HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Frau
Denise Kreiseler

Automatisches Klassifizieren von textuellen Fehlermeldungen

Mittweida, 2022

Fakultät Angewandte Computer- und
Biowissenschaften

BACHELORARBEIT

Automatisches Klassifizieren von textuellen Fehlermeldungen

Autor:
Frau

Denise Kreiseler

Studiengang:
IT-Forensik/Cybercrime

Seminargruppe:
CC18w1-B

Erstprüfer:
Prof. Dr. rer. Nat. Dirk Labudde

Zweitprüfer:
B.Sc. Martin Klöden

Einreichung:
Wernau, 07.08.2022

Verteidigung/Bewertung:
Wernau, 2022

Faculty Applied Computer Sciences and Biosciences

BACHELOR THESIS

Automated classification of textual error messages

author:
Ms.

Denise Kreiseler

course of studies:
IT-Forensic/Cyber Crime

seminar group:
CC18w1-B

first examiner:
Prof. Dr. rer. Nat. Dirk Labudde

second examiner:
B.Sc. Martin Klöden

submission:
Wernau, 07.08.2022

defence/ evaluation:
Wernau, 2022

Bibliografische Beschreibung:

Kreiseler, Denise:
Automatisches Klassifizieren von textuellen Fehlermeldungen, 55 Seiten, 8 Seiten Verzeichnisse, 38 Seiten Inhalt, 5 Seiten Anlagen,
Mittweida, Hochschule Mittweida, Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2022

Referat:

Die vorliegende Arbeit dient als Grundlage zur Umsetzung für eine automatisierte Klassifizierung von textuellen Fehlermeldungen. Das Hauptziel ist ein grundlegendes Verständnis für die Herangehensweise zum Aufbau eines maschinellen Lernsystems zu erreichen. Es werden verschiedene Arten des maschinellen Lernens erläutert. Auswahl und Aufbau eines Lernmodells werden von unterschiedlichen Seiten beleuchtet, um einen Überblick der einzelnen Schritte zu gewinnen. Zur Gewährleistung eines praktischen Lösungsansatz wurden bereits erste Tests mit einem ausgewählten Lernmodell durchgeführt.

As a basic elaboration for automated classification of textual error messages this thesis will show what steps are needed to set up a learning system. A short overview about different types of machine learning will introduce into the general topic. A closer look will be taken to the evaluation of an example learning system including performing of several tests and result analyses.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation und Zielsetzung

Automatisieren, aus dem altgriechischen „sich selbstbewegend“, keine weiteren menschlichen Ressourcen benötigen und Aufgaben selbstständig lösen, manuelle Abläufe durch automatisierte Prozesse schneller, akkurater und mit weniger Mitarbeiterbindung durchzuführen, das ist der Geist der Zeit. Mitarbeiter, die bisher diese manuelle Tätigkeit durchgeführt haben bekommen Raum für andere Tätigkeiten. Ebenso gilt die Annahme, dass automatisierte Prozesse schneller genauer sind. Ein psychologischer Faktor, der hinzukommt, ist die Schuldzuweisung bei fehlerhaften manuell durchgeführten Arbeitsprozessen. Mitarbeiter werden demotiviert durch etwaige Schuldzuweisungen. Auch für die hier vorliegende Abschlussarbeit, sind das einige Beweggründe gewesen, das Thema „Automatisierte Klassifikation“ aufzubereiten.

Eine kurze Einleitung erläutert die aktuelle Situation sowie die fachliche und technische Arbeitsumgebung. Im Entwicklungsverwaltungsprogramm werden heute alle Arbeitselemente angelegt. Eine bestimmte Art von Arbeitselement zeichnet sich dadurch aus, dass im Anhang eine Textdatei vorhanden ist. Zusätzlich werden diese Arbeitselemente mit einer Kennzeichnung „Exception“ versehen, um sie leichter durch Abfragen herauszufiltern. In den angehängten Textdateien der Arbeitselemente sind die Informationen enthalten, welche zur Klassifikation genutzt werden können. Aktuell werden die Textdateien manuell geöffnet, gelesen und dann nach bestem Wissen des Mitarbeiters in die entsprechenden Teambereiche verschoben. Intern werden heute vier verschiedene Teambereiche unterschieden:

- PCSF - Platform Component und Server Framework
- BOM - Bill of Material
- CALC – Profitability
- TECH - Technology

Alle Teambereiche werden weiter untergliedert in detaillierte Anwendungsfunktionen der Module. Abbildung 1 veranschaulicht diese Untergliederung der Teambereiche. Für das Einordnen der Fehlermeldung sind die Teambereiche allerdings ausreichend.

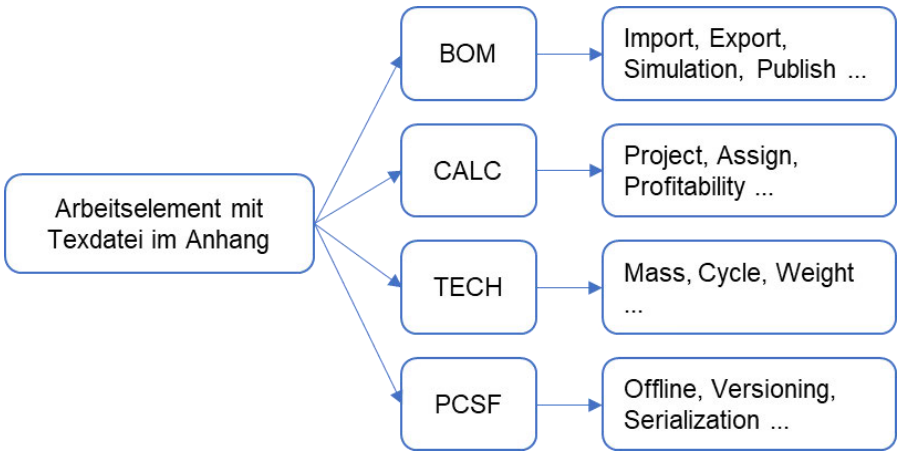

Abbildung 1 – Zuordnungsmöglichkeiten eines Arbeitselementes in die internen Teambereiche und deren Unterbereiche

Zielsetzung der vorliegenden Abschlussarbeit ist es, ein Lernmodell aufzubauen und Versuche mit dem Lernmodell und unterschiedlichen Einstellungen durchzuführen. Es soll ein Grundverständnis vermittelt werden, wie sich diese verschiedenen Einstellungen des Lernmodells auf das Lernergebnis auswirken. Weiterhin soll auf eventuell neu gefundene Problemstellungen hingewiesen werden.

Dazu werden zunächst verschiedene Arten des maschinellen Lernens und deren Vorgehensweise betrachtet. Es werden einzelne Schritte zum Aufbau eines Lernmodells erläutert und ausgewählte Möglichkeiten zur Textanalyse vorgestellt. Dabei geht es um Möglichkeiten zur Datenbereinigung und Datenvorbereitung. Ebenso wird anhand von Beispielen die Auswirkung der Merkmalsauswahl dargestellt.

Bei der Lernmodelldefinition wird erläutert welche Kriterien zur Auswahl eines geeigneten Modells zu beachten sind. Ebenfalls findet ein kurzer Exkurs in die Bestandteile eines Lernalgorithmus statt. Dabei ist es nicht das Ziel der vorliegenden Abschlussarbeit die einzelnen Algorithmen und mathematischen Formeln im Detail zu beleuchten oder zu beweisen. Zum besseren Verständnis werden dennoch verschiedene Grundsätze erläutert. Für ein tiefgehendes Verständnis ist jedoch selbstständig auf mathematische Fachliteratur zurückzugreifen.

Eine Versuchsreihe soll das ausgewählte Lernmodell testen. Die Ergebnisse werden verglichen und die dafür genutzten Parameter erläutert. Dabei soll die vorhandene Testumgebung und das aufgebaute Netz durchaus bestehen bleiben und weiter genutzt werden. Im Ausblick werden potentielle Probleme oder Verbesserungsvorschläge aufgezeigt.

2 Automatisierung durch maschinelles Lernen

Hinter maschinellem Lernen steht die Idee, auf Daten basierend eine Aufgabe zu erlernen. Ein maschinell lernendes System lernt aus Beispieldaten. Dabei lernt es nicht auswendig, sondern kann Muster und Gesetzmäßigkeiten erkennen, sodass es auch unbekannte Daten einordnen kann. [1] Lernalgorithmen sind die Grundlage eines maschinell lernenden Systems und werden innerhalb eines Lernmodells abgebildet. Einen Lernalgorithmus auszuwählen, bedarf einiger Fragestellungen vorab:

Wie nachvollziehbar muss mein Modell sein? Das heißt, kann ich mit einem Black-Box Modell arbeiten oder müssen alle getroffenen Vorhersagen genau erläutert werden können. In einem Black-Box Modell bleibt der genaue Aufbau oder Ablauf oft unbekannt. Welche Datenmengen sind zu erwarten? Muss mein Lernalgorithmus mit vielen verschiedenen Merkmalen zurechtkommen? Wie wichtig sind die Geschwindigkeit des Trainings und der Vorhersage? All diese Fragen beeinflussen die Auswahl des passenden Lernalgorithmus. [2] Auch wenn die Lernalgorithmen entscheidend für das Lernmodell sind, gilt es oftmals viel Zeit in die Datenanalyse zu investieren. Denn entscheidend für die Auswahl des Lernmodells ist auch die Art des maschinellen Lernens. Hierfür müssen die Datengrundlage und die Fragestellung, die beantwortet werden soll, verstanden sein. Einen Überblick über verschiedene Anwendungsgebiete für drei verschiedene Arten des maschinellen Lernens zeigt Abbildung 2.

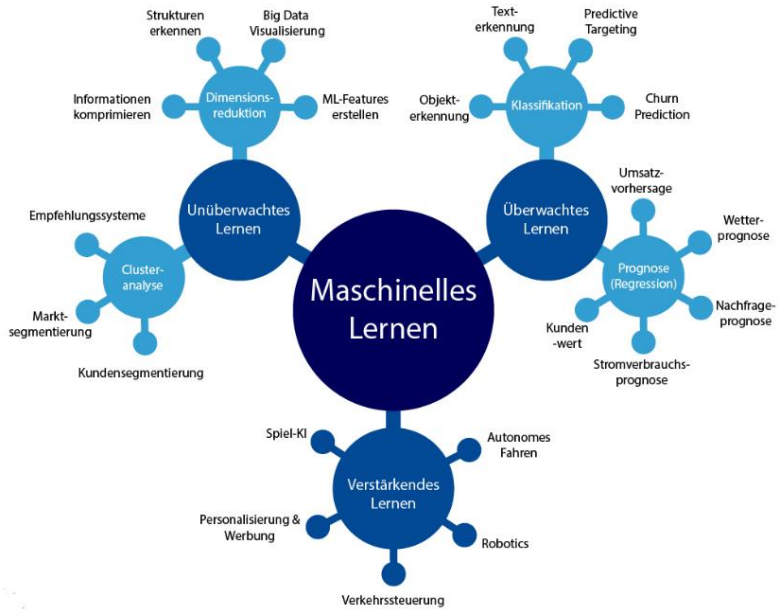

Abbildung 2 - Anwendungsbeispiele nach Arten des maschinellen Lernens [3]

2.1 Arten des maschinellen Lernens

Es gibt mehrere Arten des maschinellen Lernens. Dazu gehören beispielsweise das überwachte, das unüberwachte, oder auch das bestärkende Lernen. Folgend werden Unterschiede und ein allgemeines Verständnis für diese drei Ansätze herausgestellt. Auf weitere Arten des maschinellen Lernens wird während dieser Abschlussarbeit nicht weiter eingegangen.

2.1.1 Überwachtes Lernen

Beim überwachten Lernen gibt es bereits gekennzeichnete Daten. Das bedeutet, dass für diese Daten die Eingaben und Ausgaben bekannt sind. Es ist Ziel des Algorithmus Zusammenhänge zwischen den Eingaben und der Ausgabe festzustellen beziehungsweise zu modellieren.

Es gibt zwei Aufgabenstellungen, die Klassifikation und die Regression, für die überwachtes Lernen bevorzugt angewendet wird. Grundsätzlich zeichnet sich die Klassifikation dadurch aus, dass die Ausgabe in bereits vorgegebene Klassen vorgenommen wird. Hierbei wird nach binärer und Multi-Class-Klassifikation unterschieden. Im Gegensatz zur binären Klassifikation, welche genau zwei Klassen für die Einteilung zur Verfügung hat, sind bei der Multi-Class-Klassifikation mehrere Klassen möglich. Bekannte Lernalgorithmen für das Problem der Klassifikation sind: [4]

- Logistische Regression
- Support Vektor Maschinen
- Nächste-Nachbarn-Klassifikation (engl.: k-nearest-neighbor)
- Entscheidungsbäume
- Naiver Bayes-Klassifikator

Überwachtes Lernen wird ebenfalls für die Aufgabenstellung der Regression genutzt. Hier wird eine numerische Vorhersage erwartet, welche sich Anhand der bereits bekannten Datensätze und deren Zusammenhänge vorhersagen lässt. Bekannte Lernalgorithmen für das Problem der Regression sind: [3]

- Entscheidungsbäume
- Lineare Regression

Alle Algorithmen bringen Ihre eigenen Vor- und Nachteile zur Nutzbarkeit mit. So ist der k-nearest-neighbor-Algorithmus nicht besonders performant, weil er alle Datenpunkte hinsichtlich Ihrer Entfernung zu einem bestimmten Datenpunkt misst, was sich bei einer sehr großen Datenmenge als Nachteil erweist. Oder es sind Entscheidungsbäume die bei kleinen Anpassungen in den Trainingsdaten zu großen Änderungen im Baum und den Ergebnissen neigen. Somit muss bewusst sein, dass die Auswahl der Algorithmen zum vorhanden zu lösenden Problem passen sollte. In der Praxis bleibt jedoch oft nicht die Zeit

sich mit allen verfügbaren Algorithmen auseinanderzusetzen oder Sie zu testen. Dies führt dazu, dass oftmals nicht alle Kriterien ideal erfüllt werden.

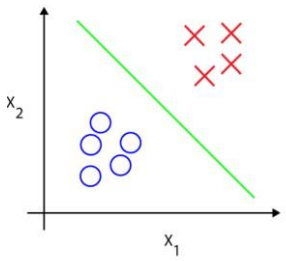

Abbildung 3 - Darstellung der Einteilung in bereits vorgegebene Klassen beim überwachten Lernen [3]

2.1.2 Unüberwachtes Lernen

Beim unüberwachten Lernen ist die vorhandene Datenmenge nicht gekennzeichnet. Ziel des Lernalgorithmus ist es, ein Modell zu entwickeln, welches Merkmale und Ähnlichkeiten ermittelt. Das Ergebnis wird somit durch den Algorithmus bestimmt. [2] Bekannte Lernverfahren samt Beispielen für Lernalgorithmen sind:

- Clustering (k-means, DB-Scan)
- Assoziationen (Eclat-Algorithmus, Apriori-Algorithmus)
- Dimensionsreduktion (Principal Components Analysis, k-nearest-neighbor)

Der geringe manuelle Aufwand zur Datenaufbereitung ist ein Vorteil des unüberwachten Lernens, da die Datenmenge nicht vorab gekennzeichnet ist. Somit wird unüberwachtes Lernen beispielsweise genutzt für das Herausfinden von ähnlichen Merkmalen. Beispiele hierfür sind Clustering von Kundenmerkmalen oder Vorschlagsunterstützungen auf Webseiten (Wer Produkt x kauft, kauft auch Produkt y) [2].

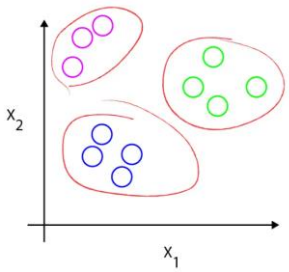

Abbildung 4 - Darstellung der gefunden Clusterzuordnung beim unüberwachten Lernen [3]

2.1.3 Bestärkendes Lernen

Beim bestärkendem Lernen ist das Erlernen einer Vorschrift das Ziel des Lernalgorithmus. Eine Maschine befindet sich in einer Umgebung und führt verschiedene Aktionen aus. Auf manche Aktionen folgen Belohnungen und auf andere Strafen. Somit wird selbstständig erlernt wann die Belohnungsfunktion ihr Maximum ergibt. Das Ergebnis ist eine erlernte Vorschrift, bei der möglichst hoch belohnt wird. Für diesen Ansatz sind im Vorfeld keine Daten notwendig. Während der Trainingsabläufe werden die notwendigen Daten generiert und in angelegten Simulationsszenarien durchlaufen. [2] Lernalgorithmen des bestärkenden Lernens teilen sich grob in modellfreie und modellbasierte Algorithmen ein. Beispiele für modellfreie Algorithmen sind die Monte-Carlo-Methode oder Temporal Difference Learning. MuZero von Deepmind kann genannt werden, als Beispiel für einen modellbasierten Algorithmus. Das Basisverfahren des bestärkenden Lernens ist nichts anderes als das sogenannte Versuch-und-Irrtum (engl.: Trial-and-Error) Verfahren. Das bedeutet, solange auszuprobieren, bis die gewünschte Belohnung eintritt. Abbildung 5 zeigt ein typisches Szenario für bestärkendes Lernen. [5]

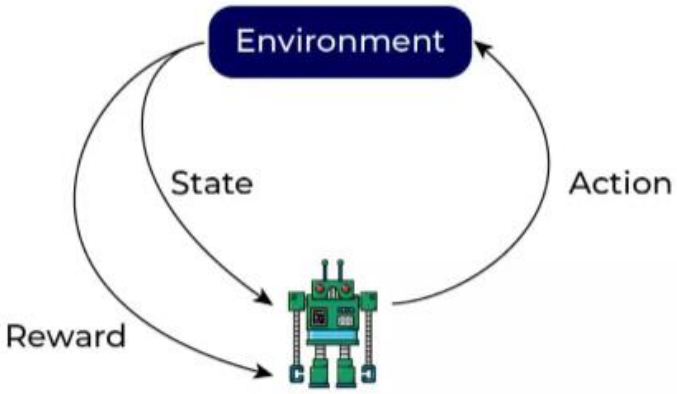

Abbildung 5 - Darstellung eines typischen Szenarios für bestärkendes Lernen [5]

2.2 Zusammenfassung

Die Methoden des überwachten und unüberwachten Lernens werden mit Ihren Unterschieden in Tabelle 1 zusammenfassend dar- und gegenübergestellt. Hierbei wird die allgemeine Vorgehensweise, die Anzahl der Merkmale und die Möglichkeit eines Echtzeiteinsatzes verglichen.

Tabelle 1 - Vergleich der Eigenschaften von überwachtem und unüberwachtem Lernen [3]

	Überwachtes Lernen	**Unüberwachtes Lernen**
Vorgehensweise	Eingabe und Ausgabe sind bekannt	Nur die Eingabe ist bekannt
Anzahl der Merkmale	Bekannt	Unbekannt
Echtzeiteinsatz	Das Lernen passiert vor dem Einsatz	Lernen in Echtzeit möglich
Anwendungsbeispiele	Stromverbrauch vorhersagen, Kaufwahrscheinlichkeiten erkennen, Spam Erkennung bei Emails	Kundenmerkmale clustern, Große Datensätze in verschieden Cluster reduzieren, amazon personalize[1] für das personalisierte Einkaufserlebnis

Wie aus Tabelle 1 zu erkennen ist und in Kapitel 2.1.1 beschrieben, befindet sich die Aufgabenstellung zur Klassifikation im Bereich des überwachten Lernens. Für die Aufgabenstellung der vorliegenden Abschlussarbeit liegen für die textuellen Fehlermeldungen bereits die Klassen, welche ausgegeben werden, vor. Die Anzahl der Merkmale wird festgelegt und als Parameter in das Lernmodell übergeben. Im Rahmen der vorliegenden Aufgabenstellung wird demnach das Prinzip des überwachten Lernens angewandt. Tabelle 2 zeigt den direkten Vergleich zwischen dem bestärkenden und

[1] https://aws.amazon.com/de/personalize/ - Erläuterung der Funktionalität des personalisierten Einkaufs von Amazon

überwachten Lernen anhand ausgewählter Kriterien wie die Vorgehensweise beim Training oder auch Abhängigkeiten der getroffenen Entscheidungen.

Tabelle 2 - Vergleich der Eigenschaften von bestärkendem und überwachtem Lernen [3]

	Überwachtes Lernen	**Bestärkendes Lernen**
Stil	Entscheidungen zu Beginn der Eingabe	Sequentielle Entscheidungen
Training	Mit Beispieldaten	Durch Interaktion mit einer Umgebung
Abhängigkeiten	Entscheidungen sind unabhängig voneinander	Entscheidungen sind abhängig voneinander
Anwendungsbeispiele	Objekterkennung	Schachspiel

Am Beispiel der textuellen Fehlermeldungen veranschaulicht sind für diese die zu erzielenden Klassen bereits bekannt. Es werden die Klassen PCSF, CALC, BOMAPI und TECH gekennzeichnet. Eine bereits gekennzeichnete Datenmenge ist ebenfalls vorhanden, da die Klassifizierung bisher manuell durchgeführt wurde. Die Textdateien stellen in diesem Beispiel die Eingabe dar. Der Algorithmus wird aus den bisher gekennzeichneten Daten Muster und Zusammenhänge erkennen und diese auf neue Daten anwenden und somit diese klassifizieren. In den weiteren Kapiteln werden die einzelnen Schritte für die Herangehensweise zum Aufsetzen eines überwachten Lernsystems erläutert.

3 Aufbau eines überwachten Lernsystems

Die weiteren Ausführungen zur Vorbereitung und zum Aufbau eines Lernsystems beziehen sich auf das überwachte Lernen. Abbildung 6 zeigt einzelne Schritte zum Aufbau eines überwachten Lernsystems. Diese werden in den folgenden Kapiteln einzeln erläutert.

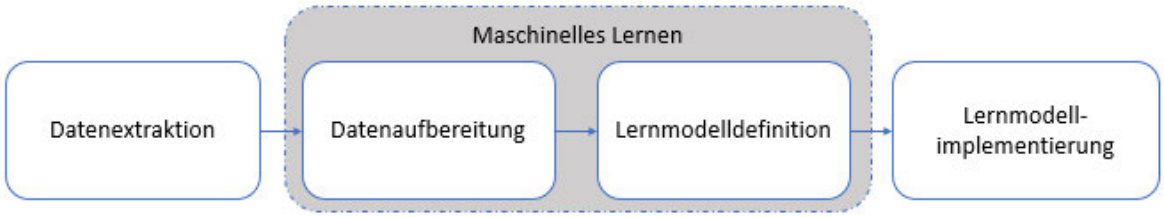

Abbildung 6 – typischer Ablauf für ein überwachtes Lernsystem, adaptiert nach [6]

3.1 Datenextraktion

In diesem ersten Schritt geht es um das Gewinnen und Sammeln von Datensätzen. Es gilt möglichst viele Datensätze zu gewinnen, um diese später für das Lernmodell zu nutzen. Für die vorliegenden textuellen Fehlermeldung werden die Arbeitselemente inklusive einer Textdatei im Anhang im Entwicklungsverwaltungssystem angelegt. Um die Textdateien zu extrahieren, wird ein Anwendungsprogramm genutzt. Alle Anhänge von Arbeitselementen, die in einer definierten Abfrage enthalten sind, werden in eine lokale Verzeichnisstruktur heruntergeladen. Da die Daten im Entwicklungsverwaltungssystem bereits klassifiziert sind, wird diese Information ebenfalls abgefragt. Die Dateien werden in einer Ordnerstruktur abgelegt, welche den Klassen entspricht. Das Programm ist bereits integriert in die Entwicklungslandschaft und soll zukünftig automatisiert laufen, um immer wieder neue Daten in das Lernmodell zu geben. Um etwaigen Ressourcenproblemen aus dem Weg zu gehen, muss ebenfalls darauf geachtet werden, ob bereits extrahierte Daten nach der Initialisierung erneut angefragt werden. Zum Beispiel können fortfolgend nur noch Dateien von neu angelegten Arbeitselementen geladen werden. Durch dieses inkrementelle Laden kann verhindert werden, dass bei jedem Zugriff alle Ergebnisse erneut heruntergeladen werden.

Ebenfalls können zur Datenextraktion auch weitere Möglichkeiten, wie zum Beispiel der direkte Zugriff auf die Datenbank der Arbeitselemente, evaluiert werden. Hierzu muss die Datenbankstruktur und speziell das Referenzieren der Anhänge untersucht werden.

Optionen für weitere Steigerungen der Performance sollten im Nachgang evaluiert werden. Im Rahmen der vorliegenden Abschlussarbeit wurden die Dateien mit dem bereits bestehenden Programm erfasst, welches im Anhang Teil 2 dargestellt ist. Informationen die einen Rückschluss auf die lokale Infrastruktur zulassen wurden ausgelassen.

3.2 Datenaufbereitung

Die Einordnung von Text, das Strukturieren von Daten, dass Akquirieren von Wissen aus einer Menge von Informationen kann mit Hilfe von Text Mining vorgenommen werden. [7] Text Mining hilft beim Klassifizieren von digitalen Daten und kann ebenfalls automatisiert werden. Es gibt eine Reihe von Methoden und Algorithmen, um Texte zu strukturieren und zu analysieren

Für den Schritt Datenaufbereitung werden einige ausgewählte Möglichkeiten zur Textanalyse und Datenbereinigung vorgestellt. Ebenso wird die Herausforderung zur Festlegung von geeigneten Merkmalen dargestellt. Die Datenaufbereitung ist in der Praxis häufig der zeitaufwendigste Schritt während des Erstellens eines maschinellen Lernsystems.

3.2.1 Textanalysemethoden

Mit Datenbereinigung ist in diesem Kontext gemeint, möglichst viel Inhalt aus den textuellen Fehlermeldungen herauszufiltern. Oftmals werden, schon automatisiert, sogenannte Stoppwörter aus Texten herausgefiltert. Beispiele für Gruppen der Stoppwörter sind bestimmte und unbestimmte Artikel, Präpositionen oder Konjunktionen. Ziel ist es durch die Datenbereinigung weniger verrauschte Daten zubekommen. Unter Datenrauschen versteht man unnötige, irrelevante Daten. Je mehr solcher statistisch auffälligen Wortformen bereinigt werden, desto mehr themenbezogene Daten können in das Lernmodel eingespeist werden. Ebenso wird verhindert, dass beim Training des Lernmodells falsche oder unwichtige Bezüge mit den irrelevanten Daten erstellt werden. Dadurch wird potentiell die Vorhersagegenauigkeit des Lernmodells verbessert. Es ist auch möglich diese im Lernmodel mit einer sehr niedrigen Relevanz zu versehen. Es benötigt eine gute Datenanalyse und teils auch fachliche Kenntnisse der vorliegenden Daten, um zu entscheiden, was irrelevant für das Lernmodell ist.

In Abbildung 7 wird eine solche Häufigkeit von Wörtern nach einer n-gramm Textanalyse in einem Balkendiagramm dargestellt. Abgebildet sind die am häufigsten vorkommenden Wörter nach rechts absteigend sortiert.

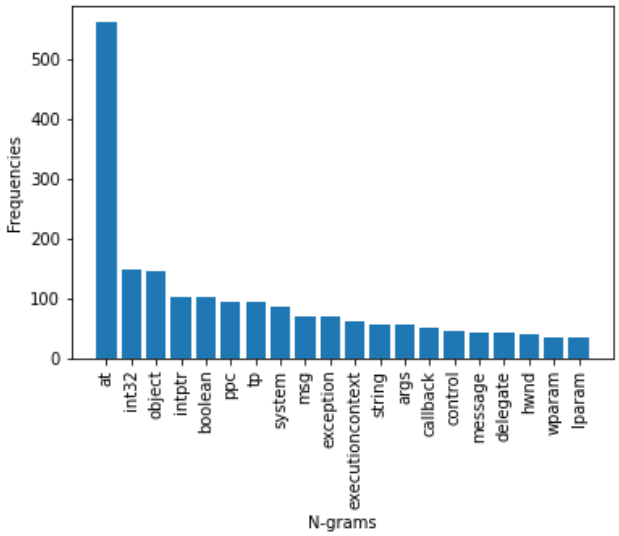

Abbildung 7 – Balkendiagramm mit den am häufigsten vorkommenden Wörtern in beispielhaft untersuchten textuellen Fehlermeldungen

N-Gramm ist das Ergebnis der Zerlegung eines Textes in einzelne Fragmente [8]. Dabei können Fragmente in Buchstaben, Sätzen oder Wörtern eingeteilt werden. N-Gramm Analysen werden oftmals zur statistischen Auswertung herangezogen [8]. Ebenfalls werden N-Gramm Analysen in Programmen wie der Rechtschreibprüfung oder in forensischen Auswertungswerkzeugen angewandt. Es wird unterschieden zwischen Unigramms, Bigramms, Trigramms bis hin zu Multigramms. Die Art des N-Gramms entscheidet darüber wie viele Buchstaben, Sätze oder Wörter als ein Ergebnis ausgegeben werden. Abbildung 8**Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt eine Übersicht über einige Arten von N-Gramms mit dazugehörigem Beispiel.

N-Gramm-Name	N	Beispiel
Monogramm	1	A
Bigramm	2	AB
Trigramm	3	UNO
Tetragramm	4	HAUS
Pentagramm	5	HEUTE
Hexagramm	6	SCHIRM
Heptagramm	7	TELEFON
Oktogramm	8	COMPUTER
…	…	…
Multigramm	17	BEOBACHTUNGSLISTE

Abbildung 8 - Allgemeine Darstellung Arten von N-Gramms [8]

Eine weitere Vorgehensweise zur Textanalyse und Datenaufbereitung ist das sogenannte Clustering. Clustering wird beispielsweise verwendet um ähnliche Wortformen oder Wortstämme zu analysieren. Als fachliche Betrachtung der textuellen Fehlermeldungen können beispielsweise Wörter mit den Grundformen „Kalkul“, „Calcul“, „kalk“ oder „calc“ ausgeschlossen oder mit einer sehr niedrigen Relevanz bewertet werden. Da es sich im vorliegenden Beispiel um eine Kalkulationssoftware handelt, sind Wörter mit diesem Wortstamm potentiell in jedem Modul vorhanden. Somit sind diese Wörter als Merkmale für das Lernmodell wenig geeignet.

Clustering kann auf Sätze, Wörter oder auch Buchstaben angewandt werden. Allgemein definiert ist es ein Verfahren zur Entdeckung von ähnlichen Strukturen in den vorliegenden Datenbeständen. Die gefundenen Gruppen von ähnlichen Objekten werden als Cluster bezeichnet, die Gruppenzuordnung als Clustering. [9] Allerdings muss für das Clustering vorab ein Algorithmus definiert werden, der die ähnlichen Objekte clustert. Dafür benötigt es die Definition eines Ähnlichkeitsmaßes. Das heißt, es muss ein Maß festgelegt werden, wie sehr sich zwei Objekte gleichen. Hierfür gibt es gängige mathematische Ansätze, wie die euklidische Distanz oder Skalarprodukte.

Ein anderer Ansatz der Datenaufbereitung ist die Musteranalyse von Wörtern wie beispielsweise das Nutzen von regulären Ausdrücken. Reguläre Ausdrücke werden als eigene „Kunstsprache“ verstanden und kommen in jeder Programmiersprache vor, was ebenfalls nicht wegzudenken ist beim Einsatz von Spracherkennung und Text Mining [7]. Reguläre Ausdrücke bestehen aus Operatoren und Atomen. Atome stellen Buchstaben, Wörter oder Wortformen dar. Operatoren können Atome verknüpfen. Sogenannte Wildcards, sind ein mächtiges Werkzeug der regulären Ausdrücke und bringen in Verbindung mit den verknüpften Atomen und Operatoren oft hilfreiche Suchmasken hervor. Mit dem Begriff Wildcards, sind Sonderzeichen gemeint. Somit lässt sich einem Text mit der Kombination aus Punkt und Asterisk „ .* „, nach beliebigen Zeichen durchsuchen. [7]

Folgend einige nützliche Beispiele bezogen auf die zu untersuchenden textuellen Fehlermeldungen: (mass*)|(cycle*) – alle Ausdrücke die mit „mass“ oder mit „cycle“ anfangen werden gefunden. Für die vorhanden Module Massenrechner und Zykluszeitrechner sind das Kombinationen, die alle damit verbundenen Bereiche finden würden. Im Bereich der Wirtschaftlichkeitsrechnung kann nach Wortformen wie *profitability* analysiert werden. Dabei wird ignoriert, ob davor oder danach weitere Zeichen vorhanden sind.

Eine ebenfalls interessante Option zur Datenaufbereitung ist das Filtern, sodass zum Beispiel nur Nomina zugelassen werden und in das Lernmodell als Merkmale eingehen. Dies wird bevorzugt im Bereich von Fachtexten angewendet. Da der Kontext im vorliegenden Beispiel Fachvokabular ist, sind andere Wortarten oft von keiner großen Relevanz. Es würde ausreichen, Modulwörter wie QuickEditor zu analysieren und Verben wie save oder load herauszufiltern. Nach Betrachtung der textuellen Fehlermeldungen hat sich allerdings herausgestellt, dass diese Art des Filterns für die vorliegenden Texte keinen

Mehrwert bringt. Gegen diese Möglichkeit spricht, dass in der Softwareentwicklung oft Funktionen und Klassen Großgeschrieben und zusammengeschrieben werden. Beispielhaft wird dies dargestellt an dem Aufruf „LoadServiceRelationExtensions“, welcher kein relevantes fachliches Nomina darstellt.

Zum Abschnitt Textanalysemethoden kann abschließend gesagt werden, dass es eine Vielzahl an Möglichkeiten gibt, die vorliegenden Daten zu bereinigen und zu analysieren bevor Sie in das Lernmodell gegeben werden. Einige werden bereits in manchem Lernmodellen berücksichtigt. Alle Methoden sollten vorab nach Aufwand und Nutzen abgewogen werden. Eine gute Datenbereinigung durch Textanalyse kann unter anderem zu einer besseren Performance des Lernmodells und höheren Genauigkeit des Lernergebnisses führen.

3.2.2 Auswahl von Klassifikationsmerkmalen

Oft ist Fachkenntnis notwendig, um gute Merkmale festzulegen. Es ist durchaus empfohlen nicht alle Merkmale eines Dokumentes zu nutzen. Zu viele Merkmale im Lernalgorithmus können dazu führen, dass Verbindungen zwischen den Merkmalen gefunden werden, die nicht gewichtig sind. Ebenfalls ist ein Modell einfacher nachzuvollziehen, welches anstatt 1 Millionen Merkmale nur 1000 Merkmale nutzt. Natürlich kann die Anzahl der Merkmale auch die Performance des gesamten Modells beeinflussen. Es gibt für den Prozess der Merkmalsgewinnung keine richtige oder falsche Vorgehensweise. Somit wird in der Praxis oft Verschiedenes ausprobiert.

Ziel ist es, Merkmale zu extrahieren, die zu einer möglichst genauen Klassifikation führen. Folgend wird an einem Beispiel veranschaulicht, wie wichtig die Auswahl passender Merkmale ist. In diesem Beispiel sollen Tiere als Reptil klassifiziert werden. Da hier nur zwischen Reptil oder nicht Reptil unterschieden wird, handelt es sich um eine binäre Klassifikation. In der ersten Spalte von Tabelle 3 wird das zu klassifizierende Tier aufgeführt. Danach folgen fünf festgelegte Merkmale, die auf Reptilien zu treffen. Die vorletzte Spalte zeigt, ob es sich real um ein Reptil handelt oder nicht. Zum Schluss folgt die Spalte mit dem Klassifizierungsergebnis des Modells. Zutreffende Merkmale werden mit 1 und nichtzutreffende Merkmale mit 0 markiert. Die letzten beiden Ergebnisspalten werden mit wahr und falsch dargestellt.

Im ersten Beispiel in Tabelle 3 werden die fünf Merkmale Eierlegend, Schuppen, giftig, kaltblütig und die Anzahl der Beine = 4 ausgewählt, um das Tier als Reptil zu klassifizieren. Die Merkmale wurden vorab anhand des Alligators verifiziert und ergaben eine korrekte Klassifikation. Somit wurden alle Merkmale in das Modell aufgenommen. Als nächstes wird das Modell mit der Kobra getestet. Hier zeigt sich sofort, dass das Modell nach der aktuellen Auswahl der Merkmale die Kobra nicht als Reptil klassifiziert, da sie keine 4 Beine hat. Die Kobra wäre ein falsch negatives Ergebnis des Lernmodells, da sie ein Reptil ist, aber vom Lernmodell nicht als solches klassifiziert wird.

Tabelle 3 – Übersicht der Merkmale zur Klassifizierung eines Tieres als Reptil, adaptiert nach [10]

	Eier-legend	Schuppen	Giftig	Kalt-blütig	Anzahl Beine	Reptil	Ergebnis Lernmodell
Alligator	1	1	0	1	4	Wahr	Wahr
Kobra	1	1	1	1	0	Wahr	Falsch

Anhand dieses Beispiels kann man für das Modell ableiten, dass die ausgewählten Merkmale keine gute Vorhersage treffen würden. Zum Anpassen des Modells werden die Merkmale auf Eierlegend, Schuppen und kaltblütig reduziert. Mit diesem Modell werden beide Tiere als Reptil klassifiziert, was Sie beide auch sind. Es gilt also, die Merkmale so generalisierend wie möglich festzulegen, ohne dabei zu viel an Genauigkeit zu verlieren.

Es folgt ein weiteres Beispiel, um darzustellen wie sehr die Merkmalsauswahl das Ergebnis des Modells beeinflusst. Die Merkmale, die nun in das Modell eingehen, sind weiterhin Eierlegend, Schuppen und kaltblütig. Im dargestellten Beispiel in Tabelle 4 wird gezeigt, dass bei der Anwendung der ausgewählten Merkmale, der Lachs ebenfalls als Reptil klassifiziert wäre. In Wirklichkeit ist der Lachs kein Reptil und wäre damit ein falsch positives Ergebnis des Models. Gleichzeitig wären der Alligator und die Kobra weiterhin korrekt klassifiziert.

Tabelle 4 - Übersicht der Merkmale zur Klassifizierung eines Tieres als Reptil, dabei gehen die hervorgehobenen Merkmale in das Lernmodel ein, adaptiert nach [10]

	Eier-legend	Schuppen	Giftig	Kaltblütig	Anzahl Beine	Reptil	Ergebnis Lernmodell
Alligator	1	1	0	1	4	Wahr	Wahr
Kobra	1	1	1	1	0	Wahr	Falsch
Lachs	1	1	0	1	0	Falsch	Wahr
Python	1	1	0	1	0	Wahr	Wahr

Anhand der aufgeführten Beispiele lässt sich ableiten, dass das Nutzen aller Merkmale eher falsch negative Ergebnisse hervorbringt. Das Reduzieren der Merkmale führt eher zu falsch positiven Ergebnissen. Ob falsch negative oder falsch positive Ergebnisse akzeptabel sind, ist eine grundlegende Designentscheidung für das Lernmodell. Oftmals werden die Anzahl der Merkmale so lange erweitert, bis möglichst alle Beispieldatensätze sehr gut vorhergesagt werden. Wendet man das Modell dann auf unbekannte Daten an, kommt es jedoch zu überdurchschnittlich vielen Fehlern. Dies bezeichnet man auch als Überanpassung (engl. overfitting). Ein Grund für eine Überanpassung ist die Gestaltung von zu komplexen Modellen für die vorhandenen Daten. Ein weiterer Grund sind zu viele Merkmale für zu wenige Testbeispiele [2]. Aus der Statistik ist ebenfalls der Begriff zu „hohe Varianz“ bekannt, was als Synonym für Überanpassung verstanden werden kann [2]. Um Überanpassung zu vermeiden gibt es verschiedene Möglichkeiten:

- Das Modell vereinfachen, zum Beispiel bei neuronalen Netzen weniger Schichten verwenden.
- Das Hinzufügen von Trainingsdaten
- Merkmalsreduzierung

Zur Vervollständigung sei an der Stelle auch die Unteranpassung (engl.: underfitting) erwähnt. Hierfür sind häufige Gründe, dass die ausgewählten Merkmale nicht aussagekräftig genug sind oder das Lernmodell zu einfach gewählt ist. [2].

Das Beispiel in Tabelle 4 zeigt ebenfalls, dass bei der Python und dem Lachs alle ausgewählten Merkmale gleich sind. Da solche Fälle immer auftreten können, ist es sinnvoll, Merkmalen durch Bewertung oder Sortierung nach Wichtigkeit eine unterschiedliche Relevanz zu geben.

3.2.3 Vermeidung von Duplikaten

Da für jede Fehlermeldung im Entwicklungsverwaltungssystem Arbeitselemente angelegt werden und jedes Arbeitselement analysiert werden muss, empfiehlt es sich, Duplikate zu vermeiden. Diese Duplikate können zum Beispiel als ein Arbeitselement zusammengefasst werden.

Eine Methode ist die Berechnung der Dokumentenähnlichkeit. Ähnlichkeiten von Dokumenten werden über Dokumentenvektoren gebildet und in einer Term-Dokumenten-Matrix gespeichert [7]. Hierfür wird sich der Clusteranalyse bedient. Für die Analyse werden zuerst charakteristische Merkmale des Dokumentes identifiziert. Dabei werden im Dokument alle Terme analysiert. Eine zu oberflächliche thematische Einordnung der Dokumente reicht im vorliegenden Fall nicht aus. Zu oft gibt es Fehlermeldungen, die sich ähnlich sind, aber unterschiedliche Ursachen haben. Es bedarf einer fast identischen vorliegenden Fehlermeldung, um sicher zu sein, dass es sich um den gleichen aufgetretenen Fehlerfall handelt. Dennoch können bei der Analyse geschlossene Wortklassen ausgenommen werden. Bei geschlossenen Wortklassen handelt es sich

vorwiegend um Artikel, Konjunktionen oder Präpositionen [7]. Wortformen aus geschlossenen Wortklassen kommen in der Regel in allen Dokumenten in einer ähnlich verteilten Häufigkeit vor [7]. Danach wird die Termfrequenz bestimmt, welche angibt wie oft ein Term in einem Dokument vorkommt. Wie auch im vorliegenden Beispiel, gilt es zu beachten, dass die Länge der Dokumente unterschiedlich ist. Da dies aber wichtig ist für die Interpretation der Häufigkeit eines Terms, muss die Termfrequenz normalisiert werden. Ebenso gilt es zu berücksichtigen, dass charakteristische Merkmale in wenigen Dokumenten häufig auftreten. Allgemein betrachtet treten Sie jedoch weniger häufig auf [7]. Werden diese zwei Aspekte berücksichtig, lässt sich für jeden Term seine Wichtigkeit für ein Dokument berechnen. Zur Vereinfachung kann ebenfalls ein Schwellenwert festgelegt werden. Unter dem Schwellenwert gelten Terme nicht mehr als wichtig für ein Dokument [7]. Die berechneten Gewichtungen werden in einer sogenannten Term-Dokument-Matrix dargestellt, wie in Tabelle 5 dargestellt. Da es sich hier ebenfalls um eine Art Clustering handelt, bedarf es auch hier einer Festlegung des Ähnlichkeitsmaßes. Das heißt, es muss ein Maß festgelegt werden, wie sehr sich zwei Dokumente gleichen. Hierfür dienen gängige mathematische Ansätze wie die euklidische Distanz oder Skalarprodukte. Die dadurch errechnete Dokumentenähnlichkeit wird ebenfalls in einer Matrix dargestellt, in der sogenannten Dokument-Dokument-Matrix, welche in Tabelle 6 verallgemeinert dargestellt ist. Hier werden die paarweisen Ähnlichkeiten der Dokumente dargestellt [7].

Tabelle 5 - allgemeine Term-Dokument-Matrix, adaptiert nach [7]

	t1	**t2**	**…**	**t(n)**
d1	w (1,1)	w (2,1)	…	w (n,1)
d2	w (1,2)	w (2,2)	…	w (n,2)
d(k)	w (1,k)	w (2,k)	…	w (n,k)

Tabelle 6 - allgemeine Dokument-Dokument-Matrix, adaptiert nach [7]

	d1	**d2**	**…**	**d(k)**
d1	sim (d1,d1)	sim (d2,d1)	…	sim (dk,d1)
d2	sim (d1,d2)	sim (d2,d2)	…	sim (dk,d2)
d(k)	sim (d1,dk)	sim (d2,dk)	…	sim (dk,dk)

Zur Cluster Analyse dient die Dokument-Dokument-Matrix als Grundlage. Als Ergebnis liegen dann Teilmengen der Gesamtdokumentenmenge vor, welche sich thematisch ähneln.

Eine weitere Vorgehensweise zur Aufdeckung von Ähnlichkeiten ist der Levenshtein Algorithmus. Dieser Algorithmus beschreibt den Abstand zweier Zeichenketten in Form der Mindestanzahl an Editieroperationen [11] und wird auch als Levenshtein-Distanz bezeichnet. Als Editieren wird dabei das Hinzufügen, Löschen oder Ersetzen einzelner Zeichen verstanden. In Tabelle 7 ist ein Beispiel für eine Angabe der Levenshtein-Distanz dargestellt.

Tabelle 7 - Beispiel für Levenshtein-Distanz, übernommen nach [11]

	Distanz	**Operation**
HAUS - MAUS	1	Ersetze H durch M
QUALITÄT - QUANTITÄT	2	Ersetze l durch n und füge t hinzu

Eine solche Distanz kann ebenfalls als Matrix wie in Tabelle 8 angegeben werden und nutzt dabei das Prinzip der dynamischen Programmierung. Von links oben der Position (0,0) wird der Weg mit den geringsten Kosten gesucht. Rechts unten an letzter Position (4,4), kann das Ergebnis der Distanz dann abgelesen werden. Dabei werden die entstandenen Kosten

bis zum Ziel aufsummiert, sodass die Position (4,4) in der Matrix in Tabelle 8 für das Beispiel Haus – Maus eine Distanz von 1 ausgibt.

Tabelle 8 - Matrix zur Bestimmung der Levenshtein-Distanz, adaptiert nach [11]

		M	**A**	**U**	**S**
	0	1	2	3	4
H	1	1	1	2	3
A	2	1	1	1	2
U	3	2	2	1	1
S	4	3	3	1	1

3.3 Lernmodelldefinition

Wie komplex die Auswahl eines geeigneten Algorithmus für das Lernmodell ist, wird beispielhaft in Abbildung 9 dargestellt. Verschiedene Kriterien wie die Anzahl der Daten, sowie die Struktur und Art der Daten beeinflussen das Finden eines passenden Algorithmus. In der Praxis werden oft iterative Runden gefahren, was bedeutet, es werden verschiedene Algorithmen ausprobiert und Ergebnisse verglichen um zu entscheiden, ob dieser Lernalgorithmus zur Lösung des vorliegenden Problems geeignet ist. Ebenfalls beeinflusst wird die Entscheidung von Parametern wie die Geschwindigkeit der Vorhersage oder des Trainings, die Anzahl der Merkmale oder gar die Nachvollziehbarkeit des Modells.

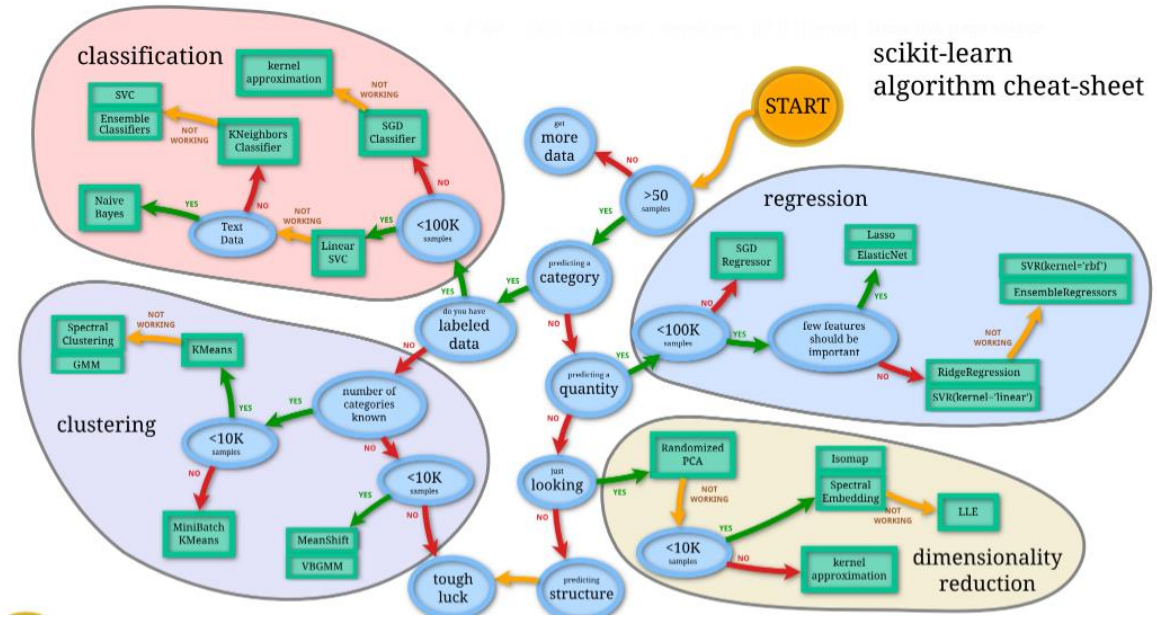

Abbildung 9 - Faktoren zur Auswahl des richtigen Lernalgorithmus [4]

Bislang wurde in der vorliegenden Abschlussarbeit lediglich von Beispieldatensätzen gesprochen. Ein vollständiges Lernmodell besteht jedoch aus verschiedenen Mengen von bereits gekennzeichneten Datensätzen. Idealerweise gibt es einen Trainingsdatensatz, welcher im Minimum 70% der gesamten Beispieldaten ausmacht. Die restliche Menge kann als Validierungsdatensatz genutzt werden oder gleich aufgeteilt als Validierungs- und Testdatensatz. Für sehr große Datenmengen wird die Aufteilung der Datensätze mittlerweile mit mindestens 95% für die Trainingsdaten angegeben. Das heißt es werden lediglich 5% als Validierungsdatensatz verwendet. Der Trainingsdatensatz wird zur Modellbildung verwendet und hier gilt grundsätzlich, je mehr Daten desto besser. Die Test- und Validierungsdaten werden genutzt, um den Algorithmus auszuwählen beziehungsweise diesen nach Fertigstellung des Modells, mit unbekannten Daten zu testen.

Grundlegend lässt sich sagen, dass die meisten Lernalgorithmen aus einer Verlustfunktion, einem Optimierungskriterium und einem Optimierungsverfahren bestehen. Eine Verlustfunktion ist ein Maß für die Strafe einer Fehlklassifizierung. [12] Es ist also Ziel diese möglichst gering zu halten. Das Optimierungskriterium beruht auf der Verlustfunktion und das Optimierungsverfahren versucht eine Lösung zu finden, die das Optimierungskriterium erfüllt [2]. Dabei gibt es einige Algorithmen, die sehr oft als Bausteine für Lernalgorithmen verwendet werden.

Sehr verbreitet ist die Lineare Regression, um Linearkombinationen aus Eingaben zu erlernen [2]. Ein angewandtes Beispiel sind Vorhersagen zum Immobilienpreis. Es gibt bereits einen Datensatz, welcher die Größe der Immobilie sowie den Preis der Immobilie beinhalten. Erwünscht ist nun eine Vorhersage für einen Preis zu einer neuen noch nicht vorhandenen Größe der Immobilie. Der Algorithmus wird für alle bekannten Daten versuchen, eine optimale Linie zu finden und den unbekannten Datenpunkt auf sein bestmöglichstes platzieren. Somit lässt sich der bisher unbekannte Preis vorhersagen.

Ein weiteres Beispiel sind Support Vector Maschinen. Ihre Stärke ist es, in hochdimensionalen Vektorräumen die optimalen Entscheidungsgrenzen zu finden, um unbekannten Eingaben optimalen Ausgaben zuzuweisen. Wie bei allen Lernmodellen wird auch das Ergebnis der Support Vector Maschinen besser, je größer die Trainingsmengen sind. Dabei sinkt die Wahrscheinlichkeit, dass eine neue Eingabe den für das Training verwendeten Datensätzen unähnlich ist [2].

Als Beispiel für die Optimierungsfunktion kann hier der Gradientenabstieg genannt werden. Der Gradientenabstieg wird verwendet, um optimale Parameter für beispielsweise die lineare Regression, Support Vector Maschinen oder für neuronale Netze zu finden. Das Gradientenabstiegsverfahren kann genutzt werden, um für ein Optimierungsproblem das Minimum zu finden. Der Gradientenabstieg geht iterativ vor und startet an einem zufälligen Punkt. Vereinfacht ausgedrückt wird von dem zufällig ausgesuchten Punkt in die Richtung des steilsten Abstieges solange iteriert, bis das Minimum der Funktion erreicht wurde, welches gleich Null ist. In der Praxis wird das exakte Minimum oftmals nicht erreicht, da meist Grenzwerte vorgegeben werden, um die Anzahl der Durchläufe zu begrenzen.[6]

Ein Regressionsalgorithmus erzeugt ein Modell anhand bereits gekennzeichneter Daten und gibt für ungezeichnete Eingaben einen Zielwert aus.

An dieser Stelle sollen auch Neuronale Netze genannt werden. Allerdings basieren diese auf dem Algorithmus der logistischen Regression. Dies ist kein Algorithmus der Regression, sondern der Klassifikation. Bei der Klassifikation werden nicht klassifizierte Daten einer Klasse zugewiesen. Handelt es sich um mehr als zwei Klassen zur Auswahl, spricht man von einer Multi-Class-Klassifikation. Bei exakt zwei Klassen handelt es sich um eine binäre Klassifikation. Vereinfacht beschrieben, besteht ein Neuronales Netz aus einem oder mehreren Neuronen, deren Verbindungen mit Gewichtungen sowie Eingaben und Ausgaben dargestellt werden. Zwischen Ein- und Ausgabe können sich sogenannte

Zwischenschichten befinden. Wie viele Schichten es gibt, wird bei der Lernmodellauswahl definiert. Eine zu hohe Anzahl an Schichten kann zu einer Überanpassung des Netzes führen. Wenn die Schichten zwischen Ein- und Ausgabe immer Ihre Vorgänger und Nachfolger kennen, ist das Netz vollständig verbunden.

Ein Klassifikationsalgorithmus erzeugt ein Modell anhand der bereits vorhanden klassifizierten Daten und gibt für nicht klassifizierte Daten eine Klasse aus.

Um ein Lernmodell zu beurteilen gibt es verschiedene Vorgehensweisen und Kennzahlen. Ein bekanntes Beispiel aus dem Bereich der Klassifikation ist die Wahrheitsmatrix. Hier wird in einer Matrix angegeben wie erfolgreich das Modell die Klassen zugeordnet hat. Dargestellt in Tabelle 9, für das Beispiel der textuellen Fehlermeldungen könnte eine solche Matrix wie folgt aussehen:

Tabelle 9 – Wahrheitsmatrix: Beispielhaft dargestellt für die Klassen der textuellen Fehlermeldungen

		VORHERGESAGT			
		PCSF	BOM	CALC	TECH
TATSÄCHLICH	PCSF	1000	0	3	0
	BOM	0	1000	0	0
	CALC	0	0	1000	0
	TECH	0	2	1	1000

Die Wahrheitsmatrix aus Tabelle 9 stellt dar, dass 4000 Dateien tatsächlich als das klassifiziert wurden, was sie sind. Diese insgesamt 4000 richtig positiven Ergebnisse sind in Tabelle 9 dunkelgrau markiert. Nur 6 Dateien wurden einer anderen Klasse zugeordnet, als sie in Wirklichkeit angehören. Diese Ergebnisse sind in Tabelle 9 hellgrau markiert. In solch einer Matrix lässt sich zum Beispiel aufdecken, dass ein Modell dazu tendiert immer Dateien aus dem Bereich BOM als CALC falsch zu klassifizieren. Als Lösungsschritt werden dann mehr Trainingsdateien für den Bereich BOM hinzugefügt oder weitere Merkmale dem Lernmodell eingespeist, damit differenzierter zwischen den Klassen entschieden werden

kann. Ebenso lässt sich aus dieser Ansicht ablesen, wenn eine Klasse überdurchschnittlich vertreten ist. Typischerweise wird eine solche Matrix auch genutzt um Leistungskriterien des Modells zu berechnen [2].

Die Genauigkeit lässt sich durch das Verhältnis von richtig positiven Ergebnissen zu der Summe von richtig und falsch positiven Ergebnissen feststellen. [2]

Die Trefferquote lässt sich durch das Verhältnis der richtig positiven Ergebnisse zu der Summe von richtig positiven und falsch negativen Ergebnissen feststellen.[2]

Je nach Aufgabenstellung ist eine hohe Genauigkeit oder Trefferquote erwünscht. Beides lässt sich oftmals nicht zusammen erreichen [2]. Werden die Kennzahlen auf den vorliegenden Fall der textuellen Fehlermeldungen angewandt, dann ist eine hohe Trefferquote wichtiger als eine hohe Genauigkeit. Eine weitere Kennzahl ist die Korrektklassifikationsrate. Diese setzt die korrekt klassifizierten Ergebnisse mit der Gesamtzahl aller klassifizierten Dateien in ein Verhältnis. Da im vorliegenden Beispiel alle Klassen gleichwertig sind, ist dies ebenfalls eine nützliche Kennzahl.

3.4 Implementierung

Für die vorliegende Abschlussarbeit zum Thema „Automatische Klassifizierung textueller Fehlermeldungen“ wurde in der Programmiersprache Python eine beispielhafte Umsetzung der einzelnen Schritte durchgeführt. Hierfür wurden unter anderem die folgenden frei verfügbare Software Bibliotheken benutzt:

- TensorFlowCore
- Scikit-Learn
- Matplotlib
- Numpy

Die komplette Liste der genutzten Softwarebibliotheken befindet sich im Anhang Teil 1. TensorFlowCore bietet als Bibliothek spezialisiert auf maschinelles Lernen bereits eine Fülle an Möglichkeiten. Zusätzlich ist eine frei verfügbare umfangreiche Dokumentation mit theoretischen und praktischen Umsetzungsbeispielen vorhanden. Im Rahmen des Projektes muss vorausschauend auf eine finale Implementierung, auch darauf geachtet werden, was für Drittanbieter Produkte integriert, beziehungsweise genutzt werden dürfen. Hierzu gibt es firmenintern vorgeschriebene Prozesse.

3.4.1 Vorabanalyse

Nachdem die Dateien aus einer frei definierten Ablage geladen wurden, werden diese vorab analysiert. Diese Analyse dient dazu, einen besseren Überblick über die vorliegenden Daten zu bekommen. Zur Veranschaulichung zeigt Abbildung 10 die Anzahl der Dateien, die eingelesen wurden und deren Länge. Die Länge bezieht sich auf die Anzahl der Wörter. Dies dient hier zur Vorarbeit, um sich einen Überblick zu verschaffen, welche Längen an Dokumenten zu erwarten sind. Die Länge und damit die Anzahl der Wörter wird in einem späteren Schritt als Eingabe in das Lernmodell angegeben.

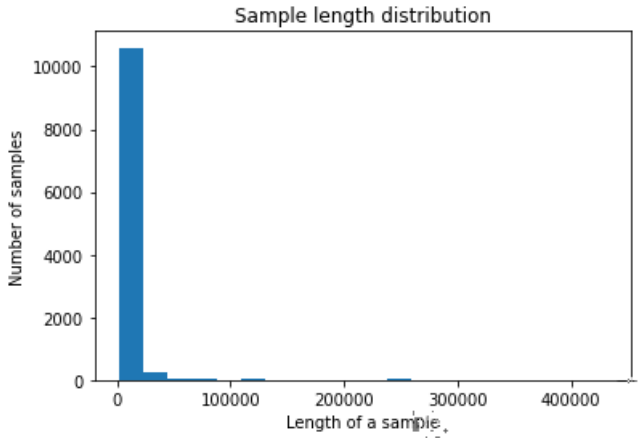

Abbildung 10 - Balkendiagramm mit Anzahl und Länge der eingelesenen Dokumente

Abbildung 11 zeigt eine weitere Analyse der Dateien mit den einzelnen Wörtern, die am häufigsten vorkommen und wie oft sie vorkommen. Dies kann genutzt werden, um die Dateien zu bereinigen, bevor sie in das Lernmodell eingespeist werden. In allen Analysen kann die Anzahl der angegeben Wörter, die an der x-Achse dargestellt werden, stets variiert werden. Eine Anzeige von 30 Wörtern hat sich als gut lesbar herausgestellt. Ebenfalls kann variiert werden, ob mit n=1, n=2 oder n=n analysiert werden soll. Wird n = 1 genutzt, dann wird die Häufigkeit einzelner Wörter analysiert wie in Abbildung 11 bereits gesehen. Bei n =2 werden einzelne Wörter analysiert sowie auch zwei aufeinanderfolgende Wörter zu einer Einheit zusammengefasst und analysiert. Zum Vergleich dargestellt in Abbildung 12. In Kapitel 3.2.1 Textanalysemethoden wurde bereits auf die Erklärung von n-gramm Analysen eingegangen. In Kapitel 3.4.2 Tokenisierung wird nochmals ein detailliertes Beispiel der Zerlegung der verschiedenen n-gramm Einstellungen aufgegriffen. Ebenso kann man sich die am wenigsten häufig vorkommenden Wörter anzeigen lassen, wie in Abbildung 13 zu sehen. Auch hier kann man aufgrund der Ergebnisse entscheiden, ob man diese filtert. In der vorliegenden Auswertung auf Abbildung 13 handelt es sich meist um eindeutige Identifikationsschlüssel (engl.: Unique Identifier) technischer Objekte aus der Anwendung selber. Sie geben aber keinen effizienten Hinweis aus welchem Modul diese stammen.

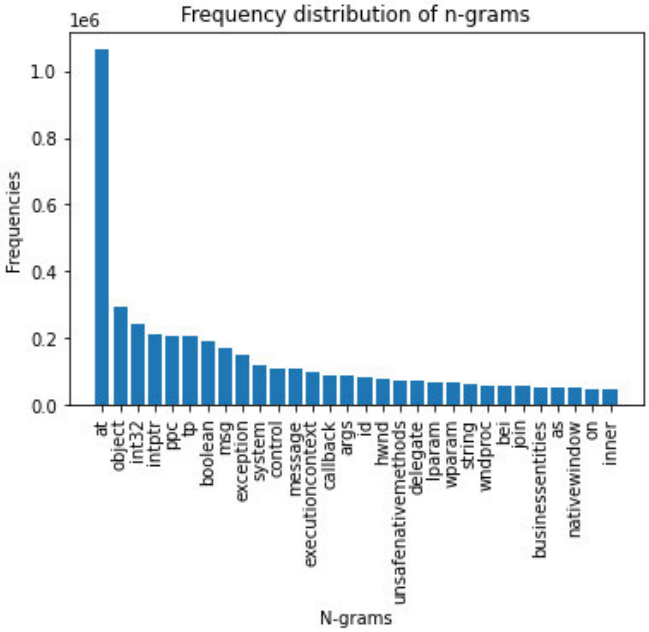

Abbildung 11 - Balkendiagramm mit den am häufigsten vorkommenden Wörtern mit dem n-gramm range (1,1)

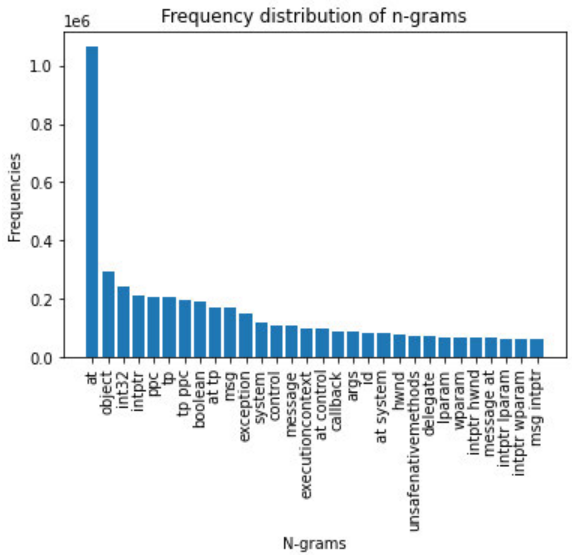

Abbildung 12 - Balkendiagramm mit den am häufigsten vorkommenden Wörtern mit dem n-gramm range (1,2)

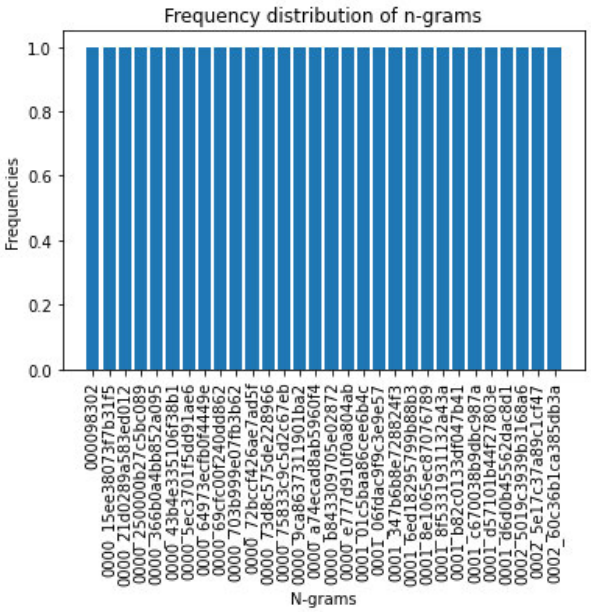

Abbildung 13 - Balkendiagramm mit den am wenigsten häufig vorkommenden Wörtern

In Abbildung 14 wird dargestellt, dass die ausgegebenen Wörter in den Fehlermeldungen durch Sonderzeichen getrennt sind. Das heißt, ohne weiteren Eingriff wird fast die ganze Zeile als ein Wort gesehen. Bereinigt man die Sonderzeichen vorab, oder hier im speziellen Fall den Satzpunkt, der als Trennzeichen genutzt wird, kann man durchaus auch mit n = 2 arbeiten. Dann werden die Wörter wie in Abbildung 14 Zeile 2 fragmentiert.

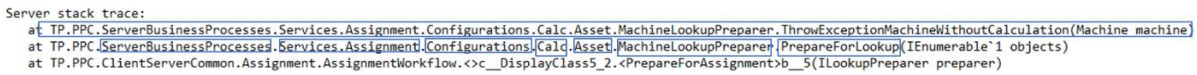

Abbildung 14 - Beispieltext aus einer Fehlermeldung, welcher die zusammenhängende Schreibweise zeigt und eine mögliche Fragmentierung, wenn der Satzpunkt gefiltert wird

In Abbildung 15 wurde zur Veranschaulichung die Einteilung der Klassen einmalig als Ausgabe dargestellt. Hier wird die Zuordnung der Klassen auf einen Identifikator vorgenommen. Die Dateien werden dem entsprechenden Identifikator zugewiesen und somit einer Klasse zugeteilt.

	traces	area	identifier	path
0	11/4/2021 2:15:29 PM: Category = Exception, Se...	PCSF	0	Stacktrace19a089c1-9b48-4289-a6f2-89606bbfab9f...
1	01/11/2021 07:20:21: Category = Exception, Sev...	BOMAPI	1	Stacktrace9bf50284-a63b-4e36-98db-c900ec4b60e5...
2	01/11/2021 07:20:03: Category = Exception, Sev...	PCSF	0	Stacktrace9833b4cf-537b-42c7-956b-8e9a06975217...
3	10/13/2016 8:46:27 AM: Category = Exception, S...	TECH	3	Stacktrace21d22016-8ca7-4a48-9ca9-8855a9998d5b...
4	11/4/2021 2:15:22 PM: Category = Exception, Se...	CALC	2	Stacktraceb59f6042-4403-4d95-8013-8e78ac19d9c4...

Abbildung 15 - Beispielhafte Ausgabe der Zuordnung zwischen Textdatei, Klasse und Identifikator

3.4.2 Tokenisierung

Tokenisierung bezeichnet allgemein die Zerlegung von Text in kleine Bausteine wie Wörter oder Sätze. In der vorliegenden Funktion def_ngram_vectorize wird definiert, welche Art eines n-gramms zur Tokenisierung genutzt wird. Dabei ist es möglich ein Minimum und Maximum anzugeben. Wird für den n-gram_range(2,2) definiert, gehen 2 Wörter als eine Einheit ein. Bei einem n-gramm_range(1,1) geht jedes Wort einzeln in die Analyse ein. Ein Beispielsatz soll dies veranschaulichen:

Text: ‚Error thrown loading object failed‘

Analyse n-gramm_range(2,2): ‚Error thrown', 'thrown loading’, ‘loading object’, ‘object failed’

Analyse n-gramm_range(1,1): ‘Error’, ‘thrown‘, ‚loading‘, ‚object‘, ‚failed‘

Wie bereits erwähnt, muss definiert werden, ob für die n-gramm Analyse, Wörter, Buchstaben oder Sätze eingehen. Ebenso kann ein n-gramm_range(1,2) definiert werden. Dann werden beide Arten von n-gramms zur Tokenisierung genutzt. Die scikit-learn Klasse TfidfVectorizer wird in dieser Funktion ebenfalls verwendet. Hier werden Termfrequenzen unter Berücksichtigung der inversen Dokumentenfrequenz ermittelt. Somit wird für jeden Term ein Maß für seine Wichtigkeit angegeben [7]. Stoppwörter oder andere unerwünschte

Wörter werden an dieser Stelle ebenfalls herausgefiltert und gehen somit nicht in die Merkmalsselektion mit ein. Danach wird festgelegt wie viele Merkmale in das Lernmodell eingehen. Mit der Angabe TOP_K_FEATURES kann hier flexibel variiert werden. Die Vorabanalyse zeigte, dass die meisten Dokumente mit einer Länge bis zu 50.000 Wörter vorhanden sind und es nur einige Ausnahmen mit größer 100.000 Wörtern gibt. Die Anzahl der Merkmale wird ebenfalls in der Funktion def_ngram_vectorize innerhalb der integrierten Klasse SelectKBest als Minimumwert übergeben. SelectKBest bewertet die Merkmale nach Ihrer Wichtigkeit und führt mit Hilfe des Parameters f_classif eine Varianzanalyse durch [13]. Der komplette Bereich Tokenisierung ist in den Anlagen Teil 3 beigefügt.

3.4.3 Modellierung

Zur Modellierung des Lernmodells wird ein mehrschichtiges Netzwerk aufgebaut. Es werden eine Eingabeschicht und eine Ausgabeschicht definiert. Zwischenschichten können hinzugefügt werden. Wie viele Zwischenschichten es gibt, wird im nächsten Schritt festgelegt, wenn die Funktion des Netzwerkaufbaus aufgerufen wird. Das Modell wird als sequentielles Modell definiert, was bedeutet, dass Informationen zur Reihenfolge der Zeichen genutzt werden. Abbildung 16 zeigt eine der vielzähligen Möglichkeiten, um ein neuronales Netz zu definieren.

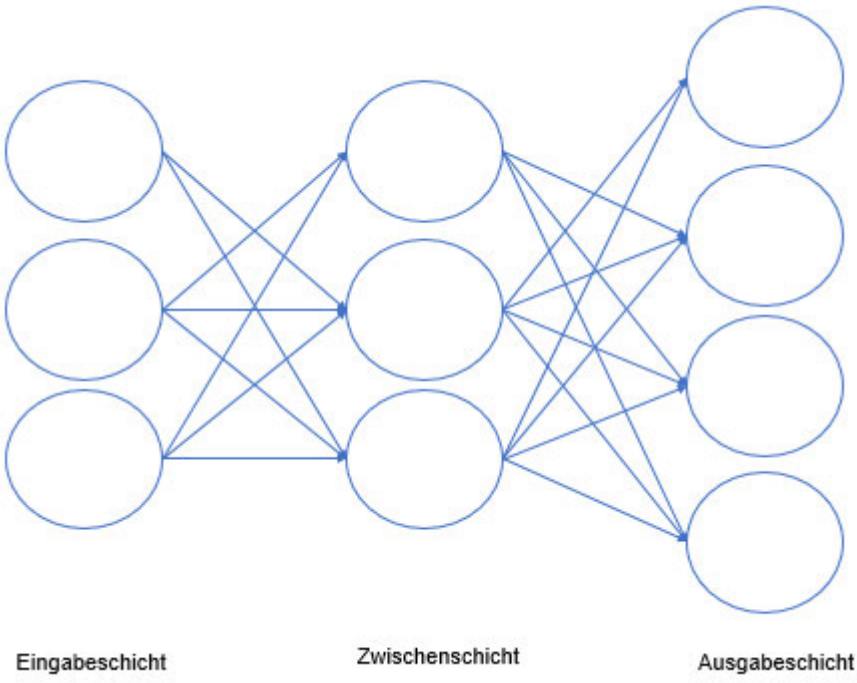

Abbildung 16 – Beispiel einer Netztopografie eines zweischichtigen neuronalen Netzes

In der Funktion zum Modellieren des Netzwerkes wird ebenfalls eine sogenannte Aktivierungsfunktionen für die Neuronen integriert, die ReLU-Funktion (engl.: Rectified Linear Unit function). Die ReLU-Funktion ist immer Null, wenn Ihre Eingabe negativ ist und nimmt sonst den Eingabewert selbst an [2]. Zusätzlich wird die Softmax-Funktion angewandt. Wie Abbildung 17 darstellt, wird die Softmax-Funktion bei Multi-Class-Klassifikationen angewandt, während bei binären Klassifikationen die Sigmoid-Funktion genutzt wird. Die Softmax-Funktion wird genutzt, um Wahrscheinlichkeiten der einzelnen Klassen zuzuordnen. Die Wahrscheinlichkeiten aller Klassen müssen immer 1 ergeben. Gleichzeitig wird bei der Softmax-Funktion erwartet, dass jedes Ergebnis nur einer Klasse zugehörig ist. Einheiten in dieser Klasse beschreibt die Anzahl der Neuronen. Wobei die Neuronen der Ausgabeschicht der Anzahl der definierten Klassen gleicht und die Anzahl der Eingabeschicht ist gleich der Anzahl der Merkmale. Variiert werden kann im Training des Modells mit der Anzahl der Zwischenschichtneuronen.

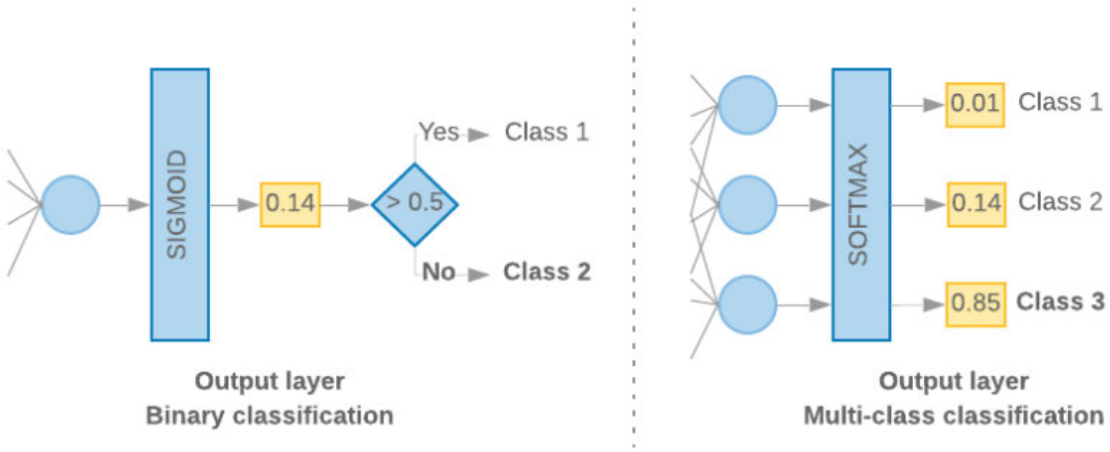

Abbildung 17 – Unterschied der Ausgabeschichten für eine binäre Klassifikation, welche die Sigmoid-Funktion nutzt und eine Multi-Class-Klassifikation, welche die Softmax-Funktion nutzt [14]

Es ist in diesem Modell auch möglich während des Trainings eine sogenannte Dropout rate zu bestimmen. Ein Dropout, zu Deutsch herausfallen, wird unter anderem angewandt, um die bereits erwähnte Überanpassung zu vermeiden. Dieser Vorgang wird Regularisierung genannt. Regularisierung umfasst alle Methoden, die den Lernalgortihmus dazu zwingen, das Modell weniger komplex zu erzeugen [2]. Weitere Methoden sind zum Beispiel ein früher Abbruch (engl. early stopping) oder Datenanreicherung (engl. data augmentation). Die Modellierung der gesamten Funktion ‚def build_mlp' kann in der Anlage Teil 3 eingesehen werden.

Es kann an dieser Stelle ebenfalls eine Erweiterung des neuronalen Netzes modelliert werden, durch ein sogenanntes Convolutional Neural Network. Diese gelten als performanter für Netze, die schnell größer werden und deren Anzahl an Merkmalen

zunimmt. Auch wenn im Zuge der Aufgabenstellung durchaus damit experimentiert wurde, wird zur Ergebnisdarstellung das einfache mehrschichtige Netzwerk genommen. Auch zur späteren Nutzung wird das mehrschichtige Netzwerk empfohlen, da dies übersichtlicher ist und bereits sehr gute Werte hervorbringt. Mittelfristig ist nicht zu erwarten, dass die Eingabe der Merkmale sich drastisch erhöht. Somit erscheint aktuell die Nutzung eines Convolutional Neural Networks überdimensioniert.

3.4.4 Training

Für das Training der Daten werden in einer eigenen Funktion verschiedene Parameter definiert und mit den tokenisierten Daten in das vorher definierte Lernmodell gegeben. Verschiedenen Parameter können dabei beeinflusst werden. Dazu zählen zum Beispiel:

- Epochen: Die Anzahl der Durchläufe der Daten durch das Netzwerk
- Batch size: Losgröße
- Layer: Zwischenschichten des Netzes
- Units: Neuronenanzahl der Zwischenschichten
- Dropout rate: Regulierungsquote

Die hier genutzte Verlustfunktion beruht auf dem Prinzip der Kreuzentropie (engl: crossentropy), welches ein statistisches Maß für die Qualität eines Modells ist. [15] In der angewendeten Aufgabe wird durch die genutzte sparse categorical crossentropy die Differenz zwischen der Wahrscheinlichkeit einer richtigen Klassifizierung und des realen Ergebnisses festgestellt. Das heißt, dass hier die Wahrscheinlichkeiten, welche durch die Softmax-Funktion errechnet werden, mit dem tatsächlichen Ergebnis verglichen wird. Wiederum kann durch Anpassen der Gewichtungen im Netzwerk, das Ergebnis verbessert werden. Aus diesem Grund werden mehrere Epochen beziehungsweise Durchläufe angestrebt. Um dies in der Realität umzusetzen, wendet das Netzwerk Rückpropagierung (engl.: Backpropagation) an. Bei diesem Vorgang kann es den gegangenen Weg durch das Netzwerk zurück gehen und die Gewichtungen anpassen, um wiederum vorwärts ein neues Ergebnis zu erzielen.

Weiterhin kommen noch Optimierungsparameter sowie weitere Regulierungsmethoden hinzu. Der Adam-Algorithmus ist eine optimierte Variante des Gradientenabstieges und verbindet mehrere Algorithmen, um das beste Ergebnis zu erzielen. Es hat sich besonders bewährt in Netzwerken mit vielen Merkmalen. [16]

Als weitere Regulierungsmethode wird der frühe Abbruch (engl.: early stopping) integriert. Hierbei werden nach jeder Epoche das Lernergebnis gespeichert und mit der Validierungsmenge verglichen. Wird bemerkt, dass die Leistung bei der Validierungsmenge sinkt, aber das Trainingsergebnis sehr gut ausfällt, wird abgebrochen, um eine Überanpassung zu verhindern. Zusätzlich, kann auch die Anzahl der Epochen vorab festgelegt werden. [2]

Es bietet sich an, die Verlaufsdaten für die Vorhersagegenauigkeit sowohl für Trainingsdaten als auch Validierungsdaten abzuspeichern. So lässt sich am Ende durch die Ausgabe dieser Werte vergleichen, wie gut das Lernmodell mit welchen Parametern agiert hat. Durch die Ausgabe der Werte für den Verlust, kann beurteilt werden, wie hoch die Wahrscheinlichkeit einer guten Einsortierung in die Klassen ist. Dabei hält sich das Netz neben den Trainingsdatensatz einen kleinen Teil Testdaten, um mit diesen gegen zu testen. Eine komplette Übersicht des Programmteils Training, Ausgabe Ergebnis und Zusammenfassung des genutzten Modells ist im Anhang Teil 4 beigefügt.

In der Praxis ist das Finden der passenden Parameter im Lernmodell immer auch ein Prozess des Ausprobierens und Evaluierens verschiedener Versuche. Jede Problemstellung benötigt dabei eigene Anwendungstests. Verschiedene Versuche werden im Folgenden dargestellt und erläutert. Als Gesamtdatensatz standen hierfür 12.943 Dateien der textuellen Fehlermeldungen zur Verfügung. Die Klassen waren dabei wie in Tabelle 10 aufgezeigt in dem vorhandenen Datensatz vertreten.

Tabelle 10 - Anzahl der vorhandenen Datensätze nach Klassen und prozentualer Anteil der Klassen am Gesamtdatensatz

Klasse	Daten	Anteil in %
BOM	4920	38,01
CALC	1384	10,69
PCSF	4747	36,68
TECH	1892	14,62
GESAMT	**12943**	**100**

Der Gesamtdatensatzes wurde wie in Tabelle 11 dargestellt, in circa 85% Trainingsdaten und 15% Validierungsdaten aufgeteilt.

Tabelle 11 - Aufteilung des vorhanden Datensatzes nach Trainings- und Validierungsdaten sowie unterteilt nach den verschiedenen Klassen

Klasse	Trainingsdaten	Testdaten	Gesamt	Anteil Testdaten in %
BOM	4182	738	4920	15
CALC	1177	207	1384	14,96
PCSF	4035	712	4747	15
TECH	1609	283	1892	14,96
GESAMT	**11003**	**1940**	**12943**	**14,99**

Dabei wurden alle der dargestellten Versuche in Tabelle 12 mit folgenden unveränderten Parametern durchgeführt:

- n_gramm_range(1,2)
- Epochen = 200
- learning rate = 0.003
- batch size = 128
- units = 64
- Dropout rate = 0.2

Tabelle 12 zeigt eine Übersicht der vier Versuche und die in den Versuchen variierten Parameter sowie das Ergebnis für die Vorhersagegenauigkeit der Trainings- und Validierungsdaten.

Tabelle 12 – Übersicht über die Versuchsreihe für das Lernmodell, die genutzten Eingabeparameters und den Ergebnissen für die Vorhersagegenauigkeit

Versuch	Merkmale	Schichten	Genauigkeit Trainingsdaten in %	Genauigkeit Validierungs-daten in %	Dauer der Ausführung
1	200	1	70,59	**72,73**	29 s
2	20000	1	98,32	**96,49**	4 min 19 s
3	40000	2	98,94	**97,42**	1 min 40 s
4	40000	200	37,79	**38,04**	2 min 50 s

Versuch 1 in Tabelle 12 zeigt die Befüllung des Lernmodells mit 200 Merkmalen. Es gehen die 200 Merkmale mit dem höchsten Score nach SelectKBest ein. Das Netz des Lernmodells ist in diesem Versuch einschichtig, was bedeutet, dass es zwischen Eingabeschicht und Ausgabeschicht keine definierten Zwischenschichten gibt. Schichten wie beispielsweise für die Softmax-Funktion oder Dropout Schichten werden hier nicht dazu gezählt. Die Vorhersagegenauigkeit der Validierungsmenge liegt bei diesem Versuch bei 79,88%. Abbildung 18 zeigt eine kurze Zusammenfassung des aufgebauten Netzwerkes aus Versuch 1. Es wird eine Dropout Schicht angezeigt, welche 200 Merkmale als Eingabe bekommt. Das Netz hat nur eine Schicht, hier dargestellt mit dem Begriff Dense. Es handelt sich in diesem einschichtigen Netz um die Ausgabeschicht., welche die 4 Klassen ausgibt. Die Gesamtanzahl der Merkmale die durch die Netzwerkschicht analysiert wurde stellt die Zahl in der Spalte Param # dar.

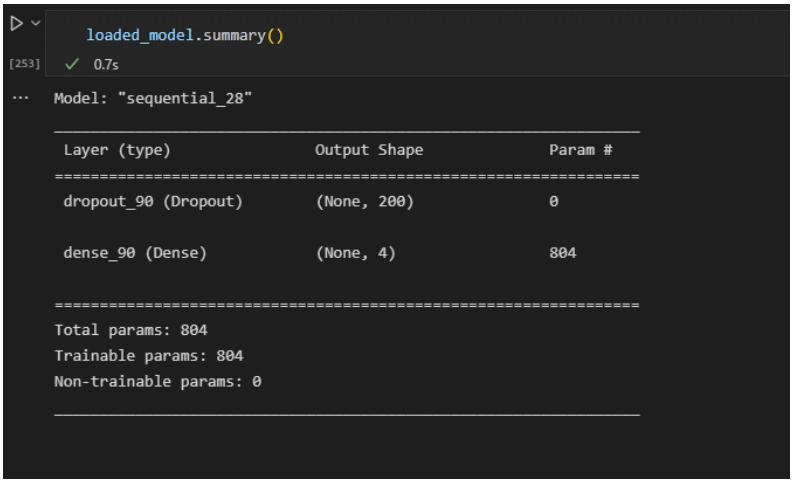

Abbildung 18 - Zusammenfassung Lernmodell aus Versuch 1

Ziel ist es nun diese Vorhersagegenauigkeit zu erhöhen. Aus diesem Grund werden im nächsten Versuch die Anzahl der eingehenden Merkmale erhöht. Es gehen im nächsten Schritt die 20.000 am höchsten bewerteten Merkmale in das Lernmodell ein. Wie in Tabelle 12 für Versuch 2 ausgewiesen, bewirkt diese Veränderung einen Sprung in der Vorhersagegenauigkeit der Validierungsdaten auf 97,95%. Somit kann festgestellt werden, dass diese Erhöhung der eingehenden Merkmale in das Lernmodell die Vorhersagegenauigkeit der Validierungsdaten erhöht hat. Abbildung 19 zeigt für Versuch 2 ebenfalls eine kurze Zusammenfassung des Lernmodells.

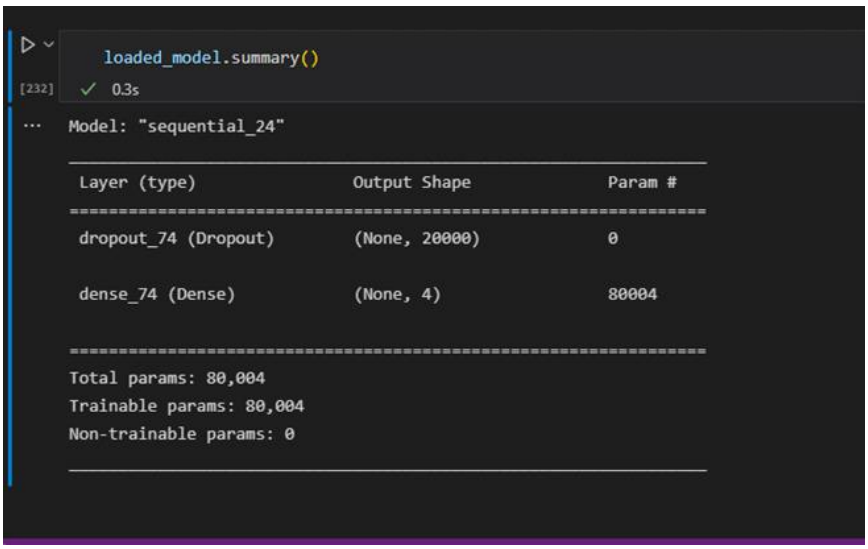

Abbildung 19 - Zusammenfassung Lernmodell aus Versuch 2

In Versuch 3 wird die Anzahl der Merkmale nochmal erhöht, auf 40.000 und gleichzeitig das Netzwerk zweischichtig aufgespannt. Auch diese Veränderung bringt eine Steigerung der Vorhersagegenauigkeit der Validierungsdaten. Ebenfalls zu beobachten ist eine Verbesserung der Dauer der Ausführung. Auch wenn in der aktuellen Aufgabenstellung die Genauigkeit wichtiger ist als die Dauer der Ausführung, gilt es das immer im Blick zu haben. Würde ein Versuch die höchste Genauigkeit erzielen, wäre er gegebenenfalls trotzdem nicht umsetzbar, falls die Ausführungszeit unverhältnismäßig lange wäre. Was genau „zu lange“ bedeutet, muss in jedem Aufgabenkontext neu geklärt werden und hängt stark von der Problemstellung und der Umgebung ab, in der sich das Lernmodell befindet. Auch für den Versuch 3 ist in Abbildung 20 eine Zusammenfassung dargestellt. Hier ist zu sehen, dass eine Zwischenschicht mit 64 Neuronen genutzt wurde. Diese Zwischenschicht entstand durch das Erhöhen der Netzwerkschichten auf zwei.

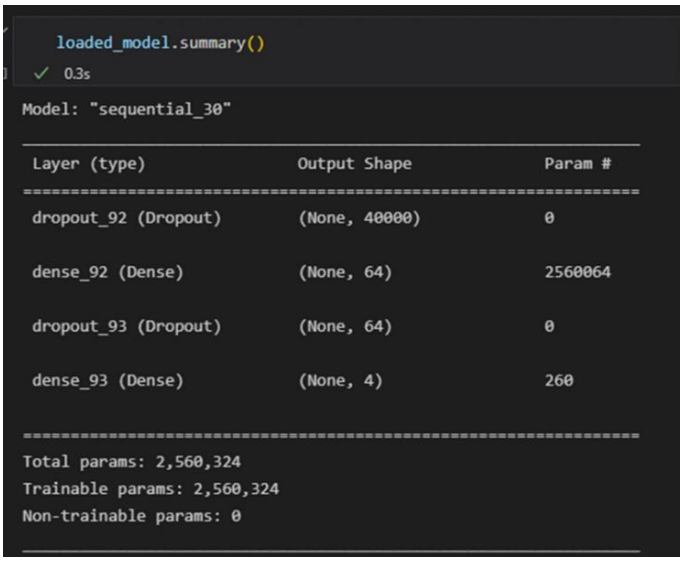

Abbildung 20 - Zusammenfassung Lernmodell aus Versuch 3

In Versuch 4 werden die Zwischenschichten auf 200 erhöht. Diese Erhöhung senkt die Vorhersagegenauigkeit des Lernmodells drastisch und verweist somit darauf, dass 200 Zwischenschichten zu komplex sind für die vorliegende Problemstellung. Die Komplexität erhöht die Ausführungs- und Analysezeit extrem und ist somit in der Praxis nicht nutzbar. In Abbildung 21 ist ein Teil der Zusammenfassung dargestellt. Es wurden nicht alle 200 Schichten aufgelistet und damit fehlt auch die Ausgabeschicht mit der Anzahl der Klassen. Diese betrug hier ebenfalls 4.

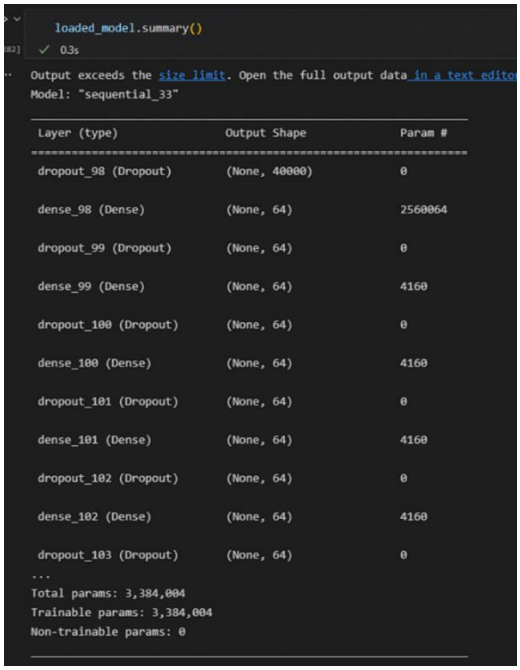

Abbildung 21 - Zusammenfassung Lernmodell aus Versuch 4

Ein zusätzlicher Versuch wurde dokumentiert, welcher besonders wenig Trainingsdaten beinhaltet, um die Auswirkung zu sehen. Die Aufteilung der Daten wird in Tabelle 13 gezeigt. Dabei wurden wieder circa 15% als Validierungsmenge eingeteilt und es wurde darauf geachtet, dass alle Klassen ähnlich Anteile am Gesamtdatensatz haben wie in der originalen Ausgangsmenge. (siehe Tabelle 10)

Das Ergebnis, welches in Tabelle 14 dargestellt wird, zeigt, dass die Vorhersagegenauigkeit der Validierungsdaten nur 40% erreicht. Die Trainingsdaten werden zu 96,47% vorhergesagt. Eine solche Vorhersagegenauigkeit der Validierungsdaten zeigt, dass die vorhandenen Datensätze nicht ausreichend sind, um ausreichend gut zu lernen, damit unbekannte Daten korrekt klassifiziert werden können.

Auf Grund der vorliegenden Ergebnisse wird empfohlen das Lernmodell mit den Parametern nach Versuch 3 aufzusetzen. Es ist einerseits ausreichend schnell in der Ausführung und andererseits erreicht es eine gute Vorhersagegenauigkeit der Validierungsdaten. Während mehrerer Versuchsreihen wurden ebenfalls andere Parameter angepasst. Es wurden in der vorliegenden Abschlussarbeit die dargestellten Ergebnisse aus dieser Versuchsreihe beispielhaft erläutert.

Tabelle 13 - Aufteilung des reduzierten Datensatzes nach Trainings- und Validierungsdaten sowie unterteilt nach den verschiedenen Klassen

Klasse	Trainingsdaten	Testdaten	Gesamt	Anteil Testdaten in %
BOM	32	6	38	15
CALC	9	2	11	15
PCSF	31	5	36	15
TECH	13	2	15	15
GESAMT	**85**	**15**	**100**	**15**

Tabelle 14 – Übersicht der Ergebnisse für Versuch 5, den genutzten Eingabeparametern und den Ergebnissen für die Vorhersagegenauigkeit

Versuch	Merkmale	Schichten	Genauigkeit Trainingsdaten in %	Genauigkeit Validierungs-daten in %	Dauer der Ausführung
5	40000	2	96,47	**40**	0.8 s

Während der Versuchsreihen ist aufgefallen, dass die Klassen unterschiedlich vertreten sind. Darauf muss generell geachtet werden, um keine Über- oder Unterrepräsentanz einer Klasse zu erhalten. Dies kann dazu führen, dass diese Datensätze aufgrund der wenigen Trainingsdaten öfter falsch klassifiziert werden als andere.

4 Zusammenfassung und Ausblick

Als Fazit für die vorliegende Abschlussarbeit werden alle gesammelten Fakten zum automatischen Klassifizieren von textueller Fehlermeldung zusammengefasst. Es wurde eingangs erörtert, dass sich diese Aufgabenstellung mit Hilfe eines überwachten Lernmodells gut lösen lässt. Gründe sind vor allem der vorliegende Datensatz, welcher bereits klassifiziert ist. Damit sind Eingangsdaten bekannt sowie die Klassen in die sie eingeteilt werden. Zum Aufbau eines geeigneten Lernmodells wurde zurück gegriffen auf den Aufbau eines neuronalen Netzes mit Hilfe verschiedener frei verfügbarer Softwarebibliotheken. Durch Anpassen von unterschiedlichen Parametern, welche die Netzwerktopografie, die Textanalyse sowie die Schnelligkeit des Lernmodells beeinflussen, konnte eine geeignete Konfiguration für die vorliegende Aufgabenstellung gefunden werden.

Eine Herausforderung, die in dieser Abschlussarbeit noch nicht angegangen wurde, ist eine voll automatisierte Datenbeschaffungsweise. Für die Daten der Versuchsreihen, wurde ein bereits vorhandenes Programm genutzt, um alle gewünschten Dateien herunterzuladen. Diese Vorgehensweise kann so übernommen werden. Allerdings sind die hinterlegten Zuordnungen der Teambereiche zu den Klassen hier fix hinterlegt. In der Realität passen sich die Teambereiche an, benennen sich um, fassen sich zusammen oder verschwinden. Dies würde in der aktuellen Form immer eine manuelle Anpassung bedeuten. Hier sollte zukünftig investiert werden, um einen Weg zu finden, auf solche Veränderungen automatisiert zu reagieren. Ebenso muss das Zugreifen auf die Dateien voll automatisiert und so gestaltet werden, dass nur neu erstellte Datensätze analysiert werden.

Das Lernmodell hat in den durchgeführten Versuchen bei jedem Versuch neu gelernt. Aus diesem Grund war es auch sinnvoll, die einzelnen Ausführzeiten mit zu betrachten. Für die endgültige Implementierung kann es sinnvoll sein, auf inkrementelles Lernen zu setzen. Hier würde bereits Erlerntes nicht wieder gelöscht werden, und neue Daten werden in ein bestehendes Lernmodell integriert.

Für die Datenvorbereitung können noch Schritte zur Datenbereinigung eingeführt werden. Dies wird die Performance des Lernmodells womöglich nochmals verbessern. Die Textdateien beinhalten oft Absätze mit nutzerspezifischen Informationen, wie Name, Rechnername, E-Mail Adresse, die für Auswertungen irrelevant sind.

Weiterhin gibt es Arbeitselemente, die keine textuellen Fehlermeldungen beinhalten. Diese waren nicht Teil dieser Abschlussarbeit, sollten aber langfristig ebenfalls in einen automatisierten Prozess eingebunden werden können. Hierfür müssen andere Textfelder aus den Arbeitselementen als Merkmalsextraktion definiert werden.

Abschließend soll noch einmal der Gedanke des unüberwachten Lernens aufgriffen werden. Dieser Versuch wäre als Experiment zu betrachten und ist wahrscheinlich nicht geeignet, wenn eine kurzfristige Umsetzung der Automatisierung gewünscht ist. Dennoch kann diese Betrachtung neue Erkenntnisse sowie andere Blickwinkel hervorbringen. Die textuellen Fehlermeldungen werden in dem Fall des unüberwachten Lernens analysiert, ohne das eine Ausgabe in vordefinierte Klassen bekannt ist. Rein aufgrund einer Ähnlichkeitsanalyse, zum Beispiel durch Clustering, werden die Dateien in Cluster gruppiert. Welche Clustergruppen dabei erkannt werden, kann eventuell neue Erkenntnisse über eine andere Einteilung der Teambereiche oder deren Unterbereiche bringen.

Literaturverzeichnis

[1] „Wikipedia,“ [Online]. Available: https://de.wikipedia.org/wiki/Maschinelles_Lernen. [Zugriff am 04 06 2022].

[2] A. Burkov, MACHINE LEARNING KOMPAKT, Frechen: mitp Verlags GmbH, 2019.

[3] d. GmbH, „datasolut,“ [Online]. Available: https://datasolut.com/was-ist-machine-learning/#machine-learning-arten. [Zugriff am 16 07 2022].

[4] datasolut GmbH, „datasolut,“ [Online]. Available: https://datasolut.com/wiki/supervised-learning/. [Zugriff am 16 07 2022].

[5] d. gmbH, „datasolut,“ [Online]. Available: https://datasolut.com/reinforcement-learning/. [Zugriff am 18 07 2022].

[6] A. Wolf, „Machine learning simplified: A gentle introduction to supervised learning,“ 2022.

[7] U. Q. T. W. Gerhard Heyer, Text Mining: Wissensrohstoff Text, Witten: W3L GmbH, 2006.

[8] „Wikipedia,“ [Online]. Available: https://de.wikipedia.org/wiki/N-Gramm. [Zugriff am 10 07 2022].

[9] „Wikipedia,“ [Online]. Available: https://de.wikipedia.org/wiki/Clusteranalyse. [Zugriff am 02 07 2022].

[10] M. I. o. Technology, Massachusetts Institute of Technology, 2016. [Online]. Available: https://ocw.mit.edu/courses/6-0002-introduction-to-computational-thinking-

and-data-science-fall-2016/resources/lecture-11-introduction-to-machine-learning/. [Zugriff am 01 07 2022].

[11] K.-U. S. Gunter Saake, Algortihmen und Datenstrukturen, Heidelberg: dpunkt.verlag GmbH, 2021.

[12] „Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Verlustfunktion. [Zugriff am 16 06 2022].

[13] s.-l. developers, „scikit learn," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#. [Zugriff am 19 07 2022].

[14] „Machine Learning," [Online]. Available: https://developers.google.com/machine-learning/guides/text-classification/step-4. [Zugriff am 19 07 2022].

[15] „Wikipedia," [Online]. Available: https://de.wikipedia.org/wiki/Kreuzentropie. [Zugriff am 06 08 2022].

[16] J. Brownlee, „machine learning mastery," [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Zugriff am 06 08 2022].

Anlagen

Anlagen, Teil 1

Als Entwicklungsumgebung stand Visual Studio Professional 2019 und Visual Studio Code zur Verfügung. Liste aller genutzen Softwarebibliotheken:

- TensorFlowCore v2.9.1
- Scikit-Learn v1.1.2
- Matplotlib v3.5.0
- Numpy v1.23.1
- Python v3.10.5
- absl-py v1.1.0
- astor v0.8.1
- astroid v2.12.2
- autopep8 v1.6.0
- bleach v5.0.1
- colorama v0.4.5
- gast v0.5.3
- graphene v3.1
- graphlql-core v3.2.1
- graphlql-relay v3.2.0
- grpcio v1.47.0
- html5lib v1.1
- isort v5.10.1
- lazy-object-proxy v1.6.0
- Markdown v3.4.1
- mccabe v0.7.0
- pandas v1.4.3
- promise v8.1.0
- protobuf v4.21.4
- pycodestyle v2.9.1
- pylint v2.14.5
- python-dateutil v2.8.2
- pytz v2022.1
- rope v1.3.0
- Rx v3.2.0
- six 1.16.0
- tensorboard v2.9.1
- termcolor v1.1.0
- typing v3.10.0.0
- Werkzeug v2.2.1
- wrapt v1.14.1

Anlagen, Teil 2

Teil 2 der Anlagen zeigt den Hauptteil des Programmes zum Herunterladen der bereits klassifizierten textuellen Fehlermeldungen. Unter Berücksichtigung der bereits zugeordneten Klassen werden die Dateien in entsprechende Ordner abgelegt.

```
….
{
        var witClient = connection.GetClient<WorkItemTrackingHttpClient>();
        var queries = witClient.GetQueriesAsync("TcPCM", depth: 2).Result;
        var myQueries = queries.FirstOrDefault(qhi => qhi.Name.Equals("My Queries"));
        if (myQueries == null)
        {
           Console.WriteLine("No work items were returned from query.");
        }

        var bugsQuery = myQueries.Children.FirstOrDefault(qhi => qhi.Name.Equals("New query
test"));
        var result = witClient.QueryByIdAsync(bugsQuery.Id).Result;
        if (!result.WorkItems.Any()) return;

        var skip = 0;
        const int batchSize = 100;
        const string baseDir = "C:\\Temp\\data\\download";
        while(true)
        {
           var workItemIds = result.WorkItems.Skip(skip).Take(batchSize).Select(wir => wir.Id);
           if (!workItemIds.Any()) return;

           var workItems = witClient.GetWorkItemsAsync(workItemIds, null, null,
WorkItemExpand.Relations).Result;
           foreach (var workItem in workItems)
           {
              var areaPath = workItem.Fields["System.AreaPath"];
              var folder = ToFolder((string)areaPath);
              if (folder == null) continue;

              var directory = Path.Combine(baseDir, folder);
              foreach (var item in workItem.Relations)
              {
                 if (item.Rel != "AttachedFile") continue;
                 var name = item.Attributes["name"].ToString();
                 if (name.Contains("Hole")) continue;
                 if (!name.Contains("Stacktrace") || !name.Contains(".txt")) continue;

                 var splitString = item.Url.Split('/');
                 var attachmentId = new Guid(splitString[8]);
```

```
                    using (var attachmentStream = witClient.GetAttachmentContentAsync(attachmentId).Result)
                    {
                        var fileFullPath = Path.Combine(directory, $"{Guid.NewGuid():N}.txt");
                        using (var writeStream = new FileStream(fileFullPath, FileMode.Create, FileAccess.ReadWrite))
                        {
                            attachmentStream.CopyTo(writeStream);
                        }
                    }
                }
            }

            Console.WriteLine($"Progress: {skip}");
            skip += batchSize;
        }
    }
}

private static string ToFolder(string areaPath)
{
    if (areaPath.Contains("BOMAPI")) return "bomapi";
    if (areaPath.Contains("Profitability")) return "calc";
    if (areaPath.Contains("Platform")) return "pcsf";
    if (areaPath.Contains("Technology")) return "tech";
    return null;
}
…
```

Anlagen, Teil 3

In Teil 3 des Anhangs werden die Funktionen zum Tokeniseren der Daten und zum Aufbau des Lernmodells dargestellt.

```
##Tokenize

TOP_K_FEATURES = 20000

def ngram_vectorize(train_traces, train_labels, test_traces):
    NGRAM_RANGE = (1, 1)

    kwargs = {
            'ngram_range': NGRAM_RANGE,
            'strip_accents': 'unicode',
            'decode_error': 'replace',
    }
    vectorizer = TfidfVectorizer(**kwargs)
    x_train = vectorizer.fit_transform(train_traces)
    x_test = vectorizer.transform(test_traces)

    selector = SelectKBest(f_classif, k=min(TOP_K_FEATURES, x_train.shape[1]))
    selector.fit(x_train, train_labels)
    x_train = selector.transform(x_train).astype('float32')
    x_test = selector.transform(x_test).astype('float32')
    return x_train, x_test, selector, vectorizer

## Build

def build_mlp(num_layers, units, dropout, input_shape, num_classes):
    model = models.Sequential()
    model.add(Dropout(rate=dropout, input_shape=input_shape))
    for _ in range(num_layers - 1):
        model.add(Dense(units=units, activation='relu'))
        model.add(Dropout(rate=dropout))

    model.add(Dense(units=num_classes, activation='softmax'))
    return model
```

Anlagen, Teil 4

In Teil 4 des Anhangs werden die Funktionen zum Trainieren und Validieren der Datensätze unter Nutzung des Lernmodells und der tokenisierten Daten dargestellt. Ebenso wird das erzeugte Modell als Zusammenfassung am Ende dargestellt.

```
#Train

def train_ngram_model(data, learning_rate=0.003, epochs=200, batch_size=128, layers=2, units=64, dropout=0.2):
        (train_traces, train_labels), (test_traces, test_labels) = data
        x_train, x_val, selector, vectorizer = ngram_vectorize(train_traces, train_labels, test_traces)
        model = build_mlp(layers, units, dropout, x_train.shape[1:], 4)
        loss = 'sparse_categorical_crossentropy'
        optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
        model.compile(optimizer=optimizer, loss=loss, metrics=['acc'])
        callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)]
        history = model.fit(x_train.toarray(), train_labels, epochs=epochs, callbacks=callbacks, validation_data=(x_test.toarray(), test_labels), verbose=2, batch_size=batch_size)
        history = history.history
        print('Validation accuracy: {acc}, loss: {loss}'.format(acc=history['val_acc'][-1], loss=history['val_loss'][-1]))
        model.save('areapath_model.h5')
        dump(vectorizer, 'vectorizer.pkl')
        dump(selector, 'selector.pkl')
        return history['val_acc'][-1], history['val_loss'][-1]

        data = ((train_df["traces"], train_df["identifier"]), (test_df["traces"], test_df["identifier"]))
         data[0]
#Show results
        result = train_ngram_model(data)

# Reload for summary

loaded_model = tf.keras.models.load_model('areapath_model.h5')
loaded_vectorizer = load('vectorizer.pkl')
loaded_selector = load('selector.pkl')

loaded_model.summary()
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Wernau, den 07.08.2022