modelle lernen ständig neu, indem sie die Ergebnisse ihrer Vorhersagen verwenden, um sich selbst zu aktualisieren

Was die Arbeitshypothesen dieser Arbeit betrifft, so ist es aus Sicht des Autors sinnvoll, für darauf aufbauende Forschungsarbeiten höheren Anforderung anzusetzen. Die Gegenüberstellung der Klassifizierungs-Ergebnisse mit jenen des Random Guessing's zeigte, dass diese allemal erfüllt wurden, jedoch dadurch nicht zwangsläufig die Praxistauglichkeit eines Verfahrens rechtfertigt. Dazu wäre eine Arbeitshypothese anzudenken, die auf einem direkten Vergleich von menschlichem Schätzverhalten zu jenem des ML-Modells gründet. Dabei sind auch die Erkenntnisse, wie auch die zuvor erwähnten Einschränkungen und Rahmenbedingungen dieser Arbeit von großer Bedeutung und zu berücksichtigen.

Abschließend ist zu sagen, dass auf der einen Seite eine Vielzahl an Fragen in zukünftigen Forschungsarbeiten noch abzuhandeln sind, um eine repräsentative Basis mit hoher praktischer Relevanz zu etablieren. Auf der anderen Seite jedoch konnte durch die Fallstudie demonstriert werden, welches Potential durch ML erschlossen werden kann, wenn es um die Verbesserung der Schätzungen in agilen Projekten geht. Zudem ist noch zu erwähnen, dass im Bereich des ML laufend Fortschritte in der Leistungsfähigkeit und der allgemeinen Verfügbarkeit der Algorithmen erzielt werden. Eine Vielzahl an ML-Verfahren sind großteils über einen relativ komfortablen Zugang (High-Level Implementierungen) via Framworks und Bibliotheken (Open Source) in unterschiedlichen Sprachen für eine breite Masse frei verfügbar geworden. Damit sind diese Technologien nicht nur jenen Personen mit tiefgehendem mathematisch- bzw. statistischen Wissen vorbehalten, sondern ebenso für, z.B. Applikations-Entwickler, zugänglich und anwendbar geworden, welche nicht über dieses Spezialwissen verfügen. Diese Umstände implizieren, dass ML zunehmend auch in der Praxis Einzug finden wird und sich möglicherweise noch weitere Anwendungsgebiete zur Unterstützung im Softwareprojektmanagement erschließen werden, die zum Zeitpunkt der Erstellung dieser Arbeit noch nicht bekannt waren bzw. dessen Potential noch nicht absehbar war.

Anhang

Anhang 1: Übersicht der Aufgaben und Artefakte des CRISP-DM



Aufgaben und Artefakte des CRISP-DM (Wirth & Hipp, 1999)

Anhang 2: Exportiertes Datenset aus dem Projektmanagement-Tool

	X	Projekte.Aufgaben	sp	Summe	datei	text	typ
0	03.03.04	a09601bf47a1b3...	1	26.11	319f9d91a38073...	08b3581e9dd09b...	projekt
1	03.03.05	c3eec41b9a11fb...	3	65.86	79462438fdbbf4...	9fb499a68baf54...	projekt
2	03.03.06	e53697f3eaa5cb...	1	16.67	cdb223e1cdb256...	220871dc8fc63c...	projekt
3	03.03.08	501085c04a3ef0...	1	21.69	217eb070e2ea6a...	1fd845be993bf5...	projekt
4	03.03.09	134a50696ca871...	1	4.81	2cd161e7bacf6f...	b5cc9e856c1ab8...	projekt

Exportiertes Datenset aus Projektmanagement-Tool

Anhang 3: Zusammenfassung (Summary) des Datensets zu Beginn der explorativen Analyse

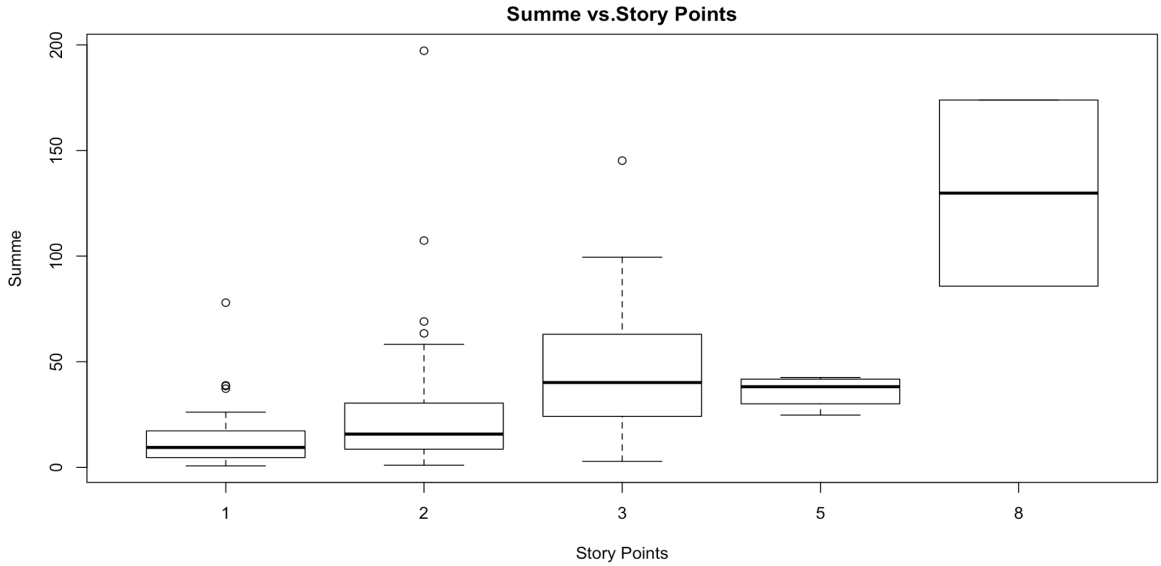
```
       X        Projekte.Aufgaben       sp           Summe            datei
 03.03.04: 1   ALETHEIA-42: 1      Min.   :1.00   Min.   :  0.670   Length:102
 03.03.05: 1   ALETHEIA-43: 1      1st Qu.:1.00   1st Qu.:  7.527   Class :character
 03.03.06: 1   AWO-215    : 1      Median :2.00   Median : 16.370   Mode  :character
 03.03.08: 1   BEV-209    : 1      Mean   :2.01   Mean   : 28.216
 03.03.09: 1   CISP-16    : 1      3rd Qu.:2.75   3rd Qu.: 37.193
 03.03.10: 1   CISP-17    : 1      Max.   :8.00   Max.   :197.210
 (Other) :96   (Other)    :96
     text                typ       textlength
 Length:102         projekt:69   Min.   : 129.0
 Class :character   core   :33   1st Qu.: 255.2
 Mode  :character                Median : 395.0
                                 Mean   : 604.2
                                 3rd Qu.: 689.5
                                 Max.   :4097.0

[1] "Anzahl der Datensätze je sp"

 1  2  3  5  8
43 33 20  4  2
[1] "Korrelation zwischen Summe und sp"
[1] 0.5118473
```

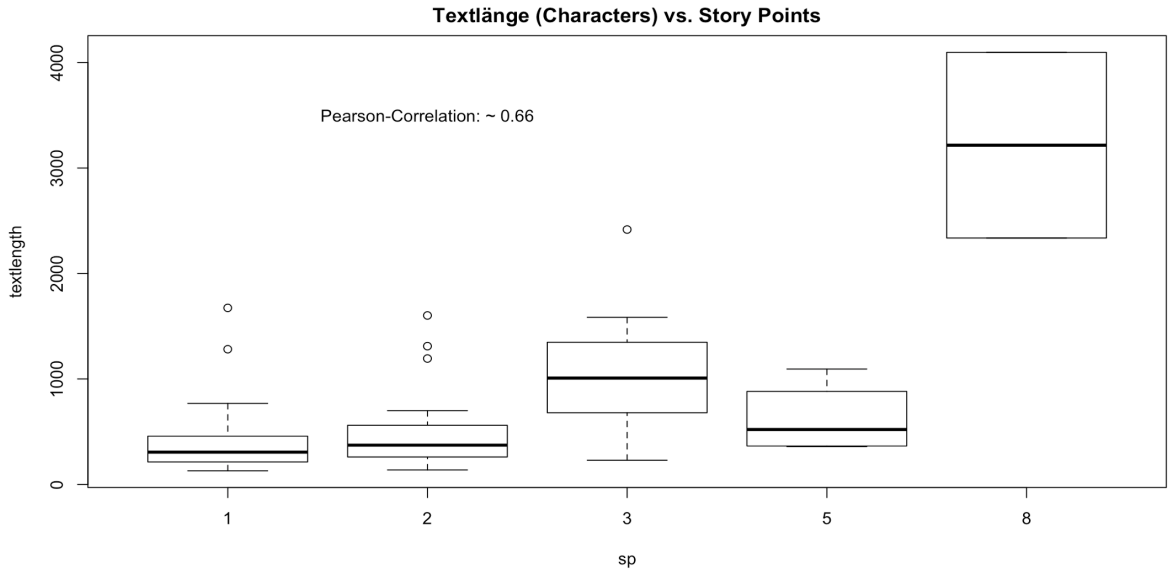
Summary aus dem Export-Datenset

Anhang 4: Boxplot zur Darstellung der Verteilung der Summe zu den geschätzten Story Points



Boxplot von Summe vs. Story Points

Anhang 5: Boxplot von Textlänge (Characters) zu den geschätzen Story Points



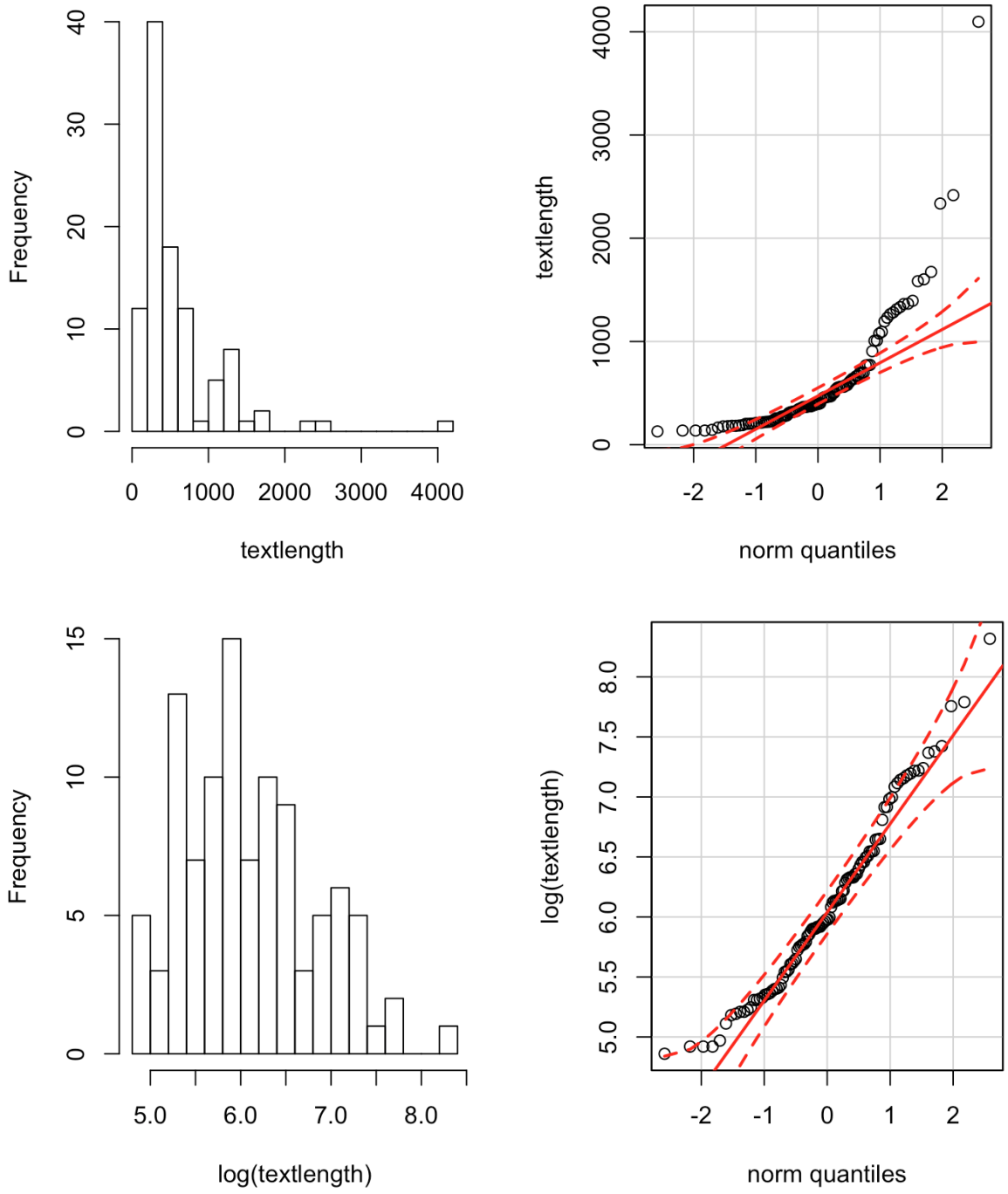
Boxplot von Textlänge vs. Story Points

Anhang 6: Datentypen des Datensets

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 102 entries, 0 to 101
Data columns (total 8 columns):
X                     102 non-null object
Projekte.Aufgaben     102 non-null object
sp                    102 non-null int64
Summe                 102 non-null float64
datei                 102 non-null object
text                  102 non-null object
typ                   102 non-null object
textlength            102 non-null int64
dtypes: float64(1), int64(2), object(5)
memory usage: 6.5+ KB
```

Datentypen des Datensets mit der Funktion .info() in Python

Anhang 7: Histogramm und Q-Q-Plot des Features textlength (Vorher-Nachher-Vergleich)



Histogramm (links) und Q-Q-Plot (rechts) von textlength und log(textlength)

Anhang 8: Code Snippet zur Bearbeitung und Transformation des Class Labels

```
1  from sklearn.preprocessing import LabelEncoder
2  # Nur Instanzen mit sp=1,2,3 verwenden
3  df_reduced = df[(df['sp'] == 1) | (df['sp'] == 2) | (df['sp'] ==
       3)]
4  # Instanzieren des Label Encoders
5  le = LabelEncoder()
6  # Transformation der Spalte sp in Class Label
7  df_sp_cat = le.fit_transform(df_reduced['sp'])
8  df_sp_cat = le.inverse_transform(df_sp_cat)
9  # Speicherung als Label Array fÃ¼r spaetere Verwendung als Ground
       Truth
10 y = df_sp_cat
```

Anhang 9: Funktion zur Textbearbeitung und Normalisierung

```
1  def text_process(text):
2    # Check char ob diese Punctuations enthalten
3     nopunc = [char for char in text if char not in
         string.punctuation]
4     # Join char um wieder einen String zu erzeugen
5     nopunc = ''.join(nopunc)
6     # Jetzt entferne ALLE Stopwords
7     tok = [word for word in nopunc.split() if word.lower() not in
         stopwords.words('german')]
8     # Nun entferne zusaetzlich eine definierte Liste mit
         Bezeichnungen,
9     # welche in jeder Story vorkommen --> Wenig aussagekraeftig
         bzw. kein
10    # geeigneter Diskriminator.
11    removalwords = ['story', 'beschreibung', 'titel', 'points',
         'id', 'akzeptanzkriterien']
12    cleandict = [word for word in tok if word.lower() not in
         removalwords]
13    # Stemming --> Char auf Wortstamm zurueckfuehren.
14    stemmer = SnowballStemmer("german")
15    return [stemmer.stem(word) for word in cleandict]
```

Anhang 10: Code Snippet der Transformation von Text in ein BOW Modell mit der Berechnung der Sparsity der Repräsentation

```
from sklearn.feature_extraction.text import CountVectorizer
# Fit und transformiere den Text in ein BOW Modell
text_bow = CountVectorizer(analyzer=text_process)
                  .fit_transform(df_reduced['text'])
print(text_bow.shape)
print('Shape of Sparse Matrix: ', text_bow.shape)
print('Amount of Non-Zero occurences: ', text_bow.nnz)
sparsity = (text_bow.nnz / (text_bow.shape[0] *
   text_bow.shape[1]))
print('sparsity: {}'.format(sparsity))
(96, 1636)
Shape of Sparse Matrix: (92, 1621)
Amount of Non-Zero occurences: 2985
sparsity: 0.02001582490679398
```

Anhang 11: Code Snippet der Implementierung einer Pipeline des Multilayer Perceptrons (MLPClassifier()) mit dessen Hyperparameter Tunings (GridSearchCV())

```
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import FunctionTransformer
from sklearn.neural_network import MLPClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.model_selection import GridSearchCV

# Funktionen die innerhalb der dnn_pipeline aufgerufen werden
def text_column(df_reduced):
   return df_reduced['text'].as_matrix()

def textlength_column(df_reduced):
   return df_reduced[['textlength']].as_matrix()
# Pipeline
dnn_pipeline = Pipeline([
   ('union', FeatureUnion(
      transformer_list= [
          ('text_transformer', Pipeline([
            ('selector', FunctionTransformer(text_column,
               validate=False)),
            ('bow', CountVectorizer(analyzer=text_process)),
            ('scale', MaxAbsScaler())
                    ])),
         ('misc_feature', Pipeline([
            ('selector', FunctionTransformer(textlength_column,
               validate=False)),
            ('logtrans', FunctionTransformer(np.log10,
               validate=False)),
            ('scale', MaxAbsScaler())
                    ]))
                ],
      transformer_weights={
         'text_transformer': 0.5,
         'misc_feature': 0.5
         }
         )),
   ('classifier', MLPClassifier(random_state=101,
                           hidden_layer_sizes=(8,),
                           solver='lbfgs',
                           activation='relu',
                           learning_rate_init=0.01,
                           warm_start=True
```

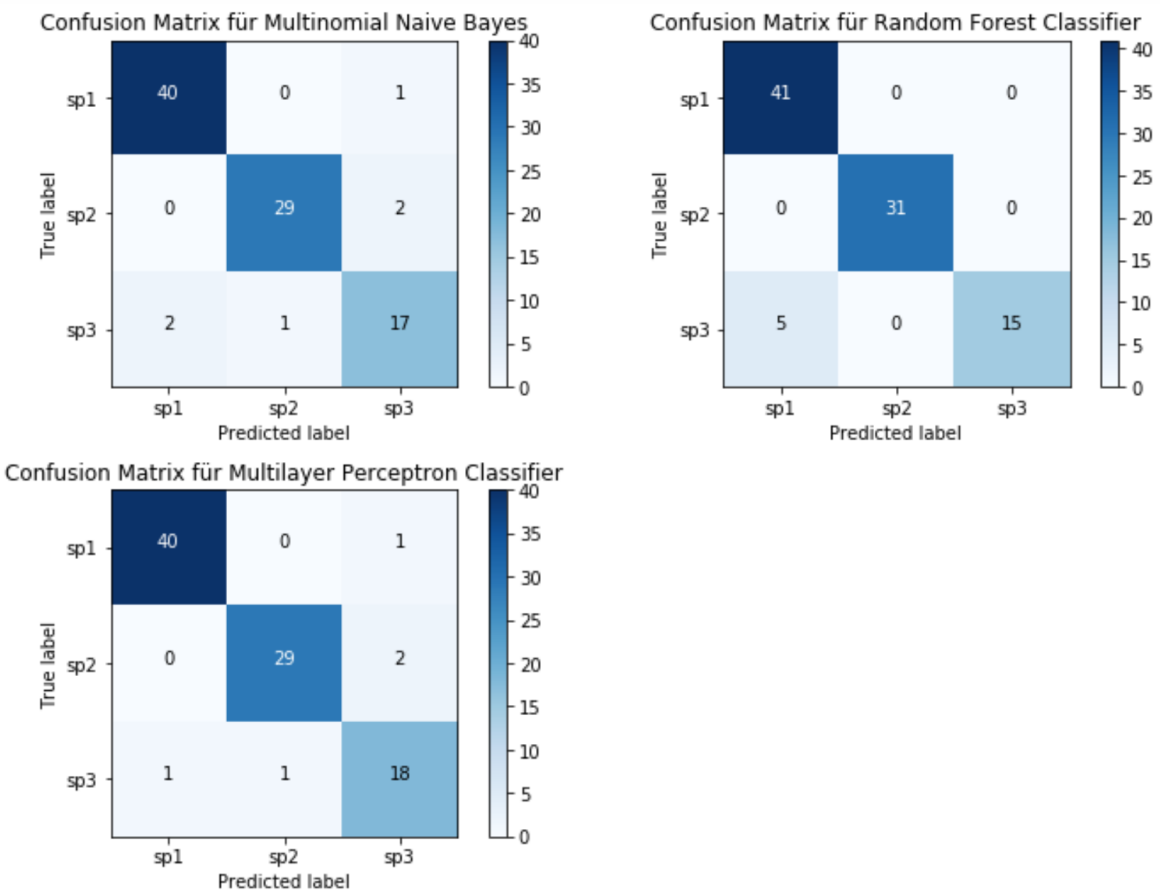
```
41                              ))
42     ])
43  # Hyper Parameter Tuning durch GridSearchCV()
44  dnn_pipe_grid = GridSearchCV(dnn_pipeline, scoring='accuracy',
       cv=5,
45                           param_grid = {
46                          "classifier__hidden_layer_sizes": [2,8,12],
47                          "classifier__activation":
                               ['logistic','relu'],
48                          "classifier__learning_rate_init":
                               [0.01,0.001,0.1]})
49  dnn_pipe_grid.fit(df_reduced, y)
50  print(dnn_pipe_grid.best_params_)
51  print(dnn_pipe_grid.best_score_)
52  {'classifier__hidden_layer_sizes': 8, 'classifier__activation':
       'relu', 'classifier__learning_rate_init': 0.01}
53  0.967391304348
```

Anhang 12: Parameter-Kombinationen für drei Klassifizierer mit deren erzielten Accuracy via GridSearchCV

RandomForestClassifier() [Acc: 0.9456]		MLPClassifier() [Acc: 0.9673]	
Param	Wert	Param	Wert
bootstrap	True	activation	relu
class_weight	None	alpha	0.0001
criterion	gini	batch_size	auto
max_depth	None	beta_1	0.9
max_features	auto	beta_2	0.999
max_leaf_nodes	None	early_stopping	False
min_impurity_decrease	1e-08 (T)	epsilon	1e-08
min_impurity_split	None	hidden_layer_sizes	(8,) (T)
min_samples_leaf	1	learning_rate	constant
min_samples_split	2	learning_rate_init	0.01 (T)
min_weight_fraction_leaf	0.0	max_iter	200
n_estimators	900	momentum	0.9
n_jobs	1	nesterovs_momentum	True
oob_score	False	power_t	0.5
random_state	101 (T)	random_state	101 (T)
verbose	0	shuffle	True
warm_start	False	solver	lbfgs (T)
		tol	0.0001
		validation_fraction	0.1
		verbose	False
		warm_start	True
MultinomialNB() [Acc: 0.9347]			
param	wert		
alpha	3 (T)		
class_prior	None		
fit_prior	False (T)		

Parameter-Kombinationen für drei Klassifizierer

Anhang 13: Confusion Matrix für drei Klassifizierungsverfahren aus einer k-Fold CV (k=5)



Confusion Matrix für drei Klassifizierungsverfahren

Anhang 14: Klassifikations-Report für die drei Klassifizierungsverfahren aus einer k-Fold CV (k=5)

Classifier	Class	Precision	Recall	$F1$-Score	Support
MultinomiaNB()	sp1	0.95	0.98	0.96	41
	sp2	0.97	0.94	0.95	31
	sp3	0.85	0.85	0.85	20
	avg./total	**0.93**	**0.93**	**0.93**	92
RandomForestClassifier()	sp1	0.89	1.00	0.94	41
	sp2	1.00	1.00	1.00	31
	sp3	1.00	0.75	0.86	20
	avg./total	**0.95**	**0.95**	**0.94**	92
MLPClassifier()	sp1	0.98	0.98	0.98	41
	sp2	0.97	0.94	0.95	31
	sp3	0.86	0.90	0.88	20
	avg./total	**0.95**	**0.95**	**0.95**	92

Klassifikations-Report

Anhang 15: Power Analyse in R zur Bestimmung der Stichprobengröße

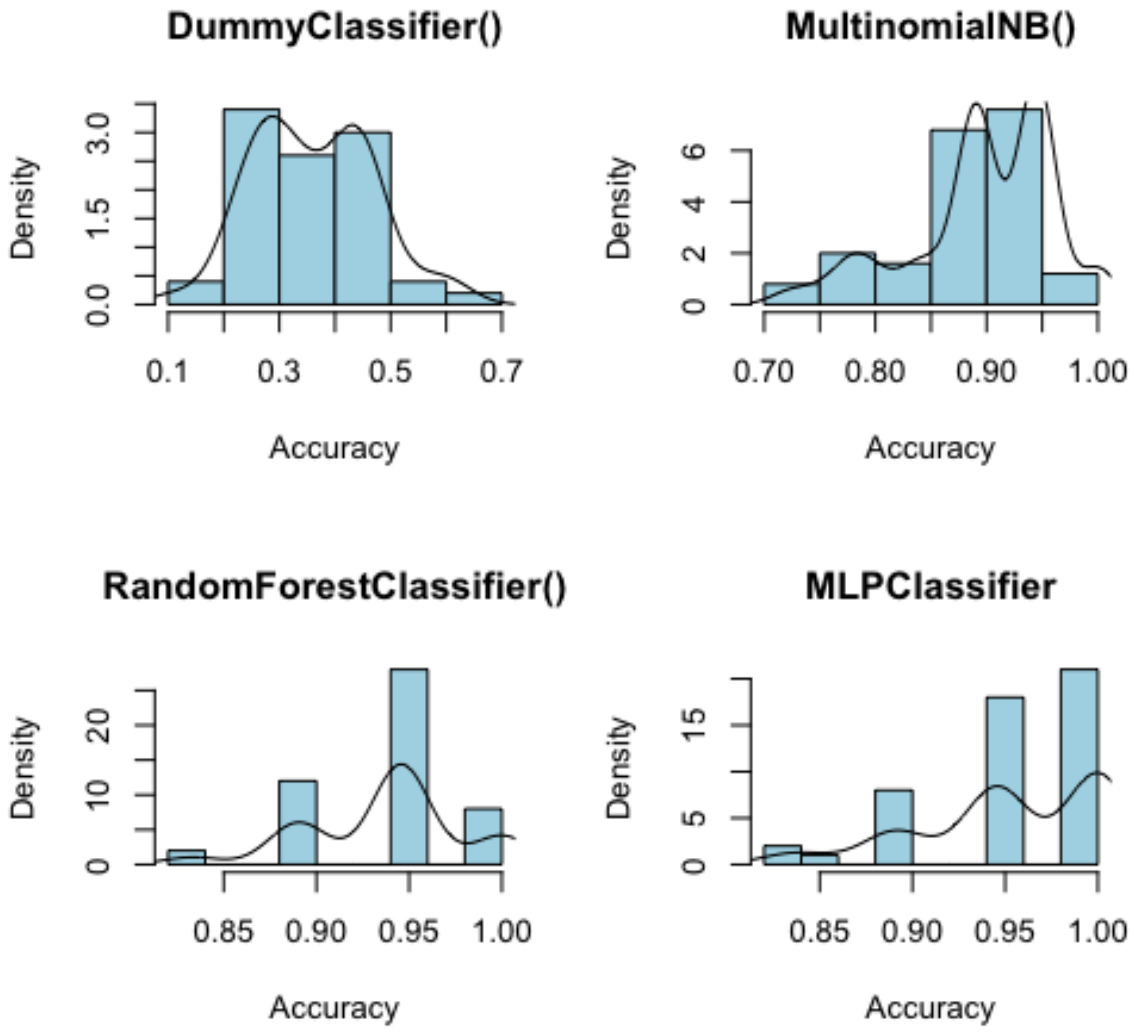
```
library(pwr)
> pwr.t.test(d = 6.07,
+          sig.level = 0.05,
+          power = 0.95)

     Two-sample t test power calculation

              n = 2.23919
              d = 6.07
      sig.level = 0.05
          power = 0.95
    alternative = two.sided

NOTE: n is number in *each* group
```

Anhang 16: Density-Plot je Klassifizierer der Accuracy-Werte aus dem Validierungs-Datenset



Density-Plot der Accuracy-Werte aus dem Validierungs-Datenset

Anhang 17: t-Test und Wilcoxon-Test in R

```
library(psych)
> t.test(valset$MLPClassifier,valset$DummyClassifier, alternative
   = 'greater')

      Welch Two Sample t-test

data: valset$MLPClassifier and valset$DummyClassifier
t = 35.595, df = 68.505, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.5654295   Inf
sample estimates:
mean of x mean of y
 0.953432 0.360214

> wilcox.test(valset$MLPClassifier,valset$DummyClassifier,
   alternative = 'greater')

      Wilcoxon rank sum test with continuity correction

data: valset$MLPClassifier and valset$DummyClassifier
W = 2500, p-value < 2.2e-16
alternative hypothesis: true location shift is greater than 0
```

Abkürzungsverzeichnis

AI	Artificial Intelligence
BOW	Bag-of-Words
COCOMO	Constructive Cost Model
CRISP-DM	Cross-Industry Standard Process for Data Mining
CV	Cross Validation
DL	Deep Learning
DM	Data Mining
EF	Einflussfaktor
FN	False Negative
FP	False Positive
KDD	Knowledge Discovery in Databases
KLOC	Kilo Lines of Code
KNN	Künstliche Neuronale Netz
LOC	Lines of Code
ML	Machine Learning
MLP	Multilayer Perceptron
NHST	Null Hypothesis Significance Test
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
POS	Part-of-Speech
Q-Q-Plot	Quantil-Quantil-Plot
QF	Qualitätsfaktor
ROC	Receiver Operating Characteristic
SE	Skalierungsexponent

TN	True Negative
TP	True Positive

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

Aggarwal, C. C. (2015). *Data Mining: The Textbook*. Springer International Publishing.

Arellano, A., Carney, E. & Austin, M. A. (2015). Natural Language Processing of Textual Requirements. In *Icons 2015 : The tenth international conference on systems.*

Berg, B., Knott, P. & Sandhaus, G. (2014). *Hybride Softwareentwicklung: Das Beste aus klassischen und agilen Methoden in einem Modell vereint*. Springer Berlin Heidelberg.

Bird, S., Klein, E. & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

Bleek, W.-G. & Wolf, H. (2011). *Agile Softwareentwicklung: Werte, Konzepte und Methoden*. dpunkt.verlag.

Boehm, B. & Turner, R. (2009). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley.

Brandstäter, J. (2013). *Agile IT-Projekte erfolgreich gestalten: Risikomanagement als ERgänzung zu Scrum*. Springer Vieweg.

Brink, H., Richards, J. W. & Fetherolf, M. (2017). *Real-World Machine Learning*. Manning Publications Co.

Brooks, F. P., Jr. (1987, April). No Silver Bullet - Essence and Accident in Software Engineering. *Computer Magazine*, *20* (4), 10–19.

Bühner, M. & Ziegler, M. (2009). *Statistik für Psychologen und Sozialwissenschaftler*. Pearson Education.

Celebi, M. E. & Aydin, K. (2016). *Unsupervised Learning Algorithms*. Springer International Publishing.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C. & Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data mining guide [Software-Handbuch].

Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T., Ghose, A. & Menzies, T. (2016). *A deep learning model for estimating story points.*

Chopra, D., Joshi, N. & Mathur, I. (2016). *Mastering Natural Language Processing with Python*. Packt Publishing.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley.

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. Pearson Education.

Combs, A. T. (2016). *Python Machine Learning Blueprints: Intuitive data projects you can relate to*. Packt Publishing.

Diederichs, H. (2005). *Komplexitätsreduktion in der Softwareentwicklung: Ein systemtheoretischer Ansatz*. Books on Demand.

Documentation of scikit-learn 0.19.1. (2017). Website. Zugriff am 18.11.2017 auf `http://scikit-learn.org/stable/documentation.html`

Dumke, R., Schmietendorf, A., Seufert, M. & Wille, C. (2014). *Handbuch der Softwareumfangsmessung und Aufwandschätzung*. Logos Verlag Berlin.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996). Knowledge Discovery and Data Mining: Towards a Unifying Framework. *Proceedings of the second inernational conference on knowledge discovery and data mining*.

Feyhl, A. (2013). *Management und Controlling von Softwareprojekten: Software wirtschaftlich auswählen, entwickeln, einsetzen und nutzen* (2. Aufl.). Gabler Verlag.

Gloger, B. (2014). *Wie schätzt man agilen Projekten - oder wieso Scrum-Projekte erfolgreicher sind*. Hanser Verlag.

Haslwanter, T. (2016). *An Introduction to Statistics with Python: With Applications in the Life Sciences*. Springer International Publishing.

Hedderich, J. & Sachs, L. (2015). *Angewandte Statistik: Methodensammlung mit R*. Springer-Verlag.

Hedges, L. V. (1981). Distribution Theory for Glass's Estimator of Effect size and Related Estimators. *Journal of Educational and Behavioral Statistics*.

Hummel, O. (2011). *Aufwandsschätzungen in der Software- und Systementwicklung kompakt*. Spektrum Akademischer Verlag.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer Science & Business Media.

Kao, A. & Poteet, S. R. (2007). *Natural Language Processing and Text Mining*. Springer Science & Business Media.

Kubat, M. (2005). *An Introduction to Machine Learning*. Springer International Publishing.

Leffingwell, D. (2010). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Pearson Education.

Luger, G. F. & Stubblefield, W. A. (1993). *Artificial intelligence: structures and strategies for complex problem solving*. Benjamin/Cummings Pub. Co.

Manifesto for Agile Software Development. (2001). Website. Zugriff am 15.07.2017 auf `http://agilemanifesto.org/`

Marbán, Ó., Mariscal, G. & Segovia, J. (2009). *Data Mining and Knowledge Discovery in Real Life Applications* (J. Ponce & A. Karahoca, Hrsg.). I-Tech Education and Publishing.

McElreath, R. (2016). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. CRC Press.

Menzies, T., Chen, Z., Hihn, J. & Lum, K. (2006). Selecting Best Practices for Effort Estimation. *IEEE Transactions on software engineering*, *32* (11).

Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.

Moll, K.-R., Broy, M., Pizka, M., Seifert, T., Bergner, K. & Rausch, A. (2004, Oktober). Erfolgreiches Management von Softwareprojekten. *Informatik-Spectrum*, *27* (5), 419–432.

Müller, A. C. & Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

NLTK 3.2.5 documentation. (2017). Website. Zugriff am 04.11.2017 auf `http://www.nltk.org/`

Palmeira, C., Chaves, R., Cavalcante, H. & Favero, E. (2012, June). A Requirements Elicitation and Analysis Aided by Text Mining. *IJCSNS International Journal of Computer Science and Network Security*, *12* (6).

Patterson, J. & Gibson, A. (2017). *Deep Learning*. O'Reilly Media.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Piatetsky-Shapiro, G. & Frawley, W. (1991). *Knowledge Discovery in Databases*. MIT Press Cambridge.

Pilz, F. (2011). Methoden zur Aufwandsschätzung von Softwareprojekten und deren Zuverlässigkeit. In S. Auer, T. Riechert & J. Schmidt (Hrsg.), *SKIL 2011 Studentenkonferenz Informatik Leipzig*.

Pohl, K. & Rupp, C. (2015). *Basiswissen Requirements Engineering* (4. Aufl.). dpunkt.verlag.

Preußig, J. (2015). *Agiles Projektmanagement: Scrum, Use Cases, Task Boards & Co.* Haufe Lexware.

Ramasubramanian, K. & Sing, A. (2016). *Machine Learning Using R*. Apress.

Raschka, S. (2015). *Python Machine Learning: Unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics*. Packt Publishing.

RDocumentation. (2017). Website. Zugriff am 18.11.2017 auf `https://www.rdocumentation.org/`

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering* (S. 328–338).

RStudio. (2016). Website. Zugriff am 18.11.2017 auf `https://www.rstudio.com/`

Russell, S. J. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

Sarkar, D. (2016). *Text Analytics with Python: Text Analytics with Python A Practical Real-World Approach to Gaining Actionable Insights from your Data*. apress.

Schwaber, K. (2004). *Agile Project Management with Scrum*. Pearson Education.

Shafique, U. & Qaiser, H. (2014). A Comparative Study of Data Mining Process Models (KDD, CRISP-DM and SEMMA). *International Journal of Innovation and Scientific Research*, *12* (1), 217–222.

Sharma, M. & Fotedar, N. (2014). Software Effort Estimation with Data Mining Techniques - A Review. *International Journal Of Engineering Sciences & Research Technology*.

Shepperd, M. & MacDonell, S. G. (2012). Evaluating prediction systems in software project estimation. *Information and Software Technology*, *54* (8), 820–827.

Sneed, H. (2014). Aufwandsschätzung in IT-Projekten. In E. Tiemeyer (Hrsg.), *Handbuch it-projekt-management* (2. Aufl., S. 273–314). Hanser Verlag.

Soria, E., Martín-Guerrero, J. D., Martinez, M., Magdalena, R. & Serrano, A. J. (2009). *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques*. IGI Global.

Standish Group 2015 Chaos Report. (2015). Zugriff am 29.04.2017 auf `https://www.infoq.com/articles/standish-chaos-2015`

Sugiyama, M. (2015). *Statistical Reinforcement Learning: Modern Machine Learning Approaches*. CRC Press.

Titanic: Machine Learning from Disaster. (2012). Website. Zugriff am 02.10.2017 auf `https://www.kaggle.com/c/titanic/data`

Trendowicz, A. & Jeffery, R. (2014). *Software Project Effort Estimation: Foundations and Best Practice Guidelines for Success*. Springer International Publishing.

Weiss, J. & Tan, E. (2004, November). *Klassische vs. agile Methoden der Softwareentwicklung.* Vortrag.

Wieczorek, I. (2001). Verbesserte Software Kostenschätzung - Eine robuste und interpretierbare Modellierungsmethode und deren umfassende empirische Überprüfung. In *Ausgezeichnete Informatikdissertationen.*

Wirdemann, R. (2017). *Scrum mit User Stories*. Hanser Verlag.

Wirth, R. & Hipp, J. (1999). *CRISP-DM: Towards a Standard Process Model for Data Mining* (Bericht). DaimlerChrysler Research & Technology and Wilhelm-Schickard-Institute, University of Tübingen.

Witten, I. H. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques* (2. Aufl.). Elsevier Science.

Wordnet: A lexical database for English. (2017). Website. Zugriff am 06.11.2017 auf `https://wordnet.princeton.edu/`