HAW HAMBURG

BACHELOR THESIS
Marcel Jankowski

# Ähnlichkeitsanalyse zur Bewertung von Softwarefehlern mit Methoden der Data Science

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Marcel Jankowski

# Ähnlichkeitsanalyse zur Bewertung von Softwarefehlern mit Methoden der Data Science

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bettina Buth
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 23.02.2023

**Marcel Jankowski**

**Thema der Arbeit**

Ähnlichkeitsanalyse zur Bewertung von Softwarefehlern mit Methoden der Data Science

**Stichworte**

Ähnlichkeitssuche, Data Science, Softwarefehler

**Kurzzusammenfassung**

Das Ziel dieser Arbeit ist die Untersuchung von Methoden der Data Science für die Ähnlichkeitssuche in Fehlermeldungen. Die Motivation ist die Unterstützung von Testern bei der Fehleranalyse. Bei einer aufgetretenen Fehlersituation eines Tests soll es einem Tester möglich sein, nach ähnlichen Fehlermeldungen in einer Datenbank zu suchen. Wenn eine ähnliche bereits gelöste Fehlersituation in der Vergangenheit aufgetreten ist, kann diese Lösung gegebenenfalls auf dem aktuellen Problem angewendet werden. Als reales Beispiel dienen über 400 000 gesammelte Fehlermeldungen des Softwareherstellers für Versicherer, msg life ag.
Für eine passende Problemlösung werden mehrere Methoden der natürlichen Sprachverarbeitung genutzt und analysiert. Eine bewährte Technik für den Vergleich von Texten ist die Verwendung von Methoden, die Texte in Vektoren umrechnen. Diese Vektoren können mit einer Vergleichsmetrik verglichen werden. Zwei in dieser Arbeit untersuchte Ansätze sind TF-IDF und SBERT. Mittels mehrerer Experimente wird die Leistung beider Methoden untersucht. Als Vergleichsmetrik wird die Kosinusähnlichkeit verwendet.
Die Ergebnisse der Experimente zeigen, dass SBERT fachlich die besseren Ergebnisse aufweist. Hingegen erweist sich TF-IDF bezogen auf den Rechenaufwand und die Speicherkapazität als die kostengünstigere Variante. Abschließend wird die SBERT Methode empfohlen. Diese Methode hat zwar einen erhöhten Aufwand bei ihrer Verwendung. Allerdings hat sie, im Vergleich zur TF-IDF Methode, eine höhere Erfolgsquote für die Ähnlichkeitssuche nach Fehlermeldungen.

**Marcel Jankowski**

**Title of Thesis**

Similarity analysis for assessing software defects using data science methods

**Keywords**

Similarity Search, Data Science, Errormessage

**Abstract**

The goal of this work is to investigate Data Science methods for similarity search in error messages. The motivation is to support testers in error analysis. When an error situation of a test occurs, it should be possible for a tester to search for similar error messages in a database. If a similar and already solved error has occurred in the past, its‘ solution could be applied to the current problem. Over 400.000 collected error messages of the software producer for insurance companies, msg life ag, serve as a real example.
Several natural language processing methods are used and analyzed for a suitable problem solution. A proven technique for the comparison of texts is the use of methods that convert texts into vectors. These vectors can be compared using a similarity metric. Two approaches studied in this work are TF-IDF and SBERT. Through several experiments, the performance of both methods is investigated. The cosine similarity is used as the comparison metric.
The results of the experiments show that SBERT has technically the better results. On the other hand, TF-IDF proves to be the more cost-effective variant in terms of computational effort and storage capacity. In conclusion, the SBERT method is recommended. This method does have an increased effort in its use. However, compared to the TF-IDF method, it has a higher success rate for similarity search for error messages.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

## 1.1 Problemstellung und Motivation

Im Arbeitszyklus eines Softwareprojekts entstehen oft Fehler in der Programmierung. Diese werden im Idealfall durch Werkzeuge, wie Compiler oder Softwaretests in Form einer Fehlermeldung gefunden und aufgezeigt. Der Prozess, indem der Fehler analysiert und vom Programm bereinigt wird, nennt sich Debugging. Das Debugging fordert einen großen Teil der Zeit in einem Projekt. In einer Studie [1], in der die Programmierzeit verschiedener Softwareentwickler untersucht wurde, betrug die durchschnittliche Zeit des Debuggen etwa die Hälfte der gesamten Programmierzeit. Eine bewährte Methodik für die Fehlerbeseitigung, die von Einsteigern als auch Experten verwendet wird, ist die Suche nach gleichen oder ähnlichen Fehlersituationen in Web-Ressourcen. Somit sollen möglichst schnell Lösungen gefunden werden. Ziel der Suche nach bereits aufgetretenen Fehlersituationen- und Lösungen ist, dass der aufgetretene Fehler bereits bekannt ist und im besten Fall schon gelöst wurde. Beispielhaft für eine solche Web-Ressource ist das Stack Overflow Q&A Forum. Das Forum verwaltet ein umfangreiches Archiv von Fragen und Antworten zu vielen Bereichen des Themas IT. Ein Großteil der Entwickler weltweit benutzen diese Ressource und besuchen sie über 100 Millionen mal monatlich[1]. Mittels solcher Anwendungen wird ein Kollektivwissen nutzbar gemacht. So ist es jeder Person mit einem Zugang zu Internet möglich, dieses Kollektivwissen zu erreichen und es zu erweitern. Dieses ist dadurch entstanden, dass in der Vergangenheit die Benutzer die Fragen erstellt und wiederum Andere diese beantwortet haben. Problematisch wird diese Suche jedoch, wenn die Fehlermeldungen von privaten oder industriellen Systemen stammen. Dafür kann keine öffentliche Online Ressource verwendet werden, da diese Daten nicht öffentlich bekannt sind. Hier sind die Fehlermeldungen nur bei den Entwicklern in dem Unternehmen bekannt. Es stellt sich die Frage nach einem System wie Stack Overflow

[1] https://stackoverflow.co/advertising/audience

Q&A, welches jedoch den Bereich der internen und nicht öffentlich zugänglichen Fehlermeldungen abdeckt. Mithilfe eines solchen Systems wäre es Mitarbeitern zum Beispiel möglich, systeminterne Fehlersituationen des Unternehmens einzusehen, die der gegenwärtig auftretenden Fehlersituation ähnelt oder sogar gleicht. Ist der gefundene Fehler bereits bekannt oder gelöst, kann dessen Lösung auf das gegenwärtige Problem angewendet werden. Eine funktionierende Lösungsanwendung könnte die Zeit der Lösungssuche und Fehlerbehebung deutlich verringern.

## 1.2 Zielsetzung

Das Ziel dieser Bachelorarbeit ist die Untersuchung und Verwendung von Methoden, welche notwendig wären, um ein zuvor beschriebenes System zu erstellen. Der Fokus soll dabei auf der Suche von Ähnlichkeiten liegen. Anhand von Methoden, die Ähnlichkeiten zweier Fehlermeldungen berechnen können, sollen vergleichbare, bereits aufgetretene Fehlersituationen gefunden werden. Als reales Beispiel dienen die Fehlermeldungen des Softwareherstellers für Versicherer msg life ag[2] (im Folgendem "msg life" genannt). In Zusammenarbeit mit dem Unternehmen wird ein bereits gegebener Datensatz des Unternehmens von über 400 000 Fehlermeldungen untersucht. Diese Fehlermeldungen stammen aus Regressionstests, welche nahezu täglich ausgeführt werden und beinhalten mehrere Tausend einzelne Tests. Sie stellen sicher, dass durch neue Änderungen bestehende Funktionalität nicht verändert wird. Aus diesen Tests können Fehlermeldungen mehrerer Komponenten des gesamten Systems der msg life auftreten. Zwecks Findung einer adäquaten Lösung, werden mithilfe von wissenschaftlichen Publikationen die verschiedenen Ansätze zur Erstellung eines solchen Systems untersucht und evaluiert.
Die Bachelorarbeit kann als erfolgreich angesehen werden, wenn sie verschiedene Kriterien erfüllt. Ein wichtiger Faktor für den Erfolg der Arbeit ist eine umfassende Literaturrecherche zu den Themengebieten der Data Science, die mit den Problemen und Lösungen der Forschungsfrage verbunden ist. Daneben muss die Ausgangssituation so erfasst werden, dass konkrete Anforderungen an die Lösung erschlossen werden können. Schlussendlich soll diese Lösung in einem letzten Schritt bewertet werden. Falls die Lösung nicht in der Praxis anwendbar ist, müssen die Gründe dafür klar ausgelegt werden. Wenn diese Kriterien erfüllt sich, gilt diese Bachelorarbeit als erfolgreich.

[2] https://www.msg-life.com/

## 1.3 Überblick über den Aufbau der Arbeit

Zum Beginn der Arbeit werden die in Kapitel 2 relevanten Grundlagen erklärt. Es werden dort die Begrifflichkeiten der Themenbereiche beschrieben und die Techniken erläutert, die für die Lösung des Problems notwendig sind. In dem nachfolgenden Kapitel 3 werden verwandte Ansätze und Lösungen vorgestellt. Hierbei wird die Tauglichkeit der Techniken aus den Grundlagen aufgezeigt. Die Ansätze werden nach Unterschieden und Gemeinsamkeiten zu dem aktuellen Problem untersucht. Das Kapitel 4 enthält eine Analyse des Ausgangspunktes und leitet die wesentlichen Anforderungen für die verwendeten Techniken sowie die Gesamtlösung ab. Der Entwurf in Kapitel 5 beinhaltet die Architektur der Lösung und beschreibt die Einbettung des Systems in den Workflows der Mitarbeiter. Um eine adäquate Lösung zu finden, werden in Kapitel 6 verschiedene Techniken experimentell miteinander verglichen. Eine Auswertung, in der die Anforderungen der Lösung gegenübergestellt werden, findet sich in Kapitel 7. Das letzte Kapitel 8 schließt die Arbeit mit einem Fazit und Ausblick ab.

# 2 Grundlagen

In diesem Kapitel werden die für diese Arbeit relevanten Grundlagen geklärt. Dazu gehören die notwendigen Werkzeuge, Begriffe und Ansätze für die Erstellung einer Lösung. Das Kapitel dient zur Verständlichkeit der nachfolgenden Kapitel.

## 2.1 Machine Learning

Für die Lösung des vorgestellten Problems werden Methoden des maschinellen Lernens (engl. Machine Learning, kurz ML) verwendet. Svensson und Söderberg definierten 2008 den Begriff wie folgt:

> "Machine learning (ML) is concerned with the design and development of algorithms and techniques that allow computers to 'learn'. The major focus of ML research is to extract information from data automatically, by computational and statistical methods." [17]

Ein Ziel der ML ist folglich die automatische Extraktion von bestimmten Informationen aus Daten. Wichtig in dem Kontext dieser Definition ist das "automatically". Bei einem großen Datensatz von über 400 000 Objekten ist es aufwändig jedes Datenobjekt einzeln zu behandeln. Die Idee ist es, mit einem Algorithmus die Informationen aus allen Objekten zu sammeln, ohne dass manuell eingegriffen wird.
Passend dazu ist Arthur Samuels Definition aus 1959:

> "[Machine Learning is the] field of study that gives computers the ability to learn without being explicity programmed" [12].

Der Programmcode einer ML Applikation zielt nicht auf jedes einzelne Objekt ab, sondern "lernt", welche Informationen extrahiert werden sollen.
Ein bekanntes Beispiel nach [7] ist der Spamfilter, welcher E-Mails, die zum Beispiel ungewünschte Werbung enthalten, als Spam markiert. Eine vorgeschlagene Lösung mit

traditionellen Programmiertechniken würde mit einer Liste von Schlagwörtern arbeiten. Wenn eine E-Mail diese Schlagwörter enthält, würde diese als Spam gekennzeichnet werden. Ist das Ergebnis nicht zufriedenstellend, muss analysiert werden, welche Schlagwörter fehlen. Der Prozess wird solange durchgeführt, bis der Spamfilter das erwünschte Ergebnis liefert.
In einem Ansatz mit maschinellem Lernen würde ein Algorithmus versuchen anhand der Trainingsdaten das Muster der Spamnachrichten zu erkennen. Somit würde das System intern eine Liste an Schlagwörter beziehungsweise auch die Zusammensetzung dieser erstellen. Es würde kein Eingriff des Programmierers notwendig sein.

Die nachfolgenden Informationen über die Arten des maschinellen Lernens und dessen Beispiele sind aus diesem Lehrbuch [7] zu entnehmen. Das maschinelle Lernen kann in die folgenden drei Arten unterteilt werden: unüberwachtes, halbüberwachtes oder überwachtes Lernen. Hauptsächlich unterscheiden sich die Arten sich darin, ob und wie stark der Lernprozess von einem Menschen überwacht und unterstützt wird.
Bei dem überwachten Lernen muss dem Algorithmus eine vorher festgelegte Lösung gegeben werden. Die Daten, mit dem der Algorithmus trainiert, müssen also vorher gekennzeichnet werden. Diese Kennzeichnung nennt sich Label. Bezogen auf das Beispiel des Spamfilters sind die E-Mails, mit denen der Algorithmus trainiert, die Trainingsdaten. Die Information, ob eine E-Mail Spam ist oder nicht, ist das Label. Eine typisches Aufgabengebiet des überwachten Lernens ist die Klassifikation. Der Algorithmus lernt zuerst, zu welcher Kategorie ein Element gehört, sodass es dann neue Elemente einordnen kann.

Das Gegenstück zum überwachten Lernen ist das unüberwachte Lernen. In diesem Fall enthalten die Trainingsdaten keine vorher festgelegte Lösung. Das System lernt, ohne dass ein Sollwert vorgegeben wird. Ein bekannter Anwendungsfall des unüberwachten Lernens ist das Clustering. Es hier versucht aus den Daten Gruppen zu identifizieren, ohne dass diese bereits vorher bekannt sind. Damit können beispielsweise Online-Plattformen herausfinden, welche typischen Nutzerprofile es gibt und welche Eigenschaften diese haben. Weitere Anwendungsfälle des unübewachten Lernens sind Visualization und Association Rule Learning.

Eine Verbindung des überwachten und unüberwachten Lernens ist das halbüberwachte Lernen. Ein Algorithmus wird im maschinellen Lernen als halbüberwachtes Lernen bezeichnet, wenn nur ein Teil der Daten ein Label besitzt. Oftmals sind die Algorithmen des halbüberwachten Lernens eine Kombination aus Algorithmen des überwachten sowie

unüberwachten Lernens.
Ein bekanntes Beispiel ist Erkennung von Personen auf Bildern in Google Photos. Zuerst findet eine Gesichtserkennung aller Fotos mit Menschen statt. Alle Bilder mit einer bestimmten Person werden einem Cluster zugeordnet. Dieser Teil ist unüberwacht. Danach muss ein Mensch manuell die Cluster benennen beziehungsweise die Namen der Personen auf jeweils einem Bild eintragen. Dieser Teil ist überwacht, da ein Labeling der Daten stattfindet. Schlussendlich können nun Namen von Personen eingeben werden und die Applikation gibt alle Bilder dieser Person aus.

Bezogen auf den Anwendungsfall dieser Arbeit kann es nun sinnvoll sein, dass die Techniken des maschinellen Lernens verwendet werden. Um die Fehlermeldungen effizient zu vergleichen bedarf es einen Algorithmus, welcher Kernelemente der Fehlermeldungen automatisch extrahiert, sodass diese miteinander verglichen werden können. Wichtig in diesem Kontext ist wieder das Wort ”automatisch”. Welche Bestandteile der Fehlermeldungen wie relevant für einen Vergleich sind, könnte in einem traditionellen Programmieransatz der Entwickler oder ein Fachexperte darlegen. In einem Ansatz des maschinellen Lernens, könnte hier der Algorithmus - wie bei einem Spamfilter - die relevanten Attribute selbst erkennen, die für einen Vergleich notwendig sind.
Es lässt sich außerdem bereits zu Anfang dieser Arbeit sagen, dass das manuelle Labeling der Daten ausgeschlossen wird. Dies liegt daran, dass diese Aufgabe sehr aufwändig ist, da vorher beschrieben werden müsste, inwieweit eine Fehlermeldung einer Anderen ähnelt. Dazu müsste jede Fehlermeldung im Trainingsdatensatz mit jeder weiteren verglichen werden. Es wird dennoch nicht von Grund auf abgesehen, Methoden des überwachten Lernens zu verwenden. Denn es können bereits trainierte Algorithmen verwendet werden, welche kein weiteres Labeling der Daten voraussetzen. Ein Beispiel sind die vortrainierten Modelle des SBERT Algorithmus in 2.5.1.
Auf der anderen Seite können ebenso die Methoden des überwachten und halbüberwachten Lernens in der Arbeit verwendet werden. Ein unüberwachter Algorithmus im maschinellen Lernen ist der TF-IDF, welcher in 2.4 weiter beleuchtet wird. Beide genannten Ansätze sind gleichzeitig Teil des maschinellen Lernens und der natürlichen Sprachverarbeitung.

## 2.2 Natural Language Processing

Natural Language Processing (kurz NLP) ist ein Themengebiet der Informatik und Computerlinguistik. Es befasst sich mit der Interaktion von Computer- und natürlicher Sprache. Der Begriff natürliche Sprache bezeichnet die menschlichen Sprachen, wie Englisch und Deutsch und dient zur Unterscheidung von den Computersprachen, wie C++ und Java [13]. Das Ziel der NLP ist die Entwicklung von Methoden, die praktische Probleme der Sprachverarbeitung lösen.
Beispiele für Anwendungsgebiete sind: Informationsextraktion, maschinelle Übersetzung und semantische Textähnlichkeit [4].

Die semantische Textähnlichkeit (engl. semantic text similarity, kurz STS) ist nach [19] ein Anwendungsfall der NLP, der sich mit den Problemen dieser Arbeit beschäftigt. Mit STS wird die inhaltlichen Ähnlichkeit zwischen zwei Texten ermittelt. Mit den NLP-Techniken wird versucht den Grad der Ähnlichkeit dieser Texte zu berechnen. Typischerweise wird diese Ähnlichkeit durch einen numerischen Werte oder einer Distanzmetrik dargestellt. Semantisch Ähnlich können beispielsweise zwei Nachrichtenartikel über das gleiche Geschehen sein. Beide Texte beschreiben das gleiche Thema, unterscheiden sich jedoch in den verwendeten Wörtern und in der unterschiedlichen Beleuchtung von Details. Haben zwei Texte eine dieselbe semantische Bedeutung, so ist von semantischer Gleichheit die Rede. Ein Gegenstück zu der semantischen Ähnlichkeit ist die Lexikalische. Hierbei werden die exakten Zeichen und Wörter miteinander verglichen. Ein Beispiel für eine semantische Gleichheit ist das Wort ”Support Vector Machine” und dessen Abkürzung ”SVM”. Die Wörter unterscheiden sich zwar lexikalisch, haben jedoch dieselbe Bedeutung. Beispiele für konkrete Anwendungsfälle sind:

- Vergleich von Nachrichtenartikeln von unterschiedlichen Produzenten [24]
- Clustering von arabischen Textdokumenten [10]
- DocBERT: Klassifikation von Dokumenten [2]
- Suche nach ähnlichsten Elementen zu einer Anfrage (Web Search) [14]

In dieser Arbeit werden Ansätze aus dem Bereich der STS untersucht. Ziel ist es, semantisch ähnliche Fehlermeldungen zu ermitteln, auch wenn diese nicht lexikalisch ähnlich sind. Mit den Lösungen für die STS soll ermittelt werden, ob Fehlermeldungen semantisch ähnliche Begriffe enthalten.

## 2.3 Einbettungen

Die Verwendung von Einbettungen (engl. Embeddings) ist nach [4] ein Kern des Erfolgs der NLP. Das Ziel ist es, diskrete Typen in einem relativ niedrigdimensionalen Vektor darzustellen. Diese diskreten Typen können beispielsweise Wörter oder Sätze sein. Mit dafür ausgelegten Algorithmen werden Wörter auf einen Punkt in einem Vektorraum abgebildet. Diese Vektoren werden Einbettungen genannt. Nachdem die Einbettungen erstellt wurden, können diese für verschiedene Ansätze der NLP verwendet werden. Beispiele sind hier das Clustering sowie die Suche nach semantischen Textähnlichkeiten.

In der Abbildung 2.1 ist ein primitives Beispiel für eine Worteinbettung zu sehen. In dem linken Teil der Abbildung sind Wörter zu sehen, welche in einen zweidimensionalen Vektor eingebettet wurden. Die Werte der Vektoren sind frei erfunden und dienen für Demonstrationszwecke. Beide Werte können durch eine Zusammensetzung mehrerer Eigenschaften der Wörter berechnet worden sein. Der rechte Teil der Abbildung zeigt ein Koordinatensystem, indem mittels der Einbettung positioniert wurden. Die Y-Achse entspricht dem ersten Wert der Einbettung, während die X-Achse den zweiten Wert der Einbettung repräsentiert. Beide Achsen haben keine Beschriftung, da die Werte mehrere Attribute repräsentieren sollen, die in einer Zahl zusammengesetzt ist. Die Abbildung

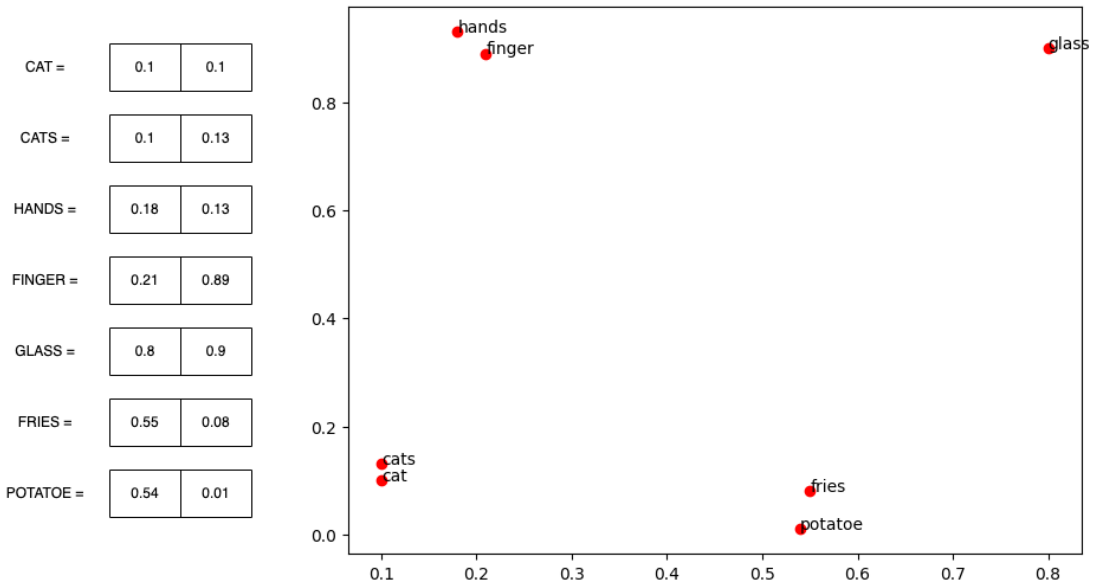


Abbildung 2.1: Beispiel für Worteinbettungen (eigene Darstellung)

von einem Text zu einer Einbettung schwankt je nachdem, welcher Algorithmus verwen-

det wird. Die Einbettungsmethode TF-IDF arbeitet beispielsweise mit den Häufigkeiten der Worte. Die Relevanz eines Wortes wird daran gemessen, wie oft es vorkommt. Ganz anders arbeitet ein SBERT. Hier wurde ein neuronales Netz trainiert, welches diese Einbettungen erstellt. Die Kernidee beider Methoden ist es, die Texte so darzustellen, dass die Distanzen im Vektorraum maßgeblich für den Grad der Ähnlichkeit sind. Im gleichen Maße wird dies im rechten Teil der Abbildung 2.1 gezeigt. Die Wörter "cat" und "glass" sollen weit voneinander entfernt sein, weil sie keinen semantischen Inhalt aufweisen. Dahingegen ist die Distanz der beiden Wörter "fries" und "potatoe" kleiner, da beides Lebensmittel sind.

Werden Wörter oder Sätze eingebettet, handelt es sich um Wort- und Satzeinbettungen. In dieser Arbeit werden ausschließlich diese Arten von Einbettungen verwendet und im Folgendem unter den Begriff Einbettungen zusammengefasst.

Einbettungsmethoden können nach [9] in die folgenden zwei Arten aufgeteilt werden: Dynamische und statische Einbettungsmethoden. Im Grunde liegt der Unterschied darin, ob in der Eingabe der Kontext berücksichtigt wird oder nicht. Die statischen Einbettungsmethoden, wie Word2Vec [18] lernen feste Repräsentationen von Wörtern. Diese werden aus einem großen Korpus an Textdaten gelernt und nicht für neue Eingabebeispiele aktualisiert. Dadurch, dass die Repräsentationen festgelegt sind, ist es einfacher diese zu berechnen.
Auf der anderen Seite gibt es die dynamischen Einbettungsmethoden, wie zum Beispiel SBERT. Diese Modelle erzeugen kontext-sensitive Einbettungen. Somit wird der Kontext des Textinhalts besser dargestellt, als bei statischen Einbettungsmethoden.

Wenn nun diese Idee auf die Problemstellung dieser Arbeit angewandt wird, können die einzelnen Fehlermeldungen mit dafür bestimmten Methoden auf Einbettungen abgebildet werden. Dies ist vor allem notwendig, wenn weitere Berechnungen getätigt werden, die Worteinbettungen als Eingabe benötigen. Ein Beispiel für einen solche Berechnung ist die Funktion der Kosinusdistanz, die in 2.7 beschrieben ist. Hier wird zwischen zwei Einbettungen eine Distanz berechnet, die ein Maß dafür geben soll, wie ähnlich diese Einbettungen sind. Außerdem finden sich in den Abschnitten 2.4 und 2.5 Details zu den Einbettungsmethoden TF-IDF und BERT.

## 2.4 TF-IDF

Bei der Suche nach einer Methode für einen Vergleich von Dokumenten beschreiben mehrere Paper die Nutzung von TF-IDF als Einbettungsmethode [21] [10] [20]. In den verschiedenen Ansätzen, die vor allem das Ziel haben, die Ähnlichkeit von Texten zu untersuchen, wird TF-IDF verwendet oder betrachtet. Obwohl die Ansätze für diese Technik bereits 1957 [16] und 1972 [11] entworfen worden sind, wird sie heutzutage weiterhin verwendet. Ein Vorteil von TF-IDF gegenüber von aktuelleren Einbettungsmethoden basierend auf neuronalen Netzen ist die Geschwindigkeit der Berechnung, da die Erstellung von Einbettung auf der unten beschriebenen Formel 2.4 basiert und nicht durch ein neuronales Netz propagiert. Anwendungsfälle finden sich beispielsweise in der Extraktion von Schlagwörtern oder Berechnung von Ähnlichkeiten aus Texten.

Das TF-IDF ist eine weit verbreitete Einbettungsmethode in der Informationsgewinnung. Sie ist eine Zusammensetzung aus term frequency (TF) und inversed document frequency (IDF). Der TF-IDF-Wert ist ein statistisches Maß für die Relevanz eines Wortes in einem Satz. Diese Relevanz steigt proportional zu der Anzahl in der ein Wort in einem Dokument vorhanden ist. Der Wert wird jedoch kleiner, wenn das Wort in mehreren Dokumenten auftaucht.

Die term frequency beschreibt das Vorkommen eines Terms in einem Dokument. Übertragen auf den Anwendungsfall dieser Arbeit beziehen sich Terme auf Wörter und Dokumente auf Fehlermeldungen. Terme, die häufig in einem bestimmten Dokument vorkommen, werden als relevant bewertet und besitzen einen hohen TF-Wert. Der TF wird wie folgt definiert:

$$TF = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document} \tag{2.1}$$

Die documente frequency (DF) beschreibt das Vorkommen eines Terms in allen Dokumenten. Es wird die Anzahl an Dokumenten, die einen Term $t$ beinhalten durch die Anzahl aller Dokumente geteilt. DF ist wie folgt definiert:

$$DF_{i,j} = \frac{|d_j \in D : t_j \in d_j|}{|D|} \tag{2.2}$$

wobei $|D|$ die Anzahl der Dokumente angibt und $|d_j \in D\ :\ t_j \in d_j|$ die Anzahl der Dokumente, die den Term $t$ beinhalten.

Die Idee vom IDF ist es, Terme, die in vielen Dokumenten vorkommen weniger stark zu gewichten. Beispiele für solche Terme sind Wörter wie ”der”, ”von” und ”ein”. Um diese Eigenschaft zu erreichen wird die inverse DF verwendet und ist wie folgt definiert:

$$IDF = log(\frac{|D|}{|d_j \in D : t_j \in d_j|}) \tag{2.3}$$

Schließlich wird der TF-IDF Wert durch eine Multiplikation des TF Werts mit dem IDF Wert realisiert:

$$TF - IDF = TF \times IDF \tag{2.4}$$

Wenn nun der TF-IDF-Wert eines Satzes berechnet werden soll, wird wie folgt vorgegangen. Zunächst werden alle Wörter aus allen Dokumenten in einer Menge gesammelt, wobei keine Wörter mehrfach vorkommen. Für jeden Satz wird nun ein Vektor erstellt, indem jeder Wert einem Wort aus dem Menge zugeordnet ist. Der Wert für jedes Wort wird nach der obigen TF-IDF Formel berechnet.
Wenn das Wort nicht in dem Satz vorkommt, ist der TF-IDF-Wert 0. Dies liegt daran, dass TF = 0 und somit auch TF*IDF = 0 ist.

Die Abbildung 2.2 zeigt ein Beispiel für die Einbettungen von zwei Sätzen. Für jeden Satz wird ein Vektor der Länge 11 erzeugt. Jeder Wert des Vektors entspricht einer Vokabel aus dem Vokabular. Wenn ein Wort in eine Satz nicht vorkommt, wird der dementsprechende Wert auf 0 gesetzt.
Ein bemerkenswerter Fall ist das Wort ”the”. In Beiden Sätzen hat dieses Wort den TF-IDF Wert von 0, obwohl es in diesen Sätzen vorkommt. Die Berechnung der Werte sieht wie folgt aus. In dem Satz A kommt das Wort ”the” einmal in dem Satz von 7 Wörtern vor. Das führt dazu, dass $TF = 1/7 = 0.142857$. Für Satz B ist der Wert $TF = 2/6 \approx 0.33$, da es zwei Vorkommen gibt. Für die Berechnung des DF Wertes für das Wort ”the” wird geschaut, in wie vielen Dokumenten das Wort enthalten ist, und dann wird diese Zahl durch die Anzahl aller Dokumente geteilt. In dem Fall ist $DF = 2/2$. Dementsprechend ist $IDF = log(2/2) = 0$. Da der IDF Wert nicht Abhängig von dem einzelnen Dokument ist, muss dieser nicht einzeln für Satz A und B berechnet werden. Schließlich zeigt sich nun, dass der TF-IDF Wert für das Wort ”the” ebenso 0 sein muss, da der IDF Wert auch 0 ist.
An diesem Beispiel ist zu sehen, wie TF-IDF mit dem IDF arbeitet. Je öfter ein Wort auch in anderen Sätzen enthalten ist, desto weniger relevant ist es. Ist ein Wort in allen Dokumenten enthalten, so ist der TF-IDF Wert dieses Wortes gleich 0.

```
documentA = 'the man went out for a walk'
documentB = 'the children sat around the fire'
```

| Word / Sentence | a | around | children | fire | for | man | out | sat | the | walk | went |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0.099 | 0 | 0 | 0 | 0.099 | 0.099 | 0.099 | 0 | 0 | 0.99 | 0.99 |
| B | 0 | 0.12 | 0.12 | 0.12 | 0 | 0 | 0 | 0.12 | 0 | 0 | 0 |

Abbildung 2.2: Beispiel von TF-IDF Einbettungen (eigene Darstellung)

Angewendet auf die Problemstellung, kann die Nutzung des TF-IDF folgende Vorteile mit sich bringen:

- Geringerer Rechenaufwand: Durch die vergleichsweise einfache Berechnung von Einbettungen wird die Rechenzeit wahrscheinlich um ein Vielfaches weniger betragen, als mit Einbettungsmethoden, die auf Neuronalen Netzen basieren. Da diese These nicht bewiesen ist, wird die Geschwindigkeit der Berechnung wird in den Experimenten in Kapitel 6 näher untersucht.
- Geringerer Speicheraufwand: Die einzelnen Einbettungen werden zu einem großen Teil aus Nullen bestehen. Dies ist vorteilhaft für die Speicherung der Vektoren. Hierbei kann der Speicherplatz dieser sogenannten dünnbesetzten Vektoren (engl. sparse vectors) durch Algorithmen um ein Vielfaches reduziert werden. Dies wird ebenso in den Experimenten in Kapitel 6 dokumentiert.

Alles in einem wäre der TF-IDF Ansatz eine vor allem kostengünstige Lösung für dieses Problems. Die vergleichsweise rechen- bzw. speicherlastigen Ansätze BERT und SBERT finden sich in den nachfolgenden Abschnitten 2.5 und 2.5.1.

## 2.5 BERT

Eine aktuellere Art, NLP Aufgaben wie die STS zu lösen, ist die Verwendung von Transformern. Mit dem Paper ”Attention is all you need” [25] wurde 2017 der Transformer eingeführt. Es ist ein Modell, welches auf neuronalen Netzen basiert und die Basis für

viele NLP Modelle des aktuellen Entwicklungsstands ist. Beispiele für Transformer Architekturen sind BERT [5], SBERT [22] und ChatGPT [8]. Mit diesen Modellen werden Sentimentanalysen auf Tweets erstellt [3], Anomalien in Logdaten erkannt [26] und ein ChatBot erstellt, welcher menschliche Sprache versteht und erzeugen kann [8].
SBERT ist ein Modell, welches speziell für die Erstellung von Einbettungen entwickelt wurde. Es ist ein Ansatz aus 2019, welches in aktuellen NLP Benchmark Tests ähnlich gute Ergebnisse, wie das Basismodell BERT geliefert hat und dabei die Rechenzeit um ein Vielfaches verringert. Belege sowie eine detailreiche Erklärung sind in Kapitel 2.5.1 zu finden. Um das SBERT Modell zu verstehen, wird zuerst das BERT Modell erläutert.

Bidirectional Encoder Representations from Transformers (BERT) ist ein transformerbasiertes neuronales Netz für die Repräsentation von Sprache, welches 2018 von Google vorgestellt wurde. Die nachfolgenden Informationen über BERT stammen aus dem Paper der Veröffentlichung "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [5]. Ziel des Transformers ist es, sowohl den Aufwand eines Trainings auf Sprache auszulagern, sowie eine Lösung für diverse NLP-Probleme zu bieten. Diese Auslagerung des Trainings geschieht, indem die neuronalen Netze bereits auf Sprache trainiert werden und somit ein vortrainiertes Modell verwendet werden kann. Wenn also ein votrainiertes BERT Modell verwendet wird, ist ein weiteres Training nicht notwendig. So kann ein bereits trainiertes Modell für mehrere Anwendungen benutzen, ohne ein separates Training zu absolvieren. Ein Teil diese Trainings besteht aus der Maskierung von einzelnen Wörtern in einem Satz (Masked Language Model). Aufgabe des Transformers ist die Füllung der Lücken.
Der zweite Ansatz im Training ist die next sentence prediction (NSP). Hier lernt das Netz, welche Sätze basierend auf der Semantik aufeinander folgen. Diesem werden zwei Sätze vorgegeben und es soll entscheiden, ob die Sätze miteinander verbunden sein könnten. Zu sehen ist dies in dem linken Teil von Grafik 2.3. In der Inputschicht werden maskierten Sätze eingegeben, während die Outputschicht die maskierten Sätze auflöst (Mask LM) und ausgibt, ob die Sätze aufeinander folgen könnten (NSP).

Das zweite Ziel - die Verwendung von einem BERT Modell für mehrere NLP-Probleme - wird durch eine gezielte Änderung der Architektur eines vortrainierten Netz realisiert. Diese Technik, welche auch fine tuning genannt wird, wird im Rahmen von BERT konkret durch das Austauschen der Eingabe- sowie Ausgabeschicht aufgezeigt. Zu sehen im rechten Teil der Abbildung 2.3 ist die an den Anwendungsfall angepasste Architketur (hier: Question Answering). Der Unterschied zu die pre-trained-Modell (links) ist hier der Input und Output.

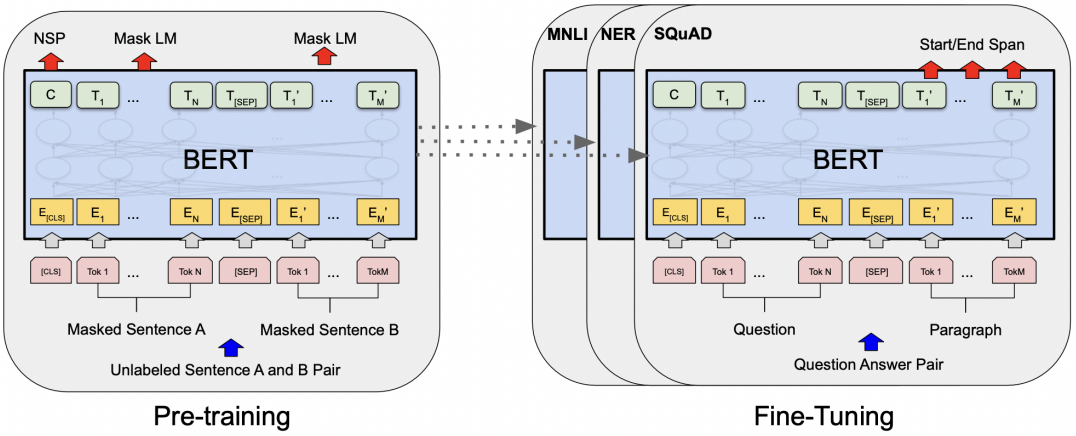


Abbildung 2.3: Aufbau der Architektur im Training (links) und die Anpassung des Netzes auf ein bestimmtest Problem (rechts) [5]

Wenn nun BERT als Lösung für das Problem der Arbeit benutzt werden würde, würden die Anspassungen wie folgt aussehen. Um zwei Fehlermeldungen zu vergleichen, muss es zwei Fehlermeldungen als Eingabe geben. Diese Eingabeschicht ist bereits aus dem Training gegeben und muss in diesem Fall nicht angepasst werden. Die Ausgabeschicht wird dahingegen so geändert, dass ein Wert ausgegeben wird, welcher ein Maß für die Ähnlichkeit gibt. Dieser könnte beispielsweise einen Wert von 0 bis 1 annehmen. In dem nachfolgenden Abschnitt 2.5.1 zeigt sich jedoch, dass diese Vorgehensweise eine hohen Rechenaufwand vorweist.
Insgesamt wird von den Autoren von BERT aufgezeigt, dass BERT in 11 NLP Aufgaben verwendet werden kann. Beispiele dafür sind: Zusammenfassung von Text, Text Generation und Textklassifikation. Zwei wichtige Vorteile von BERT sind die Auslagerung des Trainings und die Verwendung von einem Modell für unterschiedliche Aufgabengebiete der NLP.

### 2.5.1 SBERT

SBERT ist eine 2019 an der TU Darmstadt publizierte Erweiterung von BERT, welche sich auf die STS spezialisiert. Die nachfolgenden Information dieses Kapitels stammen aus der Paper der Veröffentlichung ”Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks” [22]. Laut den Autoren erzielt das Modell eine ähnlich hohe Erfolgsquote sowie schnellere Kalkulation der Ähnlichkeit im Vergleich zu dem Basismodell

BERT. Es wird das Problem adressiert, dass BERT eine sehr aufwändige Berechnung der STS aufweist. Grund dafür ist die Vorgehensweise, dass beim BERT jeder Satz mit jedem Anderen durch das Netz propagieren muss, um die Ähnlichkeiten von allen Sätzen zu berechnen. Beispielsweise müssten beim Vergleich von $n = 10\,000$ Sätzen insgesamt $\frac{(n-1)}{2} = 49\,995\,000$ Berechnungen getätigt werden. Bei einer modernen V-100 GPU würde dieser Prozess über 65 Stunden dauern.

Der Ansatz von SBERT verändert die Architektur des Netzes so, dass als Output eine Einbettung ausgegeben wird. Diese Einbettungen können im Voraus berechnet und danach gespeichert werden. Bei der Suche nach Ähnlichkeit würden die Einbettungen mit einer leichtgewichtigen Vergleichsmetrik, wie der Kosinusdistanz, miteinander verglichen werden. Dieser Ansatz ähnelt dem TF-IDF. Die Methode für die Erstellung der Einbettung ändert sich lediglich.

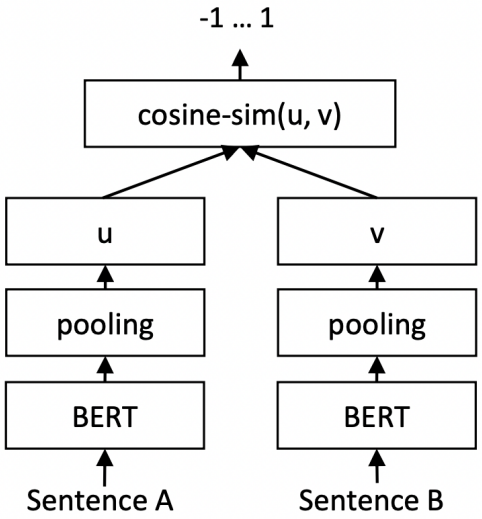


Abbildung 2.4: SBERT fine-tuned auf Ähnlichkeitsbewertung [22]

Die Abbildung 2.4 zeigt eine Netzarchitektur im Training, welche einen Similarity Score berechnet. Satz A und B propagieren zuerst durch ein BERT Netz. Wie von den Autoren des BERTs empfohlen wird, wird die Augabeschicht erweitert. Es findet sich hier ein Pooling-Layer, welcher die Ausgaben des BERTs auf einen Vektor gewünschter Größe abbildet. Per Default ist diese Größe 768. Zum Schluss werden die zwei erstellten Einbettungen von Satz A und B mit einer Vergleichsmetrik verglichen. Es ist hierbei wichtig zu erwähnen, dass es sich hier um siamesische BERT Netze handelt. Das bedeutet, dass

diese zwei neuronalen Netze dieselben Gewichte für die zwei Eingaben verwenden. Diese Architektur ist einer der Gründe für die Performance von SBERT.

An der technischen Universität Darmstadt wurden im Rahmen von SBERT mehrere Experimente durchgeführt, um die Performance des BERT basierten Sprachmodells aufzuzeigen.
Eine Art, um die Tauglichkeit und Erfolgsquote einer STS Lösung zu messen, ist es anhand des Vergleichs zu state-of-the-art Benchmarks (auch "gold labels" genannt). Dazu wurden die SBERT und SRoBERTa [1] Modelle verwendet, die auf einer Kombination von SNLI und Multi-Genre NLI trainiert wurden. Mit einer Batch-Größe von 16, dem Adam Optimizer mit einer Lernrate von 2e-5 und mit der Pooling-Strategie MEAN wurden zwei Experimente durchgeführt: Jeweils mit und ohne jegliches fine-tuning, welches auf STS Probleme spezialisiert ist.
Die Ergebnisse der Experimente zeigten, dass entweder SBERT oder SRoBERTa in fast allen Testfällen beider Experimente die höchsten Werte über den aktuellsten Standards erzielt haben.

Es zeigt sich, dass das transformerbasierte SBERT Modell sich aus den folgenden Gründen für die Suche der ähnlichsten Fehlermeldungen eignen könnte.
Zuerst ist dieser Ansatz ein solcher, welcher speziell für die STS entwickelt wurde und ebenso performante Ergebnisse liefert. Mit dem Verständnis von Sprache, welches mit dem Basismodell BERT mittels Mask LM und NSP erlernt wurde, wird erhofft, dass die Semantik der Fehlermeldungen berücksichtigt wird.
Außerdem wird gegenüber dem BERT die Rechenzeit für die Berechnung von Ähnlichkeiten drastisch gesenkt. Dies kann dazu führen, dass ein SBERT besser in der Anwendung funktioniert, wo ein Ergebnis in Sekunden gefordert wird. Der größte Aufwandsfaktor ist die Berechnung der Einbettungen. Wenn diese jedoch in einem Hintergrundprozess vorher durchgeführt wird, könnte die Anwendung praktischer sein.
Insgesamt ist der SBERT ein Ansatz, welcher wahrscheinlich kostenintensiver als TF-IDF sein wird. Mit dem zusätzlichen Wissen über Sprache werden akkuratere Ergebnisse erhofft. Wie untersucht wird, was ein Ergebnis besser macht als ein Anderes, wird in Kapitel 4.3 beschrieben. Experimente zu dem Vergleich von SBERT und TF-IDF finden sich in Kapitel 6.3.

[1] RoBERTa ist eine von Meta und der Waschington University entwickelte Optimierung von BERT. SRoBERTa ist die Weiterentwicklung von RoBERTa analog zum SBERT[15].

### 2.5.2 Modelle

Um die beiden Transformer BERT und SBERT zu verwenden, können bereits vortrainierte Sprachmodelle verwendet werden. Dies ist eine Möglichkeit, die vom Aufwand her kosteneffizient ist. Die Alternative dazu ist ein eigenes Training von dem Transformer, was gelabelte Trainingsdaten voraussetzt. Da der Aufwand Daten zu labeln zu hoch ist, wird wie in Kapitel 2.1 beschrieben, von einem solchen Ansatz von Grund auf abgesehen. Um auf der anderen Seite votrainierte Modelle zu verwenden, gibt es Open-Source Plattformen, bei denen diese Modelle heruntergeladen werden können.
Um vortrainiert Modelle herunterzuladen, wird die Python Bibliothek SentenceTransformers verwendet. Mithilfe dieser werden die Modelle über die open-source Internetseite Huggingface[2] geladen. In Kapitel 6 werden die in den Experimenten angewendeten Modelle näher beschrieben.
Die Eigenschaften und Attributausprägungen von den vortrainierten Modellen sind in Abbildung 2.5 zu sehen. Die Ausgabe ist unterteilt in mehrere Schichten. Zuerst findet sich der Transformer wieder, welcher die Basis des Modells ist. Dieser hat eine Maximallänge für die Eingabe. In diesem Fall liegt diese bei 512. Diese Zahl beschreibt die maximale Anzahl an WordPieces, die die Eingabe enthalten kann. WordPieces werden im nachfolgendem Kapitel 2.6 erklärt. Überschreitet eine Eingabe diese Länge, so wird der Text nach hinten abgeschnitten. Die zweite Schicht zeigt eine auf den Transformer aufbauende Pooling Layer. Diese bildet in diesem Fall den Output des Transformers auf einen Vektor von 768 ab. Zuletzt ist eine Schicht für das Normalisieren der Werte angehängt.
Neben diesem Attribut hat jedes SBERT Modell einen Tokenizer, der die Wörter in kleinere Einheiten teilt. Dieser wird im folgendem Kapitel mit dem WordPiece beschrieben.

```
SentenceTransformer(
  (0): Transformer({'max_seq_length': 512, 'do_lower_case': False}) with
Transformer model: MPNetModel
  (1): Pooling({'word_embedding_dimension': 768, 'pooling_mode_cls_toke
n': False, 'pooling_mode_mean_tokens': True, 'pooling_mode_max_tokens': F
alse, 'pooling_mode_mean_sqrt_len_tokens': False})
  (2): Normalize()
)
```

Abbildung 2.5: Beispiel eines SBERT Modells (eigene Darstellung)

[2] https://huggingface.co/models

## 2.6 WordPiece Tokenisierung

Die Tokenisierung ist ein Prozess in dem ein Text in mehrere kleinere Einheiten geteilt wird. Diese Einheiten werden Tokens genannt. Ein einfaches Beispiel für Tokens sind Wörter, wie sie in Wörterbüchern zu finden sind. Die Tokenisierung ist ein fundamentaler Prozess in der NLP und wird von vielen Anwendungen benutzt. Aktuelle NLP Modelle, wie die vorher beschriebenen Transformer BERT, SBERT und ChatGPT tokenisieren Text in Teilwörter, um dann mit diesen zu arbeiten.
BERT und SBERT verwenden für die Tokenisierung WordPiece [6]. Die Tokens, die WordPiece erstellt, werden WordPieces genannt. Die Idee hinter der WordPiece Tokenisierung ist die Aufteilung der Sätze in Wörter. Wenn das Wort nicht Teil des vorher festgelegten Vokabular ist, wird dieses in Teilwörter aufgeteilt. Gibt es keine Teilwörter, die dem Vokabular sind (engl. out-of-vocabulary, kurz OOV), wird ein neues Wort aus einer Verbindung dieser Teilwörter dem Vokabular hinzugefügt. Damit wird das Problem von OOV Wörtern angegangen. Auf den genauen Algorithmus von WordPiece wird im Rahmen dieser Arbeit nicht weiter eingegangen.

Ähnlich wie vortrainierte SBERT-Modelle, gibt es bereits fertige Modelle für die Tokenisierung, die Tokenizer genannt werden. In Kapitel 2.5.2 wurde beschrieben, dass jedes vortrainierte Modell einen Tokenizer besitzt. Dieser kann bei Modellen des Sentence-Transformers mit einem Aufruf ”tokenizer” ermittelt werden. Die Tokenizer der Modelle werden voraussichtlich einen Einfluss auf die Ergebnisse haben. Grund dafür ist das unterschiedliche Vokabular der Tokenizer. Ein englisches Modell weist zum Beispiel ein weit geringeres Vokabular auf, als ein Modell, welches mehrere Sprachen unterstützt.
Zusammenfassend sind die Tokenizer ein fundamentales Element der NLP und vor Allem in den aktuellen Transformermodellen weit verbreitet. Die Wahl des richtigen Tokenizers ist ausschlaggebend für die Ergebnisse. Außerdem kann die Verwendung WordPiece Tokenisierung Vorteile bei der Verarbeitung der Fehlermeldungen sein. Dies liegt daran, dass in Fehlermeldungen interner Systeme, wie dem von msg.life, Wörter beinhalten, die keinem open-source Tokenizer bekannt sind. Der WordPiece Algorithmus ist darauf ausgelegt, solche Wörter speziell zu beachten.

## 2.7 Ähnlichkeitsmaß

Das Ähnlichkeitsmaß ist nach [24] eine Funktion, welche die Ähnlichkeit zweier Elemente berechnet und einen reellen Wert zurück gibt. Dieser Wert wird berechnet, indem die Distanzen im Vektorraum auf Ähnlichkeiten abgebildet werden. Beispiele für Ähnlichkeitsmaße sind: Kosinusähnlichkeit, Jaccardähnlichkeit und Euklidische Distanz.
Die Abbildung 2.6 zeigt eine vereinfachte zweidimensionale Darstellung der Kosinusähnlichkeit. Es sind die zwei Punkt P1 und P2 gegeben. Wird die Distanz D größer, so unterscheiden sich die zwei Punkte mehr. Bei kleinerer Distanz sind die Punkt ähnlicher. Ist also der Winkel zwischen den Punkten 0, so sind beide Elemente identisch und die Kosinusfunktion liefert 1. Nimmt der Winkel einen anderen Wert als 0 ein, so liegt die Kosinusfunktion bei einem Wert unter 1.

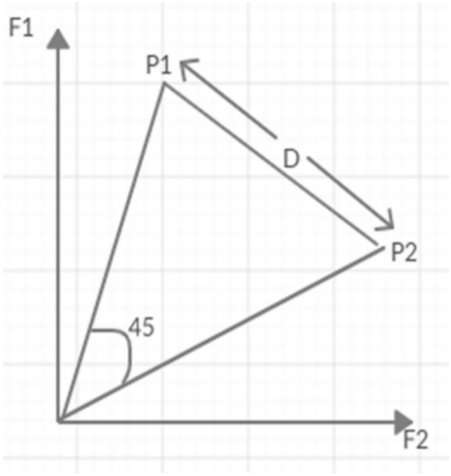


Abbildung 2.6: Beispiel Kosinusähnlichkeit [24]

Ein Ähnlichkeitsmaß soll im Rahmen dieses Projekts für den Vergleich von Einbettungen dienen. Somit soll dann ein Similarity Score berechnet werden kann. In dem SBERT Paper [22] werden dafür die Kosinusähnlichkeit, Euklidische sowie die Manhattan Distanz empfohlen. Jeder dieser Ansätze wurde in Experimenten verwendet. Jedoch zeigte sich, dass die Ergebnisse sich nur geringfügig unterscheiden. Die weiteren Experimente beschränkten sich in dem Ähnlichkeitsmaß auf die Kosinusähnlichkeit. Dies spricht für die Verwendung der Kosinusähnlichkeit.
In einem anderem Anwendungsfall, zeigen sich ähnliche Ergebnisse. Bei einem Vergleich

von mehreren Ähnlichkeitsmaßen in Newsatrikeln [24] bewies die Kosinusähnlichkeit die höchste Erfolgsquote. Die Newsartikel wurden hierbei mit TF-IDF eingebettet.

Es zeigt sich, dass das Ähnlichkeitsmaß der Kosinusähnlichkeit in den Einbettungsmethoden TF-IDF sowie SBERT praktikabel ist und angewendet werden kann. Es wird daher in dieser Arbeit von weiteren Experimenten für ein Ähnlichkeitsmaß abgesehen und die Kosinusähnlichkeit in allen nachfolgenden Experimenten verwendet.

Die Kosinusähnlichkeit zweier Vektoren $A$ und $B$ ist wie folgt definiert:

$$cos\_sim(A, B) = \frac{A.B}{||A|| \times ||B||} \tag{2.5}$$

Die Kosinusdistanz wird wie folgt definiert:

$$cos\_dis = 1 - cos\_sim(A, B) \tag{2.6}$$

# 3 Verwandte Arbeiten

In diesem Kapitel wird der Einsatz von den genannten Einbettungsmethoden TF-IDF und SBERT mit ähnlichen Ansätzen aufgezeigt.

Das 2022 publizierten Paper "LogST: Log Semi-supervised Anomaly Detection Based on Sentence-BERT" [26] beschreibt die Extraktion von semantisch relevanten Informationen aus Logs in der Erkennung von Anomalien. Die finale Softwarelösung - LogST - verwendet die 768-dimensionalen Einbettungen von SBERT und wendet nach einer Dimensionsreduktion den Clusteringalgorithmus HDBSCAN an, um Logs zu kategorisieren und anomale Logs zu identifizieren. In mehreren Experimenten werden neben anderen Methoden TF-IDF und SBERT für die Erstellung von Einbettungen verwendet und miteinander verglichen. Im Vordergrund der Experimente steht die Unterscheidung von Wort- und Satzeinbettungen. Die TF-IDF Vektoren sind ein Beispiel für wortbasierte Einbettungen, da hier die resultierende Einbettung für ein Satz aus der Verbindung der TF-IDF Werte einzelner Wörter ist. Der SBERT Ansatz hingegen, welcher satzbasierten Einbettungen produziert, berechnet einen Vektor aus dem gesamten Satz, wobei auch die Reihenfolge beachtet wird. Damit soll versucht werden, die Interaktion der Wörter in die Einbettung mit einzubringen.
Abbildung 3.1 zeigt die Ergebnisse eines Experiments, in dem SBERT und wortbasierte Einbettungen in der Anomalieerkennnung verwendet wurden. Die satzbasierten Einbettungen aus dem SBERT weisen in der Performance bezüglich Precision, Recall und F1 auf erfolgreichere Ergebnisse hin.

Verglichen mit der Problemstellung der Suche nach ähnlichen Fehlermeldungen können hier Gemeinsamkeiten gefunden werden. Zum einen betrachtet die genannte Arbeit die Ansätze des TF-IDFs und des SBERTs. Zum anderen ist die Grundlage der Daten eine Ähnliche. Denn Log-Nachrichten enthalten ebenso wie Fehlermeldungen Texte, die nicht in der natürlichen Sprache vorkommen. Hier ist eine Beispiel Log-Nachricht aus dem Paper zu sehen:

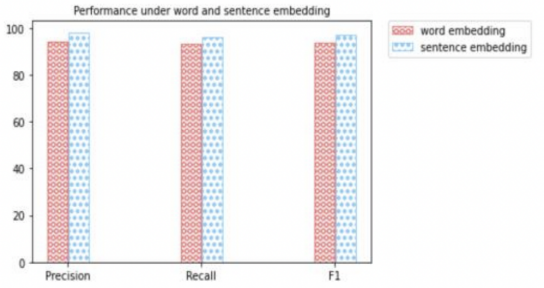


Abbildung 3.1: Vergleich von Wort- und Satzeinbettungen [26]

> PacketResponder 2 for block blk_-1608999687919862906 terminating

Und hier ist ein Beispiel für eine Fehlermeldung aus der Datenbank der Fehlermeldungen:

> Fehler bei FRNAC02 des Vertrags SHF2017_08_31, der Returncode des Batchservers ist 8.

Zu unterscheiden ist das Ziel der gesamten Arbeit. Das LogST System zielt auf eine Anomalieerkennung ab. Dafür wird ein Clustering verwendet, um anomale Log-Nachrichten zu finden, die in keine bekannte Kategorie fallen. Das Ziel dieser Bachelorarbeit ist die semantische Suche und nicht die semantische Kategorisierung.
Neben diesem Unterschied, gibt es einen weiteren in der Datenverarbeitung. In dem Paper wird eine vorherige Verarbeitung der Daten beschrieben. Aus aufwandstechnischen Gründen wird die Vorverarbeitung der Daten in den Ausblick dieser Arbeit verschoben. Schlussendlich ist aus dieser Arbeit zu entnehmen, dass es einen Vorteil haben kann, Methoden zu verwenden, die satzbasierte Einbettungen erstellen. Somit kann vermutet werden, dass der SBERT Ansatz bessere Ergebnisse liefert. Grund dafür ist die Auswertung der genannten Arbeit und die ähnliche Datengrundlage.

2021 wurde die wissenschaftliche Arbeit "Catching Unusual Traffic Behavior using TF–IDF-based Port Access Statistics Analysis" [23] in der CCCI[1] vorgestellt. Ähnlich, wie in dem vorher vorgestellten Paper [26] werden Anomalien in Logs gesucht, wobei die Autoren sich auf den Netzwerkverkehr fokussieren. Ziel der Arbeit ist es, anomale Aktivitäten an Ports anhand von Zugriffslogs festzustellen, die auf einen Angriff hindeuten könnten. Die zugrundeliegende Methode für die Analyse ist TF-IDF.
Es stellt sich heraus, dass mithilfe dieses Ansatzes nach einer 30-tägigen Datensammlung

[1] International Conference on Communications, Computing, Cybersecurity, and Informatics

mehrere Anomalien entdeckt werden konnten.
Mit dieser Arbeit zeigt sich ein Ansatz, der ebenso eine ähnliche Datengrundlage vorweist. Es ist ein weiterer Beweis dafür, dass mithilfe von TF-IDF Anomalieerkennung einsetzbar ist.

Im Jahre 2021 erschien das Paper "Text Similarity Measures in News Articles by Vector Space Model Using NLP" [24]. Das Ziel der Arbeit ist der semantische Vergleich von Nachrichtenartikeln. Als Problemlösung dieser STS Aufgabe beinhalten die meisten Ansätze die Einbettung der Artikel mittels TF-IDF. Danach wird die Ähnlichkeit der Einbettungen mit der Kosinusähnlichkeit und der Jaccardähnlichkeit berechnet. Im Fokus dieser Arbeit ist der Vergleich von mehreren Ähnlichkeitsmaßen.
Schließlich hat sich herausgestellt, dass der Ansatz, welcher den TF-IDF als Einbettungsmethode und die Kosinusähnlichkeit als Ähnlichkeitsmaß verwendet, die besten Ergebnisse aufweist.
Verglichen mit der Problemstellung und dem Ansatz dieser Bachelorarbeit, gibt es folgende Gemeinsamkeiten und Unterschiede. Zum einen ist das Ziel der Arbeiten ähnlich. Es werden die Ähnlichkeiten zweier Texte ermittelt. Zum anderen verwenden beide Lösungsansatz unter anderem TF-IDF und die Kosinusähnlichkeit als Ähnlichkeitsmaß.
Zu unterscheiden ist jedoch die Datengrundlagen. Nachrichtenartikel weisen weit mehr Textinhalte auf, die in der natürlichen Sprache angehören, als Fehlermeldungen.

# 4 Analyse der Ausgangslage

Das folgende Kapitel setzt sich mit der Problemstellung und -lösung auseinander. In dem ersten Teil wird der Ausgangspunkt inklusive der Beschaffenheit der Daten vorgestellt. Eine Analyse des Ausgangspunktes bildet dann die Bausteine für eine Anforderungsanalyse. Diese wird in funktionale und nichtfunktionale Anforderungen unterteilt. Zuletzt wird beschrieben, unter welchen Bedingungen die gesamte Arbeit ein Erfolg ist.

## 4.1 Ausgangspunkt

Der Softwarehersteller für Versicherer msg life hat, wie die meisten anderen IT-Unternehmen auch, Tests für dessen Software. Damit soll die Funktionalität sichergestellt werden. Darunter finden sich Tests, die täglich ausgeführt werden. Mit diesen soll überprüft werden, ob bestehende Funktionalität nicht durch neue Änderungen verfälscht wird und ungewollte Seiteneffekte entstehen. Diese Tests, die in einem regelmäßigen Zyklus durchgeführt werden, nennen sich Regressionstests. Sie stellen die Stabilität des Gesamtsystems bei Änderungen am Code sicher. Im Fall von msg life können Regressionstests wie folgt aussehen:

1. Mit einem vorgefertigten Vertrag wird eine Fortschreibung auf 10 Jahre in die Zukunft durchgeführt. Nun überprüft ein Test, ob die Steuerwerte richtig berechnet wurden.

2. Es wird versucht einen neue Rentenversicherung anzulegen. Die zu versichernde Person hat jedoch ein Alter von 99 Jahren. Dieses Alter ist in diesem Vertrag bei Vertragsanlage nicht erlaubt. Deshalb überprüft ein Test, ob die Anlage des Vertrags verhindert wird.

3. Auf einen Kapitalvertrag werden die unterschiedlichen planmäßigen und außerplanmäßigen Geschäftsvorfälle durchgeführt, z.B. eine vorzeitige Beitragsfreistellung. Ein Test prüft, ob die korrekten Zustände innerhalb des Vertrages zustande kommen.

Aktuell beinhaltet der größte Regressionstest etwas mehr als 14 000 Tests in fast 900 Testketten. Aus diesen Tests werden die verschieden möglichen Fehlermeldungen aus den gesamten Systemen der msg life ausgelöst. Dabei ist zu bedenken, dass ein Test nicht nur an einer Stelle fehlschlagen kann. In dem vorgestellten Beispiel 1 der Regressionstests ist es zwar offensichtlich, dass die Komponente der Steuerberechnung überprüft wird. Neben dieser Überprüfungen wird ebenso sichergestellt, dass ein Vertrag fortgeschrieben werden kann oder auch dass die Verbindung der Datenbank hergestellt wird. Es zeigt sich, dass bei einem Regressionstest die unterschiedlichsten Komponenten fehlschlagen können, da dieser sich nicht auf eine Einheit fokussiert. Damit werden Komponenten von angebundenen Drittanbietersystemen nicht ausgeschlossen. Mit bereits einem Test werden mehrere mögliche Fehlersituationen überprüft. Diese Fehlersituationen können dementsprechend viele Fehlermeldungen erzeugen. Dies soll verdeutlichen, wie viele unterschiedliche Fehlermeldungen bei den Regressionstests entstehen.

Bei jeder Änderung der Entwickler werden die Regressionstests gestartet. Die Tester von msg life überprüfen nahezu täglich die Fehlermeldungen der fehlgeschlagenen Tests. Deren Aufgabe ist mitunter die Analyse dieser Fehlermeldungen. Wenn der Fehler nicht sofort behoben werden kann, dann wird ein sogenanntes Fehlerticket erzeugt. Das Fehlerticket enthält neben der Fehlermeldung im Idealfall eine Beschreibung. Neben dieser wird dort der Fortschritt der Analyse mit Kommentaren festgehalten und es ist der Ort, wo die künftige Lösung inklusive Code abgelegt ist. Dieses neu erstellte Ticket wird an ein anderes Team weitergegeben. Zum Beispiel wird bei einem Fehler in der Berechnung von Steuerwerten das Ticket an das für Mathematik zuständige Team weitergeben. Stellt dieses Team fest, dass die Fehlermeldung nicht aus deren Bereich stammt, so wird das Ticket an ein weiteres Team abgegeben. Dieser Prozess wird so lange durchgeführt, bis eine Lösung für den fehlgeschlagenen Testfall gefunden wurde. Die verschiedenen Fehlermeldungen, die nahezu täglich erscheinen, werden seit einiger Zeit in einer Datenbank gespeichert. Diese Datenbank wird mit dem Namen TestAV bezeichnet und ist eine Abkürzung für "Test-Auswertungs-Vergleich". Das finale Werkzeug, welches die benannte Funktionalität erhalten soll, wird TestAVSearch genannt. In dem Abschnitt 4.1.2 werden weitere Informationen über die Beschaffenheit der Daten gegeben.

### 4.1.1 Motivation

Der Prozess, in dem ein Fehler behandelt wird, kann im schlimmsten Fall mehrere Tage, Wochen oder sogar Monate betragen. Dies liegt zum Teil daran, dass die Fehlerquelle in manchen Fällen unklar ist. Das Fehlerticket wird zwischen den verschiedenen Teams und Personen hin und her gereicht und es wird eine lange Kommunikationskette hervorgerufen. Die Lösung des Fehlers kann zum Beispiel eine komplexe Umstellung der Systemarchitektur erfordern. Es ist jedoch auch wahrscheinlich, dass die Fehlerursache an einer Stelle liegt und die Änderung minimal ist. Ein Beispiel ist hier die fälschliche Verwendung eines Umlauts, wie "ä", wobei "ae" zu verwenden ist. In jedem Fall muss sich eine Person mit dem Thema auseinandersetzen. Wenn diese Person den Fehler bereits kennt, wird sie die bereits bekannte Lösung schneller anwenden können, als jemand, der den Fehler nicht kennt.
Mit der Erstellung des Tools TestAVSearch soll ein Kollektivwissen erreicht werden. Die Erfahrung der einen Person, die die Lösung eines bestimmten Fehlers kennt, soll mit anderen Mitarbeitern geteilt werden. Somit soll erreicht werden, dass der Prozess der Lösungsfindung beschleunigt wird. Denn es besteht die Möglichkeit, dass die neuen Fehlersituationen bereits früher vorgekommen sind. Daher ist es naheliegend, zu Anfang der Analyse des Testers zu untersuchen, ob der aktuelle Fehler bereits in der Vergangenheit vorgekommen ist oder sogar gelöst wurde. Das Werkzeug TestAVSearch soll als eine Unterstützung des Testers und Entwicklers dienen, in dem es eine Suche nach ähnlichen Fehlermeldungen durchführt.

### 4.1.2 Datengrundlage

Wie bereits beschrieben, wurden im Vorfeld Fehlermeldungen gesammelt. Aktuell beinhaltet der aktuelle Datensatz, welcher in einem regelmäßigen Rhythmus aktualisiert wird, 400 000 ungefilterte Fehlermeldungen. Die Datenbank enthält neben der Fehlermeldung 34 weitere Attribute. Diese sind beispielsweise Primary Keys, Testlaufzeiten und verwendete Komponenten des Gesamtsystems. Die für diese Arbeit relevanten Attribute finden sich in Tabelle 4.1.
Grundvoraussetzung für die Suche sind die Fehlermeldungen selbst. Dieses Attribut ist in jeder Zeile der Datenbank vorhanden. In dem Beispiel in Tabelle 4.1 ist eine Meldung mit deutschem Text zu sehen. In der Datenbank finden sich ebenso reine englische Texte, deutsch-englische Fehlermeldungen sowie Stacktraces von mehreren Zeilen

| Attribut | Beschreibung | Beispiel (Anonymisiert) |
|---|---|---|
| ERRORMESSAGE | Gesamte Fehlermeldung | Fehler bei FRNAC02 des Vertrags SHF2017_08_31, der Returncode des Batchservers ist 8. |
| ISSUEID | Ticket/Issue Nummer | IS-12345 |
| NOTES | Notizen der Analyse | blauen Build lf-big abwarten |

Tabelle 4.1: Beispiel für einen Datensatz aus TestAV (eigene Darstellung)

und aneinander geschriebene Zeichenketten wie /ref/determineCompleteInfoResponse[1]-/return[1]/value[1]/". Für die Unterstützung des Testers wird die Issue-ID notiert, welche eine Referenz auf das Fehlerticket ist. Dieses soll mehr Auskunft über den Fehler, die Änderungen und Anmerkungen liefern. Des Weiteren gibt es das Attribut Notes, wo bereits Notizen über den Fehler stehen. Beide Felder sind optional, da nicht an jedem Fehler diese Informationen angehängt wurden. Die reine Suche geschieht ausschließlich auf den Fehlermeldungen selbst. Das bedeutet, dass keine weiteren Attribute der Datenbank Gegenstand der Suche sind. Die mögliche Verwendung von weiteren Attributen wird in dem Ausblick in Kapitel 8 diskutiert.

Außerdem sind die Daten nicht gelabelt. Das bedeutet, dass in der Suche nach Ähnlichkeiten von Fehlermeldungen keine konkrete Lösung vorgegeben ist. Der Prozess der Lernens mit den vorgestellten Einbettungsmethoden TF-IDF und SBERT ist also unüberwacht. Im Fall von SBERT bedeutet dies, dass ausschließlich vortrainierte Modelle verwendet werden. Es findet kein Training mit den Fehlermeldungen der TestAV Datenbank statt.

Mit der Abbildung 4.1 soll ein Überblick über die Länge der Fehlermeldungen in dem gesamten Datensatz geschaffen werden. Das Balkendiagramm zeigt, wie häufig die Fehlermeldungen eine bestimmte Anzahl an Worten enthalten. Dabei ist ein Wort in diesem Fall als eine Zeichenkette definiert, die per Leerzeichen von anderen Zeichenketten getrennt wird. Auf der X-Achse ist die Anzahl an Worten einer Fehlermeldung zu sehen, während auf der Y-Achse die Häufigkeit dieser Anzahlen dargestellt wird.
Diese Darstellung beschränkt sich auf eine maximale Wortanzahl von 200. Die größte Anzahl an Worten in einer Fehlermeldung liegt jedoch bei 1536. Grund für die beschränkte Darstellung ist die hohe Verteilung dieser Werte. Der Großteil der Werte, nämlich 98,3 %, liegt zwischen 1 und 200 Wörtern.
In der Abbildung ist zu sehen, dass die meisten Werte zwischen 1 und 50 liegen. Dies ist ebenso an einem Durchschnittswert von 37,03 zu erkennen. Ab 50 sind die Anzahlen vergleichsweise gering.

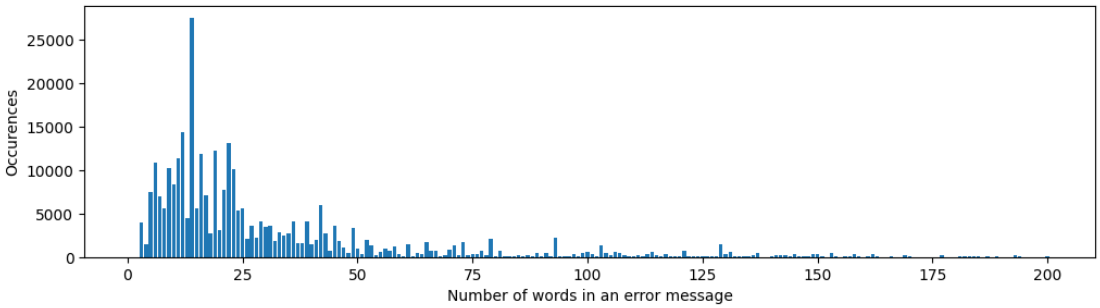


Abbildung 4.1: Häufigkeitsverteilung von Wortanzahlen in Fehlermeldungen (eigene Darstellung)

Die Abbildung 4.2 zeigt ähnlich wie Abbildung 4.1, die Häufigkeitsverteilung der Anzahlen an WordPieces der Fehlermeldungen der gesamten TestAV-Datenbank. Die WordPieces wurden mit dem Tokenizer des vortrainierten SBERT-Modells sentence-transformers_-distiluse-base-multilingual-cased-v1 erstellt. Hier ist anzumerken, dass verschiedene Modelle verschiedene Tokenizer verwenden, dessen Vokabular sich unterscheidet. Das bedeutet, dass andere Modelle hierbei andere Ergebnisse produzieren werden. Weitere Informationen zu dem Modell finden sich in den Experiment zu dem Vergleich der SBERT-Modelle in 6.2.
Die Grafik beschränkt sich hier auf der X-Achse auf eine Anzahl an WordPieces von 1000, obwohl das Maximum bei 8872 liegt. Grund hierfür ist die erhöhte Verteilung der Werte ab 1000. Denn 97,4 % der Werte liegen zwischen 3 und 1000, wobei 3 das Minimum an WordPieces einer Fehlermeldung ist. Die durchschnittliche Anzahl an WordPieces einer Fehlermeldung liegt bei 228.8. Der Median ist 107.

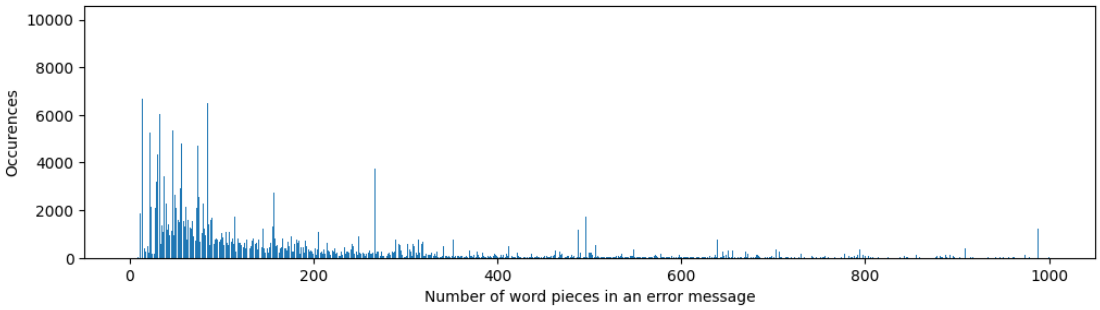


Abbildung 4.2: Häufigkeitsverteilung von WordPiece-Anzahlen in Fehlermeldungen (eigene Darstellung)

Zusammenfassend lässt sich sagen, dass die Varianz der Daten hoch ist. Die Fehlermeldung selbst können deutsche, englische und deutsch-englische Texte enthalten. Es finden sich ebenso Fehlermeldungen mit einer Syntax, welcher nicht der natürlichen Sprache entspricht. Des weiteren sind Wörter enthalten, die nur internen msg life Systemen bekannt sein sollten. Die Länge der Fehlermeldungen variiert stark nach außen, jedoch besitzt der Großteil dieser einer Länge von 1 und 200 Wörtern. Diese Analyse kann entscheidend für die Wahl einer Einbettungsmethode sein und liefert eine Basis für die Anforderungsanalyse.

## 4.2 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an das System untersucht und in funktionale sowie nichtfunktionale Anforderungen aufgeteilt. Der überwiegende Fokus der Anforderungen liegt auf den Methoden der NLP insbesondere der Einbettungsmethoden. Die Anforderungen werden in Kapitel 7 dem Istzustand der Lösung gegenübergestellt.

**Funktionale Anforderungen:**

Die zugrundeliegende funktionale Anforderung ist die Findung von gleichen oder ähnlichen Fehlermeldungen. Die Überprüfung der Qualität wird in 4.3 erläutert. Gemäß der vorher diskutierten Datengrundlage wird eine Einbettungsmethode gesucht, die die folgenden Eigenschaften besitzt. Es sollen englische sowie deutsche Texte verarbeitet werden können. Neben diesen, soll die Methode ebenso mit unbekannten Wörtern arbeiten können, wie msg-life-interne Begriffen oder zusammengesetzten Wörtern in den Fehlermeldungen. Im besten Fall soll die spezifische Syntax der Fehlermeldungen berücksichtigt werden, die sich von der natürlichen Sprache unterscheidet. Außerdem müssen Text mit einer Länge von 512 WordPieces akzeptiert werden. Schließlich muss es möglich sein, dass die Einbettungen vorher berechnet werden können. Ziel dabei ist die Auslagerung der Berechnung der Einbettungen, sodass dieser Prozess nicht für jede Anfrage des Benutzers wiederholt werden muss. Zusammengefasst sind die funktionalen Anforderungen in Tabelle 4.2.

**Nichtfunktionale Anforderungen:**

Der hauptsächliche Aspekt der nichtfunktionalen Anforderungen ist die Effizienz, das heißt der zeitliche Rahmen der Findung einer Fehlermeldung. Zum einen ist damit das

| Anforderungs-ID | Anforderung |
|---|---|
| F1 | Das System soll aus einer gegebenen Fehlermeldung die n ähnlichsten Fehlermeldungen der TestAV-Datenbank ausgeben |
| F2 | Die Berechnung der Einbettungen muss im Vorhinein möglich sein |
| F3 | Es sollen englische Texte verarbeitet werden können |
| F4 | Es sollen deutsche Texte verarbeitet werden können |
| F5 | Die Syntax von Fehlermeldungen soll berücksichtigt werden |
| F6 | Unbekannte Begriffe sollen berücksichtigt werden |
| F7 | Eingaben mit bis zu 512 WordPieces sollen akzeptiert werden |
| F8 | Die Ausgaben sollen in eine CSV-Datei geschrieben werden |

Tabelle 4.2: Funktionale Anforderungen (eigene Darstellung)

| Anforderungs-ID | Anforderung |
|---|---|
| NF1 | Die Verarbeitung der Anfrage darf nicht länger als 10 Sekunden betragen |
| NF2 | Die Berechnung der Einbettungen darf 120 Minuten nicht überschreiten |
| NF3 | Die Einbettungen dürfen nicht mehr als ein Gigabyte an Speicherplatz einnehmen |

Tabelle 4.3: Nichtfunktionale Anforderungen (eigene Darstellung)

Zeitintervall zwischen Lösungsanfrage und Lösungsausgabe gemeint. Diese soll nicht länger als 10 Sekunden betragen. Zum anderen wird damit der zeitliche Rahmen der Vorbereitung der Lösung beschrieben. Die Vorbereitung ist in diesem Fall die Erstellung von Einbettungen der Fehlermeldung, welche in einem regelmäßigen Zyklus aktualisiert und neu berechnet werden müssen. Der zeitliche Aufwand darf 2 Stunden für die Berechnung aller Einbettungen nicht überschreiten. Zusammengefasst sind die nichtfunktionalen Anforderungen in Tabelle 4.3.

## 4.3 Qualitätsprüfung

Das folgende Unterkapitel beschäftigt sich zunächst mit der Qualitätsprüfung der Ausgaben des Systems. Die Ausgaben beziehen sich auf die ähnlichsten Fehlermeldungen, die ein System zurück gibt und sind die Grundlage der Qualität eines Modells. Dadurch soll deutlich werden, wie die Systeme bewertet werden und wann ein Ergebnis einer Einbettungsmethode besser ist, als das einer Anderen. Dies ist notwendig für die Auswertung

der Experimente in 6. Da nicht im Vorhinein klar ist, wie eine Bewertung aussehen kann, werden mehrere Ansätze diskutiert.

### 4.3.1 Qualität der Ausgabe

Ein trivialer Ansatz, ein System zu bewerten, ist der Vergleich mit einem Sollwert. Ist- und Sollwert werden miteinander verglichen und es entsteht eine Metrik, mit der eine Lösung bewertet werden kann. In dieser Arbeit wäre eine Sollwert eine Fehlermeldung, die als ähnlichste Fehlermeldung zu einer Eingabefehlermeldung deklariert werden würde. Da die beiden Einbettungsmethoden TF-IDF und SBERT unüberwacht arbeiten, ist vorher kein Sollwert festgelegt. Der Aufwand für ein manuelles Labeling ist zu hoch. In dem Fall müsste ein Fachexperte jede einzelne Fehlermeldung mit einem gegebenen Datensatz vergleichen, um eine Fehlermeldung als ähnlichste deklarieren zu können. Da dieser Ansatz sehr aufwändig ist, wird dieser nicht für die Bewertung der Ausgaben verwendet.
Ein weiterer Ansatz ist die Synthetisierung von Daten. Die Idee dahinter ist die künstliche Erstellung von Daten. Um die ähnlichste Fehlermeldung zu deklarieren, wird diese künstlich erstellt. Es wird eine sehr ähnliche Fehlermeldung erstellt, die durch minimale Veränderung der Eingabefehlermeldung generiert wird. In der Evaluierung wird dann erwartet, dass die generierte Fehlermeldungen unter den ähnlichsten n zu finden sein müsste. Hiermit ist ein Vergleich von Soll- und Istwert möglich, ohne das ein manuelles Labeling notwendig ist. Das Problem an diesem Ansatz ist jedoch, dass die mutierten Fehlermeldung mit hoher Wahrscheinlichkeit keiner echten Fehlermeldung entsprechen werden. Die Sollwerte wären realitätsfern.
Außerdem besteht die Möglichkeit aktuelle Benchmarks, wie die golden Label für semantische Textähnlichkeit für die Bewertung der Lösung zu verwenden. Das Problem bei diesem Ansatz ist, dass es keine Benchmarks für msg life interne Fehlermeldungen gibt. Die Fehlermeldungen sind zu einem großen Teil systemintern und daher ist ein solcher Benchmark nicht anwendungsspezifisch.
Schließlich gibt es den Ansatz, die Ergebnisse von den unterschiedlichen Systemen von Fachexperten bzw. Testern beurteilen zu lassen. Dazu würden Eingabefehlermeldung mehreren Ausgabefehlermeldungen gegenübergestellt werden. Fachexperten könnten dann die Ausgabe der verschiedenen Systeme miteinander vergleichen. Hierbei würde genau die Personengruppe, die das Tool TestAVSearch in Zukunft verwenden würde, einen Einfluss auf die Auswahl des richtigen Systems haben. Der Aufwand ist hier gering, da die Fa-

| TF-IDF | | | SBERT | |
|---|---|---|---|---|
| Rangfolge Fehlermeldung | Fehlermeldung | | Rangfolge Fehlermeldung | Fehlermeldung |
| 1 | <FM 1> | | 1 | <FM 1> |
| 2 | <FM 2> | | 2 | <FM 2> |
| 3 | <FM 3> | | 3 | <FM 3> |
| 4 | <FM 4> | | 4 | <FM 4> |
| 5 | <FM 5> | | 5 | <FM 5> |
| 6 | <FM 6> | | 6 | <FM 6> |
| 7 | <FM 7> | | 7 | <FM 7> |
| 8 | <FM 8> | | 8 | <FM 8> |
| 9 | <FM 9> | | 9 | <FM 9> |
| 10 | <FM 10> | | 10 | <FM 10> |
| | | | | |
| **Ranking: 1** | **Hilfreich: Ja** | | **Ranking: 2** | **Hilfreich: Nein** |

Tabelle 4.4: Vorlage für Bewertung (eigene Darstellung)

chexperten oder Tester die Fehlermeldung als Eingabe mit den Ausgaben pro Lösung vergleichen müssten.

Nach Abwägung der genannten Lösungen wird aus Gründen des geringen Aufwands und der hohen Einbringung der Fachexpertise die letztere Lösung für die Beurteilung der Ergebnisse des Systems verwendet. Genauer beschrieben, soll der Prozess der Evaluierung der Ausgabe wie folgt aussehen.
Ein Fachexperte wird 10 Fehlermeldungen aussuchen, die dieser bereits kennt oder in welchen die Begriffe bekannt sind. Damit soll gewährleistet werden, dass die Fachexpertise in der Bewertung enthalten ist. Diese Fehlermeldungen propagieren durch jedes System und daraufhin werden die ähnlichsten zehn unterschiedlichen Fehlermeldungen gesammelt. Danach untersucht der Fachexperte die Ausgabefehlermeldungen und entscheidet für jede Eingabefehlermeldung, welche Softwarelösung die passenderen Ergebnisse liefert. Die Tabelle 4.4 zeigt die Vorlage für einen Fachexperte, der die Bewertung übernimmt. Die Softwarelösungen werden mit einer Rangfolge versehen, wobei 1 das beste Ergebnis markiert. Neben der Rangfolge soll ein Ja/Nein Feld beschreiben, ob die Ausgaben eines Systems hilfreich sind. Die beiden Felder, die der Experte ausfüllen soll, findet sich in der letzten Reihe der Tabelle.

Bei dem Vergleich der Ausgabefehlermeldungen ist entscheidend, wie gut das beste Ergebnis unter den zehn Ausgabelösungen ist. Bei gleichem oder fast gleichem besten Ergebnis,

wird das zweitbeste Ergebnis untersucht. Dies gilt analog bis zur zehnten Fehlermeldung. Sind alle zehn Ergebnisse gleich oder nahezu gleich, so wird der gleiche Rang für beide Systeme vergeben.

# 5 Entwurf

Dieses Kapitel beschäftigt sich mit dem Entwurf der Software und dessen zukünftige Einbettung in die Entwicklung. In dem ersten Teil findet sich ein fachlicher Überblick über die Funktionalität der Software. Danach wird mithilfe eines Komponentendiagramms die Architektur aus technischer Sicht erläutert. Zuletzt werden die Einsatzmöglichkeiten geschildert.

## 5.1 Fachliche Sicht

Die Funktionalität der Software wird in die folgenden zwei Prozesse aufgeteilt: Einbettungs- und Suchprozess. Das Flussdiagramm aus 5.1 zeigt den Einbettungsprozess aus fachlicher Perspektive und stellt die Vorbereitung der für den Suchprozess dar. Der Einbettungsprozess soll der Anforderung F2 aus Kapitel 4.2 gerecht werden. Dieser Prozess wird aktiv, sobald die TestAV-Datenbank aktualisiert wird und neue Fehlermeldungen hinzukommen. Das Ziel ist die Erstellung und Abspeicherung von Einbettungen der aktuellen Errormessage DB, sodass diese in dem Suchprozess verwendet wird. Somit wird der aufwändige Prozess ausgelagert und muss nicht bei jeder Suche neu ausgeführt werden.
Zu Anfang des Prozess werden die Fehlermeldungen der TestAV-Datenbank geladen. Dort wird die Spalte der Fehlermeldungen extrahiert. Danach wird ein sogenannter Vectorizer geladen. Dies ist die Instanz, welche mit den Techniken ausgestattet ist, Einbettungen zu erstellen. Nachdem die Einbettungen erstellt werden, werden diese in einer speichereffizienten Datenstruktur gespeichert, um Speicherplatz nach Anforderung NF3 zu sparen. Die Datenstruktur ist abhängig von der Art der Vektoren und wird in den Experimenten in Kapitel 6 näher untersucht.

Der Suchprozess wird in der Abbildung 5.2 beschrieben. Der Prozess beinhaltet die Sortierung der Fehlermeldungen nach Ähnlichkeit und richtet sich nach Anforderung F1. Es wird eine Eingabe entgegengenommen und die n ähnlichsten Fehlermeldungen werden

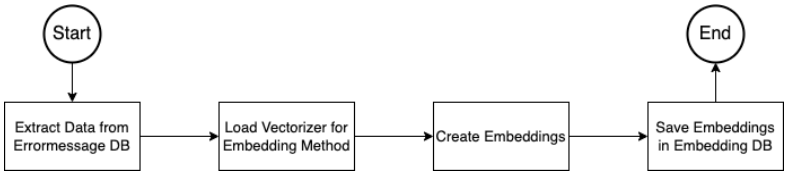


Abbildung 5.1: Flussdiagramm der Erstellung von Einbettungen (eigene Darstellung)

ausgegeben.
Zu Anfang wird die Eingabefehlermeldung mit einer festgelegten Einbettungsmethode in eine Einbettung umgerechnet. Hierauf werden alle Einbettungen mit der Einbettung der Eingabe mittels Vergleichsmetrik verglichen und danach sortiert. Es werden die n ähnlichsten, sich unterscheidenden Indexe der Fehlermeldungen gesammelt. Die Zahl n wird entweder über Parameter im Aufruf des Tools mitgegeben oder wird per Konfigurationsdatei gesetzt. Die gesammelten Indexe werden verwendet, um die Zeilen in der ursprünglichen Errormessage DB in eine Ausgabedatei, die nach Anforderung F8 im CSV Format zu schreiben ist.

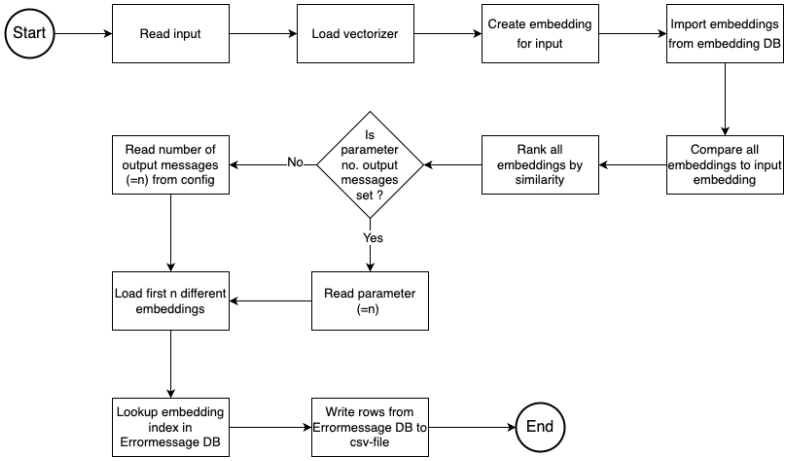


Abbildung 5.2: Flussdiagramm einer Suche nach ähnlichsten Fehlermeldungen (eigene Darstellung)

Zusammenfassend zeigt sich, wie die Anforderungen F1, F2 und F8 aus fachlicher Perspektive umgesetzt werden. Fundamental ist hier die Teilung des Systems in die Prozesse der Einbettung und der Suche. Die weiteren Anforderungen F3 bis F7 und NF1 bis NF3 können zu diesem Stand nicht untersucht werden und sind Teil der Wahl der Einbettungs-

methode. Der Abgleich mit der Anforderungen mit einer Einbettungsmethode findet sich in Kapitel 7.

## 5.2 Technische Sicht

Die Abbildung 5.3 zeigt die Architektur des Tools und stellt die technische Sicht dar. Es ist zu sehen, dass das System entsprechend der Workflows im Unterkapitel 5.1 in zwei Komponenten aufgeteilt sind: EmbeddingSystem und SearchSystem.

Das EmbeddingSystem zeichnet sich durch den EmbeddingService aus, der die Einbettungen mit einer festgelegten Einbettungsmethode berechnet. Die Daten werden aus der Datenbank Errormessage DB mit der Komponente Dataimport extrahiert. Für die Bereitstellung der Einbettungen werden diese mit der Komponente Dataexport abgespeichert. Das SearchSystem ist für die Suche nach ähnlichsten Fehlermeldungen zuständig. Es enthält eine Benutzerschnittstelle nach außen, wo der Benutzer Eingaben tätigen kann. Diese werden durch den APIController verarbeitet. Der EmbeddingService nimmt eine Fehlermeldung entgegen und produziert eine Einbettung. Der SimilarityService nimmt zum einen diese Einbettung entgegen. Zum anderen werden mithilfe der Komponente Dataimport die bereits kalkulierten Einbettungen aus der Embedding DB geladen und dem SimilarityService übergeben. In diesem werden danach die ähnlichsten Fehlermeldungen inklusive der Indexe der Datenbank ermittelt und der Outputmanagement Komponente übergeben. Diese Komponente bezieht mit den Indexen die Zeilen der Datenbank Errormessage DB und schreibt diese in eine CSV-Datei.

Die Auswahl der verwendeten Einbettungsmethode ist Gegenstand der Experimente in Kapitel 6. Es werden die in den Grundlagen vorgestellten Methoden TF-IDF und SBERT untersucht. Da die Performance von SBERT abhängig von dem zugrundeliegenden Modell ist, werden mehrere Modelle verwendet.

## 5.3 Einsatzmöglichkeiten

Nachdem die Software einsatzbereit ist, gibt es zwei angedachte Möglichkeiten, diese einzusetzen. Die erste Einsatzmöglichkeit sieht eine manuelle Ausführung des Tools vor. Sobald bei einem Test neue Fehlermeldungen erscheinen, kann der Tester die Fehlermeldung kopieren und in das Tool eingeben. Mittels eines Kommandozeilenaufrufs soll die

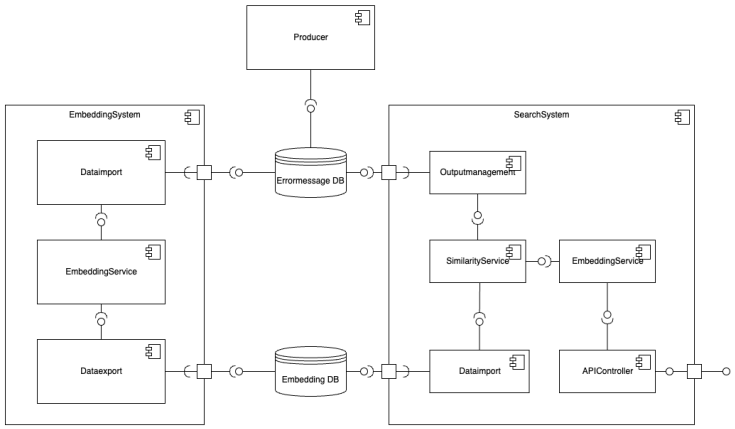


Abbildung 5.3: Architektur der TestAVSearch (eigene Darstellung)

Software gestartet werden. Angefügte Parameter an dem Aufruf sollen beispielsweise die Outputdatei festlegen oder beschreiben, wie viele Ergebnisse ausgegeben werden sollen. Weitere Parameter können in der Konfigurationsdatei gesetzt werden. Dieser Prozess sollte eine der ersten Tätigkeiten sein, wenn ein Testfall fehlschlägt.
Eine zweite Einsatzmöglichkeit ist die Anbindung des Tools an die Pipeline, die die Tests ausführt. Die Pipeline ist ein automatisierter Prozess, welcher bestimmte Aktionen, wie das Kompilieren von Software oder das Starten von Tests ausführt. Dieser Prozess wird in der Regel nach einer bestimmten Zeiteinheit oder beim Hochladen von Änderungen in dem Code gestartet. Die Idee dieser Einsatzmöglichkeit ist die Verwendung des Tools nachdem die Tests in der Pipeline ausgeführt werden. Sobald Fehler in den Tests entstehen, würde das Tool automatisch gestartet werden. So wird der Tester bei der täglichen Überprüfung der Testfehler ebenso eine Liste an ähnlichen Fehlermeldungen finden, was diesem bei der Analyse helfen würde.

# 6 Experimente

Dieses Kapitel beschäftigt sich mit Experimenten zu den Einbettungsmethoden TF-IDF und SBERT. Ziel dieser Experimente ist es eine Einbettungsmethode zu finden, die nach den festgelegten Qualitätskriterien in den Abschnitten 4.2 und 4.3 geeignet sind.
Das erste Experiment untersucht die Modelle der Einbettungsmethode SBERT. Diese sind die Basis für die Berechnungen der Ähnlichkeiten und unterscheiden sich in Parametern, wie Eingabelänge oder Outputlayer und wurden mit unterschiedlichen Daten trainiert. Das Modell, welches in diesem Experiment die besten Ergebnisse erzielt hat, wird in dem zweiten Experiment verwendet. In dem zweiten Experiment werden TF-IDF und SBERT miteinander verglichen.
In den Experimenten wird zu Beginn die Fragestellung verdeutlicht. Danach wird der Aufbau und die Durchführung erläutert, woraufhin die Ergebnisse und Auswertungen folgen. Eine finale Auswertung aller Experimente findet sich in dem nächsten Kapitel 7.

## 6.1 Technische Details

Für die Erstellung der Einbettungen mit SBERT sowie TF-IDF der gesamten TestAV-Datenbank wird eine NVIDIA Tesla V100 GPU mit 16 GB RAM verwendet, die über einen SSH-Tunnel gesteuert wird.
Für den Prozess der Erstellung der Einbettungen einzelner Fehlermeldungen, sowie die Suche nach ähnlichsten Fehlermeldungen mit der Kosinusähnlichkeit wird eine Intel(R) Core(TM) i7-9850H CPU mit 32 GB RAM verwendet. Der Grund für die Verwendung unterschiedlicher Hardware ist der erhöhte Aufwand der Berechnung der Einbettungen der gesamten Menge an Fehlermeldungen. Diese Berechnung soll idealerweise auf eine leistungsstarke GPU ausgelagert werden. Bei dem Benutzer soll es dennoch möglich sein das Tool lokal zu starten. Daher werden die Messungen des Suchprozess auf einem typischen Arbeitscomputer ausgeführt.

Beide Experimente werden mithilfe der Programmiersprache Python Version 3.8 durchgeführt. Für die Verwendung der vortrainierten Modelle wird die Python-Bibliothek SentenceTransformers verwendet, welche speziell für die SBERT Modelle programmiert wurde. Somit ist es möglich anhand der Modellnamen die aktuellen Modelle herunterzuladen und zu benutzen. Für die Verwendung von TF-IDF wird Klasse TfidfVectorizer der Python-Bibliothek sklearn verwendet.
Alle Fehlermeldungen der TestAV-Datenbank werden in einer spaltenorientierten Datenbank gespeichert. Grund dafür ist die effiziente Speichernutzung. Zum Stand der Experimente ist die Datenbank in einer parquet-Datei mit ungefähr 30 MB gespeichert und verwaltet 316 453 Fehlermeldungen. Dieser Datensatz beinhaltet die gesammelten Fehlermeldungen eines Projektes. Die Gesamtanzahl an Fehlermeldungen beträgt aktuell weit über 400 000.

## 6.2 Vergleich Modelle

Dieses Experiment soll untersuchen, welches Modell sich für den Vergleich von Fehlermeldungen am besten eignet. Alle Modelle sind auf der Huggingface Webseite unter den genannten Modellnamen zu finden. Jegliche Modelle, die die SentenceTransformers Bibliothek unterstützen, können ohne weiteres Anpassen als SBERT Modell verwendet werden. Die Auswahl der Modelle ist basierend auf den Feststellungen der Beschaffenheit der Daten in 4.1 und der daraus folgenden Anforderung.
Keines der vorgestellten Modelle kann allen Anforderungen aus 4.2 entsprechen. Hierfür müsste ein Modell aus eigenem Training erstellt werden. Die Idee hinter der Auswahl ist es, einen Teil der Anforderungen nahe zukommen. Das erste Modell weist eine der höchsten hohe Leistung auf. Das Zweite zielt auf die Sprachbarriere zwischen Deutsch und Englisch ab. Und das Dritte ist für die Verarbeitung der Syntax von Fehlermeldungen spezialisiert. In den Beschreibungen der Modelle steht als Erstes die in dieser Arbeit verwendete Abkürzung des Modells, woraufhin der genaue Namen des Modells folgt, wie dieser auf der Huggingface-Webseite zu finden ist.

**Modell Englisch/multi-qa-mpnet-base-cos-v1:** Dieses Modell erzielt laut der SBERT-Webseite[1] die höchste Leistung (57.46) in den verschiedenen Tests der semantischen Suche. Das Modell ist für das Ähnlichkeitsmaß der Kosinusähnlichkeit geeignet. Es wurde

[1] https://www.sbert.net/docs/pretrained_models.html

auf 215 Millionen englischen Frage-Antwort-Paaren trainiert und primär für die semantische Suche angepasst. Die maximale WordPiece-Länge der Eingabe ist mit 512 einer der höchsten und wird den Anforderungen aus 4.2 gerecht. Die ausgegebenen Einbettungen haben eine standardisierte Länge von 768.
Neben diesem Modell stand das all-mpnet-base-v2-Modell zur Auswahl. Diese erzielt laut SBERT.net den höchsten Durchschnitt in allen Tests und einen ähnlich hohen Wert von 57.02 in den Test der semantischen Suche. Da jedoch die maximale WordPiece Eingabelänge auf 384 beschränkt ist, wird das erste Modell verwendet.

**Modell Multilang/distiluse-base-multilingual-cased-v1:** Dieses Modell soll sich auf die deutsch-englische Sprachbarriere fokussieren. Es wurde neben deutschen und englischen Texten mit 13 weiteren Sprachen trainiert. In den Tests zu der semantischen Suche zeigt das Modell das beste Ergebnis für ein multilinguales Modell, welches jedoch mit dem Wert von 29.87 weit unter dem rein englischen Modell liegt. Über die Trainingsdaten sind keine weiteren Informationen bekannt. Außerdem hat das Modell eine maximale WordPiece-Eingabelänge von 128. Die Länge der Einbettungen als Vektoren sind mit 512 geringer als die von dem englischen Modell und entsprechen nicht den Anforderungen.

**Modell Stack/stackoverflow_mpnet-base:** Dieses Modell wurde auf 18 Millionen Frage-Antwort-Paaren von der Internetplattform Stackoverflow trainiert. Mit dem Modell wird die Syntax von Fehlermeldungen abgedeckt. Die Länge der ausgebenden Vektoren beträgt 768. Die maximale WordPiece-Eingabelänge beträgt 512.

### 6.2.1 Durchführung

Das Experiment wird in zwei separate Prozesse unterteilt, die der Architektur in 5.3 entspricht.
In dem ersten Prozess werden die Einbettungen erstellt. Dazu werden die genannten vortrainierten Modelle mithilfe der Python-Bibliothek SentenceTransformers geladen. Danach werden die Einbettungen für jedes Modell für alle Fehlermeldungen der TestAV-Datenbank erstellt und als numpy-Datei abgespeichert, sodass diese für den nachfolgenden Prozess verwendet werden können. Es findet zudem eine Zeit- und Speichermessung für die Erstellung der Einbettungen statt.

In dem zweiten Prozess werden ausgesuchte Fehlermeldungen in das Tool eingegeben und ähnliche Fehlermeldungen ausgegeben.

Zu diesem Zweck werden zehn Fehlermeldungen von dem Test-Team ausgewählt. Dabei wurde zum einen beachtet, dass die Fehlermeldungen in sich unterschiedlich sind. Zum anderen sollen in den Fehlermeldungen bereits erkennbare Elemente enthalten sein, sodass eine Bewertung der ähnlichsten Fehlermeldungen mit der Fachexpertise des Bewertenden entstehen kann.
In dem nächsten Schritt werden alle zehn Fehlermeldungen in Einbettungen umgerechnet. Danach wird jede dieser Einbettungen mit allen anderen Einbettungen aus der abgespeicherten Datei aus dem ersten Prozess verglichen. Die jeweils zehn ähnlichsten und unterschiedlichen Fehlermeldungen werden in einer Excel-Datei abgespeichert. Die Vergleichsmetrik ist die Kosinusähnlichkeit. Dieser Vorgang wird für jedes der drei Modelle wiederholt. Für jeden Vorgang findet für die Erstellung der Einbettung und Berechnung der ähnlichsten Fehlermeldungen eine gemeinsame Zeitmessung statt.

In dem letzten Schritt wird das Test-Team die Ergebnisse als Fachexperten sichten. Für jede Fehlermeldung soll bestimmt werden, welches Modell die fachlich besten Ähnlichkeiten gefunden hat. Dabei hat die ähnlichste der zehn Fehlermeldungen das größte Gewicht. Ist die erste Fehlermeldung gleich oder unterscheidet sich nur kaum, wird die nächstähnlichste Fehlermeldung betrachtet.
Den drei Modellen wird eine Rangfolge vergeben, um das beste, zweitbeste und schlechteste Ergebnis zu kennzeichnen. Daneben soll mit Ja oder Nein beschrieben werden, ob die Ausgaben pro Eingabefehlermeldung hilfreich sind.

### 6.2.2 Beobachtung

In der Abbildung 6.1 ist die Bewertung der Modelle nach deren Ausgaben zu sehen. Das Balkendiagramm beschreibt, wie häufig ein Modell einen bestimmten Rang erzielt, wobei 1 das beste Ergebnis markiert. Vorab ist zu erwähnen, dass viele Ausgaben gleich oder sehr ähnlich waren. Das führt dazu, dass bei einer Eingabefehlermeldung mehrere Modelle den gleichen Rang erhalten. In einigen Fällen zeigen mehrere Modelle identische Ausgaben. In anderen Fällen unterscheiden sich die Ausgabefehlermeldungen nur in den Attributausprägungen wie ”Wert ’-8281.68’ ”, wobei der restliche Teil der Fehlermeldungen identisch sind.
In der Grafik zeigt sich, dass die Modelle Multilang und Stack 7-mal das beste Ergebnis erzielt haben und damit die meisten besten Ergebnisse zeigen. Jedoch ist auch zu erkennen, dass das Stack Modell zweimal das schlechtestes Ergebnis erzielt hat. Von allen Ausgaben wurde lediglich eine als nicht hilfreich eingestuft. Diese Ausgabe stammt von

| Aufgabe/Modell | Multilang | Englisch | Stack |
|---|---|---|---|
| Berechnung aller Einbettungen | 4:58 min | 18:27 min | 18:30 min |
| Berechnung einer Einbettung und Ähnlichkeitssuche | 0.87 s | 1.22 s | 1.14 s |
| Speicherverbrauch aller Einbettungen | 632,9 MB | 949,4 MB | 949,4 MB |

Tabelle 6.1: Exp. 1: Ergebnisse der Zeitmessung und Speichernutzung (eigene Darstellung)

dem Stack Modell. Mit einem Durchschnittsrang von 1,3 erzielt das Multilang Modell das beste Ergebnis. Bemerkenswert ist hier die gefundene Übersetzung einer Fehlermeldung, die nur das Modell Multilang ausgegeben hat. Bei der Eingabefehlermeldung "[...] There is a difference between the defining data and the generated code[...]" war eine der Ausgaben "[...]Differenz zwischen Defdaten und generiertem Code[...]".

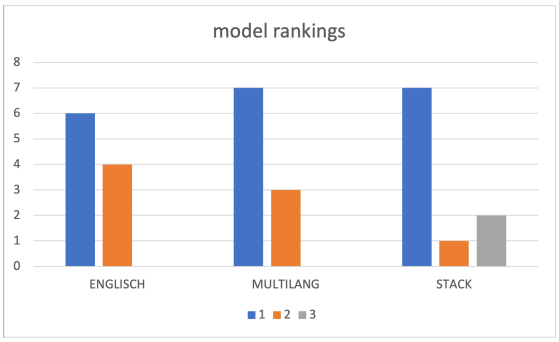


Abbildung 6.1: Ergebnisse Exp. 1: SBERT Modelle (eigene Darstellung)

In Tabelle 6.1 sind die Zeitmessungen und Speichernutzungen der verschiedenen Modelle zu sehen. Es ist zu erkennen, dass alle Ergebnisse von dem Modell Englisch und Stack nah beieinander liegen bzw. identisch sind. Die Werte des Modells Multilang unterscheiden sich stark. Das Multilang Modell berechnet die Einbettungen in fast einem Viertel der Zeit von den beiden anderen Modellen. Außerdem verbrauchen die Einbettungen dieses Modells einen Drittel weniger Speicherplatz.

### 6.2.3 Auswertung

Es zeigt sich, dass das Modell Multilang in allen Bereichen die besten Ergebnisse vorweist. Das Modell ist in der Verarbeitung der Fehlermeldungen am effizientesten und verbraucht am wenigsten Speicherplatz. Der Grund für diesen Vorteil kann die verringerte Vektorgröße der Einbettungen von 512 sein. Somit hat jede Einbettung ein Drittel weniger Werte als die 768 großen Vektoren der beiden anderen Modelle. Dies spiegelt sich im Speicher wieder, welche ebenso um ein Drittel kleiner ist.
Ebenso zeigt das Multilang Modell die besten fachlichen Ergebnisse in den Ausgabefehlermeldungen. Bemerkenswert ist an diesem Ergebnis, dass die Informationen der Fehlermeldungen auf einem kleineren Vektor abgebildet wurden. Grund hierfür kann die Spezialisierung des Modells auf verschiedene Sprachen sein. Durch einen Tokenizer, dessen Vokabular ebenso deutsche Wörter enthält, können diese Wörter besser vom Modell dargestellt werden. In dem Fall der anderen Modelle werden deutsche Wörter entweder so geteilt, dass sie dem Vokabular entsprechen oder sie werden als out-of-vocabulary angesehen. In beiden Fällen ist die Semantik hinter den deutschen Wörtern nicht gegeben.

## 6.3 Vergleich TF-IDF und SBERT

Dieses Experiment vergleicht die Einbettungsmethoden TF-IDF und SBERT angewendet auf die Ähnlichkeitssuche der Fehlermeldungen. Das in dem vorherigen Experiment ermittelte Modell Multilang wird für die Einbettungemethode des SBERTs verwendet.

### 6.3.1 Durchführung

Wie in dem vorherigen Experiment wird dieses in zwei Prozesse entsprechend der Architektur in 5.3 unterteilt.
Die Erstellung der Einbettungen für die Einbettungsmethode SBERT entspricht dem Verfahren aus dem ersten Experiment. Die Einbettungen werden in diesem Fall nicht neu berechnet, sondern aus den abgespeicherten npy-Dateien entnommen.
Für die Einbettungsmethode TF-IDF wird die Python Bibliothek TfidfVectorizer verwendet. Die Erstellung der Einbettungen wird mit einem Methodenaufruf realisiert. Die weiteren Schritte sind analog zum ersten Experiment und werden daher nicht erneut erklärt.

### 6.3.2 Beobachtung

Das Kreisdiagramm in 6.2 zeigt die prozentuale Häufigkeit des besten Ergebnis einer Einbettungsmethoden. Da in 4 von 10 Ausgaben beide Methoden gleich bewertet wurden, wird die Kategorie ”Even” hinzugefügt, die diese Gleichheit markiert. Diese Darstellung wird bewusst so gewählt um zu zeigen, wann welche Methode besser bewertet wurde, als die Andere. Es zeigt sich, dass in 5 von 10 Ausgaben, SBERT bessere Ergebnisse aufweist. In 1 von 10 Ausgaben zeigt TF-IDF ein besseres Ergebnis. Außerdem wurden 2 von den 10 Ausgaben vom TF-IDF als nicht hilfreich eingestuft. Bei dem Modell Multilang waren alle Ausgaben laut Fachexperten hilfreich.

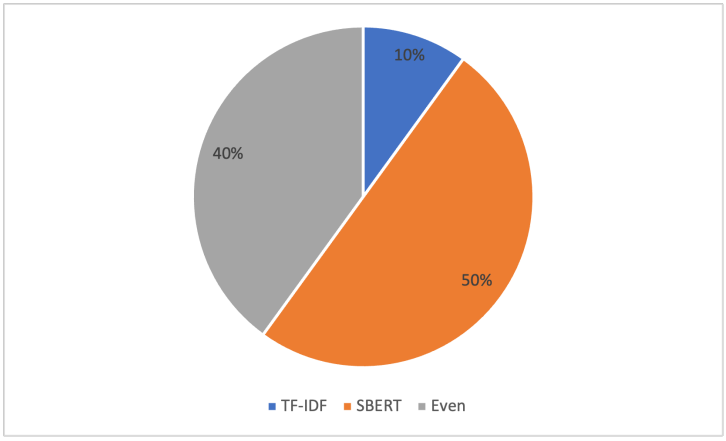


Abbildung 6.2: Ergebnisse Exp. 2: TF-IDF vs. SBERT (eigene Darstellung)

Die Tabelle 6.2 zeigt die Zeit- und Speichermesssungen von TF-IDF und SBERT. Es ist zu erkennen, dass in allen Bereichen TF-IDF einen deutlichen Vorsprung gegenüber von SBERT hat. TF-IDF berechnet alle Einbettungen der 316 453 Fehlermeldungen in 9 Sekunden und ist damit etwa 31 mal schneller als das performanteste SBERT Modell, welches in dieser Arbeit untersucht wurde. Mit 0.3 Sekunden bei TF-IDF ist die Einbettung einer Fehlermeldung und Suche nach der Ähnlichsten fast dreimal schneller als bei dem SBERT. Ebenso ist der Speicherverbrauch der TF-IDF Einbettungen um das 32-fache gesenkt.

### 6.3.3 Auswertung

Die Einbettungsmethode TF-IDF weist die besten Ergebnisse in der Geschwindigkeit der Einbettung der Fehlermeldungen und den geringsten Speicherverbrauch auf. Damit zeigt

| Aufgabe/Methode | TF-IDF | SBERT |
|---|---|---|
| Berechnung aller Einbettungen | 0:09 min | 4:58 min |
| Berechnung einer Einbettung und Ähnlichkeitssuche | 0.3 s | 0.87 s |
| Speicherverbrauch aller Einbettungen | 19,9 MB | 632,9 MB |

Tabelle 6.2: Exp. 2: Ergebnisse der Zeitmessung und Speichernutzung (eigene Darstellung)

sich, dass der TF-IDF Algorithmus eine geringere Komplexität als das neuronale Netz des SBERTs besitzt. Außerdem ist der Speicherverbrauch stark bei dem TF-IDF gesenkt, da die Vektoren viele Nullen enthalten und somit der dünnbesetzte Vektor effizienter gespeichert werden kann.
Auf der anderen Seite zeigt der SBERT die besseren Ergebnisse in der fachlichen Analyse der Ergebnisse. Durch die Verwendung von einer transformerbasierten Architektur kann die Semantik der Fehlermeldungen besser erfasst werden. Zu dem können durch das multilinguale Modell Übersetzungen von Wörtern besser gefunden werden.
Zusammengefasst lässt sich sagen, dass das TF-IDF deutlich schneller arbeiten kann, während der SBERT semantisch akkuratere Einbettungen erzeugt.

# 7 Auswertung der Ergebnisse

In diesem Kapitel werden die Ergebnisse der Experimente zusammengefasst und den Anforderungen gegenübergestellt. Es soll damit diskutiert werden, ob eine geeignete Einbettungsmethode nach den Anforderungen gefunden werden konnte.

## 7.1 Zusammenfassung

In dem ersten Experiment hat sich herausgestellt, dass die Nutzung eines mehrsprachigen SBERT Modells die besten Ergebnisse unter den untersuchten Modellen liefert. Der Vorteil an diesem Modell ist die Anwendbarkeit auf deutsche und englische Fehlermeldungen, wie diese in der Datenbank vorhanden sind. Zudem verbraucht dieses Modell in der Evaluation am wenigsten Rechenzeit und Speicherplatz für die Sicherung der Einbettungen. Verglichen mit dem TF-IDF ist dieser Verbrauch nach dem zweiten Experiment dennoch sehr hoch. Die Einbettung der Fehlermeldungen der Datenbank sowie Abspeicherung dieser ist bei dem effizientesten SBERT Modell um mehr als das 30-fache erhöht. Auf der anderen Seite zeigt die Einbettungsmethode SBERT bessere Ergebnisse in der fachlichen Bewertung der ähnlichsten Fehlermeldungen.
Schlussendlich gibt es nun zwei Lösungen, die gegeneinander abgewägt werden müssen. Die Lösung TF-IDF, welche die Kapazitäten Rechenzeit und Speicherplatz schont und die Lösung SBERT mit dem Modell Multilang, welche fachlich bessere Ergebnisse liefert. Indem beide Methoden den Anforderungen gegenübergestellt werden, wird ermittelt, welche Lösung für das Tool TestAVSearch verwendet wird.

## 7.2 Vergleich mit den Anforderungen

In dem Kapitel 4.2 werden die Anforderungen an die Software beschrieben. Es werden nun diese Anforderungen in den Tabellen 4.2 und 4.3 nacheinander mit den Merkmalen

der zwei Lösungen abgeglichen. Die folgenden Feststellung zu dem SBERT beziehen sich lediglich auf das untersuchte Modell Multilang.

**F1 - Das System soll aus einer gegebenen Fehlermeldung die n ähnlichsten Fehlermeldungen der TestAV-Datenbank ausgeben**: Sowohl TF-IDF als auch SBERT erfüllen diese Anforderung und zeigen dies in den Experimenten. Mit einem Abgleich aller Einbettungen der Fehlermeldungen mit der Einbettung der Eingabefehlermeldungen können alle Ähnlichkeiten in Form eines numerischen Wertes beschrieben werden. Die Einbettungen können nach diesen Werten sortiert werden. Danach können maximal n viele ähnlichste Fehlermeldungen ausgegeben werden. Die Zahl n beschränkt sich auf die Anzahl aller Fehlermeldungen der Datenbank.
**F2 - Die Berechnung der Einbettungen muss im Vorhinein möglich sein**: Durch eine Abspeicherung der Einbettungen ist es möglich diese im Vorhinein zu berechnen. Diese können dann zu einem beliebigen Zeitpunkt verwendet werden. In TF-IDF und SBERT werden diese mit der Python-Bibliothek numpy realisiert.
**F3 - Es sollen englische Texte verarbeitet werden können**: Beide Einbettungsmethoden akzeptieren alle Wörter. Sie akzeptieren jegliche Wörter, unabhängig davon, ob sie einer Sprache angehören. Jedoch stellt sich die Frage, ob die Einbettungsmethoden darauf ausgelegt sind, englische Wörter zu verarbeiten. TF-IDF berechnet die Einbettungen auf Basis der Wortanzahlen. Es gibt hier keinen semantischen Bezug zu den Wörtern. Daher macht es keinen Unterschied für den TF-IDF, welcher Sprache die Wörter angehören. Es ist nicht klar zu sagen, dass diese Anforderung erfüllt ist. Die nachfolgenden Anforderungen F4 und F5 sind im Fall von TF-IDF analog zu behandeln. In dem Fall von dem SBERT Modell Multilang ist es deutlicher. In dem Trainingsprozess des Transformers wurden englische Datensätze verwendet, sodass das Modell mit dieser Sprache arbeiten kann. Daher ist diese Anforderung im Fall SBERT erfüllt.
**F4 - Es sollen deutsche Texte verarbeitet werden können**: Der Fall TF-IDF wird in dem Vergleich der Anforderung F3 erklärt. Das SBERT Modell ist wie in der Anforderung ebenso für deutsche Texte ausgelegt. Daher ist diese Anforderung ebenso erfüllt.
**F5 - Die Syntax von Fehlermeldungen soll berücksichtigt werden**: Der Fall TF-IDF wird in dem Vergleich der Anforderung F3 erklärt. Das SBERT Modell ist nicht für diese Syntax ausgelegt. Jedoch akzeptiert das Modell solche Eingaben und kann diese verarbeiten. Diese Anforderung ist wie TF-IDF weder vollständig erfüllt, noch nicht erfüllt.
**F6 - Unbekannte Begriffe sollen berücksichtigt werden**: Der TF-IDF untersucht, wie beschrieben, lediglich die Wortanzahlen. Gehören Wörter keiner Sprache an, so kön-

nen diese genauso gut verarbeitet werden. Es ist hier ebenso unklar, wie in F3, F4 und F5. SBERT hingegen verwendet die WordPiece Tokenisierung. Diese hat ein spezifisches Vorgehen für out-of-vocabulary Wörter. Diese werden somit gesondert behandelt und die Anforderung ist erfüllt.
**F7 - Eingaben mit bis zu 512 WordPieces sollen akzeptiert werden**: Dies ist eine SBERT spezifische Anforderung. Dennoch kann erschlossen werden, dass TF-IDF diese Anforderung erfüllt. Grund dafür ist, dass TF-IDF theoretisch kein Limit für Eingaben besitzt. SBERT hingegen hat ein Limit für die Eingabelänge. Bei dem Multilang Modell ist diese 128 WordPieces. Damit ist diese Anforderung nicht erfüllt.
**F8 - Die Ausgaben sollen in eine CSV-Datei geschrieben werden**: Alle Ausgaben können mit Python-Bibliotheken in eine CSV-Datei geschrieben werden. Dies ist unabhängig von der Einbettungsmethode und daher ist diese Anforderung in beiden Fällen erfüllt.

**NF1 - Die Verarbeitung der Anfrage darf nicht länger als 10 Sekunden betragen**: Beide Einbettungsmethoden brauchen für die Einbettung der einzelnen Eingabefehlermeldung und für den Vergleich aller anderen Einbettungen weniger als eine Sekunde. Die Anforderung ist in beiden Fällen erfüllt.
**NF2 - Die Berechnung der Einbettungen darf 120 Minuten nicht überschreiten**: Auf einer modernen Cloud-GPU unterschreiten beide Einbettungsmethoden 5 Minuten. Dennoch ist erwähnenswert, dass mit TF-IDF diese Aufgabe in 9 Sekunden erledigt wurde.
**NF3 - Die Einbettungen dürfen nicht mehr als ein Gigabyte an Speicherplatz einnehmen**: Beide Einbettungsmethoden erfüllen diese Anforderung. Es ist dennoch wieder anzumerken, dass die TF-IDF Einbettungen mit etwa 20 MB SBERT deutlich weniger Speicherplatz benötigen, als die des SBERTS.

Es zeigt sich, dass TF-IDF keine Anforderung nicht erfüllt. Dennoch gibt es einige sprachbasierte Anforderungen, wie F3, die TF-IDF nicht klar und eindeutig erfüllt. Denn es ist schwer möglich mit der Analyse der Wortanzahlen Schlussfolgerungen die die Semantik der Wörter in Einbettungen zu fassen. Das zeigt sich gut an dem Beispiel von übersetzten Fehlermeldungen. Da die Wörter ”Differenz” und ”Difference” nicht gleich sind, kann TF-IDF hier keine Ähnlichkeiten finden.
Anders ist es im Fall von SBERT. Hier können die semantische Inhalte der Fehlermeldungen in der Suche berücksichtigt werden, indem ein Modell auf solchen Daten trainiert wurde. Trotzdem ist es wichtig zu erwähnen, dass SBERT Modell nicht allen Anforderungen entspricht. Zum einen wird die Syntax von Fehlermeldungen nicht berücksichtigt

und die Eingaben werden ab einer Länge von 128 WordPieces abgeschnitten. Da jedoch die Berücksichtigung der Semantik laut den Bewertungen der Fachexperten bessere Ergebnisse liefert, wird SBERT mit dem Modell Multilang als eignungsreifer für den Anwendungsfall der Suche nach ähnlichsten Fehlermeldungen erklärt.

# 8 Fazit und Ausblick

Dieses letzte Kapitel dieser Arbeit beinhaltet das Fazit der Ergebnisse der gesamten Arbeit und teilt mit, wie an dieser Arbeit weiter gearbeitet werden kann. Zu Anfang werden die Problemstellung und die Anforderungen zusammengefasst und aufgegriffen. Danach wird die Problemlösung beschrieben. Dabei wird insbesondere auf die Ergebnisse der Experimente hingewiesen. Schließlich wird das Ergebnis der Arbeit beschrieben und es wird ausgewertet, ob die gesamte Arbeit nach den Kriterien aus 1.2 erfolgreich ist. Der Ausblick wird in einem separaten Unterkapitel ausgeführt.

## 8.1 Fazit

Das Ziel dieser Arbeit ist die Untersuchung einer Methode für einen effizienten Vergleich von Fehlermeldungen. Diese entstammen aus der TestAV-Datenbank, in der die Fehlermeldungen von den Systemen der msg life gesammelt werden. Die Fehlermeldungen werden durch mehrere Tausende Regressionstests ausgelöst, welche nahezu täglich ausgeführt werden. Die Motivation hinter der Suche nach Ähnlichkeiten ist die Unterstützung des Tester bei der Analyse der Fehlermeldungen. Denn es besteht die Möglichkeit, dass die neuen Fehlermeldungen bereits vorherigen Tests entstanden sind und die Lösung bereits bekannt ist. Wenn also ein Tester ein Werkzeug für die Findung gleicher oder ähnlicher Fehlermeldungen besitzt, dann können die Erfahrungen und Lösungen der Vergangenheit genutzt werden. Ähnlich gehen Entwickler vor, die Fehlermeldungen untersuchen, welche aus einem Open-Source-System entstammen. Das Stack Overflow Q&A Forum ist ein Beispiel dafür. Hier kann ein Entwickler Fehlermeldungen oder Probleme eingeben und erhält im besten Fall eine Lösung, die von anderen Entwicklern empfohlen wurde. Ähnlich soll so ein Tester oder Entwickler der msg life mit systeminternen Fehlermeldungen arbeiten können. Es soll die Suche nach ähnlichsten Fehlermeldungen in der TestAV-Datenbank möglich sein.
Da dieses Problem Teil der natürlichen Sprachverarbeitung ist, werden die Methoden

der NLP angewendet. Der grobe Ansatz der Lösung aus Kapitel 5 ist die Verwendung einer Methode, die die Fehlermeldungen in Zahlen übersetzen, sodass diese besser von einem Computer erfasst werden können. Danach können diese Darstellungen der Fehlermeldungen mit Vergleichsmetriken verglichen werden. Zwei in dieser Arbeit untersuchte Ansätze für die die Übersetzung der Fehlermeldungen sind TF-IDF und SBERT. Beide Methoden erhalten einen Text als Eingabe und bilden diese auf einen Vektor ab. TF-IDF ist ein Ansatz, dessen Ursprung bereits in dem Jahre 1957 liegt. Der vergleichsweise einfache Algorithmus stützt sich auf die Analyse der Wortanzahlen. SBERT hingegen ist eine transformerbasierte Methode, die 2019 zum ersten mal eingeführt wurde. Diese Methode arbeitet mit neuronalen Netzen und wird auf Sprache trainiert. Es wird damit versucht die Semantik der Sprache in den Vektoren zu repräsentieren. Die Vergleichsmetrik, die zum Vergleich der Vektoren verwendet wird, ist die Kosinusähnlichkeit. Mehrere Anwendungsfälle, die in 2.7 untersucht werden, unterstützen die Verwendung der Kosinusähnlichkeit.
Da die Vor- und Nachteile der Einbettungsmethoden TF-IDF und SBERT unterschiedlich sind, werden diese in 6 experimentell untersucht. Die Experimente untersuchen zum einen die geeignetsten Modelle für SBERT und zum anderen die Leistungen in einem Vergleich von TF-IDF und SBERT. Es stellte sich heraus, dass SBERT fachlich die besseren Ergebnisse in der Findung der Ähnlichkeit vorweist. Dennoch ist wichtig zu erwähnen, dass dieser Ansatz erheblich mehr Rechen- und Speicherkapazitäten erfordert, als TF-IDF. Während SBERT für die Berechnung aller Fehlermeldungen etwa 5 Minuten braucht, schafft TF-IDF dies in 9 Sekunden. Bei dem finalen Abgleich der Anforderungen in Kapitel 7.2 stellt sich heraus, dass SBERT mit dem Modell Mutlilang die geeignetste Einbettungsmethode für den Anwendungsfall ist. Dadurch, dass die Berechnung der Einbettungen in einem separaten Prozess geschehen kann, wird dieser ausgelagert. Das hat den Vorteil, dass beim Anwender lediglich die Eingabefehlermeldung eingebettet werden muss und diese mit allen anderen verglichen werden muss. Bei den Experimenten zeigt sich, dass dieser Aufwand auf eine handelsüblichen CPU ungefähr 1 Sekunde dauert.

Es hat sich nun eine Lösung ergeben, die anwendbar für die Problemstellung ist. Die Einbettungsmethode SBERT entspricht nicht allen Anforderungen. Dennoch zeigt sich in den Experimenten, die Anwendbarkeit dieser Methode. Somit lässt sich sagen, dass eine Einbettungsmethode erfolgreich gefunden werden konnte. Der Erfolg dieser Arbeit ist ebenso gelungen. Im Rahmen der Arbeit werden verschiedene Ansätze und Themengebiete der Problemstellung mithilfe von wissenschaftlichen Quellen dargelegt. In Kapitel 4.1 wird die Ausgangslage erfasst, woraufhin die Anforderungen gebildet werden konnten.

Durch die Analyse der verschieden möglichen Ansätze zeigen mehrere Methoden, dass sie dieses Problem lösen können. Alle Methoden werden fachlich analysiert und bewertet. Die Auswertung der Ergebnisse in Kapitel 7 zeigt erfolgreiche Ergebnisse.

## 8.2 Ausblick

In diesem Unterkapitel werden Möglichkeiten für die Erweiterung dieser Arbeit beschrieben. Zum einen gibt es ein Potenzial, um das Ergebnis weiter zu verbessern. Zum anderen gibt es weitere Problemstellungen, die mit diesen oder ähnlichen Methoden der NLP gelöst werden können.
Eine Möglichkeit die Ergebnisse weiter zu verbessern ist eine Vorverarbeitung der Daten. Vor allem in dem Bereich der Fehlermeldungen gibt es verschiedene Stellen, wo zusammenhängende Wörter getrennt werden können. Ein Beispiel ist diese Zeichenkette aus einer Fehlermeldung: "/ref/determineCompleteInfoResponse[1]/return[1]/value[1]/". Mit einer Vorverarbeitung der Daten könnten die "/" als mit einem Leerzeichen getrennt werden.
Es gibt auch die Möglichkeit, weitere Daten aus der TestAV-Datenbank zu verwenden. Felder, wie getestete Komponenten und Rechenzeit können ebenso eingebracht werden. In dem Fall würde die bisherige Eingabe diese Felder ergänzt werden. Es gibt sogenannte multimodale Modelle, welche tabulare Eingabedaten verarbeiten.
Ein weiterer Ansatz ist die Erweiterung des SBERT Modells. Hier gibt es zwei Möglichkeiten, sodass das Modell auf die eigenen Daten spezialisiert ist. Zum einen können Trainingsdaten selbst erstellt werden, die die internen Begriffe der msg life Systeme beinhalten. Mit den Trainingsmethoden der Next-Sentence-Prediction und dem Masked Language Model aus 2.5 wird dies realisiert. Zum anderen kann ein bestehendes und bereits trainiertes Modell verwendet werden, welches um die hauseigenen Trainingsdaten ergänzt wird.

Es gibt neben den Erweiterungen dieser Lösung die Möglichkeit, eine ähnliche Lösung für ähnliche Anwendungsfälle zu verwenden. Es gibt beispielsweise ein Projekt in der msg life, welches die Ähnlichkeiten in den Tickets ermittelt. Tickets, die zum Beispiel die Aufgabe beinhalten Fehlermeldungen zu lösen oder neue Features einzubauen, haben eine Beschreibung, Kommentare sowie ein Titel. Es gibt hier auch die Möglichkeit, dass eine neue Aufgabe bereits in einem anderen Projekt erledigt wurde. Dazu soll diese Suche nach ähnlichen Tickets diese Arbeit vereinfachen. Es ist möglich, die Lösung dieser Arbeit

auf das Problem anzusetzen. Mit einem einfachen Austausch des SBERT Modells können die spezifischen Merkmale der Tickets berücksichtigt werden.

# Literaturverzeichnis


[1] Abdulaziz Alaboudi, Thomas D. L.: An Exploratory Study of Debugging Episodes. (2021). – URL https://arxiv.org/abs/2105.02162

[2] Adhikari, Ashutosh ; Ram, Achyudh ; Tang, Raphael ; Lin, Jimmy: *DocBERT: BERT for Document Classification.* 2019. – URL https://arxiv.org/abs/1904.08398

[3] Bello, Abayomi ; Ng, Sin-Chun ; Leung, Man-Fai: A BERT Framework to Sentiment Analysis of Tweets. In: *Sensors* 23 (2023), Nr. 1. – URL https://www.mdpi.com/1424-8220/23/1/506. – ISSN 1424-8220

[4] Delip Rao, Brian M.: *Natural Language Processing with PyTorch.* O'Reilly, 2020. – ISBN 978-3-96009-118-9

[5] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2018. – URL https://arxiv.org/abs/1810.04805

[6] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2018. – URL https://arxiv.org/abs/1810.04805

[7] Géron, Aurélien: *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow.* O'Reilly, 2019. – ISBN 987-1-492-03264-9

[8] Haque, Mubin U. ; Dharmadasa, Isuru ; Sworna, Zarrin T. ; Rajapakse, Roshan N. ; Ahmad, Hussain: *Ï think this is the most disruptive technology": Exploring Sentiments of ChatGPT Early Adopters using Twitter Data.* 2022. – URL https://arxiv.org/abs/2212.05856

[9] Hasan, Fatema ; Roy, Arpita ; Pan, Shimei: Integrating Text Embedding with Traditional NLP Features for Clinical Relation Extraction. In: *2020 IEEE 32nd*

*International Conference on Tools with Artificial Intelligence (ICTAI)*, 2020, S. 418–425

[10] Huang, Anna: Similarity measures for text document clustering. In: *Proceedings of the 6th New Zealand Computer Science Research Student Conference* (2008), 01

[11] Jones, K.: A Statistical Interpretation of Term Specificity in Retrieval. In: *Journal of Documentation* 60 (2004), 01, S. 493–502

[12] Kashyap, Pantanjali: *Machine Learning for Decision Makers*. Apress, 2017. – ISBN 978-1-4842-2988-0

[13] Kumar, Ela: *Natural Language Processing*. I.K. International Publishing House Pvt. Ltd., 2011. – ISBN 987-93-80578-77-4

[14] Li, Hang ; Xu, Jun: Semantic Matching in Search. In: *Foundations and Trends® in Information Retrieval* 7 (2014), Nr. 5, S. 343–469. – URL http://dx.doi.org/10.1561/1500000035. – ISSN 1554-0669

[15] Liu, Yinhan ; Ott, Myle ; Goyal, Naman ; Du, Jingfei ; Joshi, Mandar ; Chen, Danqi ; Levy, Omer ; Lewis, Mike ; Zettlemoyer, Luke ; Stoyanov, Veselin: *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. – URL https://arxiv.org/abs/1907.11692

[16] Luhn, H. P.: A Statistical Approach to Mechanized Encoding and Searching of Literary Information. In: *IBM Journal of Research and Development* 1 (1957), Nr. 4, S. 309–317

[17] Martin Svensson, Joakim S.: Machine-learning technologies in telecommunications. (2008). – URL https://www.academia.edu/34634797

[18] Mikolov, Tomas ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: *Efficient Estimation of Word Representations in Vector Space*. 2013. – URL https://arxiv.org/abs/1301.3781

[19] Pradhan, Nitesh ; Gyanchandani, Manasi ; Wadhvani, Rajesh: A Review on Text Similarity Technique used in IR and its Application. In: *International Journal of Computer Applications* 120 (2015), 06, S. 29–34

[20] Prakoso, Dimas ; Abdi, Asad ; Amrit, Chintan: Short text similarity measurement methods: a review. In: *Soft Computing* 25 (2021), 03, S. 1–25

[21] Ramos, Juan: Using TF-IDF to determine word relevance in document queries. (2003), 01

[22] Reimers, Nils ; Gurevych, Iryna: *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. – URL https://arxiv.org/abs/1908.10084

[23] Shima, Keiichi: Catching Unusual Traffic Behavior using TF–IDF-based Port Access Statistics Analysis. In: *2021 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, 2021, S. 1–5

[24] Singh, S.: Text Similarity Measures in News Articles by Vector Space Model Using NLP, 2021, S. 329–338

[25] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: *Attention Is All You Need*. 2017. – URL https://arxiv.org/abs/1706.03762

[26] Zhang, Mingyang ; Chen, Jianfei ; Liu, Jianyi ; Wang, Jingchu ; Shi, Rui ; Sheng, Hua: LogST: Log Semi-supervised Anomaly Detection Based on Sentence-BERT. In: *2022 7th International Conference on Signal and Image Processing (ICSIP)*, 2022, S. 356–361

## Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

| Ort | Datum | Unterschrift im Original |
|---|---|---|