Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Identifizierung von Erklärungsbedarf via User-Feedback-Analyse

Identifying Explanation Needs via User Feedback Analysis

Masterarbeit

im Studiengang Informatik

von

Marvin Kuhnke

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. Jil Klünder
Betreuer: Larissa Chazette, Wasja Brunotte

Hannover, 21.07.2020

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 21.07.2020

Marvin Kuhnke

Zusammenfassung

Identifizierung von Erklärungsbedarf via User-Feedback-Analyse

Die Analyse von Benutzerfeedback in digitalen Distributionsplattformen gewinnt immer mehr an Bedeutung. Es existieren viele Ansätze, wie Benutzerfeedback ausgewertet und in Klassen wie Problemberichte, Feature-Wünsche oder Beschwerden klassifiziert werden kann. Diese Informationen können wertvoll für App-Anbieter und -Entwickler sein, um die Qualität ihrer Produkte zu optimieren und neue Anforderungen zu entdecken. Eine Art Anforderung, die bisher noch nicht in Benutzerfeedback gesucht wurde, ist Erklärungsbedarf. Erklärbarkeit in Softwaresystemen wird durch den Einzug komplexer Entscheidungssysteme in unseren Alltag immer wichtiger. Nutzer wünschen sich Erklärungen über die Entscheidungen solcher Systeme.

Doch geben Nutzer auch Benutzerfeedback ab, in dem sie ihren Erklärungsbedarf kundtun? Diese Arbeit analysiert 1200 Reviews von je sechs Apps aus Apples App Store und Googles Play Store. Das Ergebnis ist, dass die Menge an gefundenem Erklärungsbedarf je nach App schwankt und, dass schlechte Reviews im Schnitt mehr Erklärungsbedarf enthalten. Jedoch ist Erklärungsbedarf insgesamt relativ rar, er konnte in nur circa 5.6% der analysierten Sätze festgestellt werden.

Weiterhin wurden diverse Klassifizierer mit dem Ziel entwickelt, Erklärungsbedarf in Texten zu erkennen. Das beste Ergebnis konnte ein *Convolutional Neural Network* mit einer Präzision von 0.41, einem Recall von 0.53 und einer Spezifität von 0.95. Er übertraf sowohl einfachere Heuristiken, andere Arten von neuronalen Netzen sowie einen komplexeren, auf Satzfragmenten basierenden Klassifizierer.

Abstract

Identifying Explanation Needs via User Feedback Analysis

User feedback analysis in digital distribution platforms is growing in popularity. Many tools and techniques on how to evaluate and sort feedback into classes like bug reports, feature requests and complaints have been developed. This information can be useful for app vendors and app developers alike, to improve the quality of their products and discover new requirements. A type of requirement that so far has not been mined in user feedback is explanation need. Explainability in software systems grows in importance as complex decision systems become part of our every day life. Users desire explanations about the decisions of these systems.

But are users voicing those desires in user feedback? This thesis analyzes 1200 reviews for six apps out of each Apples App Store and Googles Play Store. The findings are that the amount of explanation need varies between different apps and that bad reviews contain, on average, more explanation need. However, on the whole, explanation need is relatively rare with only about 5.6% of analyzed sentences containing it.

Furthermore, various classifiers for explanation need in text have been devised. A *convolutional neural network* was able to produce the best results with a precision of 0.41, recall of 0.53 and specificity of 0.95. It surpassed simple heuristics, other types of neural networks, as well as a more complex, sentence fragment based classifier.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Täglich wird in digitalen App-Vertriebsplattformen wie Apples App Store und Googles Play Store viel Nutzerfeedback in Form von Reviews verfasst. Diese Reviews haben verschiedenste Themen wie Problemberichte, Beschwerden, Feature-Wünsche und mehr. Dieses Nutzerfeedback kann wertvoll für App-Anbieter und -Entwickler sein, um die Produktqualität zu erhöhen und neue Anforderungen zu finden. Eine Kategorie von Feedback, die noch nicht in Reviews gesucht wurde, ist der Erklärungsbedarf. In dieser Arbeit soll untersucht werden, ob und in welcher Menge Nutzer Erklärungsbedarf in Reviews kundtun. Zunächst muss eine Datengrundlage von Reviews geschaffen werden, auf Basis derer weitere Analysen ausgeführt werden können. Zusätzlich soll eine Möglichkeit entwickelt werden, Erklärungsbedarf in Nutzerfeedback möglichst autonom zu erkennen, sodass Feedback in dieser Kategorie für Entwickler einfach zu erfassen ist.

1.2 Lösungsansatz

Um eine Datengrundlage zu schaffen, müssen zunächst Möglichkeiten gefunden werden, um Reviews aus App Store und Play Store herunterzuladen. Da keiner dieser Vertriebsplattformen ein öffentliches *Application Programming Interface* (API) bietet, wurden *Web Crawler* und *Web Scraper* entwickelt, die App-Informationen und Reviews von den menschenlesbaren Internetplattformen des Play Stores und App Stores herunterladen und in strukturierte Informationen umwandeln kann. Mithilfe ihnen wurden 1200 Reviews von sechs Apps je Store heruntergeladen, die als Datengrundlage dienen. Zur Analyse wurden zunächst diese Reviews in Sätze und schließlich die Sätze in Tokens zerlegt. Die resultierende 7222 Sätze wurden jeweils manuell auf Erklärungsbedarf untersucht und markiert, um eine *Ground Truth* für die Entwicklung von automatischen Klassifizierungsverfahren aufzubauen.

Es wurden sechs Ansätze für die automatische Klassifizierung von Sätzen mit Erklärungsbedarf getestet: Zuerst eine einfache Heuristik, basierend auf der Wortanzahl von in Sätzen mit Erklärungsbedarf häufig vorkommender Wörter. Zweitens ein naiver Bayes-Klassifizierer, gefolgt von drei verschiedene Arten von neuronalen Netzwerken. Schließlich ein kombiniertes Verfahren, welches zählt, wie viele Fragmente eines Satzes von einem neuronalen Netzwerk mit Erklärungsbedarf klassifiziert werden.

1.3 Struktur der Arbeit

Kapitel 2 vermittelt zunächst wesentliche Grundlagen, die für die folgenden Kapitel benötigt werden. Kapitel 3 erläutert die in dieser Arbeit umgesetzten Konzepte. Folgend zeigt Kapitel 4 einige Implementierungsdetails. Kapitel 5 diskutiert die Evaluationsresultate dieser Arbeit. Kapitel 6 beschreibt einige verwandte Arbeiten. Schließlich zieht Kapitel 7 ein Fazit über die Resultate dieser Arbeit und bietet einen Ausblick.

Kapitel 2

Grundlagen

Dieses Kapitel vermittelt wesentliche Grundlagen, auf denen diese Arbeit aufbaut. Abschnitt 2.1 erklärt, was Nutzerfeedback ist und wofür es verwendet wird. Folgend erläutert 2.2 den Begriff der Erklärbarkeit und des Erklärungsbedarfs. Abschnitt 2.3 vermittelt Grundlagen der natürlichen Sprachverarbeitung. Es Folgen die Abschnitte 2.4 und 2.5, die die Begriffe Crawler und Scraper erläutern. Abschnitt 2.6 erklärt den Begriff des Klassifizierungsverfahrens. Folgend zeigt Abschnitt 2.7 die Funktionsweise eines naiven Bayes-Klassifikators. Schließlich erklärt Abschnitt 2.8 die Funktionsweise von verschiedenen Arten von neuronalen Netzen.

2.1 Benutzerfeedback

Benutzerfeedback in Form von App-Store-Reviews ist für Benutzer ein wichtiger Indikator für die Qualität einer Applikation [MN15, 117]. Auch werden Applikationen mit besseren Bewertungen von den Stores zum Beispiel höher auf Toplisten gezeigt oder von der jeweiligen Store-Redaktion in einem Feature vorgestellt. Dies erhöht die Sichtbarkeit der Apps und führt zu einer höheren Nachfrage [IJ14];

Doch nicht nur für die Benutzer eines Stores ist solches Feedback interessant. Studien zeigen, dass circa ein Drittel der verfassten App-Store-Reviews Informationen für die Weiterentwicklung einer App von Bedeutung sein können. So beschreiben Guzman et al. [GEHB15, 771-772] in ihrer *User Review Taxonomy for Software Evolution* sieben Kategorien von Feedback: *Bug Reports*, *Feature Strength*, *Feature Shortcoming*, *User Request*, *Praise*, *Complaint* und *Usage Scenario*.

2.2 Erklärbarkeit

Erklärbarkeit in Software-Systemen gewinnt immer stärker an Bedeutung [CS20]. Entscheidungen über Freiheitsstrafen, Kredit-Scores,

Krankheitsbilder und mehr werden immer öfter von komplexen Systemen getroffen, die Algorithmen der künstliche Intelligenz einsetzen. Dadurch kommt es gerade in den vergangenen Jahren vermehrt zu Forderungen für mehr Erklärbarkeit und Transparenz in solchen Systemen, um ihre Entscheidungen nachvollziehen zu können [Sam15].

Doch auch Anwendungen für Endbenutzer werden immer komplexer und gewinnen gleichzeitig an Wichtigkeit. Ein hoher Anteil an Nutzern wünscht sich auch hier mehr Erklärungen, wenn Erwartungen an das System gebrochen werden und wenn unerwartete Ereignisse eintreffen [CKS19, 232].

Software-Systeme, die nachvollziehbare Entscheidungen treffen, genießen ein höheres Vertrauen bei Benutzern [BSO15].

Konkreter Erklärungsbedarf besteht, wenn ein Mangel an Erklärungen für eine Entscheidung eines Systems besteht [Vog19], beispielsweise, wenn ein Nutzerkonto ohne Erklärung gesperrt wird.

2.3 Natürliche Sprachverarbeitung

Natürliche Sprachverarbeitung (*Natural Language Processing*, *NLP*) bezeichnet eine Disziplin aus Linguistik, Informatik und künstlichen Intelligenz, die sich mit Verständnis und Generierung von natürliche Sprache sowie Erkennung von gesprochener natürliche Sprache beschäftigt.

Natürliche Sprache ist ein System, welches so komplex und vielfältig ist, dass nie alle Möglichkeiten modelliert werden können [GAL+06]. Um NLP-Probleme zu lösen, müssen Wortsinn, Wortkategorien, syntaktische Strukturen und semantischer Kontext akkurat erörtert werden. Jedoch steht das Ziel, eine maximale Sprachabdeckung zu erreichen, im Widerspruch zu einer Minimierung der sprachlichen Ambiguität. Eine Erweiterung der Grammatik auf obskure Sprachkonstrukte erhöht die Ambiguität für einfache Sätze und umgekehrt [MS99, 18]. Die Konstruktion von handgefertigten Regeln ist aufwendig. Zudem skalieren sie schlecht und ihre Fragilität skaliert mit der Häufigkeit von Metaphern im Quellentext.

2.3.1 Tokenisierung

Tokenisierung bezeichnet den Prozess, einen gegebenen Text in Sektionen (Tokens) zu zerlegen. Diese Tokens können ganze Sätze, einzelne Wörter oder wortähnliche Konstrukte sein. Betrachtet man den Satz

„I don't expect free, but $145 is a lot!!“, (2.1)

wäre eine einfache Zerlegung in Wörter die Trennung an Trennzeichen wie Leerräumen und Satzzeichen:

$$(I \quad don't \quad expect \quad free \quad but \quad \$145 \quad is \quad a \quad lot)$$

Dies lässt sich beispielsweise um Kontraktionen und Satzzeichen erweitern:

$$(I \quad do \quad not \quad expect \quad free \quad , \quad but \quad \$ \quad 145 \quad is \quad a \quad lot \quad !!)$$

Diese Zerlegung, insbesondere von Kontraktionen, hilft, ein Problem namens Datensparsität abzuschwächen. Datensparsität in natürlicher Sprache zeichnet sich dadurch aus, dass selbst ein großer Textkorpus nur einen kleinen Teil aller durch die Sprache formbaren Sätze enthält [AGG06].

2.3.2 Lemmatisierung

Lemmatisierung bezeichnet die Reduktion von Flexionsformen eines Wortes auf seine Grundform. So kann zum Beispiel „best“ auf „good“ und „going“ auf „go“ reduziert werden. Zusätzlich können Familien von abgeleiteten Wörtern wie „democratic“, „democracy“ und „democratization“ auf eine gemeinsame Grundform reduziert werden. Eine Herausforderung für Lemmatisierer sind Homographen, Wörter mit gleicher Orthographie, aber unterschiedlicher Semantik. Um Homographen auf die richtige Grundform zu reduzieren, muss der Lemmatisierer dessen Kontext betrachten. So muss das Wort „evening“ im Satz „I'm **evening** out the floor.“ auf die Grundform „even“ (*to even*, ebnen) reduziert werden, im Satz „It's a nice **evening**.“ jedoch ist „evening“ bereits seine eigene Grundform (*the evening*, der Abend).

Bergmains und Goldwater zeigen mit *Lematus*, dass Kontext auf Buchstabenebene zu einer signifikant besseren Lemmatisierung von mehrdeutigen Wörtern führt [BG18].

2.3.3 N-Gramm

Ein n-Gramm ist ein Teil einer Zerlegung eines Texts in Fragmente. Diese Fragmente bestehen jeweils aus n aufeinanderfolgenden Tokens des Texts. Diese Tokens können je nach Anwendungsfall zwischen Symbolen, Silben, Wörtern oder anderen Zeichen variieren. Oft nutzt man für n-Gramme mit n gleich eins, zwei oder drei jeweils die Bezeichnungen Monogramm, Bigramm und Trigramm. Monogramme sind lediglich die Menge der Tokens eines des Alphabets. Formal definiert sind n-Gramme aus einem Satz $S = (w_1, w_2, \ldots, w_m)$ die Menge

$$\{(w_{i_1}, w_{i_2}, \ldots, w_{i_n}) | i_j + 1 = i_{j+1}\}. \tag{2.2}$$

Aus dem Satz

„The pricing was extremely reasonable“ (2.3)

ergeben sich die Folgenden n-Gramme:

Ein Vorteil von n-Grammen mit $n > 1$ im Gegensatz zu Monogrammen ist, dass sie effizient mit wachsendem n einen größeren Kontext abdecken.

Bigramme	=	{The pricing, pricing was, was extremely, extremely reasonable}
Trigramme	=	{The pricing was, pricing was extremely, was extremely reasonable}

2.3.4 Skip-Gramm

Skip-Gramme sind eine Generalisierung von n-Grammen. Bei einem k-Skip-n-Gramm folgen die Tokens eines Fragments nicht unbedingt **direkt** aufeinander. Stattdessen besteht es aus n Tokens mit maximal k Sprüngen (*Skips*), bzw. n Tokens aus einem Fenster der Größe $n+k$ [GAL+06]. Formal definiert sind k-Skip-n-Gramme aus einem Satz $S = (w_1, w_2, \ldots, w_m)$ die Menge

$$\{(w_{i_1}, w_{i_2}, \ldots, w_{i_n}) | \sum_{j=2}^{n} i_j - i_{j-1} - 1 \leq k \wedge i_j < i_{j+1}\}\,. \tag{2.4}$$

Skip-Gramme mindern das Problem der Datensparsität. Trennt man beispielsweise den Satz aus Eintrag 2.3 in Trigramme, erhält man lediglich die Trigramme „The pricing was“, „pricing was extremely“ und „was extremely reasonable“, welche nicht die Kernaussage „pricing was reasonable“ erfassen.

Nutzt man 2-Skip-Gramme, um den Satz in Eintrag 2.3 zu zerlegen, erhält man folgende n-Gramme:

2-*Skip-Bigramme*	=	{The pricing, The was, The extremely, pricing was, pricing extremely, **pricing reasonable**, was extremely, was reasonable, extremely reasonable}
2-*Skip-Trigramme*	=	{The pricing was, The pricing extremely, The pricing reasonable, The was extremely, The was reasonable, The extremely reasonable, pricing was extremely, **pricing was reasonable**, **pricing extremely reasonable**, was extremely reasonable}

Sowohl mit 2-Skip-Bigrammen als auch mit 2-Skip-Trigrammen lässt sich die Kernaussage des Satzes erfassen.

2.3.5 Word2Vec

Word2Vec ist ein von Google entwickeltes und patentiertes Word-Embedding-Verfahren [MCCD15]. Wörter werden in Vektoren umgewandelt die oft mehrere hundert Dimensionen umfassen. Semantische und syntaktische Relationen bestehen bleiben. So kann man beispielsweise die Aussage

`Königin` verhält sich zu `Frau`, wie `König` sich zu `Mann` verhält

über die arithmetische Operation

```
Königin = König - Mann + Frau
```

repräsentieren.

2.4 Crawler

Ein Crawler ist eine Software, die Internetseiten indiziert und/oder herunterlädt und archiviert. Ausgehend von einer parametrisierten Startseite werden rekursiv Unterseiten besucht, indiziert und gespeichert. Der Prozess läuft fort, bis entweder keine Unterseiten unbesucht sind, oder eine gewisse Rekursionstiefe erreicht wurde. Internetseiten werden entweder direkt über HTTP heruntergeladen oder über einen Browser aufgerufen, mit dem programmatisch interagiert werden kann. Ein solcher Browser kann nötig sein, wenn benötigte Daten nicht direkt aus der heruntergeladenen Datei erfasst werden können, beispielsweise, wenn Daten dynamisch beim scrollen oder klicken nachgeladen werden. Im Kontext dieser Arbeit werden Crawler genutzt, um App- und Review-Seiten herunterzuladen. Heruntergeladene Seiten bestehen haben jedoch noch nicht die benötigte Informationsstruktur. Um Informationen wie App-Metadaten und Reviews zu extrahieren, kann ein Web-Scraper eingesetzt werden.

2.5 Web Scraping

Web Scraping bezeichnet verschiedene Verfahren zum Extrahieren von Daten aus Internetseiten. Der Unterschied zu einem Crawler ist, dass ein Crawler lediglich Internetseiten herunterlädt, ein Scraper jedoch Informationen aus heruntergeladenen Seiten extrahiert. Dies ist nötig, wenn man Daten nutzen möchte, die nicht über eine öffentliche API, jedoch menschenlesbar auf Internetseiten zur Verfügung stehen. Einen Scraper, welcher Zielinformationen aus strukturierten Daten, wie einem HTML-DOM, extrahiert nennt man auch Wrapper. Man unterscheidet zwischen zwei verschiedenen Ansätzen [Liu07, 323]:

Manueller Ansatz Manuelle Analyse des Quelltexts einer Internetseite, um Muster zu erkennen. Aus diesen Mustern können Regeln für den Wrapper extrahiert werden. Dieser Ansatz skaliert schlecht und ist anfällig für Änderungen in der Quelltextstruktur.

Wrapper-Induktion Überwachtes Lernen über manuell gekennzeichnete Internetseite und weiterführend die Anwendung der gelernten Regeln auf ähnlich formatierte Internetseite.

Automatische Extrahierung Ansatz des unüberwachten Lernens, bei dem Muster automatisch erkannt werden.

Diese Arbeit nutzt den manuellen Ansatz, da lediglich Reviews einiger weniger Apps aus zwei verschiedenen Stores ausgewertet werden. Eine hohe Skalierbarkeit wird nicht benötigt.

2.6 Klassifikationsverfahren

Klassifizierungsverfahren sind Verfahren, welches unbekannte Beobachtungen eine oder mehrere vordefinierte Klassen zuordnet. Solche Verfahren werden oft mithilfe von Trainingsbeobachtungen entwickelt, denen im Voraus, oft händisch, korrekte Klassen zugeordnet werden. Klassifikationsverfahren kommen in verschiedensten Bereichen unterschiedlichen Beobachtungsformaten zum Einsatz. Abbildung 2.1 veranschaulicht einige Beispiele. Zuerst ein Beispiel aus der Medizin, bei dem eine Diagnose basierend auf vom Patienten empirisch erhobenen Daten wie Geschlecht, Blutdruck, Medikamenten, etc. gestellt wird. Es folgt optische Zeichenerkennung, hier ist das Ziel, Text in Bildeingaben zu erkennen. Das letzte ist ein bekanntes Beispiel aus der Textklassifizierung, die Spamerkennung. Texte, meist E-Mails, werden hier nach *Spam* und *nicht Spam* klassifiziert.

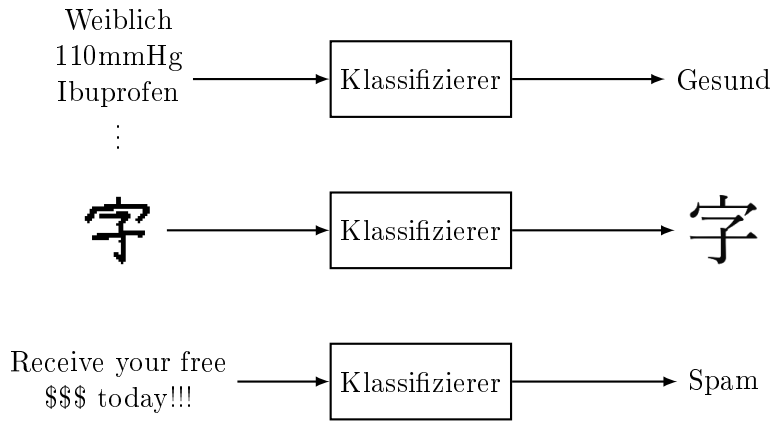

Abbildung 2.1: Verschiedene Klassifizierer (vereinfacht)

In dieser Arbeit werden Klassifizierer mit dem Ziel entwickelt, Erklärungsbedarf in App- und Play-Store-Reviews zu erkennen. Wie die Spam-Klassifizierung ist auch die Erklärungsbedarf-Klassifizierung eine binäre Klassifizierung, das heißt, es wird lediglich zwischen zwei sich gegenseitig ausschließenden Klassen unterschieden. Die Leistung eines binären Klassifizierers lässt sich anhand der Wahrheitsmatrix messen. In der Wahrheitsmatrix gibt es vier Zähler: Als *True Positive* (Richtig-Positiv) werden Vorhersagen bezeichnet, wenn sie positiv (im konkreten fall also *mit Erklärungsbedarf*)

sind und die korrekte Klasse ebenfalls positiv ist. *True Negative* (Richtig -Negativ) ist der entgegengesetzte Fall: Eine negative Vorhersage für eine negativ markierte Beobachtung. Als *Type I Error*, auch False Positive (Falsch-Positiv), bezeichnet man eine positive Vorhersage für eine als negativ markierte Beobachtung. Schließlich gibt es den *Type II Error*, auch *False Negative* (Falsch-Negativ). So bezeichnet man negative Vorhersagen für eine positiv markierte Beobachtung. Ein Klassifizierer, der perfekte Vorhersagen trifft, hat weder Type I noch Type II Error.

		True condition	
		Condition Positive	Condition Negative
Predicted Condition	Predicted Condition Positive	True Positive (TP)	False Positive (FP) (Type I Error)
	Predicted Condition Negative	False Negative (FN) (Type II Error)	True Negative (TN)

Tabelle 2.1: Wahrheitsmatrix

Neben diesen einfachen Metriken gibt es auch kombinierte Metriken. Die *Precision* (Präzision)

$$Precision = \frac{\sum TP}{\sum TP + \sum FP}$$

gibt an, wie hoch der Anteil der als positiv vorhergesagten Beobachtungen tatsächlich positiv ist. Der *Recall* (Sensitivität)

$$Recall = \frac{\sum TP}{\sum TP + \sum FN}$$

gibt an, welcher Anteil der tatsächlich positiven Beobachtungen vom Klassifizierer als positiv erkannt wird. Weiterhin gibt es die *Specificity* (Spezifität)

$$Specificity = \frac{\sum TN}{\sum FP + \sum TN},$$

welche angibt, welcher Anteil an tatsächlich negativen Beobachtungen tatsächlich als Negativ erkannt wird. Eine vierte genutzte Metrik ist der F_1-*Score* (F_1-Maß)

$$F_1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}},$$

welcher das harmonische Mittel zwischen Präzision und Recall bildet. Durch diese Kombination lässt sich die Güte von Klassifizierern im Bezug auf Präzision **und** Recall in einer einzigen Metrik vergleichen.

2.7 Naiver Bayes-Klassifizierer

Der naive Bayes-Klassifizierer ist ein Klassifizierer, der auf dem Satz von Bayes basiert.

Satz von Bayes

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{2.5}$$

Der Satz von Bayes (2.5) beschreibt die Wahrscheinlichkeit eines Ereignisses A basierend auf Vorwissen über dieses Ereignis B. $p(A|B)$ ist die bedingte Wahrscheinlichkeit, dass A eintrifft, vorausgesetzt B ist wahr. $p(B|A)$ ist die bedingte Wahrscheinlichkeit, dass B eintrifft, vorausgesetzt A ist wahr. $p(A)$ und $p(B)$ sind jeweils die Wahrscheinlichkeiten, dass A bzw. B auftreten.

Klassifizierer

Den Satz von Bayes auf ein höherdimensionales Dokument $S = w_1, w_2, \ldots, w_n$ anzuwenden ist oft nicht plausibel. Die Anzahl der verschiedenen Eingabemöglichkeiten wächst exponentiell mit der Anzahl der Dimensionen, sodass jedes zu klassifizierende Dokument ein Novum für den Klassifizierer ist. Um $p(K_j|S = \{w_1, w_2 \ldots, w_i\})$ dennoch berechnen zu können, lässt sich das Dokument unter der Annahme, dass alle Features w_i unabhängig voneinander sind, zerlegen:

$$p(K_j|w_i) = \frac{p(w_i|K_j)p(K_j)}{p(w_i)} \tag{2.6}$$

Unter der beschriebenen Annahme erhält man nach Umformung

$$p(K_j|S = \{w_1, w_2 \ldots, w_i\}) = \frac{p(K_j)}{p(S)} \prod_{i=1}^{n} p(w_i|K_j). \tag{2.7}$$

Ein Klassifizierer wählt in der Regel die Klasse, die die höchste Wahrscheinlichkeit erreicht:

$$\hat{K} = \underset{j \in \{1,2,\ldots,J\}}{argmax}\ p(K_j) \prod_{i=1}^{n} p(w_i|K_j) \tag{2.8}$$

Der Divisor $p(S)$ kann gekürzt werden, weil er bei einem gegebenen S für alle Klassen K_j identisch ist, da er nicht von der Klasse abhängt.

2.8 Neuronale Netzwerke

Neuronale Netzwerke sind Systeme, die die Funktion von Neuronen im Gehirn abstrahieren. Diese Netze werden genutzt, um komplexe Zusammenhänge zu modellieren. Sie haben ein Eingabe- und ein Ausgabeset, welche durch künstliche Neuronen verbunden ist und lernen durch Verarbeiten von Beispielen mit bekannten Ein- und Ausgaben. Diese Neuronen haben selbst Ein- und Ausgaben, welche mit anderen Neuronen verbunden sind. Diese

Verbindungen zwischen Neuronen besitzen Gewichte, welche beim Lernen mit bekannten Werten angepasst werden können, um die Präzision des Netzwerks zu erhöhen. Des Weiteren hat ein Neuron eine Übertragungsfunktion, welche die Eingaben aggregiert, meist über eine gewichtete Summe mit Gewichten aus den Verbindungen. Der letzte Bestandteil des Neurons ist die Aktivierungsfunktion, welche aus dem Ergebnis der Übertragungsfunktion die Ausgabe des Neurons bestimmt und sie auf einen bestimmten Bereich begrenzt. Diese Ausgabe nennt man auch Aktivierung.

Das Trainieren eines Netzes findet in sogenannten Epochen statt. In einer Epoche werden die Trainingsdaten auf das Netz angewendet, die Ausgabe notiert und die Differenz zwischen erwarteter und tatsächlicher Eingabe, der Fehler, berechnet. Dieser Fehler wird in einem Prozess namens Backpropagation genutzt, um die Gewichte des Netzes anzupassen. Gewichte werden gewichtet nach ihrem Einfluss auf den Fehler angepasst. Diese Anpassung kann nach jeder Eingabe, je nach einer gewissen Anzahl von Eingaben oder am Ende einer Epoche passieren. Zwei weitere Hyperparameter steuern das Lernverhalten: Die Lernrate steuert, wie stark die Gewichte bei der Backpropagation angepasst werden. Eine statische Lernrate birgt jedoch Probleme mit lokalen Extremstellen: Ist die Lernrate am Anfang des Lernens zu niedrig, besteht die Gefahr, nicht über ein lokales Maximum hinauszukommen, während eine hohe Lernrate spät im Lernprozess dazu führen kann, dass man ein lokales Maximum nicht genau trifft. Daher wird ein weiterer Parameter, der Verfall (Decay), eingeführt. Nach jeder Epoche wird der Verfall von der Lernrate abgezogen. So ist es möglich, gleichzeitig eine hohe Lernrate am Anfang des Prozesses zu haben, während gegen Ende die Lernrate immer kleiner wird.

In dieser Sektion werden verschiedene Netzwerktypen vorgestellt. Zunächst werden in Sektion 2.8.1 Feedforward-Netzwerke erläutert, die älteste und einfachste Art von neuronalen Netzen. Folgend werden in Sektion 2.8.2 Convolutional Neural Networks erklärt, die Vorteile im Erkennen von Räumlich abhängigen Daten haben. Zuletzt werden in Sektion 2.8.3 Rekurrente Neuronale Netzwerke vorgestellt, die auf das Erkennen von Zeitserien ausgelegt sind.

2.8.1 Feedforward-Netzwerke

Feedforward-Netzwerke (FFN) waren die ersten und in ihrer Struktur simpelsten neuronalen Netze [Sch14]. Sie bestehen neben den in allen Netzwerken präsenten Eingabe- und Ausgabeschichten aus einer einzigen versteckten Schicht (*Hidden Layer*), siehe Abbildung 2.2. Das Beispielnetzwerk in dieser Abbildung besitzt drei Eingabeneuronen x_i, vier Neuronen in der versteckten Schicht h_j und zwei Ausgabeneuronen y_k. Die Aktivierungsfunktion der Eingabeschicht ist im Allgemeinen die Identität der Eingabe. Die Aktivierungen der Eingabeneuronen werden nun an die Neuronen der versteckten

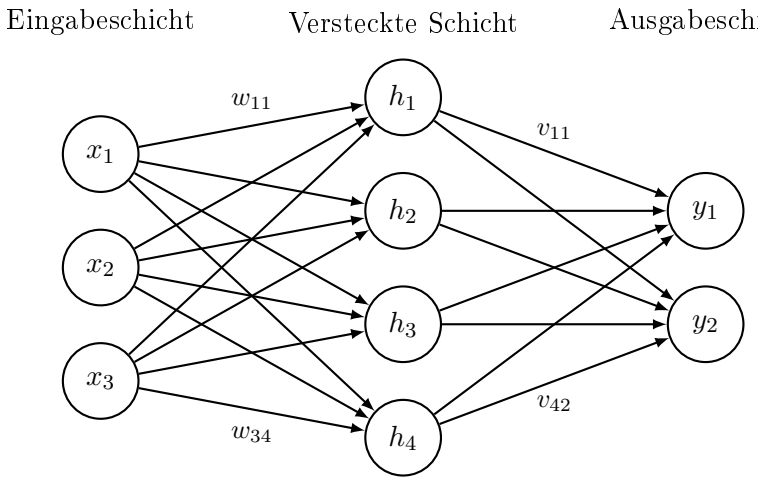

Abbildung 2.2: Beispielaufbau eines einfachen Feedforward-Netzwerks

Schicht weitergegeben. Dabei werden sie mit den zugehörigen Gewichten w_{ij} multipliziert, beispielsweise zwischen den Neuronen x_1 und h_1 mit dem Gewicht w_{11}. Die Eingaben aus x_1, x_2 und x_3, multipliziert mit den zugehörigen Gewichten, werden in jedem Neuron der versteckten Schicht durch die Übertragungsfunktion akkumuliert. Dieses Ergebnis wird von der Aktivierungsfunktion in die Aktivierung transformiert. Die Aktivierungen der versteckten Schicht werden dann, multipliziert mit den Gewichten v_{jk}, an die Neuronen der Ausgabeschicht weitergegeben. Diese Ausgabeschicht funktioniert analog zur versteckten Schicht, hat aber häufig eine andere Aktivierungsfunktion.

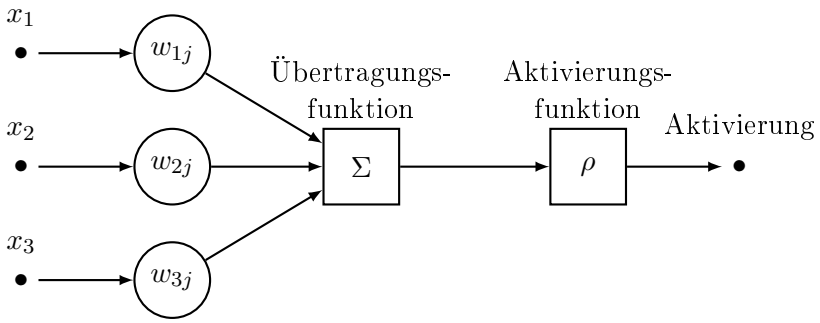

Abbildung 2.3: Aufbau eines Neurons am Beispiel eines Neurons h_j der versteckten Schicht aus Abbildung 2.2

Eine Erweiterung dieser einfachen Feedforward-Netzwerke sind *Deep-Feedforward-Netzwerke* (DFF), welche sich in der Anzahl der versteckten Schichten unterscheiden. Sie sind mehrere Schichten tief (*deep*), daher ihr Name.

2.8.2 Convolutional Neural Networks

Ein *Convolutional Neural Network* (CNN) ist eine Klasse von neuronalen Netzen, welche besser als traditionelle neuronale Netze mit der größe der Eingabe skaliert. Ebenfalls sind sie besser darauf ausgelegt, räumliche Strukturen zu erkennen. Sie bestehen wie konventionelle Netzwerke aus einem Input Layer, einem Output Layer und mehreren Hidden Layern. Die Hidden Layer kann man in zwei Abschnitte unterteilen. Der mehrdimensionale Teil besteht aus einem oder mehr Convolutional Layer, welche den Input via Skalarprodukt falten, gefolgt von einem Pooling Layer, welcher die Anzahl der Aktivierungen reduziert. Diese beiden Layer-Typen können sich wiederholen. Hierauf folgt, verbunden mit einem Flatten Layer, ein Teil des Netzwerks, welcher, wie traditionelle Netzwerke, *fully connected* ist.

Convolutional Layer

In einem bestimmten Convolutional Layer wird zunächst die Größe c der Faltungsmatrix (*Kernel*) bestimmt. In bestimmten vordefinierten Abständen (*Stride*),üblicherweise 1, seltener 2, 3 oder mehr [VL], wird nun eine für jeden Punkt k mit eigenen Gewichten definierte Faltungsmatrix $F_k \in \mathbb{R}^{c \times c}$ über die Eingangsmatrix M gelegt. Das Ergebnis einer Faltung ist wie folgt definiert:

$$x_k = f(F_k \cdot M_k + b_k) \tag{2.9}$$

b_k ist hier der Bias-Term für den Punkt k und M_k die Submatrix von M am Punkt k im Umkreis c. f ist eine Aktivierungsfunktion, üblicherweise *Rectified Linear Unit* mit $ReLU(x) = max(0, x)$.

Unabhängig von der Dimensionalität des Eingabe-Layers werden häufig höherdimensionale Convolutional Layer genutzt. Diese auch *Feature Map Channel* genannte Dimension dient dazu, um verschiedene Merkmale an derselben Stelle der Eingabe zu erkennen.

Max-Pooling-Layer

Auch beim Max-Pooling wird in vordefinierten Abständen auf der Eingangsmatrix M agiert. Das Ergebnis definiert sich allerdings für jeden Punkt k einfach über den höchsten Wert in Submatrix M_k:

$$x_k = max\, M_k \tag{2.10}$$

Die meistgenutzte Filtermaske hat eine Dimension von 2×2. Da beim Pooling üblicherweise ein Stride gewählt wird, welcher der Größe der Filtermaske gleicht, entspricht dies einer Reduktion der Aktivierungen um 75%.

Flatten Layer

Der Flatten Layer reduziert den Output des höherdimensionalen Faltungsteils auf eine Dimension. Ist beispielsweise die Output-Dimension des letzten Pooling Layers $\mathbb{R}^{10\times10\times2}$, dann ist die Dimension nach der Flatten-Operation $\mathbb{R}^{200}$.

2.8.3 Rekurrente Neuronale Netzwerke

Rekurrente Neuronale Netzwerke (RNN) bezeichnet eine Klasse von neuronalen Netzen, die sich selbst Eingaben zuführen. „Entrollt“ man ein solches Netzwerk, kann man es sich als wiederholende Kopie eines Netzwerks, auch Zelle genannt, vorstellen, welches als zusätzliche Eingabe die Ausgabe der Vorgängerzelle enthält, siehe Abbildungen 2.4 und 2.5.

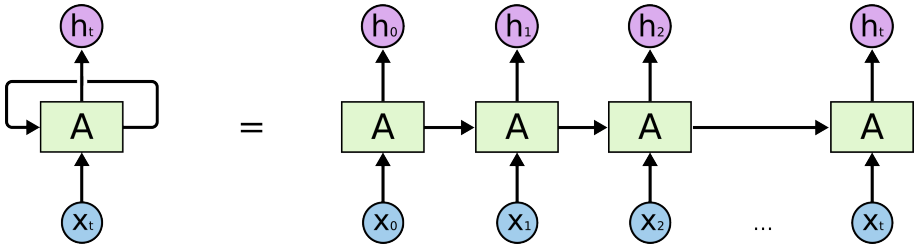

Abbildung 2.4: Entrolltes RNN [Ola15]

Eine solche Netzwerkarchitektur kann genutzt werden, um aufeinanderfolgende, voneinander abhängige Eingaben wie natürlichsprachliche Sätze, Aktienpreise oder Videobilder zu analysieren oder synthetisieren. Frühe RNN-Architekturen haben jedoch Probleme, Kontext mit großer zeitliche Diskrepanz richtig zu deuten [BSF94]. Dieses „Problem der Langzeitabhängigkeit“ soll das folgende Beispiel illustrieren: Möchte man das letzte Wort des Satzes „Japan ist mein Heimatland. Ich spreche flüssig *Japanisch*.“ voraussagen zeigt der direkte Kontext zwar, dass es sich vermutlich um eine Sprache handelt, jedoch ist die Information „Japan“ eine lange Distanz entfernt [Ola15].

Long Short-Term Memory

Long short-term memory (LSTM) ist eine RNN-Architektur, die zuerst 1997 von Hochreiter und Schmidhuber in ihrer gleichnamigen Publikation vorgestellt wurde [HS97]. LSTM ist speziell entworfen, um das Problem der Langzeitabhängigkeit zu umgehen. Eine Zelle gibt nun zusätzlich zum Ausgabewert einen Zellzustand weiter, der von der Zelle verändert werden kann. Zellen besitzen Strukturen, auch *Gates* genannt, bestehend aus einem NN-Layer (σ) und einem Punktweisen Produkt ($\times$). Diese Gates geben keine Binärwerte zurück, sondern operieren auf einer Skala von 0 bis 1.

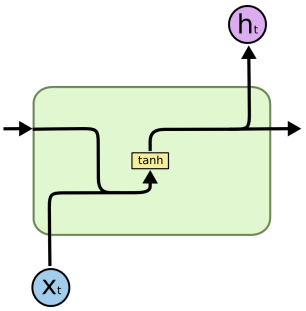

Abbildung 2.5: Detail-Blick auf eine RNN-Zelle [Ola15]

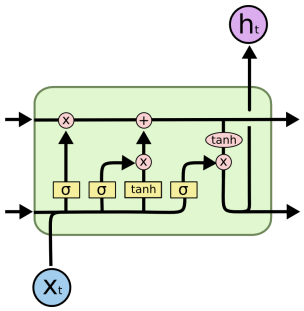

Abbildung 2.6: Detail-Blick auf eine LSTM-Zelle [Ola15]

Das erste dieser Gates ist das Forget Gate, welches bestimmt, welche Werte aus dem Zellstatus gelöscht werden. Das zweite Gate ist das Input Gate, welches bestimmt, welche der im Netzwerk (*tanh*) berechneten Werte an den Zellstatus weitergereicht werden. Das Output Gate ist das finale Gate, es legt fest, welche Werte des Zellstatus nach einer *tanh*-Operation die Ausgabe für die Zelle sind.

LSTM wird seit Mitte der 2010er Jahre vermehrt mit Erfolg verwendet. So haben bei Facebook 2017 LSTM-basierte automatischen Übersetzungen die vorher üblichen phrasenbasierten Modelle vollständig abgelöst [Sid17]. Auch Amazon nutzt LSTM-Technologie bei Amazon Polly, einem Sprachsynthese-Service, der auch die Sprachsynthese von Amazons digitalem Assistenten Alexa antreibt [Vog17]. Polly nutzt LSTM-basierte Netzwerke mit einer „kolossalen Menge“ an Daten, um Modelle zu trainieren, die Zeichen zu Audiospuren umwandeln und Intonationskonturen vorhersagen.

Kapitel 3

Konzepte

Dieses Kapitel erläutert die in dieser Arbeit verwendeten Konzepte. Abschnitte 3.1 und 3.2 geben einen Überblick über die Datenquellen und wie Rohdaten erhoben werden können. Abschnitt 3.3 erläutert den Prozess der Erzeugung einer Ground Truth. Folgend erklärt Abschnitt 3.4, wie Textdaten in für ein neuronales Netzwerk verständliches Format umgewandelt werden. Abschließend zeigt Abschnitt 3.5, wie die Entwickelten Klassifizierer funktionieren und welche Parameter ausgesucht wurden.

3.1 Play Store

Google bietet zwar seinen Entwicklern eine Schnittstelle, um auf App-Informationen zuzugreifen, jedoch ist diese nicht öffentlich und beschränkt sich auf die Apps eines jeweiligen Entwicklers. Eine echte öffentliche Schnittstelle existiert nicht.

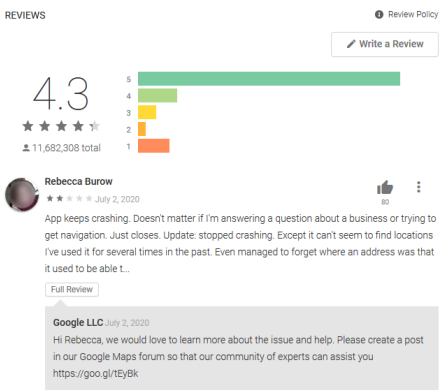

Abbildung 3.1: Aggregierte Bewertungen und ein Review mit Entwicklerantwort im Play Store [Kuhd]

3.1.1 Apps

Es existieren verschiedene Möglichkeiten, den Play Store zu durchsuchen. Es lässt sich einfach nach einer bestimmten Query suchen, oder man kann nach bestimmten App-Kategorien filtern. Filtert man nach Kategorien, gibt die Toplisten „Recommended for you“, „Top Apps“, „Top Grossing“, „Trending“ und

„Top Selling“. Der Unterschied zwischen „Top Grossing“ und „Top Selling“ scheint zu sein, dass „Top Grossing“ Apps mit In-App-Käufen, einer Art Mikrotransaktion in Apps, beinhaltet, während „Top Selling“ lediglich den initialen Verkaufspreis in Betracht nimmt.

Die Detailseiten einer App unter `https://play.google.com/store/apps/details?id=<APP_ID>` bieten eine Vielzahl von Informationen über die jeweilige App. Diese menschenlesbaren Seiten zeigen Informationen wie App-Name, ID, Anbieter, Anbieterwebsite, Beschreibung, Kategorie, Icon, Altersfreigabe, Durchschnittsbewertung in $[1, 5]$, Preis mit Währung, einige Reviews und mehr. Diese können direkt über HTTP heruntergeladen werden und aus dem HTML-DOM extrahiert werden.

3.1.2 Reviews

Auf einer App-Detailseite gibt es die Möglichkeit, mehr als die wenigen gezeigte Reviews anzuzeigen. Klickt man auf den *Read All Reviews*-Knopf, bzw. fügt man den `showAllReviews=true` zu der Query-Komponente der URL hinzu, öffnet sich eine Seite, auf der durch herunterscrollen kontinuierlich neue Reviews geladen werden. Dies passiert über eine via Javascript ausgelöste HTTP-POST-Anfrage, die jedoch hochgradig obfuskiert ist. So ist nicht klar, wie man die Anfrage auf Reviews für andere Apps generalisieren kann.

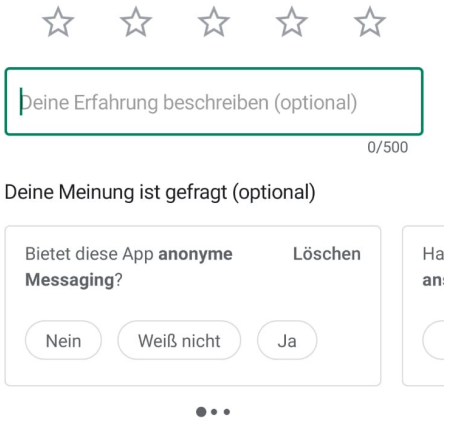

Abbildung 3.2: Review-Interface auf Android-Endgerät [Sch]

Das Extrahieren via Web Scraping ist möglich. So lassen sich Autorname, Autoravatar, Erstelldatum im `dd. MMMM yyyy`-Format, Text, Bewertung zwischen 1 und 5, Anzahl von „hilfreich“-Bewertungen, und eine eventuelle Entwicklerantwort aus dem HTML-DOM einer Review-Seite extrahieren.

Bemerkenswert sind Unterschiede im Review-Prozess zwischen Desktop und mobilen Endgeräten. Während es auf dem Desktop eine Eingabemaske mit Wortzähler gibt, die bis zu 4096 Zeichen akzeptiert, ist die Maximallänge für ein Review auf mobilen Endgeräten 500 Zeichen. Zusätzlich gibt es dort Entscheidungsfragen, siehe Abbildung 3.2.

3.2 App Store

Apple bietet auf `http://rss.itunes.apple.com/` einige Schnittstellen für Metadaten über Apps, Musik und andere von Apple angebotene Medien an. Eine Schnittstelle für App-Informationen ist vorhanden, beschränkt sich aber auf einige wenige, teils kuratierte Listen wie „New Apps We Love“, „Top Free“ oder „Top Grossing“. Eine öffentliche Schnittstelle für App-Reviews wird hier nicht angeboten.

3.2.1 Apps

Die Detailseiten von App-Store-Apps unter `https://apps.apple.com/us/app/id<APP-ID>` sind analog zu Sektion 3.1.1 via HTTP herunterladbar. Aus ihrem HTML-DOM lassen sich App-Name, ID, Beschreibung, Kurzbeschreibung, Anbieter, Anbieter-ID, Durchschnittsbewertung in $[1, 1.1, 1.2, \ldots, 5]$ und Anzahl der Bewertungen extrahieren.

Auf `https://apps.apple.com/us/genre/ios/id36` sind Links App-Kategorien wie „Business“, „Education“ oder „Lifestyle“ aufgelistet, die einen A-Z-Index von Apps der jeweiligen Kategorie sowie einen „Popular Apps“-Liste enthalten. Dieser Teil von Apples Web-Präsenz scheint nicht aktiv erhalten zu werden und befindet sich im Verfall. Darauf lassen in erster Linie die „Popular Apps“-Listen schließen. Diese zeigen Stand Juli 2020 für viele Kategorien lediglich eine einzige App oder eine vom App Store auf einem iPhone divergierende Liste an, siehe Abbildung 3.3. Hinzu kommt das archaische Design der Seite sowie fehlende Backlinks von Apples Onlinepräsenz.

3.2.2 Reviews

Eine private Schnittstelle für Reviews lässt sich auf `https://itunes.apple.com/us/rss/customerreviews/id=<ID>/sortBy=mostRecent/xml` finden. Diese Schnittstelle ist limitiert, da unabhängig von der zur Verfügung stehenden Anzahl von Reviews maximal 10 Seiten des Feeds existieren [Rib]. Ein weiteres Problem ist die Unzuverlässigkeit der Schnittstelle: Stackoverflow-Nutzer Brent Traut konnte beobachten, dass bestimmte Feeds manchmal oder immer unvollständig oder komplett leer erschienen, obwohl in Apples Weboberfläche eine hohe Anzahl Reviews beobachtet werden konnte. Traut folgerte, dass sich diese Schnittstelle aufgrund dieses erratischen Verhaltens nicht für reale Nutzung eignet [Tra]. Dies konnte durch eigene Experimente belegt werden und die Schnittstelle wurde daher verworfen.

Ein weiterer Ansatz war, die privaten APIs zu finden, die Apple nutzt, um Reviews auf ihren Endgeräten anzuzeigen. Dies erwies sich als unmöglich, da sich normaler HTTPS-Traffic zwar über einen Debugging-

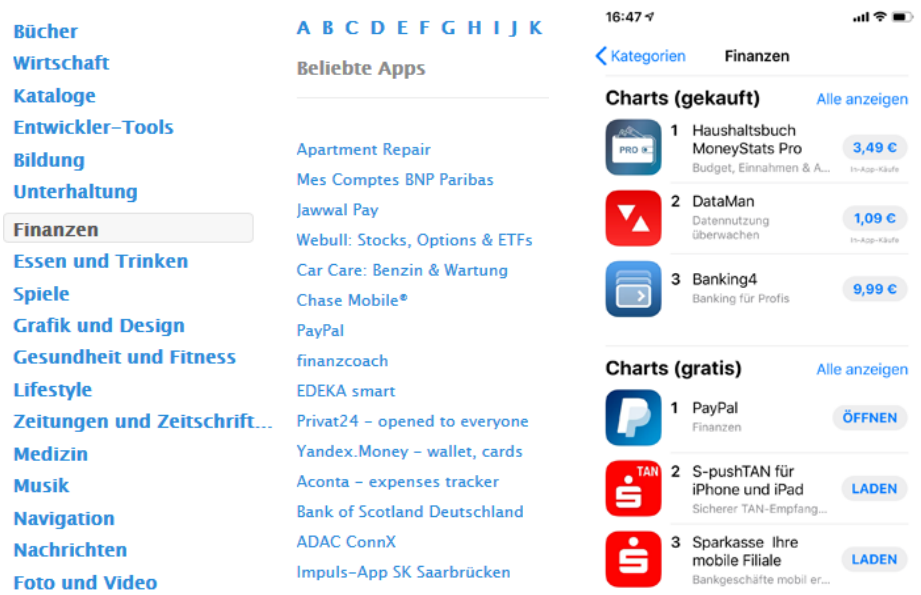

Abbildung 3.3: Diskrepanz zwischen Webliste (links) [Kuhb] und App-Listen (rechts) [Kuha] von Top Finanzapps im App Store am 06. Juli 2020

Proxy-Server wie Fiddler auslesen lässt, der Zugriff auf Apples eigene Dienste wie den App Store jedoch nicht möglich ist. Abhilfe schuf Apple

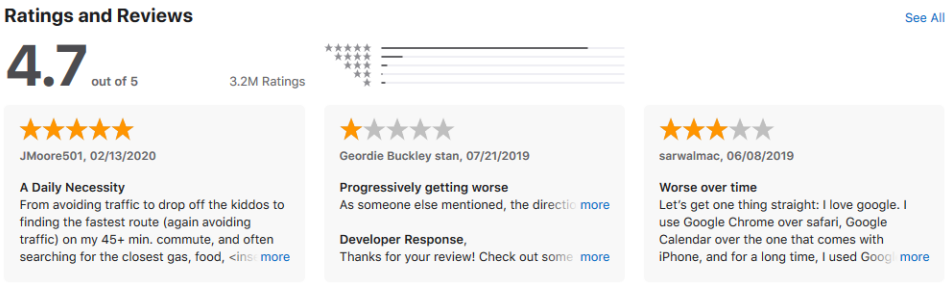

Abbildung 3.4: Review-Übersicht im Web-App-Store mit *See All* button oben rechts [Kuhc]

zufällig, indem während der Schnittstellensuche ein *See All*-Knopf auf die App-Übersichtsseiten eingebaut wurde. Über diesen Button lässt sich analog zum *Read All Reviews*-Knopf im Play Store eine Seite öffnen, durch kontinuierliches Scrollen eine große Anzahl an Reviews geladen werden kann. Diese Reviews werden über HTTP-GET-Anfragen an eine private JSON-Schnittstelle erhalten. Jedoch erfordert diese das übermittelt eines Bearer Tokens, welches vermutlich über einen Cookie vergeben wird.

Da die Haltbarkeit, Fragilität sowie das Verfahren des Erhalts eines

Bearer Tokens unbekannt ist, wurde stattdessen die simple Methode des Web Scrapings gewählt. Aus dem HTML-DOM des menschenlesbaren Formats lassen sich Autorname, Erstelldatum als *ISO 8601*, Bewertung in $\mathbb{Z} \cap [1, 5]$, Titel, Text und eine eventuelle Entwicklerantwort extrahieren.

3.3 Ground Truth

Um Klassifizierer zu entwickeln, wurde zunächst eine Ground Truth ermittelt. Um zu observieren, ob Daten aus verschiedenen Stores verschiedene Ergebnisse liefern, wurden Daten aus Apples App Store und Googles Play Store erhoben. Um diverse Reviews zu erheben, wurden Apps aus verschiedenen Kategorien gewählt, die in beiden Stores existieren. Es wurden Kategorien gewählt die einen Echtweltbezug haben. Ein solcher Bezug lässt echte Konsequenzen eines erklärungsbedürftigen Prozesses zu. Die Vermutung war, dass so mehr Reviews mit Erklärungsbedarf gefunden werden können. Die gewählten Kategorien waren „Finance“, „Medical“ und „Travel & Local“ für den Play Store bzw „Travel“ für den App Store. Für jede Kategorie wurde je Store eine App ausgewählt, die auch in ähnlicher Form im jeweils anderen Store vorkommt, siehe 3.1. So lassen sich gleiche Apps aus verschiedenen Stores vergleichen. Für den Play Store wurde aus der in Sektion 3.1.1 erwähnten „Top Apps“-Liste je eine App gewählt, für den App Store aus der in Sektion 3.2.1 beschriebenen „Popular Apps“-Liste.

	Finance	Travel	Medical
App Store	Venmo	United Airlines	MyChart
Play Store	Turbo Tax	Google Maps	Medscape

Tabelle 3.1: Aus den jeweiligen Toplisten der Stores gewählte Apps

Venmo App, die es Nutzern ermöglicht, Sofortzahlungen zwischen Nutzern durchzuführen, bietet teils ähnlichen Service wie PayPal

Turbo Tax App, um Elektronisch eine US-Amerikanische Steuererklärung auszufüllen

United Airlines App für Flugbuchung, Check-In und ähnliche Services der Fluglinie United Airlines

Google Maps Navigations- und Karten-App

MyChart App, die es Nutzern ermöglicht, Gesundheitsinformationen direkt von ihren Gesundheitsversorgern zu erhalten, Termine mit Gesundheitsversorgern abzuschließen und mit Gesundheitsversorgern zu kommunizieren

Medscape App für Gesundheitspersonal für Neuigkeiten, Medikamenten- und Krankheitsinformationen und berufliche Aus- und Weiterbildung

Für jede dieser Apps wurden aus beiden Stores mehrere Tausend Reviews, jeweils sortiert nach der Standardsortierung, heruntergeladen. Da eine fünfstellige Anzahl von Reviews eine zu große Datenmenge für eine manuelle Analyse ist, wurde ein Limit von 100 Reviews je App gesetzt. Um eine ausgewogene Verteilung von Reviews mit unterschiedlichen Bewertungen zu gewährleisten, wurden für die Bewertungen von eins bis fünf Sternen für jede App je die ersten 20 Reviews ausgewählt.

3.3.1 Review-Länge

Abbildungen 3.7 und 3.8 zeigen respektive die Anzahl der Sätze und die Anzahl der Zeichen in Reviews für verschiedene Apps je Store und ihre jeweiligen Durchschnittswerte, sowie ein Durchschnitt der gesamten Population. Auffällig sind augenscheinlich die höhere Länge von Reviews aus dem App Store im Vergleich zum Play Store sowie die hohe Länge von schlechten Bewertungen.

Abbildung 3.5: Icon für den Google Play Store [LLC]

Abbildung 3.6: Icon für den Apple App Store [Inc]

„Schlechte Bewertungen sind länger“

Als „schlecht“ seien fortan Reviews mit Ein- oder Zwei-Sterne-Bewertungen gemeint. Dies führt die Bezeichnung von Vasa et al. fort, die 2012 in ihrer Veröffentlichung *A preliminary analysis of mobile app user reviews* dieselben Beobachtungen machten [VHMN12, 243].

Bemerkenswert ist die Verteilung von Reviewlängen im Play Store. Sie bilden keine Normalverteilung, stattdessen es gibt eine Häufung in der Kategorie 400–500 Zeichen. Aus den 600 betrachteten Reviews haben lediglich vier eine Länge von 500 oder mehr Zeichen. Der Grund für diese Häufung ist vermutlich, dass übermäßig viele Reviews über mobile Endgeräte abgegeben werden, welche die in Sektion 3.1.2 erwähnte 500-Zeichen-Begrenzung für Reviews besitzen. App Store Reviewlängen sind keiner Limitierung ausgesetzt, sie ähneln eher einer Normalverteilung.

Um die Nullhypothese verwerfen zu können, wird der Mann-Whitney-U-Test genutzt. Ein Z-Test ist nicht möglich, da mindestens bei den Reviewlängen im Play Store keine Normalverteilung gegeben ist.

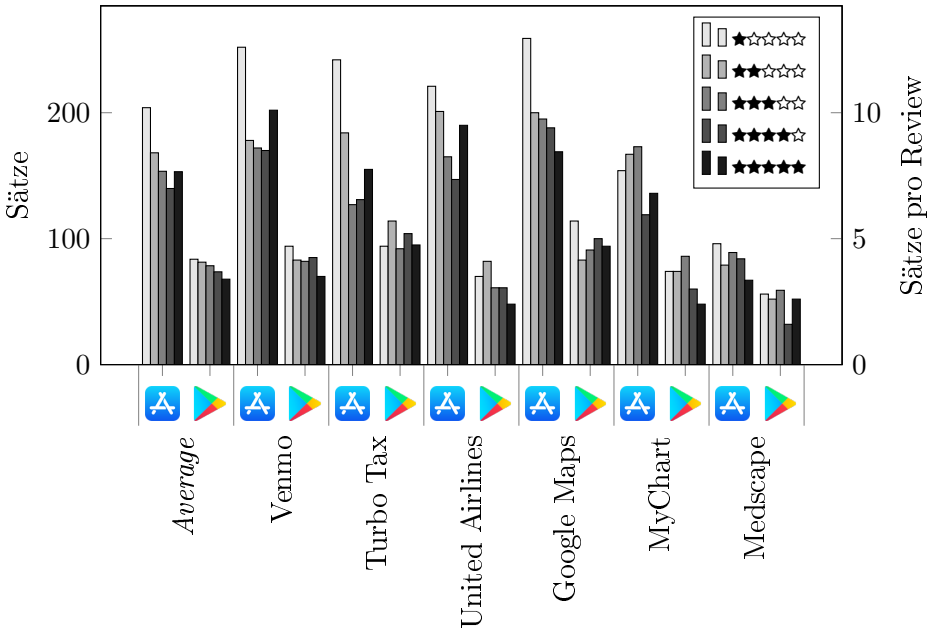

Abbildung 3.7: Anzahl Sätze in Reviews

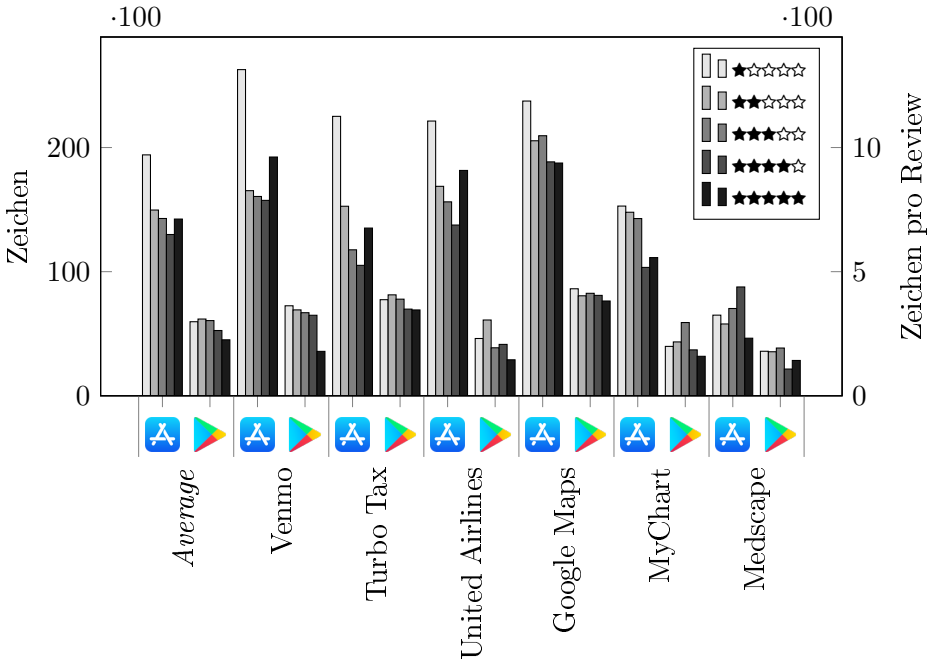

Abbildung 3.8: Anzahl Zeichen in Reviews

Seien

$$\mu_1 = \text{Median von Reviewlängen mit Bewertung} \geq 3$$
$$\mu_2 = \text{Median von Reviewlängen mit Bewertung} < 3$$
$$H_0 : \mu_1 = \mu_2$$
$$H_1 : \mu_1 < \mu_2.$$

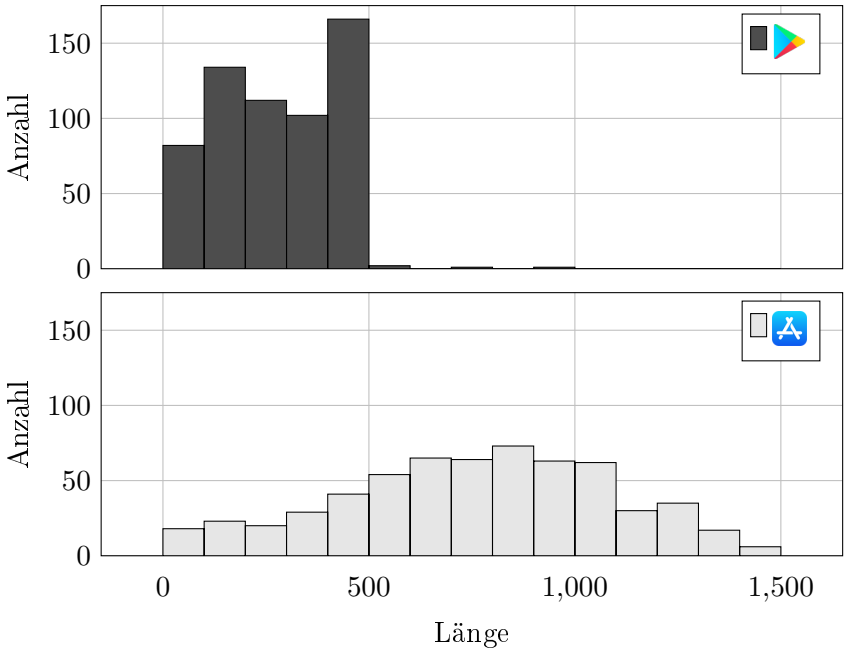

Abbildung 3.9: Verteilung Zeichenlängen in Reviews

H_0 lässt sich für den App Store mit $p \approx 1.27 \times 10^{-10}$ und für den Play store mit $p = 2.75 \times 10^{-4}$ verwerfen. Dies deckt sich mit den Ergebnissen von Vasa et al, die ebenfalls längere schlechte Reviews beobachten konnten [VHMN12, 243].

Zwei-Sterne-Reviews der Ground Truth aus dem Play Store sind im Schnitt länger als andere Reviews, im App Store sind Ein-Sterne-Reviews im Schnitt am längsten. Die Beobachtungen aus dem App Store decken sich nicht mit den Beobachtungen von Vasa et al, die in ihrer Probe aus reinen App Store Reviews Zwei-Sterne-Reviews als längste Reviews identifizieren konnten [VHMN12, 243].

3.3.2 Erklärungsbedarf

Erklärungsbedarf kann sich in verschiedenen Arten und Weisen explizit zeigen. Zum einen kann es ein Mangel von Erklärungen über die Entscheidungen eines Systems sein, wenn beispielsweise beworbene Features vom System versteckt wurden:

„**I can't tell if** supposed features of the app are missing, disabled or what (or unsupported by my provider?)“

Review von MyChart im App Store

Auch ein Navigationssystem, welches ohne Erklärung die Route ändert, kann zu diesem Erklärungsbedarf führen:

„**For some reason** [Google] maps **decided on its own** that I needed to go a different way."

Review von Google Maps im App Store

Dieser Erklärungsbedarf bei Routenänderung wurde 2019 von Chazette et al. in ihrer Veröffentlichung *Do End-Users Want Explanations? Analyzing the Role of Explainability as an Emerging Aspect of Non-Functional Requirements* untersucht [CKS19].

Eine ähnliche Art Erklärungsbedarf gibt es bei einem Mangel an Erklärungen, wie ein Nutzer Systementscheidungen beeinflussen kann. Beispielsweise, wenn ein Nutzer ohne Vorbringung von Gründen von seinem Account ausgeschlossen wurde:

„**No real explanation as to how** I could fix the issue going forward or things to avoid to ensure I don't have this issue going forward."

Review von Venmo im App Store

Eine weitere Art von Erklärungsbedarf ist fehlende, falsche oder verspätete Information. Beispielsweise, wenn ein Flug storniert wird und in einer App diese Stornierung nicht ersichtlich ist:

„Yesterday, my flight was canceled and **the App did not update to show this** for almost 2 hours after the cancellation."

Review von United im App Store

Oder aber, wenn in einer Arznei-Auskunftsapp Informationen über weitverbreitete Medikamente fehlen:

„**I can't find references** for so many common medications on medscape."

Review von MedScape im Play Store

Neben diesen Arten von *explizitem* Erklärungsbedarf gibt es noch *impliziten* Erklärungsbedarf, der lateral dazu existiert. Beispielsweise ein implizierter Erklärungsbedarf über die hohe Verzögerung für einen Geldtransfer:

„2-3 business Days just to transfer your money from your Venmo account to your bank or debit card?"

Review von Venmo im App Store

Auch das routenändernde Navigationssystem kann impliziten Erklärungsbedarf verursachen:

„Every time we would stop for gas, it would then re-route how it wanted us to go."

Review von Google Maps im Play Store

Manuelle Kennzeichnung von Erklärungsbedarf

In einem ersten Durchgang wurden die in Sätze gespaltenen Reviews auf alle Arten von Erklärungsbedarf untersucht und gekennzeichnet. Dabei wurde nicht zwischen implizitem und explizitem Erklärungsbedarf unterschieden. Aus den insgesamt 7222 Sätzen konnten 406 Sätze mit Erklärungsbedarf ermittelt werden, das sind etwa $5,64\%$. Von den 2310 Sätzen im Play Store enthielten 100 Erklärungsbedarf, also etwa 4.33%. Aus dem App Store konnten von den 4912 Sätzen 307 mit Erklärungsbedarf identifiziert werden, was genau 6.25% entspricht.

Die Verteilung von Erklärungsbedarf über App, Bewertung und Store ist Abbildung 3.10 zu entnehmen. Dies lässt die Hypothese zu, dass schlechte Reviews mehr Erklärungsbedarf enthalten. Seien

$$\begin{aligned}
\mu_1 &= \text{Median von Sätzen mit Erklärungsbedarf pro Review} \\
&\quad \text{mit Bewertung} \geq 3 \\
\mu_2 &= \text{Median von Sätzen mit Erklärungsbedarf pro Review} \\
&\quad \text{mit Bewertung} < 3 \\
H_0 &: \mu_1 = \mu_2 \\
H_1 &: \mu_1 < \mu_2.
\end{aligned}$$

Die Nullhypothesen H_0 lassen sich mittels Mann-Whitney-U-Test für den Play Store mit $p \approx 0.022$ und für den App Store mit $p \approx 2.92 \times 10^{-10}$ verwefen.

Manuelle Kennzeichnung von explizitem Erklärungsbedarf

In einem zweiten Durchgang wurden Sätze, die bereits als Erklärungsbedarf enthaltend markiert wurden, erneut auf expliziten Erklärungsbedarf untersucht. Die Tokens dieser Sätze, die den expliziten Erklärungsbedarf darstellen, wurden markiert. Expliziter Erklärungsbedarf manifestiert sich häufig in Verbindung mit einem oder mehr der „Five Ws and How“ (5W1H): *Who*, *What*, *When*, *Where*, *Why* und *How*. Sie haben eine Frequenz von 7.11% für *Where* bis 17% für *When*, siehe Abbildung 3.11. Mindestens eins der 5W1H-Wörter tritt mit einer Frequenz von ungefähr 52.17% auf, was 132 der 253 Sätze mit explizitem Erklärungsbedarf entspricht.

In den folgenden Beispielen ist der explizite Erklärungsbedarf jeweils Fett gedruckt:

„Also, **why is there no way** to cancel a payment and **why was** my payment flagged for review?!?“

Review von Venmo im App Store

„Except the card gets turned down **for no apparent reason** at places like...Target.“

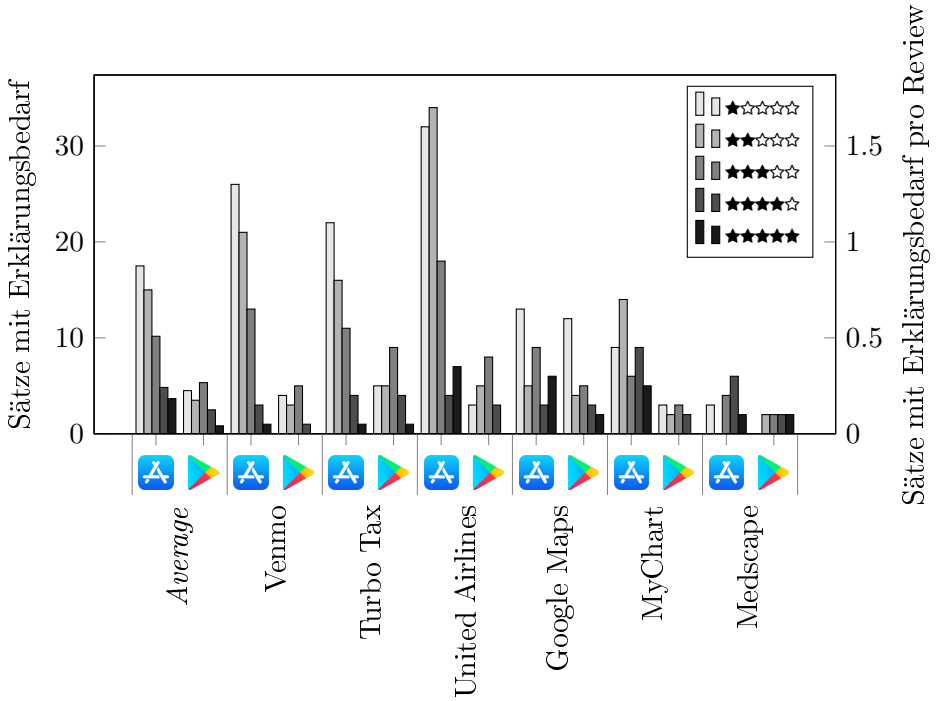

Abbildung 3.10: Häufigkeit von Erklärungsbedarf in Reviews

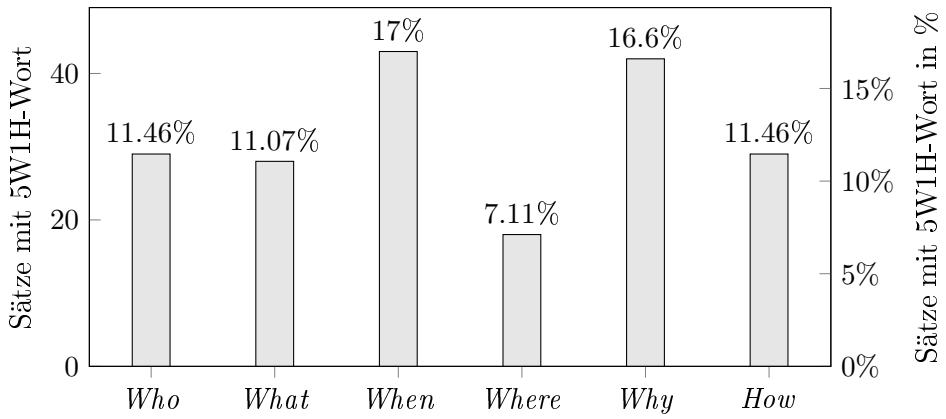

Abbildung 3.11: Frequenz von Sätzen mit mindestens einem 5W1H-Wort

Review von Venmo im Play Store

„Also **why does it assume** I didn't see the faster route already and decided I didn't want to go on that route beforehand, does it think I'm stupid or something?"

Review von Google Maps im App Store

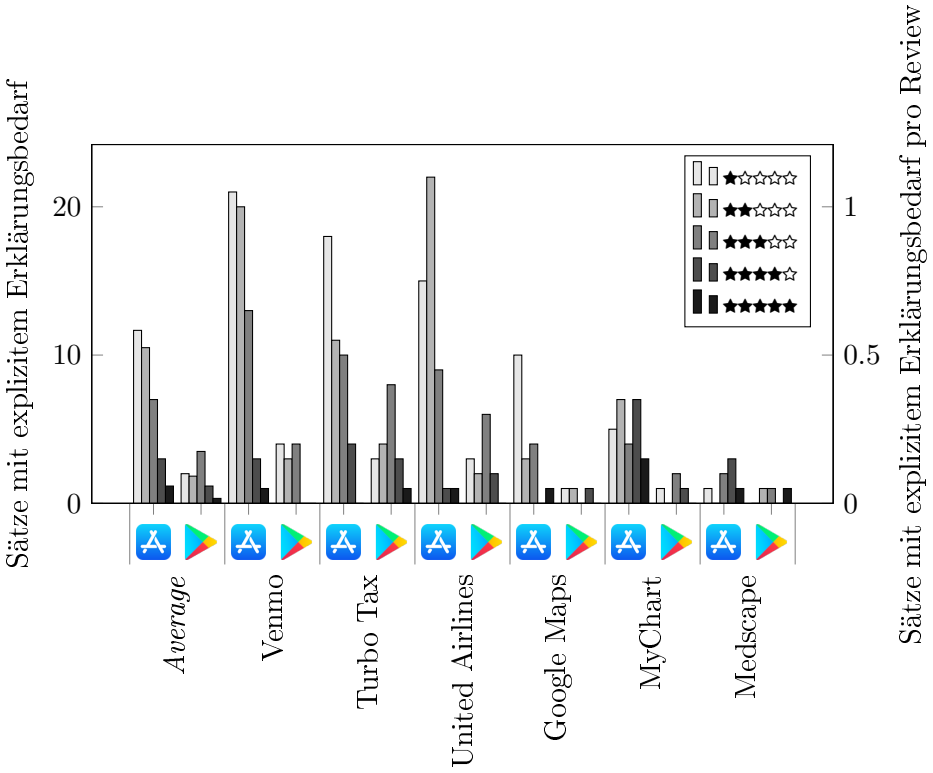

Abbildung 3.12: Häufigkeit von explizitem Erklärungsbedarf in Reviews

3.4 Datenvorverarbeitung

3.4.1 Sätze zu Matrizen

Um Sätze in ein für neuronale Netzwerke verständliches Format zu transformieren, kommt Word2Vec zum Einsatz. Ein Word2Vec-Modell wurde zuvor mit dem kompletten Korpus an gesammelten Reviews trainiert, die Word-Embedding-Dimension wird auf 100 festgelegt. Zunächst werden die Tokens w_i der Sätze $S = \{w_1, w_2, \ldots, w_n\}$ mittels Word2Vec in 100-Dimensionale Vektoren $v_i \in \mathbb{R}^{100}$ umgewandelt. Diese Vektoren werden zu der Satzmatrix $M = (v_1^T | v_2^T | \ldots | v_n^T)^T \in \mathbb{R}^{n \times 100}$ erweitert. Sie bildet die Grundlage für die weitere Verarbeitung.

3.4.2 Paarweise Merkmalskorrelation nach Zhang et al.

Feedforward-Netzwerke und Convolutional Networks haben die Limitierung, dass Eingaben dieselben Dimensionen besitzen müssen. Es ergeben sich aus zwei Sätzen S_a und S_b mit unterschiedlicher Länge, beispielsweise 10 und 20 Tokens, nach der Transformation von Satz zu Matrix (siehe die vorherige Sektion 3.4.1) zwei unterschiedlich dimensionierte Eingabema-

trizen $M_a \in \mathbb{R}^{10\times100}$ und $M_b \in \mathbb{R}^{20\times100}$. Um dieses Problem zu umgehen, wird ein Verfahren von Zhang et al. genutzt, bei dem für die gegebenen Word-Embedding-Dimension 100 eine paarweise Merkmalskorrelationsmatrix $S \in \mathbb{R}^{100\times100}$ erstellt wird [ZFDY16, 1376]. Da die Größe dieser Matrix nunmehr nicht von der Anzahl der Tokens in einem Satz, sondern lediglich von der fixen Größe der Word-Embedding-Dimension abhängt, lässt sie sich als Eingangsmatrix für die genannten Netze verwenden.

Jedes Wort w_i aus einem Dokument $S = (w_1, w_2, \ldots, w_n)$ wird mithilfe von Word2Vec in einen m-Dimensionalen Vektor $v(w_i) \in \mathbb{R}^100$ umgewandelt. Indem die Vektoren aufeinandergestapelt werden, erhält man die Satzmatrix $M \in \mathbb{R}^{n\times100}$. Nun kann man die paarweise Merkmalskorrelationsmatrix $Q \in \mathbb{R}^{m\times m}$ berechnen, die wie folgt definiert ist:

$$Q = \begin{bmatrix} sim(M_1, M_1) & sim(M_1, M_2) & \ldots & sim(M_1, M_100) \\ sim(M_2, M_1) & sim(M_2, M_2) & \ldots & sim(M_2, M_100) \\ \vdots & \vdots & \ddots & \vdots \\ sim(M_100, M_1) & sim(M_100, M_2) & \ldots & sim(M_100, M_100) \end{bmatrix} \tag{3.1}$$

M_i bezeichnet die $i$te Spalte der Satzmatrix M. $sim(M_i, M_j)$ ist die Kosinus-Ähnlichkeit zwischen den beiden Merkmalsvektoren M_i^T und M_j^T. Sie beschreibt die Ähnlichkeit zweier Vektoren über den Kosinus des zwischen ihnen eingeschlossenen Winkels θ:

$$sim(M_i, M_j) = cos(\theta) = \frac{M_i^T \cdot M_j^T}{||M_i^T||\,||M_j^T||} \tag{3.2}$$

Ein weiterer Vorteil des genannten Verfahrens ist, dass ähnliche semantische Transitionen erfasst werden können, auch wenn die Dokumente nicht viele gleiche Wörter enthalten [ZFDY16, 1376].

3.5 Klassifizierung

Die Ergebnisse, die von den in diesem Kapitel vorgestellten Klassifizierern erzielt wurden, wurden mit den Validierungsmengen erzeugt. Es sind jeweils die besten Klassifizierer in verschiedenen Metriken markiert und benannt, die in Kapitel 5 mit der Testmenge evaluiert werden, um unabhängige Ergebnisse zu erhalten.

3.5.1 Trainings-, Validierungs- und Testmengen

Um eine gemeinsame Vergleichsbasis für Klassifizierer zu schaffen, wurde die Ground Truth in je drei Mengen gespalten: Trainings-, Validierungs- und Testmenge. Lernende Klassifizierer nutzen die Testmenge, um ihr Modell zu trainieren. Mithilfe der Validierungsmenge werden die Hyperparameter

eines Klassifizierers justiert. Man lässt den Klassifizierer mit verschiedenen Hyperparametern trainieren und kontrolliert dann seine Leistung mithilfe der Validierungsmenge. Die Testmenge dient schließlich zum Vergleichen von Klassifizierern, sie darf nicht zum Verbessern der Klassifiziererleistung genutzt werden. Die 7222 Dokumente wurden zunächst je in „Erklärungsbedarf“ und „kein Erklärungsbedarf“ geteilt, diese beiden Mengen wurden gemischt und die ersten 70% als Trainingsmenge, die folgenden 15% als Validierungsmenge und schließlich die letzten 15% als Testmenge markiert. Dann wurden die Trainings-, Validierungs- und Testmengen jeweils wieder zusammengeführt. Es ergeben sich die Aufteilungen aus den Tabellen 3.2 und 3.3.

	Anteil	Dokumente	Eb.	kein Eb.
Training	70%	5055	284	4771
Validation	15%	1084	61	1023
Test	15%	1083	61	1022
Gesamt	100%	7222	406	6816

Tabelle 3.2: Aufteilung von Trainings-, Validierungs- und Testmengen für generellen Erklärungsbedarf (Eb.)

	Anteil	Dokumente	Eb.	kein Eb.
Training	70%	5055	177	4878
Validation	15%	1084	38	1046
Test	15%	1083	38	1045
Gesamt	100%	7222	253	6969

Tabelle 3.3: Aufteilung von Trainings-, Validierungs- und Testmengen für expliziten Erklärungsbedarf (Eb.)

3.5.2 Messbasis

Die Klassen in der Ground Truth haben ein hohes Ungleichgewicht, für generellen Erklärungsbedarf liegt die Verteilung bei 406 zu 6816, für expliziten Erklärungsbedarf bei 253 zu 6969. Die Präzisions-Messbasis ist die Wahrscheinlichkeit, dass ein naiver Klassifizierer C_N ein Dokument korrekt klassifiziert. Dies ergibt sich aus dem Verhältnis von positiven Dokumenten P (Dokumenten mit Erklärungsbedarf) zu der Gesamtanzahl der Dokumente N, also $\frac{P}{N}$. Die Präzisions-Messbasis liegt bei $\frac{406}{7222} \approx 0.0562$ für generellen und bei $\frac{253}{7222} \approx 0.035$ für expliziten Erklärungsbedarf.

3.5.3 Klassifizierer: Einfache 5W1H-Heuristik

Basierend auf der Feststellung, dass Erklärungsbedarf häufig in Kombination mit einem der 5W1H-Wörter auftaucht, wurde Klassifizierungsverfahren entwickelt, das Sätze basierend auf der Frequenz von diesen Wörtern klassifiziert. Sei $5W1H = \{who, what, when, where, why, how\}$ die Menge von 5W1H-Wörtern. Ein Klassifizierer besteht aus einem Tupel von Hyperparametern $W = (U, t)$. $U \subseteq 5W1H$ ist eine Untermenge von 5W1H-Wörtern, $t \in \mathbb{N}_0$ ist einen Schwellenwert. Tokens w_i eines Satzes $S = (w_1, w_2, \ldots, w_n)$ werden in Kleinschrift umgewandelt. Um S zu klassifizieren wird gezählt, wie oft eines der Tokens in U in diesem Satz vorkommt:

$$s = \sum_{i=1}^{n} \begin{cases} 1 & w_i \in U \\ 0 & w_i \notin U \end{cases} \tag{3.3}$$

Ist $s \geq t$, so wird der Satz als Erklärungsbedarf klassifiziert.

Der beschriebene Klassifizierer wurde je für generellen und expliziten Erklärungsbedarf getestet. Für alle Mengen in $\mathcal{P}(5W1H) \setminus \emptyset$ wurden alle Werte für $t > 0$, die mindestens einmal in der Test- bzw. Validierungsmenge vorkamen, getestet.

Genereller Erklärungsbedarf

Für generellen Erklärungsbedarf erreichen mehrere Klassifizierer einen F_1-Score von über 0.2, siehe Abbildung 3.13. Der Klassifizierer $W_{g1} = (5W1H, 1)$ erreicht einen F_1-Score von 0.2107 bei einem Recall von 0.4261 und einer Präzision von 0.14. Er ist der Klassifizierer mit dem Höchsten Recall. Den höchsten F_1-Score mit 0.2137 erreicht $W_{g2} = (\{what, when, why, how\}, 1)$ bei einer Präzision von 0.1483 und einem Recall von 0.3818. $W_{g3} = (\{why\}, 1)$ erreicht lediglich einen F_1-Score von 0.1591, jedoch bei einem Recall von noch 0.1084 eine Präzision von Klassifizierer 0.2993.

Alle getesteten Klassifizierer, die einen Recall von 0.1 oder höher erreichen und somit auch alle Klassifizierer, die einen F_1-Score von mindestens 0.15 haben, haben einen Schwellenwert von 1. Einige wenige Klassifizierer mit höherem Schwellenwert haben einen besseren Präzisionswert als die genannten, jedoch mit einem Recall von unter 0.1 und entsprechend niedrigem F_1-Score.

Mehrere Klassifizierer erreichen eine Präzision von 0.5, die Anzahl von Richtig-Positiven ist jedoch so gering, dass keine gewichtige Aussage über den Genauigkeitsgrad dieser Metriken getroffen werden können.

Alle Klassifizierer, die den maximalen Recall von 1.0 erreichen, schaffen es nicht über die Messbasis hinaus und sind daher nicht besser, als ein naiver Klassifizierer. Sie sind in Abbildung 3.13 nicht abgebildet.

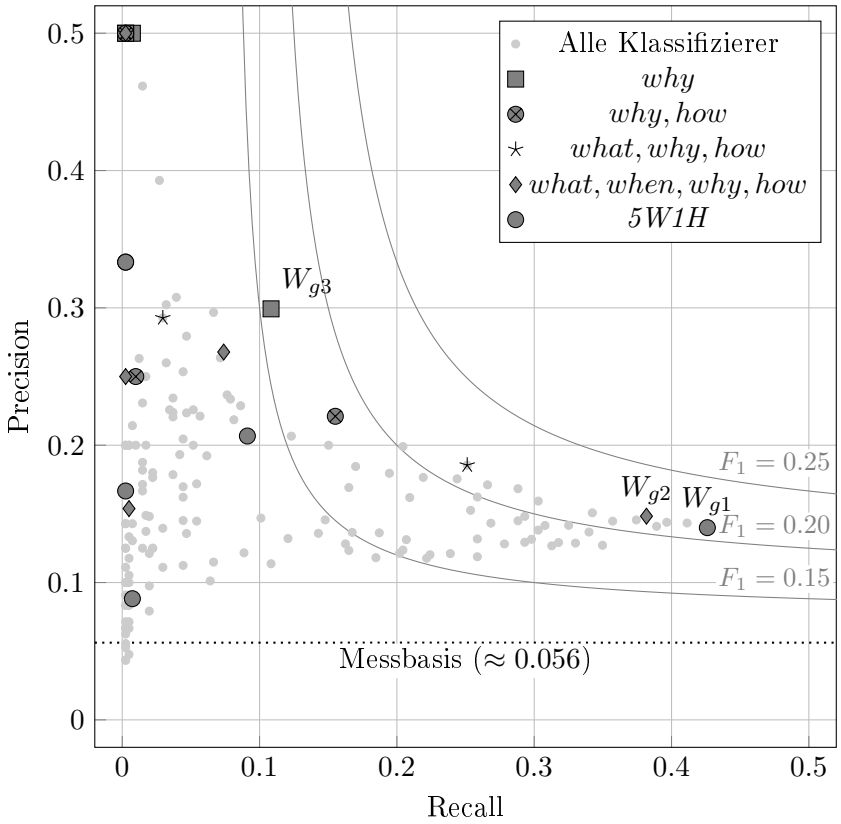

Abbildung 3.13: Precision-Recall-Kurve verschiedener 5W1H-Klassifizierer für generellen Erklärungsbedarf

Expliziter Erklärungsbedarf

Für expliziten Erklärungsbedarf erreichen vier Klassifizierer einen F_1-Score von über 0.2. Einer von ihnen ist $W_{e2} = (\{why, how\}, 1)$, er erreichte mit 0.2156 den höchsten F_1-Score bei einer Präzision von 0.2035 und einem Recall von 0.2293.

Die beste Präzision von ihnen mit 0.2857 $W_{e3} = (\{why\}, 1)$ bei einem F_1-Score von 0.21 und einem Recall von 0.166. Den höchsten Recall von 0.4901 erreicht $W_{e1} = (5W1H, 1)$ analog zu W_{g1} bei einem F_1-Score von 0.1666 und einer Präzision von 0.1003.

Auch hier erreicht kein Klassifizierer mit einem Schwellenwert von 2 oder höher einen F_1-Score von mindestens 0.15.

Ebenfalls existieren wieder Klassifizierer mit einer Präzision von 0.5, jedoch ist die Anzahl von True Positives auch hier wieder sehr gering.

Wieder sind alle Klassifizierer, die den Maximalen Recall von 1.0 erreichen, nicht besser gewesen als ein naiver Klassifizierer. Sie sind in Abbildung 3.13 nicht abgebildet.

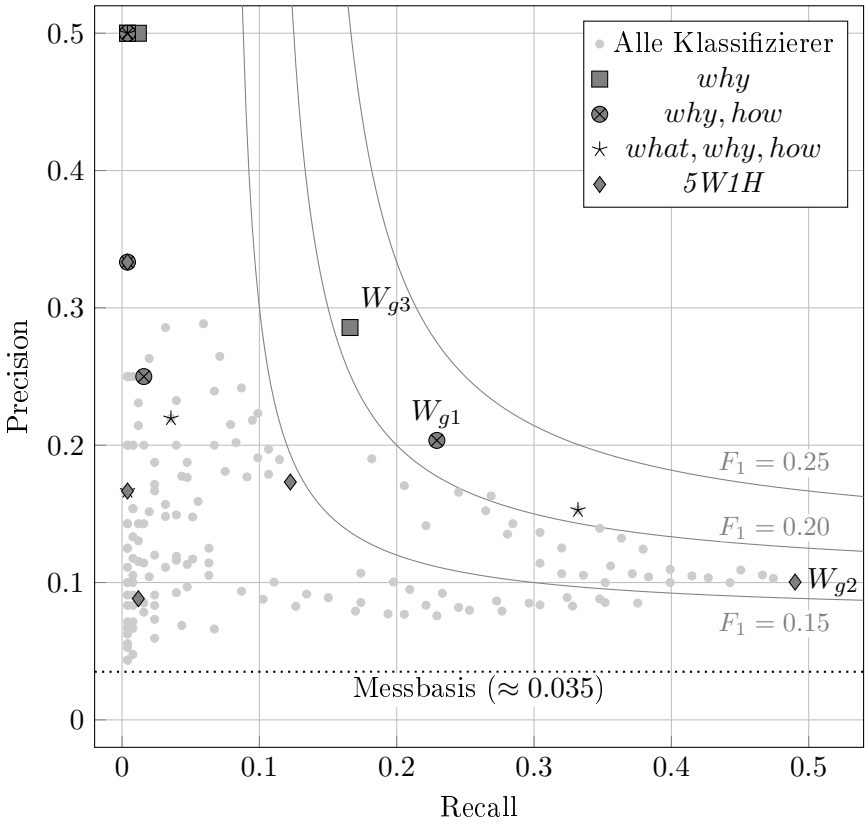

Abbildung 3.14: Precision-Recall-Kurve verschiedener 5W1H-Klassifizierer für expliziten Erklärungsbedarf

3.5.4 Klassifizierer: Naïve Bayes

Der Entwickelte naive Bayes-Klassifizierer B hat zwei Hyperparameter $B = (t, q)$. $t \in \{\text{Token}, \text{Lemma}\}$ bestimmt, ob die Originaltoken oder ihre Lemma zur klassifizierung genutzt werden. $q \in [0, 1]$ bestimmt den Parameter $p(K_{\text{Erklärungsbedarf}})$. $p(K_j)$ kann prinzipiell auf die Wahrscheinlichkeit des Vorkommens der Klasse K_j im Trainingsset gesetzt werden, jedoch ist es denkbar, dass der Klassifizierer mit anderen Werten bessere Ergebnisse liefert.

Im Folgenden wurde der Klassifizierer für $t =$ Token und $t =$ Lemma unter verschiedenen q-Werten in $[0, 1]$ getestet mit der Validierungsmenge getestet. Herausgehoben sind Klassifizierer, die in den verschiedenen Metriken vergleichsweise gute Ergebnisse erzielen konnten.

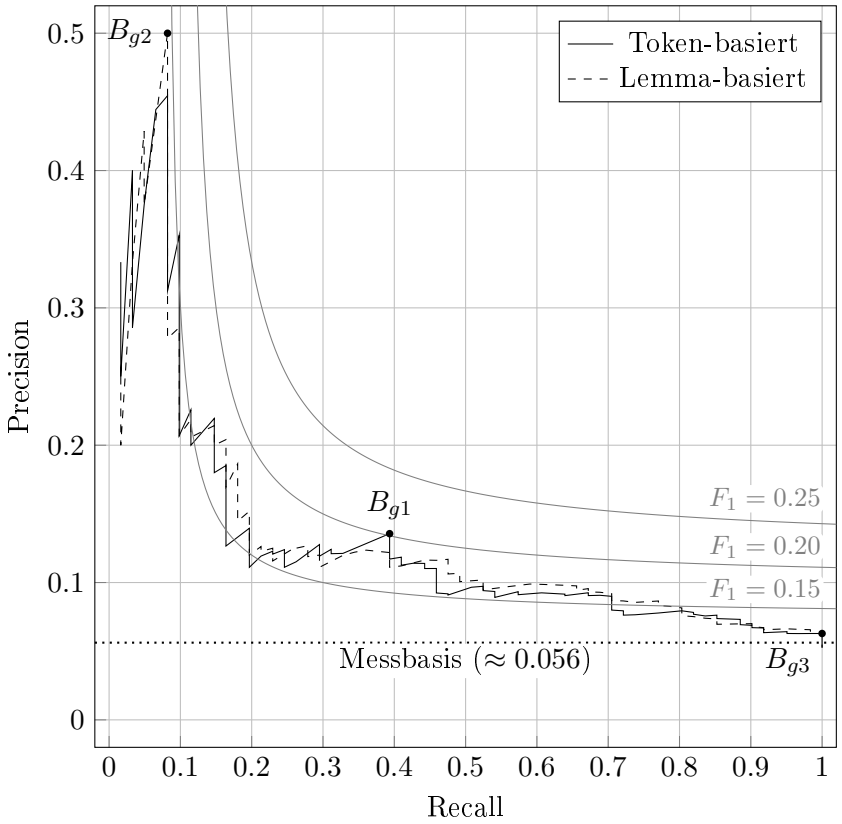

Abbildung 3.15: Precision-Recall-Kurve verschiedener Naïve-Bayes-Klassifizierer für generellen Erklärungsbedarf

General Erklärungsbedarf

Für generellen Erklärungsbedarf (siehe Abbildung 3.15) ist $B_{g1} = (Token, 1.1922 \times 10^{-7})$ mit einem F_1-Score von 0.2017 der einzige Klassifizierer, der eine F_1-Score von über 0.2 erzielt. Er erreicht dabei eine Präzision von 0.136 bei einem Recall von 0.3934.

$B_{g2} = (Lemma, 3.5252 \times 10^{-16})$ zeigt mit 0.5 den höchsten Präzisionswert bei einem Recall von 0.082 und einem F_1-Score von 0.1408. Bei einem maximalen Recall von 1.0 erreicht $B_{g3} = (Lemma, 0.0788)$ eine Präzision von nur 0.0629, was 14% über der Messbasis liegt.

Expliziter Erklärungsbedarf

Für expliziten Erklärungsbedarf zeigt der naive Bayes-Klassifizierer einen etwas besseren maximalen F_1-Score von 0.2308. $B_{e1} = (Lemma, 4.4767 \times 10^{-15})$ erreicht ihn bei einer Präzision von 0.4286 und einem Recall von 0.1578.

Die höchste Präzision von $0.\overline{45}$ wird von $B_{e2} = (Token, 1.0844 \times 10^{-19})$

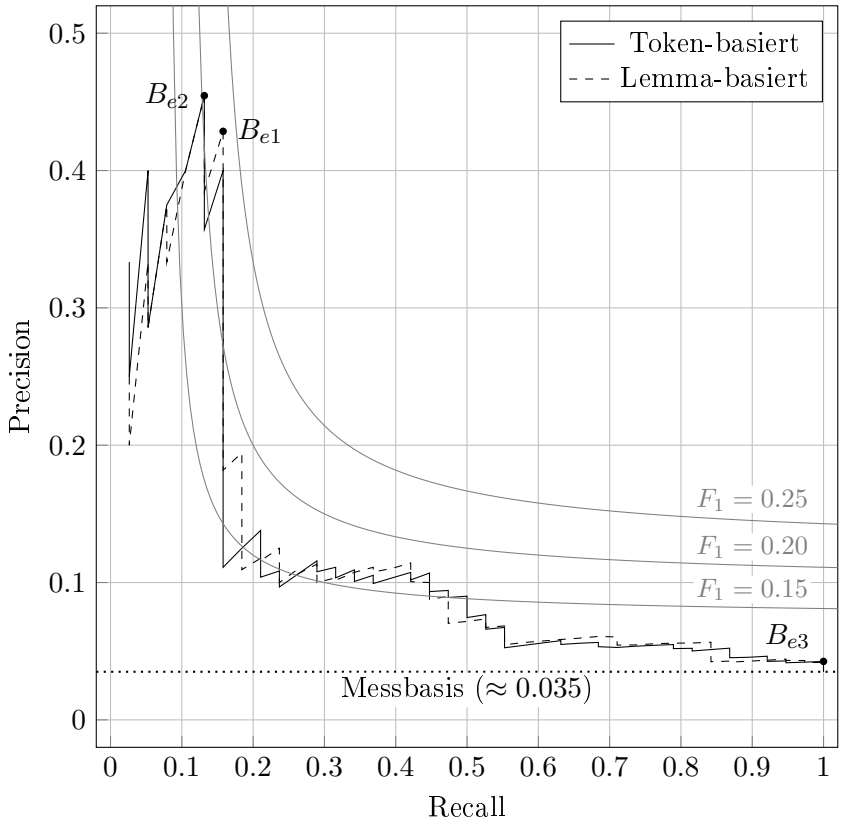

Abbildung 3.16: Precision-Recall-Kurve verschiedener Naïve-Bayes-Klassifizierer für expliziten Erklärungsbedarf

mit einem Recall von 0.1316 bei einem F_1-Score von 0.2041 erzielt. Einen maximalen Recall von 1.0 erzielt $B_{e1} = (Token, 0.0313)$ bei einem F_1-Score von 0.0816 und einer Präzision von 0.0426. Dies entspricht einer 21%-Verbesserung über der Messbasis.

3.5.5 Klassifizierer: Feedforward-Netzwerk

Da Deep-Feedforward-Networks lediglich eindimensionale Eingaben akzeptieren, wird das Eingabedokument nach einer paarweisen Merkmalskorrelation (Siehe Sektion 3.4.2) noch auf eine Dimension reduziert. Dazu werden alle Zeilen Q_j der Merkmalskorrelationsmatrix Q in einen eindimensionalen Vektor $\hat{q} = \{Q_1|Q_2|\dots|Q_{100}\} \in \mathbb{R}^{100}$ gereiht, welcher die Eingabe für das Netzwerk bildet.

Der entwickelte Deep-Feedforward-Klassifizierer ist $D = (M, t)$. M ist ein trainiertes Netzwerkmodell, welches aus der Netzwerkkonfiguration N, der Anzahl der Epochen e, der Lernrate r und dem Decay d besteht. r und d sind für die folgenden jeweils Klassifizierer 2×10^{-5} und 1×10^{-7}. Solange sie klein genug gewählt wurden, hatten sie keinen großen Einfluss auf das Ergebnis. Die Epochenanzahl e wurde mit 50 so gewählt, dass weder Unter- noch Überanpassung vorhanden ist. Für N wurden ebenfalls mehrere Möglichkeiten getestet. Vorgestellt wird die Konfiguration N_0 aus drei aufeinanderfolgenden Hidden Layern mit jeweils 10000 Neuronen und *ReLU*-Aktivierung, gefolgt von einem Output-Layer aus einem Neuron mit *Sigmoid*-Aktivierung. Sei dieses Netzwerkmodell M_0. $t \in [0, 1]$ ist ein Schwellenwert, der bestimmt, ab wann ein Ergebnis als Erklärungsbedarf klassifiziert wird.

Folgend wurde ein Klassifizierer mit M_0 unter verschiedenen Schwellenwerten t mit der Validierungsmenge getestet.

Genereller Erklärungsbedarf

Bei generellem Erklärungsbedarf gibt es zwei Klassifizierer mit einem F_1-Score von über 0.35. $D_{g1} = (M_0, 0.8721)$ erreicht einen 0.3576 bei einer Präzision von 0.3 und einem Recall von 0.4426. Dicht gefolgt liegt $D_{g1} = (M_0, 0.0.7695)$ mit 0.3524 bei einer Präzision von 0.2483 und einem Recall von 0.6066.

Die beste Präzision von 0.4231 erreicht $D_{g3} = (M_0, 0.8940)$ mit einem Recall von 0.1803 bei einem F_1-Score von 0.2529. Einen Recall von 1.0 erzielt $D_{g4} = (M_0, 0.0058)$ mit einem F_1-Score von 0.1612 und einer Präzision von 0.0876. Dies repräsentiert eine Verbesserung von 56% über der Messbasis.

Expliziter Erklärungsbedarf

Für expliziten Erklärungsbedarf werden zunächst drei Klassifizierer mit einem F_1-Score von über 0.35 vorgestellt. Den besten F_1-Score von 0.3881 erreicht $D_{e1} = (M_0, 0.9518)$ bei einer Präzision von 0.4483 und einem Recall von 0.3421, gefolgt von $D_{e2} = (M_0, 0.9261)$ mit einem F_1-Score von 0.3594, einer Präzision von $0.\overline{5}$ und einem Recall von 0.6053. Den drittbesten F_1-Score von 0.3607 erreicht $D_{e3} = (M_0, 0.9548)$ mit einer Präzision von 0.4783 und eine Recall von 0.2895.

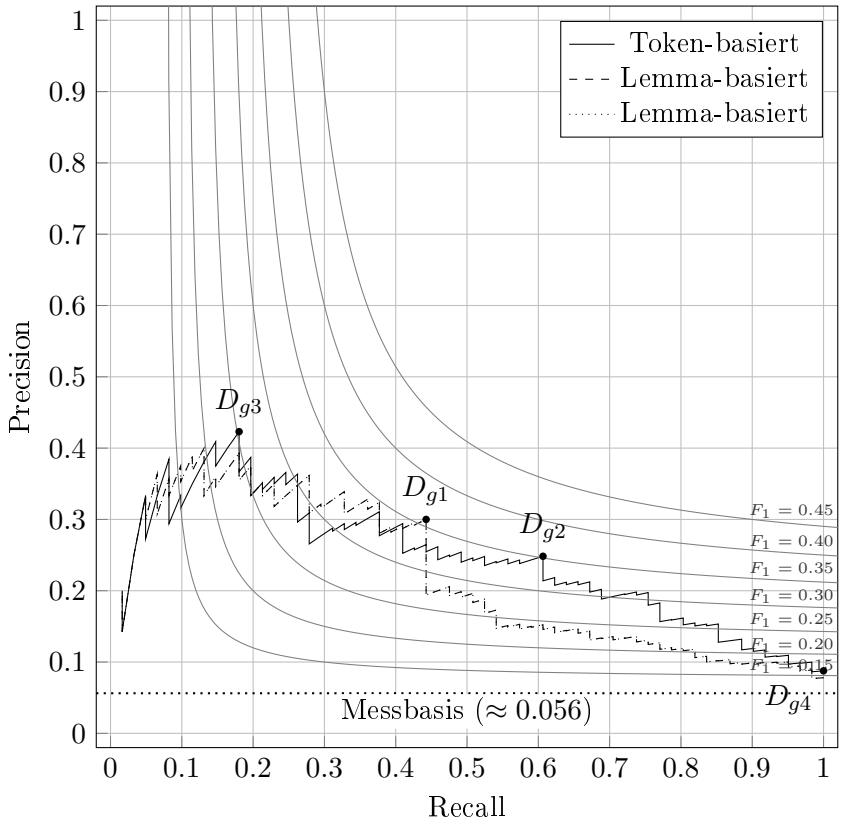

Abbildung 3.17: Precision-Recall-Kurve verschiedener FNN-Klassifizierer für generellen Erklärungsbedarf

$D_{e4} = (M_0, 0.9823)$ erreicht mit 0.4783 die beste Präzision bei einem Recall von 0.2895 und einem F_1-Score von 0.3607. Den maximalen Recall von 1.0 erreicht schließlich D_{e5} bei einem F_1-Score von 0.1066 und einer Präzision von 0.0563. Dies entspricht einer Verbesserung der Messbasis von 61%.

3.5.6 Klassifizierer: Convolutional Network

Der entwickelte Klassifizierer $C = (M, t)$ besteht analog zu Sektion 3.5.5 aus dem trainierten Netzwerkmodell M und dem Schwellenwert t. Die Epochenanzahl $e = 50$, Lernrate $r = 2 \times 10^{-5}$ und Decay $d = 1 \times 10^{-7}$ konnten aus M_0 übernommen werden. Es wurden wieder verschiedene Netzwerkkonfigurationen getestet. Viele zeigten ähnliche Ergebnisse, entschieden wurde sich daher für ein Modell, dass ähnlich bereits mit Erfolg von Zhang et al. genutzt wurde [ZFDY16, 1377].

In N_1 folgen auf den Input-Layer zwei Tripel von Convolutional Layer, Max Pooling Layer und Dropout Layer. Der erste Convolutional Layer hat

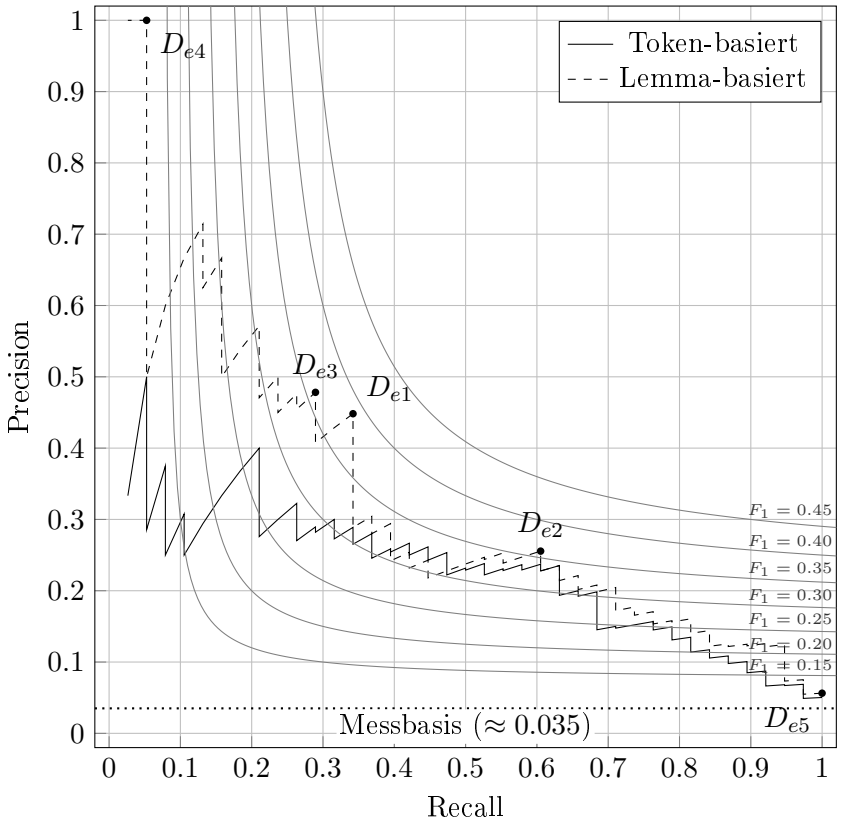

Abbildung 3.18: Precision-Recall-Kurve verschiedener FNN-Klassifizierer für expliziten Erklärungsbedarf

23 Ebenen, bei einer Kernelgröße von 5×5, Stride von 1×1 und *ReLU*-Aktivierung. Der zweite hat 32 Ebenen, eine Kernelgröße von 3×3, ebenfalls einen Stride von 1×1 und *ReLU*-Aktivierung. Beide Max Pooling Layer haben Poolgrößen von 2×2 und Strides von 2×2. Die Dropout Layer haben eine Dropout-Rate von 0.1. Auf diese beiden Tripel folgen vier Hidden Layer mit *ReLU*-Aktivierung und Neuronenanzahlen von 2500, 800, 200 und 20 sowie ein Output Layer mit einem Neuron und *Sigmoid*-Aktivierung. Sei dieses Netzwerkmodell M_1. $t \in [0, 1]$ ist ein Schwellenwert, der bestimmt, ab welchem Ergebniswert eine Eingabe als Erklärungsbedarf klassifiziert wird.

Das beschriebene Netzwerkmodell M_1 wird im folgenden mit der Validierungsmenge unter verschiedenen Schwellenwerten getestet.

Genereller Erklärungsbedarf

Für generellen Erklärungsbedarf liefert $C_{g1} = (M_1, 0.9245)$ den besten F_1-Score mit 0.3881 bei einer Präzision von 0.3562 und einem Recall von 0.4262. Einen weiterer interessanter Klassifizierer ist $C_{g2} = (M_1, 0.9226)$, er erreicht

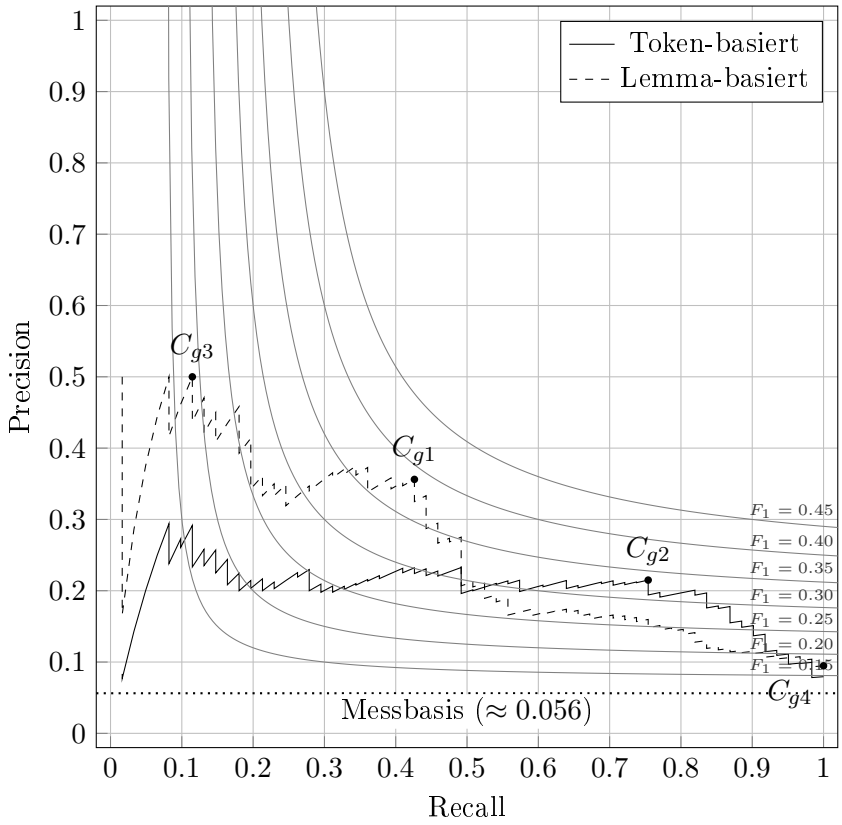

Abbildung 3.19: Precision-Recall-Kurve verschiedener CNN-Klassifizierer für generellen Erklärungsbedarf

einen F_1-Score von $0.33\overline{45}$ bei einer Präzision von 0.215 und einem Recall von 0.7541.

Die beste Präzision mit 0.5 erreicht $C_{g3} = (M_1, 0.9738)$ bei einem Recall von 0.1148 und einem F_1-Score von $0.18\overline{66}$. Einen maximalen Recall von 1.0 erreicht $C_{g4} = (M_1, 0.3049)$ bei einem F_1-Score von 0.1731 und einer Präzision von 0.0947, was einer Verbesserung über der Messbasis von 69% entspricht.

Expliziter Erklärungsbedarf

Für expliziten Erklärungsbedarf erreichen mehrere Klassifizierer einen F_1-Score von über 0.4. Einer von ihnen ist $C_{e1} = (M_1, 0.9202)$ mit einem F_1-Score von 0.4286, einer Präzision von 0.35 und einem Recall von 0.5526. Ein weiterer ist $C_{e2} = (M_1, 0.9421)$ mit einem F_1-Score von 0.4096 bei einer Präzision von $0.3\overline{7}$ und einem Recall von 0.4474.

Die höchste Präzision erreicht mit 0.5 $C_{e3} = (M_1, 0.9820)$ bei einem Recall von 0.079 und einem F_1-Score von $0.1\overline{36}$. Den maximalen Recall von

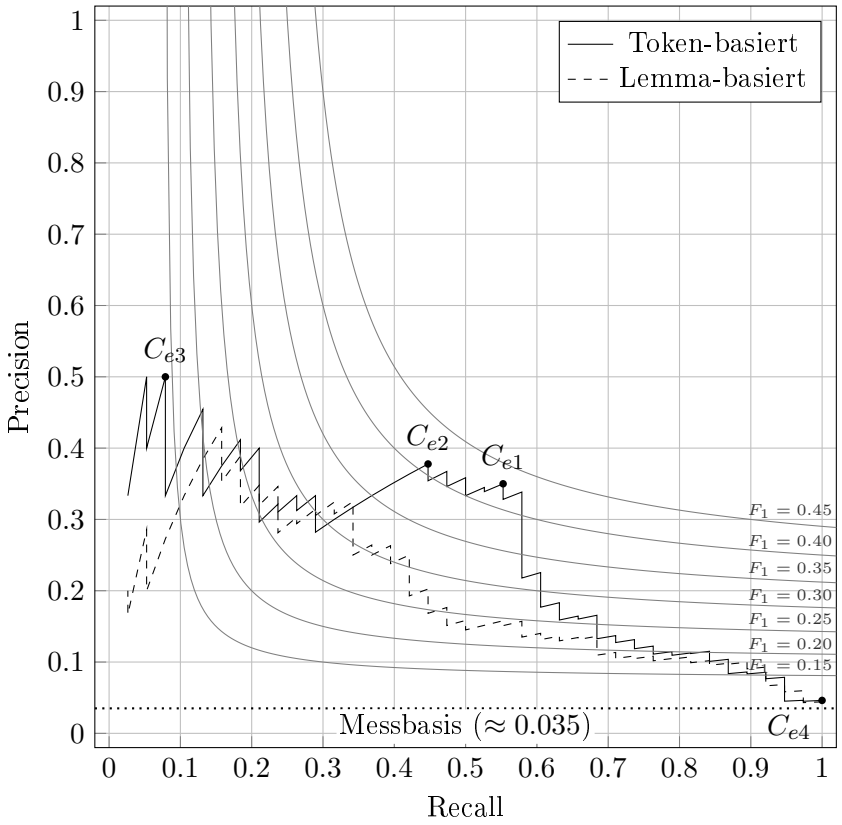

Abbildung 3.20: Precision-Recall-Kurve verschiedener CNN-Klassifizierer für expliziten Erklärungsbedarf

1.0 erreicht $C_{e4} = (M_1, 0.0082)$ mit einem F_1-Score von 0.0883 und einer Präzision von 0.0462. Dies entspricht einer Verbesserung über der Messbasis von 32%.

3.5.7 Klassifizierer: Long short-term memory

Da LSTM-Netzwerke mit Eingaben unterschiedlicher Länge arbeiten können, wird die paarweise Merkmalskorellation aus Sektion 3.4.2 nicht benötigt. Als Eingabe dient die Satzmatrix Q aus Sektion 3.4.1.

Analog zu den vorherigen Sektionen 3.5.5 und 3.5.6 besteht der LSTM-Klassifizierer $L = (M, t)$ aus einem trainierten Netzwerkmodell M und einem Schwellenwert t. Lernrate $r = 2 \times 10^{-5}$ und Decay $d = 1 \times 10^{-7}$ konnten wieder aus M_0 übernommen werden, die Epochenanzahl e wurde auf 40 gesetzt, da das Modell bei einer höheren Epochenzahl mehr Anzeichen von Überanpassung gezeigt hat. Im Netzwerk N_2 folgen auf den Eingabe-Layer zwei LSTM-Layer mit je 128 Zellen. Der Output-Layer umfasst wieder ein Neuron mit *Sigmoid*-Aktivierung. Sei das beschriebene Netzwerkmodell M_2.

$t \in [0, 1]$ ist analog zu den vorherigen Sektionen ein Schwellenwert, der bestimmt, ab welchem Ergebnis eine Eingabe als Erklärungsbedarf klassifiziert wird.

Folgend wird das Netzwerkmodell M_2 unter verschiedenen Schwellenwerten mit der Validierungsmenge getestet.

Genereller Erklärungsbedarf

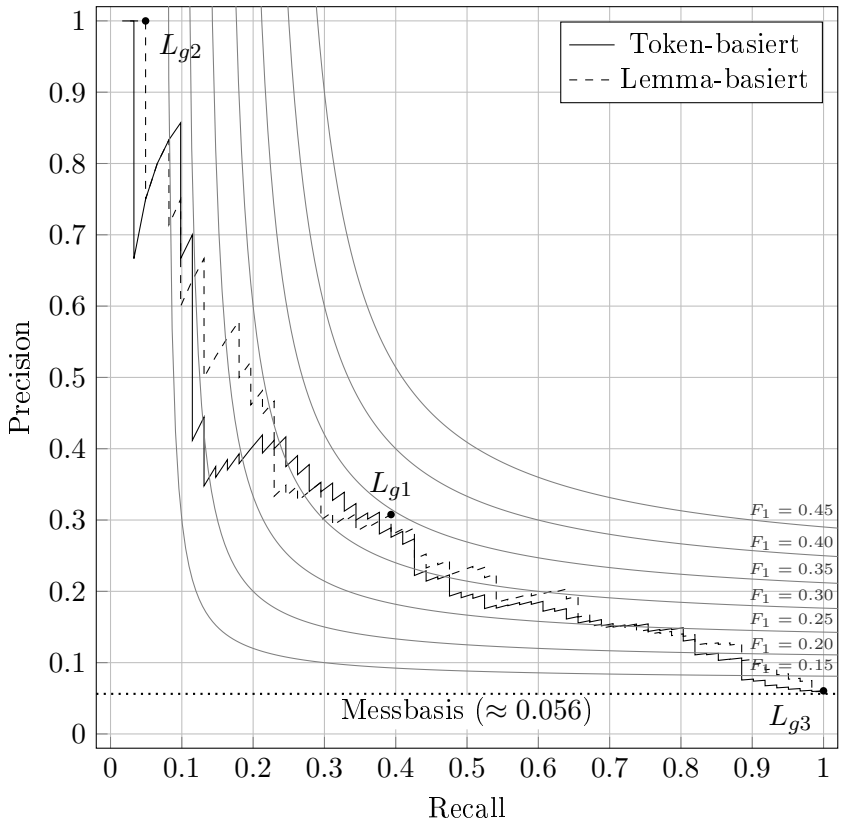

Abbildung 3.21: Precision-Recall-Kurve verschiedener LSTM-Klassifizierer für generellen Erklärungsbedarf

Den besten F_1-Score für generellen Erklärungsbedarf liefert mit 0.3453 der Klassifizierer $L_{g1} = (M_2, 0.7409)$ bei einer Präzision von 0.3077 und einem Recall von 0.3934.

Die höchste Präzision mit 1.0 erreicht $L_{g2} = (M_2, 0.93)$ bei einem Recall von 0.049 und einem F_1-Score von 0.0938. Einen maximalen Recall von 1.0 erreicht schließlich $L_{g3} = (M_2, 0.013)$ bei einem F_1-Score von 0.1140 und einer Präzision von 0.0605, was einer Verbesserung von 72% über der Messbasis entspricht.

Expliziter Erklärungsbedarf

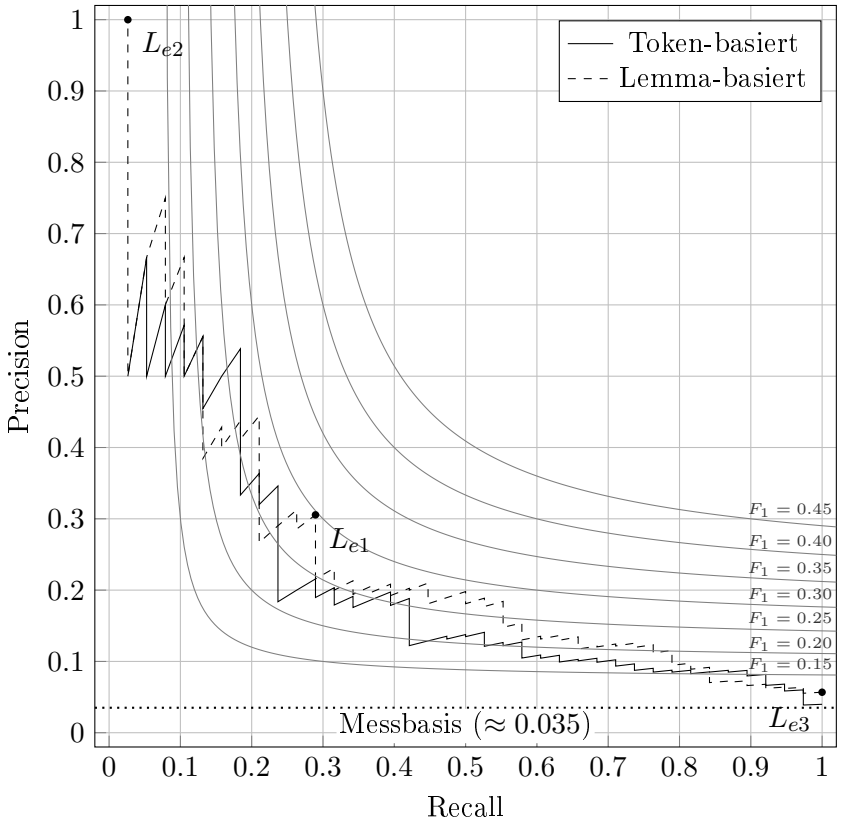

Abbildung 3.22: Precision-Recall-Kurve verschiedener LSTM-Klassifizierer für expliziten Erklärungsbedarf

Den maximalen F_1-Score für expliziten Erklärungsbedarf erzielt mit $0.\overline{297}$ $L_{e1} = (M_2, 0.9015)$ bei einer Präzision von $0.30\overline{5}$ und einem Recall von 0.2895.

Die maximale Präzision von 1.0 erreicht $L_{e2} = (M_2, 0.9822)$ bei einem Recall von 0.0263 und einem F_1-Score von 0.0513. Den maximalen Recall von 1.0 erzielt schließlich $L_{e3} = (M_2, 0.1732)$ bei einem F_1-Score von 0.1075 und einer Präzision von 0.0568. Dies entspricht einer Verbesserung der Messbasis von 62%.

3.5.8 Klassifizierer: Skip-Gramme

Dieser Klassifizierer basiert auf einer zweistufigen Klassifizierung, bei der zuerst aus den zu klassifizierenden Sätzen Skip-Gramme gebildet werden. Diese Skip-Gramme sind die eigentlichen Dokumente, die durch ein neuronales Netz klassifiziert werden. Ein Satz wird dann auf Grundlage der Anzahl

der als Erklärungsbedarf klassifizierten Skip-Gramme klassifiziert.

Die Eingabedokumente werden aus der Ground Truth für expliziten Erklärungsbedarf generiert. Folgend wird ein Beispiel für 1-Skip-Bigramme betrachtet, die eigentliche genutzte Art von Skip-Grammen ist eine andere. Der folgende Satz ist ein Satz mit explizitem Erklärungsbedarf, er ist fett gedruckt:

„**I can not find** it anywhere“

Hieraus ergeben sich die folgenden 1-Skip-Bigramme:

{**I can**, **I not**, **can not**, **can find**, **not find**, not it, find it, find anywhere, it anywhere}

Skip-Gramme, dessen Tokens im Ausgangssatz als Erklärungsbedarf markiert waren, werden als Skip-Gramme mit Erklärungsbedarf markiert (fett gedruckt). Alle anderen Skip-Gramme bilden die Menge von Skip-Grammen ohne Erklärungsbedarf.

Soll nun ein Satz klassifiziert werden, wird er wie die Trainingsdaten in Skip-Gramme zerlegt. Diese Skip-Gramme werden dann vom neuronalen Netz klassifiziert. Liegt der Wert für ein bestimmtes Skip-Gramm über einem Schwellenwert t, wird es als Skip-Gram mit Erklärungsbedarf klassifiziert. Ob ein Satz nun Erklärungsbedarf ist, wird durch die folgende Ungleichung bestimmt:

$$n_{\text{Erklärungsbedarf}} \geq a * n + b$$

Ist die Anzahl an Skip-Grammen mit Erklärungsbedarf $n_{\text{Erklärungsbedarf}}$ im Satz größer als eine durch die Anzahl von Skip-Gramm in einem Satz n parametrisierte Grade, so wird der Satz als Erklärungsbedarf eingestuft.

Der Klassifizierer $S = (M, t, a, b)$ hat wieder M als einen Hyperparameter. N_3 ist ein Feedforward-Netzwerk, bestehend aus 5 Hidden Layern mit je 2000 Neuronen und *ReLU*-Aktivierung, jeweils gefolgt von einem Dropout Layer mit Dropout-Rate von 0.1. Abschließend folgt ein Hidden Layer mit einem Neuron und *Sigmoid*-Aktivierung.

Die Lernrate wurde auf $r = 2 \times 10^{-6}$ und der Decay $d = 1 \times 10^{-8}$ festgelegt. Der Grund für die Verringerung der Lernrate ist, dass dieses Netzwerk deutlich schneller lernt als vorherige Netzwerke, was an der geringeren Eingabedimension liegen könnte. Daher musste die Epochenanzahl ebenfalls auf 20 reduziert werden, das Netzwerk leidet sonst deutlich an Überanpassung. Sei dieses Modell M_3.

Um eine performante Konfiguration zu finden, wurde zunächst die Validierungsmenge mit M_3 ausgewertet. Dann wurde durch mithilfe der Brute-Force-Methode performante konfigurationen von t, a und b gesucht. Da a und b eine Gerade definieren, die Ausgaben in Erklärungsbedarf und nicht Erklärungsbedarf trennen, hätte hier auch eine Support Vector Machine

genutzt werden können. Da der Suchraum jedoch nur zwei Dimensionen umfasst und die Werte n und $n_{\text{Erklärungsbedarf}}$ für ein gegebenes t für jeden Satz in der Validerungsmenge im Voraus berechnet werden können, konnten mehrere hundert Millionen Konfigurationen in wenigen Minuten getestet werden. Abbildung 3.23 zeigt einige Tausend dieser Konfigurationen und hebt einige von ihnen hervor.

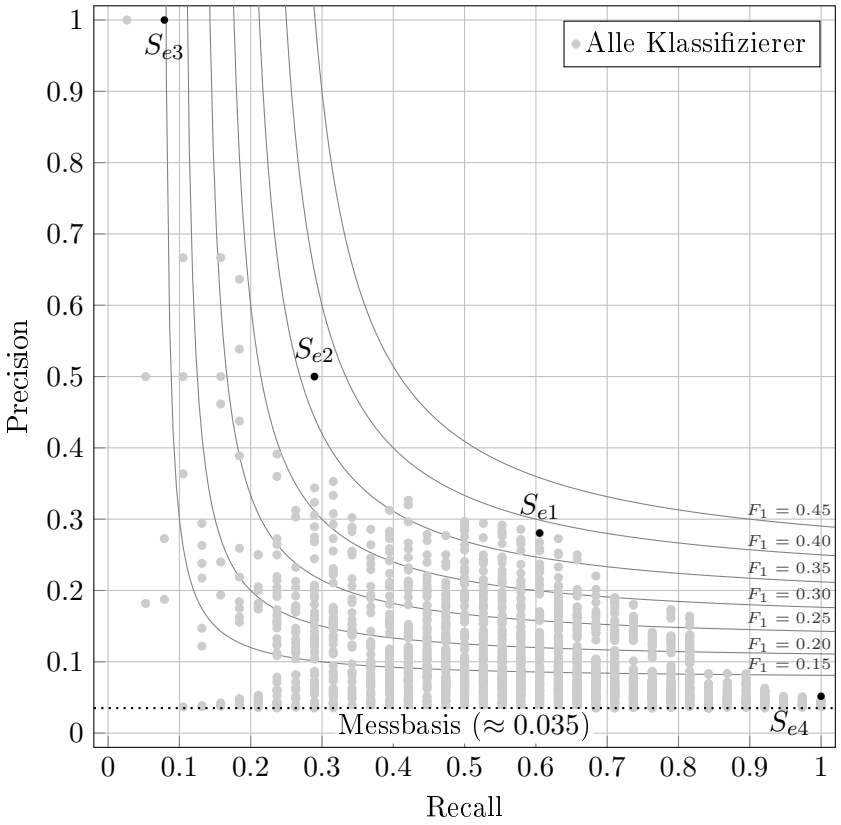

Abbildung 3.23: Precision-Recall-Kurve verschiedener LSTM-Klassifizierer für expliziten Erklärungsbedarf

Den besten F_1-Score mit $0.38\overline{3}$ erreicht $S_{e1} = (M_3, 0.9914, 0, 5.5)$ bei einer Präzision von 0.2805 und einem Recall von 0.6053. Einen weiteren guten F_1-Score mit 0.3726 erzielt $S_{e2} = (M_3, 0.9946, 0, 4.5)$ bei einer Präzision von 0.2969 und einem Recall von 0.5.

Die maximale Präzision von 1.0 erreicht $S_{e3} = (M_3, 0.958, 0, 540)$ bei einem Recall von 0.079 und einem F_1-Score von 0.1463. Den maximalen Recall von 1.0 erreicht schließlich $S_{e4} = (M_3, 0.031, 0.0685, 3.4)$ bei einem F_1-Score von 0.0982 und einer Präzision von 0.0516, was einer Verbesserung über der Messbasis von 47% entspricht.

Auffällig ist, für alle herausgehobenen Klassifizierer, ausgenommen S_{e4},

$a = 0$ ist. Dies suggeriert, dass für eine Klassifizierung mit höherer Präzision nicht angenommen werden kann, dass $n_{\text{Erklärungsbedarf}}$ Proportional zu n wächst.

Kapitel 4

Implementierung

In diesem Kapitel werden die Implementierungsteile der Arbeit vorgestellt. In Abschnitt 4.1 wird zunächst gezeigt, wie der Crawler aufgebaut ist. Abschnitt 4.2 erklärt, in welcher Struktur Daten gespeichert werden. Schließlich zeigt Abschnitt 4.3, wie die neuronalen Netze implementiert sind.

4.1 Web Scraper

Der entwickelte Web Scraper ist modular aufgebaut und wurde in Java entwickelt. Für Scraping, welches Eingaben erfordert, um dynamische Inhalte nachzuladen, wird *Selenium* verwendet, siehe 4.1.1. Als HTML-Parser kommt *jsoup* zum Einsatz, eine Bibliothek, die umfangreiche Möglichkeiten zur Analyse von HTML-DOM bietet.

4.1.1 *Selenium*

Scraping, welches nicht einfach über HTTP-Downloads verrichtet werden kann, weil Beispielsweise Nutzereingaben wie Knopfdrücke oder Scrollen benötigt werden, um dynamische Inhalte nachzuladen, erfolgt über ein Framework namens *Selenium*. *Selenium*s Primärfunktion ist als ein Framework für automatisierte, browserbasierte Tests. Munzert et al. bemerkten 2017, dass *Selenium* umfassende Möglichkeiten bietet, einen Browser automatisiert zu steuern, was es auch für testfremde Anwendungen wie Web Scraping interessant macht [MRMN14, 253].

4.1.2 Modul: `crawler-base`

Das Modul `crawler-base` beinhaltet die Interfaces `CrawlerStrategy<Parameter, Result>` und `CrawlerStrategyIterator<Result>`. `CrawlerStrategy` ist eine *Strategy* im Sinne des Strategy-Designmusters, da das Extrahieren von Reviews und App-Informationen viele verschiedene Implementationsmöglichkeiten hat.

Es bietet lediglich eine Methode als eine grobe Implementierungsvorgabe. Parameter sind eine Instanz des generischen Parameters `Parameter`, Rückgabe ist ein `CrawlerStrategyIterator<Result>`. Ein `CrawlerStrategyIterator` ist angelehnt an `java.util.Iterator`, kann jedoch auf den beiden Methoden `next` und `hasNext` eine `CrawlerException` werfen.

4.1.3 Modul: `crawler-playstore`

Im Modul `crawler-playstore` gibt zwei Kernklassen.

Die Klasse `PlayStoreAppCrawlerStrategy` implementiert `CrawlerStrategy<Function<PlayStoreUrlGenerator, String>, PlayStoreApp>` und ist eine HTTP-Basierte Scraper-Strategie, welche als Parameter eine Mapping-Funktion zwischen `PlayStoreUrlGenerator` und `String` erhält. So können verschiedene Modi wie Suche nach einer bestimmten Query und Topcharts oder Neuerscheinungen einer bestimmten Kategorie mit einer Strategie gescraped werden. `PlayStoreUrlGenerator` ist ein Interface, verschiedene Methoden zur Bildung einer URL bereitstellt. Dies ist als Interface implementiert, damit eventuelle Änderungen in der URL-Struktur des Play Stores nicht in mehreren Klassen oder Modulen zu Änderungen führen. Die vom Iterator gelieferte Klasse `PlayStoreApp` enthält alle in Sektion 3.1.1 genannten Daten.

Die Zweite Kernklasse ist `SingleAppPlayStoreReviewCrawlerStrategy` und implementiert `CrawlerStrategy<String, PlayStoreReview>`. Sie ist eine Crawler-Strategie, welche Selenium aufbaut, da auf der Review-Seite einer App durch scrollen dynamisch neue Reviews nachgeladen werden. Der `String`-Parameter ist der URI der gewünschten App, zum Beispiel `com.google.android.apps.maps` für Googles *Google Maps*. Das Result-Objekt dieser Strategy ist `PlayStoreReview`, welches alle in Sektion 3.1.2 genannten Daten enthält.

4.1.4 Modul: `crawler-appstore`

Das Modul `crawler-appstore` bietet zwei verschiedene Strategien für App-Scraping. `FixedListAppStoreAppCrawlerStrategy` implementiert `CrawlerStrategy<AppCategory, AppStoreApp>` und ermöglicht Scraping von Toplisten-Apps aus einer vorgegebenen `AppCategory`, beispielsweise `AppCategory.FINANCE`. `FeedTopListAppStoreAppCrawlerStrategy` bietet die Möglichkeit, die in Sektion 3.2.1 beschriebenen kuratierte App-Listen zu scrapen. Beide Klassen sind HTTP-basiert und produzieren `AppStoreApp`-Instanzen, welche die in Sektion 3.2.1 erwähnten Daten enthalten.

Die Klasse `SingleAppAppStoreReviewCrawlerStrategy` implementiert `CrawlerStrategy<String, AppStoreReview>` und ist analog zu `SingleAppPlayStoreReviewCrawlerStrategy` aus Sektion 4.1.3 ein

scrollender Scraper mit Selenium. Der `String`-Parameter ist die ID der App, beispielsweise `585027354` für Googles *Google Maps*. Das Resultat-Objekt `AppStoreReview` enthält die in Sektion 3.2.2 genannten Daten.

4.2 Persistenz

Diese Sektion beschreibt, in welcherart Datenstruktur heruntergeladene Reviews, analysierte Sätze und Ground Truth gespeichert wurden. Um einen einfachen Transfer von Java-Objekten in die Datenbank zu ermöglichen, wurde Hibernate eingesetzt, siehe Sektion 4.2.1. Das Datenbankschema wird schließlich in Sektion 4.2.2 vorgestellt.

4.2.1 Hibernate

Eine Implementation von Data Access Objects (DAO) mit der klassischen Java-Datenbankschnittstelle JDBC ist relativ aufwendig. Für jede zu persistierende Klasse muss händisch ein DAO geschrieben werden, welches bei eventuellen Änderungen an der Klasse angepasst werden muss, was eine zusätzliche Fehlerquelle bedeutet. Aus Gründen der Fehlervermeidung, Erweiterbarkeit und Einfachheit wurde daher Hibernate eingesetzt. Hibernate ist ein Werkzeug zur Objektrelationalen Abbildung, welches eine aufwandsarme Abbildung von Java-Objekten zu relationalen Datenbanken ermöglicht.

Es wurde eine Klasse `HibernateRepository<E extends Serializable> implements Repository<E>` realisiert, die bereits DAO-Methoden wie `create(E entity)`, `delete(E entity)` und `findById(final long id)` implementiert. Soll nun ein DAO für ein bestimmtes Java-Objekt geschaffen werden, beispielsweise `Review`, muss die zu implementierende Klasse lediglich `HibernateRepository<Review>` erweitern. Alle Standard-DAO-Methoden stehen dann ohne weiteren Implementierungsaufwand bereit.

4.2.2 Datenbankschema

Das Datenbankschema, siehe Abbildung 4.1 lässt sich grob in drei Regionen spalten: Storefront-Spezifisch, statisch und Ground Truth. Die erste Region enthält die Tabellen `PLAYSTORE_APP` und `PLAYSTORE_REVIEW` für den Play Store und `APPSTORE_APP` und `APPSTORE_REVIEW` für den App Store. Der Grund für diese Trennung ist, dass die beiden Stores für Apps und Reviews abweichende Informationen bereitstellen.

Die zweite Region besteht aus den Tabellen `REVIEW` und `SENTENCE`. `REVIEW` konsolidiert Reviews aus beiden Review-Tabellen in eine Tabelle, dabei werden nur Daten berücksichtigt, die in beiden vorhanden ist. Nachdem ein Review in Sätze gespaltet wird und diese Sätze weiter auf Tokens,

Lemmas und Parts of Speech untersucht werden, werden die bei der Analyse gewonnenen Daten in `SENTENCE` gespeichert.

Die Tabelle `SENTENCE_GROUND_TRUTH` hält Informationen aus der manuellen Ground Truth Analyse, die auf ganze Sätze zutreffen, beispielsweise, ob ein Satz Erklärungsbedarf enthält. Schließlich hält die Tabelle `NGRAM` Skip-Gramme, die konkreten Erklärungsbedarf darstellen. Diese Skip-Gramme sind Sätzen zugeordnet.

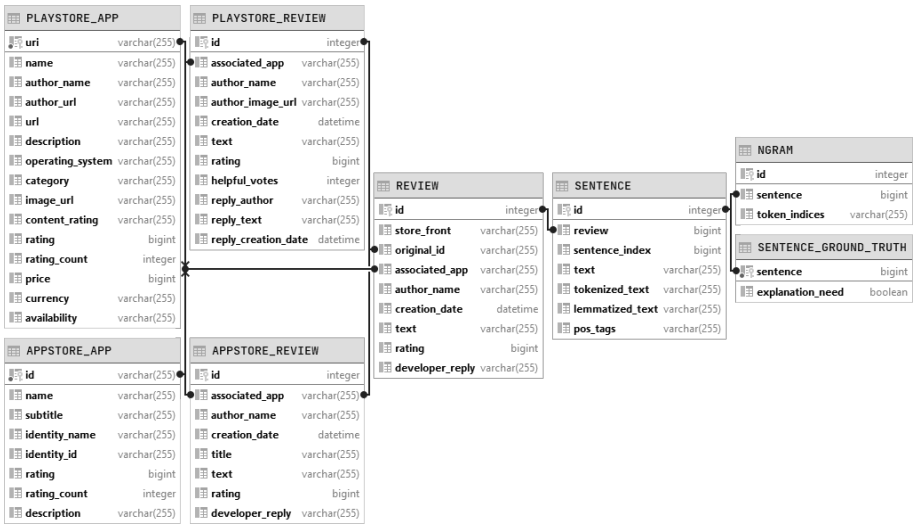

Abbildung 4.1: Datenbankschema

4.3 Neuronale Netze

Klassifizierer, die neuronale Netzwerke benutzt, benutzen dafür die Python-Bibliothek *TensorFlow*. Sie bietet Implementationen für eine umfangreiche Anzahl neuronaler Netzwerke. Für das Trainieren eines Word2Vec-Modells und die umwandlung eines Satzes in eine Word2Vec-Satzmatrix wurde die Python-Bibliothek *Gensim* eingesetzt, die eine implementierung von Word2Vec beinhaltet. Als Trainingsdaten dienten 193666 Sätze aus 70008 Reviews.

Kapitel 5

Evaluation

In diesem Kapitel werden die Ergebnisse der Arbeit diskutiert. Abschnitt 5.1 zeigt Erkenntnisse über die Gewinnung einer Ground Truth. Folgend evaluiert Abschnitt 5.2 die Entwickelten Klassifizierer. Abschnitt 5.3 beleuchtet anhand von Beispielen die Güte einiger Klassifizierer. Schließlicht zieht Abschnitt 5.4 ein Fazit über die Resultate.

5.1 Ground Truth

In dieser Sektion werden die Erkenntnisse über die Ground Truth den Prozess der Erstellung diskutiert. Zunächst wird in Sektion 5.1.1 beschrieben, wie hoch der zeitliche Aufwand war und welche mentalen Effekte beachtet werden sollten. Folgend werden in Sektion 5.1.2 Auswirkungen der festgestellten Ground Truth auf die Klassifikation besprochen.

5.1.1 Aufwand

Im Voraus der Arbeit war es schwer abzuschätzen, wie viel Zeit die manuelle Klassifikation von Sätzen in Anspruch nimmt. Es sollte eine möglichst hohe Anzahl von Sätzen klassifiziert werden, damit für die Klassifizierer eine möglichst große Datenbasis vorhanden ist. Gleichzeitig sollte dieser Prozess aber auch in angemessener Zeit stattfinden. Die gewählte Anzahl von 12 Apps mit je 100 Reviews wurde mit dem Plan gewählt, die Klassifikation in circa einer Woche abschließen zu können, also circa 200 Reviews am Tag bei sechs Arbeitstagen.

Die tatsächlich benötigte Zeit war etwa das doppelte, also im Schnitt die Reviews von etwa eine App am Tag. Der erste Grund ist ein unterschätzer Aufwand pro Satz. Entscheidungen, ob ein Satz Erklärungsbedarf enthält oder nicht ist oft nicht auf den ersten Blick erkennbar. Die beschriebenen 5W1H-Wörter waren oft ein guter Indikator, jedoch kommen sie, absolut betrachtet, viel häufiger in Sätzen ohne Erklärungsbedarf vor

(siehe Abbildung 5.1). Zusätzlich kommt in über der Hälfte der Sätze mit Erklärungsbedarf keines dieser Wörter vor, sodass jeder Satz trotzdem genau untersucht werden musste.

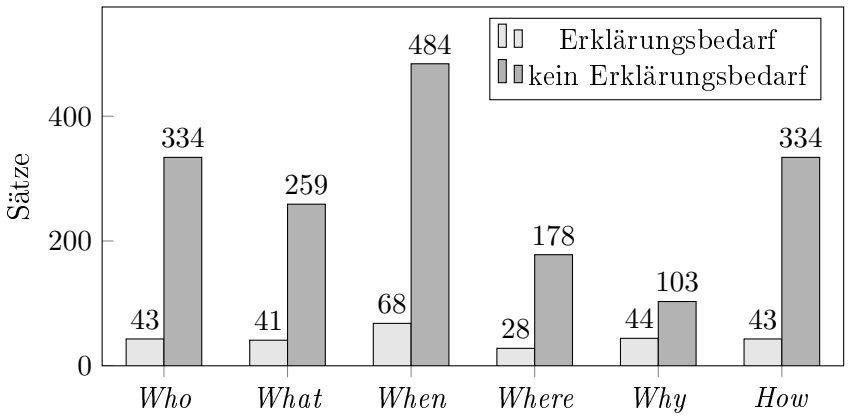

Abbildung 5.1: Absolute Frequenz von Sätzen mit mindestens einem 5W1H-Wort in Sätzen mit generellem Erklärungsbedarf

Ein weiterer Grund für den erhöhten Zeitaufwand war eine nach einiger Zeit einsetzende geistige Ermüdung. Trotz der benötigten Konzentration ist das Klassifizieren eine monotone Tätigkeit. Ohne häufige Pausen nahm subjektiv die Qualität der Klassifikation ab und musste teils nachträglich berichtigt werden. Subjektiv haben häufigere Pausen, etwa 10 Minuten pro halbe Stunde einen positiveren Effekt als etwa längere aber seltenere Pausen, etwa 20 Minuten pro Stunde.

Diese mit der Zeit sinkende Entscheidungsfähigkeit deckt sich mit Ergebnissen von Robertson und Kortum. Sie konnten in ihrer Veröffentlichung *Extraneous Factors in Usability Testing: Evidence of Decision Fatigue During Sequential Usability Judgments* nachweisen, dass aufeinanderfolgende Entscheidungen, in ihrem Fall über die Usability einer Software, dazu führt, dass zunehmend in eine bestimmte Richtig entschieden wurde [RK18]. Dieser Effekt heißt *Decision Fatigue* (Entscheidungsmüdigkeit). Sie empfehlen ebenfalls, Pausen einzulegen, konnten aber keine genaue Pausenlänge empfehlen.

5.1.2 Limitationen

Die hohe Schiefe der Verteilungen von Erklärungsbedarf und Nicht-Erklärungsbedarf in der Ground Truth führt dazu, dass die Anzahl von Sätzen mit Erklärungsbedarf relativ niedrig. So konnten für manche Apps nur einige wenige Sätze mit Erklärungsbedarf ermittelt werden. Beispielsweise sind bei MyChart für Android nur 10 aus 342 Sätzen solche mit Erklärungsbedarf. Eine hohe Schiefe verstärkt den Effekt, den eventuelle Ausreißer auf das Ergebnis haben können. Zusätzlich kann es in neuronalen

Netzen einfacher zu Überanpassung kommen. Um eine Überanpassung auszuschließen, wurde die Testmenge lediglich zur Leistungsüberprüfung genutzt. Wenn auch in der Testmenge ein gutes Ergebnis erzielt wird, ist eine Überanpassung unwahrscheinlicher.

5.2 Klassifizierer

In dieser Sektion werden die Ergebnisse der entwickelten Klassifizierer verglichen. Es wird gezeigt, welche Klassifizierer mit den Testmengen die besten Ergebnisse erzielt haben. Dann wird diskutiert, wie die Ergebnisse der Klassifizierer zustande kommen. Folgend werden genereller und expliziter Erklärungsbedarf verglichen. Zuletzt wird gezeigt, welche Ergebnisse der beste Klassifizierer für Reviews einer App einer ungesehenen Kategorie erzielt.

5.2.1 Genereller Erklärungsbedarf

Abbildung 5.2 zeigt die Leistung der in Sektion 3.5 vorgestellten Klassifizierer für generellen Erklärungsbedarf mit der Testmenge. Den höchsten F_1-Score mit 0.4571 erreicht der CNN-Klassifizierer C_{g1}. Er konnte sich im Vergleich zu der Validierungsmenge sogar noch verbessern und ist, expliziten Erklärungsbedarf eingeschlossen, der beste Klassifizierer. Im F_1-Rang folgen die FNN-Klassifizierer D_{g1}, D_{g2} und D_{g3} mit F_1-Scores zwischen 0.3804 bis 0.3. Auch sie haben, ausgenommen D_{g3}, einen leicht erhöhten F_1-Score. Es folgt ein weiterer CNN-Klassifizierer, C_{g2}, mit 0.28 und der LSTM-Klassifizierer L_{g1} mit 0.2791. Ihr F_1-Score ist jeweils leicht gesunken. Schließlich folgen die 5W1H-Klassifizierer und die übrigen, nicht genannten Klassifizierer.

Einen Klassifizierer für generellen Erklärungsbedarf mit einer Präzision von 1.0 für die Testmenge gibt es nicht. Der Klassifizierer mit der besten Präzision ist C_{g3} mit 0.4286, was nur marginal höher ist als die Präzision von C_{g1}, bei einem 82% geringeren Recall. Er hatte für die Validierungsmenge eine Präzision von 0.5 bei etwa 16% höherem Recall. Alle anderen Klassifizierer, die wegen ihrer hohen Präzision mit der Validierungsmenge ausgesucht wurden, haben mit dem Testset eine zum Teil stark reduzierte Präzision. Diese Diskrepanz rührt vermutlich daher, dass Klassifizierer, die mit der Validierungsmenge eine hohe Präzision erreichen konnten nur einige wenige Sätze als True Positive klassifiziert haben. Sie waren also auf wenige „Ausreißer“ in der Validierungsmenge optimiert, die nicht in gleicher Form in der Testmenge auftauchen.

Ein ähnliches Bild zeigt sich für Klassifizierer, die nach hohem Recall ausgewählt wurden. Lediglich einer von ihnen, B_{g3}, erreicht für die Testmenge einen Recall von 1.0, er gleicht jedoch im Ergebnis einem naiven Klassifizierer und klassifiziert alle Sätze als Sätze mit Erklärungsbedarf. L_{g3}

zeigt bei einem Recall von nunmehr 0.9836 eine Präzision, die lediglich 5% über der Messlinie liegt.

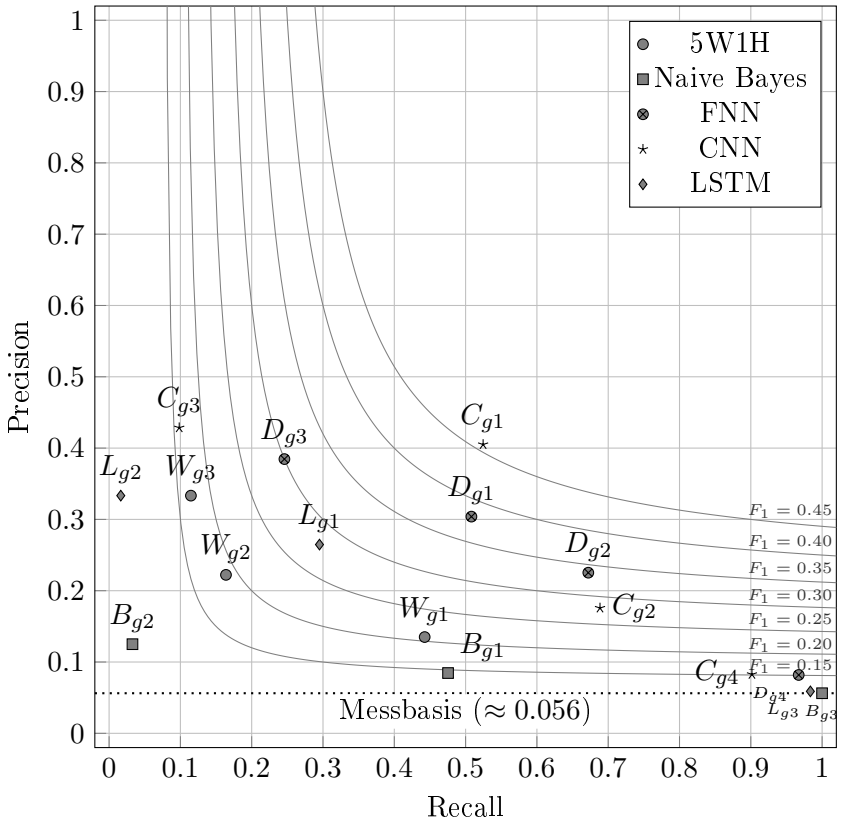

Abbildung 5.2: Precision-Recall-Werte verschiedener Klassifizierer für generellen Erklärungsbedarf

5.2.2 Expliziter Erklärungsbedarf

Abbildung 5.3 zeigt die Leistung der in Sektion 3.5 vorgestellten Klassifizierer für expliziten Erklärungsbedarf mit der Testmenge. Der Klassifizierer mit dem besten F_1-Score bleibt der CNN-Klassifizierer C_{e1} mit 0.3226, gefolgt von C_{e2} mit 0.2895. Sie haben im Vergleich zu der Testmenge beide an Leistung verloren. Die nächstbesten Klassifizierer nach F_1-Score sind der Skip-Gramm-Klassifizierer S_{e1} mit 0.2830 und der LSTM-Klassifizierer L_{e1} mit 0.2153, gefolgt von den FNN-Klassifizierern D_{e2}, D_{e3} und D_{e1} mit einem F_1-Score von 0.224 bis 0.2029. Es folgen die 5W1K-Klassifizierer und die restlichen nicht genannten Klassifizierer.

Zwei Klassifizierer konnten ihre Präzision von 1.0 aufrechterhalten, D_{e4} und L_{e2}. Sie klassifizieren jedoch jeweils nur einen Satz als Satz mit Erklärungsbedarf, was diese Ergebnisse wiederum sehr von „Ausreißern“

abhängig macht.

Den maximalen Recall von 1.0 erreicht L_{e3}, er konnte alle Sätze mit Erklärungsbedarf korrekt klassifizieren und hat eine um 82% erhöhte die Präzision gegenüber des naiven Klassifizierers.

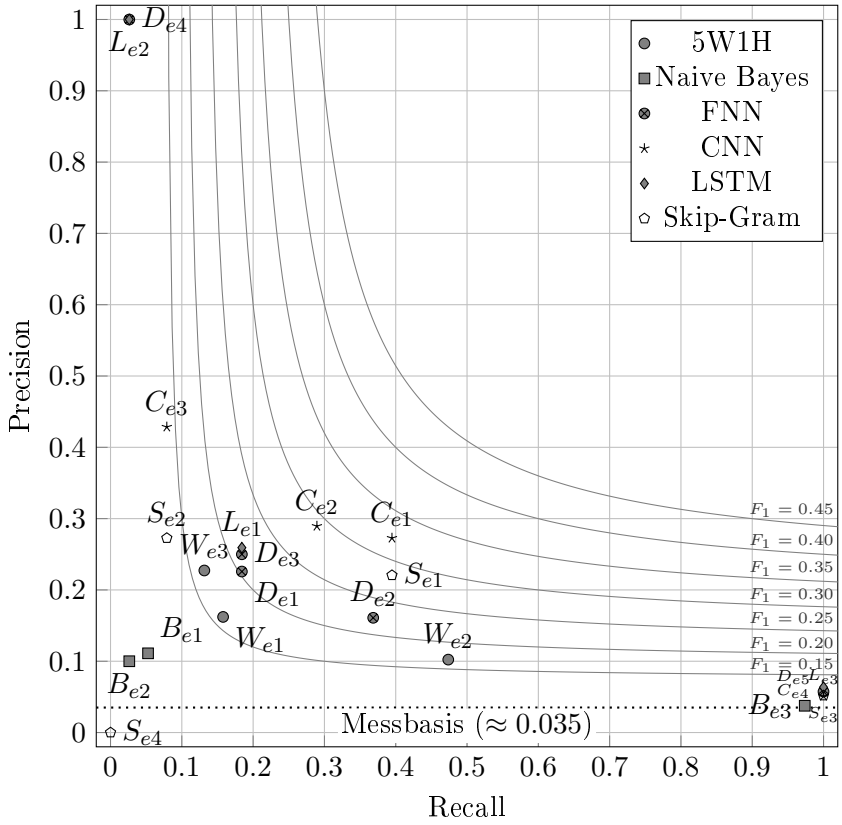

Abbildung 5.3: Precision-Recall-Werte verschiedener Klassifizierer für expliziten Erklärungsbedarf

5.2.3 Vergleich Explizit vs Generell

Mit den Testmengen konnten für generellen Erklärungsbedarf bessere Ergebnisse erzielt werden als für generellen Erklärungsbedarf. Der beste Klassifizierer für generellen Erklärungsbedarf hat einen 41% höheren F_1-Score als der beste Klassifizierer für expliziten Erklärungsbedarf. Abbildung 5.4 zeigt die Verteilung von F_1-Scores von Klassifizierern, die in Sektion 3.5 wegen ihres hohen F_1-Scores hervorgehoben wurden. Klassifizierer, die wegen hoher Präzision oder wegen hohem Recall ausgewählt wurden, sind nicht enthalten.

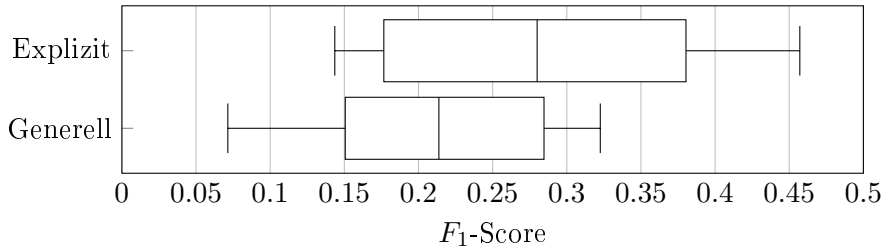

Abbildung 5.4: Verteilung von F_1-Scores in Klassifizierern, die wegen ihres hohen F_1-Scores mit der Validierungsmenge ausgewählt wurden

5.3 Fallstudien

Im folgenden Abschnitt sollen die Limitationen der vorgestellten Klassifizierer vorgestellt werden. Am Beispiel der besten Klassifizierer für generellen Erklärungsbedarfs wird mit Beispielsätzen und Statistiken gezeigt, woran Klassifizierer scheitern und welche Sätze korrekt klassifiziert werden konnten.

5.3.1 Fallstudie: 5W1H

Durch den simplen Aufbau des 5W1H-Klassifizierers lässt sich seine Leistung einfach Erklären. Als beispiel soll W_{g1} dienen, der von den 5W1H-Klassifizierern den besten F_1-Score erreicht hat. Er klassifiziert alle Sätze als Erklärungsbedarf, die mindestens ein 5W1H-Wort enthalten.

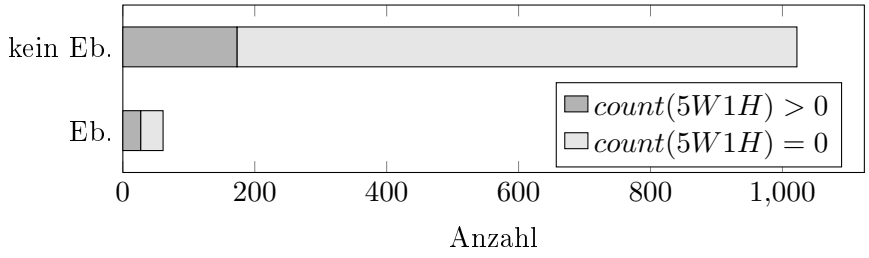

Abbildung 5.5: Verteilung von Sätzen mit mindestens einem 5W1H-Wort und keinem 5W1H-Wort in Sätzen mit und ohne Erklärungsbedarf (Eb.)

Abbildung 5.5 zeigt die Limitierungen dieses Klassifizierers: Die hohe Schräge der Daten kombiniert mit dem Fakt, dass auch Sätze ohne Erklärungsbedarf 5W1H-Wörter enthalten. Diese Verteilung lässt sich direkt in die Wahrheitsmatrix des Klassifizierers umwandeln, siehe Tabelle 5.1. Der Klassifizierer kann zwar die Präzision eines naiven Klassifizierers um 141% überbieten, schließt dabei jedoch jeglichen Erklärungsbedarf aus, der nicht mindestens eines der 5W1H-Wörter enthält. Dies führt zu Falsch-Negativen wie

		Tatsächliche Klasse	
		Eb.	kein Eb.
Klassifiziert als	Eb.	27	173
	kein Eb.	34	849

Tabelle 5.1: Wahrheitsmatrix für W_{g1}

„The information it offers is short and vague."

oder

„The support article is of no help because the app itself is acting crazy due to poor design and functionality.",

die Erklärungsbedarf auf andere Weise mitteilen. Gleichzeitig gibt es viele Falsch-Positive wie

„This is **why** I only gave 4 out of 5 stars."

oder

„**What** is not to like about it?",

die 5W1H-Wörter beinhalten, jedoch keinen Erklärungsbedarf mitteilen.

5.3.2 Fallstudie: CNN

Neuronale Netzwerke sind Blackboxes, durch die Komplexität der Berechnungen der Ausgabe ist nur schwer verständlich, wie ein neuronales Netzwerk eine Entscheidung berechnet. Daher soll am Beispiel von C_{g1} anhand von einigen Satzbeispielen gezeigt werden, welche Sätze das Netzwerk korrekt klassifizieren kann und mit welchen Sätzen es Schwierigkeiten hat. Tabelle 5.2 zeigt die Wahrheitsmatrix des Klassifizierers mit der Testmenge. 41% der erkannten Sätze beinhalten Erklärungsbedarf, was einer Erhöhung von 623% über der Messbasis entspricht. Dabei erkennt er 52% der Sätze mit Erklärungsbedarf.

		Tatsächliche Klasse	
		Eb.	kein Eb.
Klassifiziert als	Eb.	32	47
	kein Eb.	29	975

Tabelle 5.2: Wahrheitsmatrix für W_{g1}

Das Netzwerk konnte sowohl Sätze wie

„I have no idea of if or **when** this pending charge will fall off the account."

oder

„I want to know **where** it is and how to retrieve it.“

erkennen, die 5W1H-Wörter enthalten, als auch Sätze wie

„Can they see me too?“

und

„The information it offers is short and vague.“,

die beispielsweise von W_{g1} falsch klassifiziert wurden.

Falsch-Negative beinhalten beispielsweise den Satz

„I have had money frozen or inexplicably disappear.“.

Hier zeichnet sich der Erklärungsbedarf vor allem durch das Wort „inexplicably“ aus. Dieses Wort kam in der Trainingsmenge in drei weiteren Sätzen vor, von denen zwei Sätze Erklärungsbedarf beinhalteten. Dies ist vermutlich nicht häufig genug für das Modell, um eine korrekte Klassifizierung durchzuführen, insbesondere, weil es im Training nicht immer mit Erklärungsbedarf verbunden war. Hier würde das Modell vermutlich von mehr Trainingsdaten profitieren. Ein weiteres Falsch-Negativ ist der Satz

„The app was simple to use and very helpful, but I was just given false hope and information on when the money was expected to be in my direct deposit.“.

Der Erklärungsbedarf in diesem Satz steckt ausschließlich im Nebensatz. Der Hauptsatz hingegen ist eine positive Bemerkung, welche, wie in Sektion 3.3.1 gezeigt, seltener Erklärungsbedarf enthalten. Dieser Hauptsatz ist vermutlich der Auslöser für die Fehlklassifikation, das Netzwerk war nicht in der Lage, die Wichtigkeit des Nebensatzes zu erfassen.

Ein Beispiel für Falsch-Positive ist der Satz

„In the explanation of what was in the update it stated “Nothing new-new. [sic]“.

Vermutlich war das Wort „explanation“ der Auslöser für die Falschklassifizierung. In der Testmenge gibt es fünf Sätze, die ein Wort beinhalten, das mit den Silben „expla-“ beginnt (beispielsweise „explanation“ oder „explained“). Drei dieser fünf haben Erklärungsbedarf und von diesen fünf ist er der einzige Satz, der falsch klassifiziert wurde. Über viele Falsch-Positive lassen sich allerdings keine einfachen Hypothesen aufstellen. Die Sätze

„I figured it would be a priority with a new app.“

und

„The app literally tells you what to do and how to do it.“

werden beide als Sätze mit Erklärungsbedarf klassifiziert, lassen aber keinen einfachen Schluss auf den Grund zu.

Folgend ein beispiel für ein Richtig-Negativ, welches trotz vermeintlicher Schwierigkeit richtig erkannt wurden:

„Without leaving the page and having to do research, it explains my non-taxable pay from combat/hazardous zones and helps guide me.“

Dies ist der zweite der im vorherigen Absatz angesprochenen Sätze ohne Erklärungsbedarf, die Wörter enthalten, die mit der Silbe „expla-“ beginnen. Hier konnte der Klassifizierer vermutlich erfolgreich den Kontext, in dem „explains“ steht, bewerten und den Satz als Satz ohne Erklärungsbedarf klassifizierern.

5.3.3 Fallstudie: Skip-Gramm-Klassifizierer

Auch für Skip-Gramm-Klassifizierer wird ein neuronales Netzwerk eingesetzt, was eine Analyse erschwert. Abbildung 5.6 zeigt am Beispiel von S_{e1} die Verteilung von Skip-Grammen mit Erklärungsbedarf im Verhältnis zu Skip-Grammen in einem Satz. Die Trennlinie bildet sich aus den beiden Parametern $a = 0$ und $b = 5.5$. Sätze über der Trennlinie werden vom Klassifizierer als Sätze mit Erklärungsbedarf klassifiziert. Auffällig ist, dass in vielen Sätze mit Erklärungsbedarf keine Skip-Gramme mit Erklärungsbedarf erkannt werden konnten. Ein Grund hierfür sind der hohe Schwellenwert in S_{e1} von $t = 0.991656$. Bei einem ausgewogeneren Schwellenwert von $t = 0.5$ zeigt sich ein anderes Bild. Hier konnten in beinahe allen Sätzen mit Erklärungsbedarf Skip-Gramme mit Erklärungsbedarf klassifiziert werden, jedoch werden auch in vielen Sätzen ohne Erklärungsbedarf mehr Skip-Gramme mit Erklärungsbedarf gefunden.

		Tatsächliche Klasse	
		Eb.	kein Eb.
Klassifiziert als	Eb.	15	53
	kein Eb.	23	931

Tabelle 5.3: Wahrheitsmatrix für S_{e1}

Die meisten Skip-Gramme mit Erklärungsbedarf, 156, konnten im Richtig-Positiven Satz

„I asked **who** would be able to give me that information and then they said there is no information available.“

gefunden werden. Er enthält das 5W1H-Wort „who“, welches jedoch nicht in den vorhergesagten Skip-Grammen mit Erklärungsbedarf auftaucht.

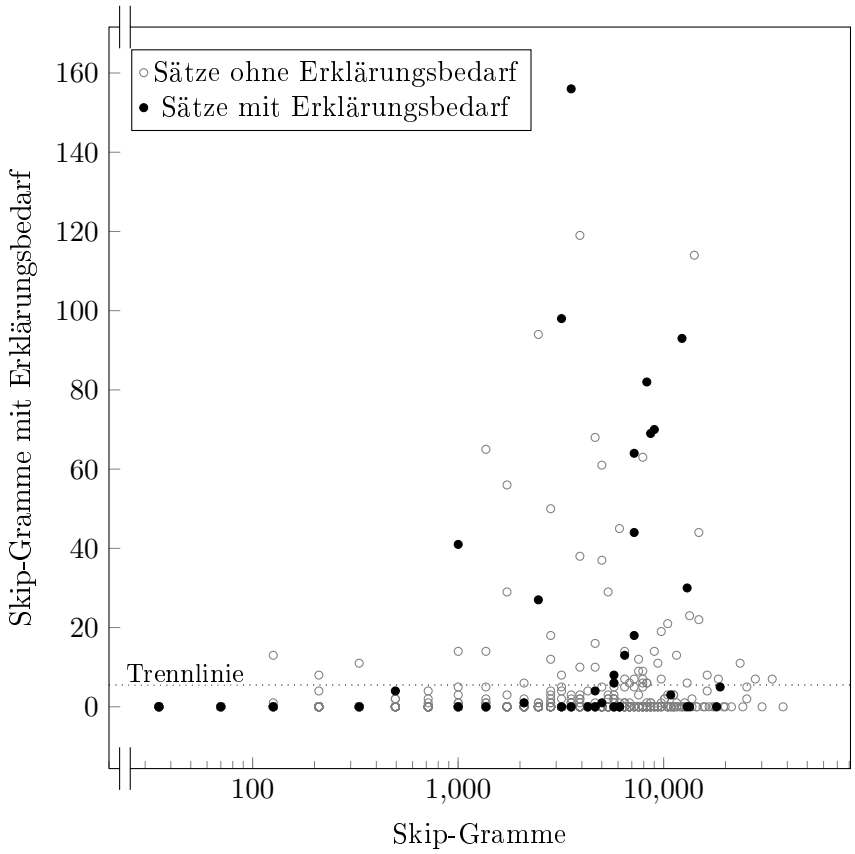

Abbildung 5.6: Verteilung von Skip-Grammen mit Erklärungsbedarf im Verhältnis zu Skip-Grammen für S_{e1}

Stattdessen sind es Skip-Gramme, die ein oder sogar zwei mal das Wort „information“ enthalten, wie $(give, information, no, information)$ oder $(there, be, no, information)$. Im Richtig-Positiven Satz

„Also, **why** is there no way to cancel a payment and **why** was my payment flagged for review?!?“

wurden 98 Skip-Gramme mit Erklärungsbedarf vorhergesagt. Es sind zumeist Skip-Gramme wie (why, no, to, why) oder $(why, be, there, no)$, die das 5W1H-Wort „why“ enthalten. Richtig-Positive beschränken sich jedoch nicht nur auf Sätze mit 5W1H-Wörtern. Auch

„If Venmo will correct my issues or even atleast give me an explanation I'd be happy to update my review but as of now I unfortunately have left this type of review.“

wurde mit 82 als Erklärungsbedarf erkannten Skip-Grammen korrekt klassifiziert. Hier drehen sich die erkannten Skip-Gramme um das Wort „explanation“: $(atleast, give, a, explanation)$ oder $(issue, give, a, explanation)$.

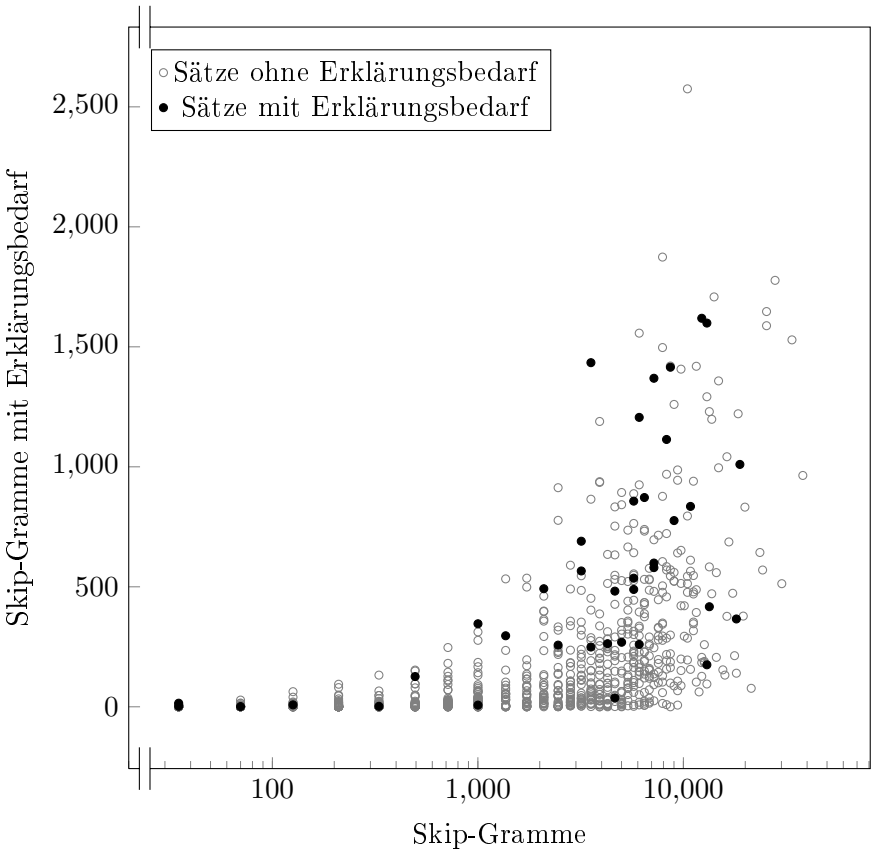

Abbildung 5.7: Verteilung von Skip-Grammen mit Erklärungsbedarf im Verhältnis zu Skip-Grammen für einen Klassifizierer mit Schwellenwert $t = 0.5$

Ein Beispiel für Falsch-Positive mit vielen Skip-Grammen, die als Erklärungsbedarf erkannt wurden, ist der Satz

„The first time I did it, they send me an email telling me some information was as not imported correctly.“,

bei dem viele Skip-Gramme, die das Wort „information“ beinhalten, wie $(I, information, as, not)$ und $(information, be, not, correctly)$, als Erklärungsbedarf klassifiziert wurden. Auch der Satz

„So far, on the iPhone I have not had a single issue with this app, unlike the dreaded Apple maps that to this day, doesn't always give me the proper directions, and takes too long to figure out when I veer off the path.“.

wurde als Falsch-Positiv klassifiziert. Der zweite Teil des Reviews ist ein negatives Erlebnis mit einer konkurrierenden App, hier wurden Skip-Gramme

wie $(not, give, the, proper)$ oder $(to, figure, out, when)$ als Erklärungsbedarf klassifiziert.

Der Satz

> „I used to have no issues with this but lately it has been changing my destination with no notification and taking me to the wrong spot."

ist ein Beispiel für ein Falsch-Negativ, kein Skip-Gramm liegt über dem Grenzwert.

5.4 Fazit

Es wurden diverse Klassifizierer entwickelt und für generellen sowie expliziten Erklärungsbedarf getestet. Klassifizierer für generellen Erklärungsbedarf konnten dabei sowohl eine höhere Präzision als auch einen höheren Recall erzielen. Da das Erstellen einer Ground Truth für expliziten Erklärungsbedarf mindestens mit demselben Aufwand verbunden ist, wie das Erstellen einer Ground Truth für generellen Erklärungsbedarf ist daher der CNN-Klassifizierer C_{g1} zu empfehlen. Er erreicht mit der Testmenge eine Präzision von 0.4051 und einen Recall von 0.5246, was einem F_1-Score von 0.4571 entspricht. 95% der Sätze, die keinen Erklärungsbedarf enthielten, konnten mit diesem Klassifizierer herausgefiltert werden, während lediglich 48% der Sätze mit Erklärungsbedarf nicht erkannt wurden.

Einfache Klassifizierer wie der 5WH1-Klassifizierer oder der naive Bayes-Klassifizierer konnten keine guten Leistungen erzielen und der naive Bayes-Klassifizierer konnte schlecht von den Trainings- auf die Testdaten generalisieren.

Der Skip-Gramm-Klassifizierer S_{e1} konnte für expliziten Erklärungsbedarf überdurchschnittliche Ergebnisse erzielen, wurde jedoch von den CNN-Klassifizierern C_{e1} und C_{e2} übertroffen. Für diese Art klassifizierer muss mehr Arbeit beim Erstellen einer Ground Truth aufgewendet werden, da in jedem Satz mit Erklärungsbedarf Tokens, die Erklärungsbedarf darstellen, markiert werden müssen. Auch das Trainieren und Klassifizieren ist etwas aufwendiger, da Sätze in Skip-Gramme aufgeteilt und jedes Skip-Gramm einzeln klassifiziert werden muss. Es ist zweifelhaft, ob sich dieser Aufwand lohnt, wurden doch mit CNN-Klassifizieren ähnliche und sogar bessere Ergebnisse erzielt.

Klassifizierer mit hoher Präzision konnten, wenn überhaupt, nur eine geringe einstellige Anzahl an Sätzen als Erklärungsbedarf klassifizieren. Über ihre Güte lässt sich daher keine Aussagen machen, da sie stark von leichten Schwankungen in der Verteilung der Testmenge beeinflusst werden.

Klassifizierer mit hohem Recall, wie der LSTM-Klassifizierer L_{e3} konnten bei einem Recall von 1.0 bis zu 46% der Sätze ohne Erklärungsbedarf herausfiltern, durch die hohe Schiefe der Daten verbleibt jedoch trotzdem

ein großer Anteil an Falsch-Positiven, sodass nur eine Präzision von 0.0635 zustande kommt. In einer Echtweltanwendung wären die Kosten für so viele verbleibende Falsch-Positive vermutlich sehr hoch, da Personen, die klassifiziertes Benutzerfeedback sichten, trotzdem sehr viele Sätze ohne Erklärungsbedarf lesen müssten, was wiederum zu Decision Fatigue (Siehe Abschnitt 5.1.1) führen und so Ergebnisse weiter verschlechtern könnte. Hier wäre vermutlich ein Klassifizierer wie C_{g1} vorzuziehen, der zwar einige Sätze mit Erklärungsbedarf nicht erkennt, bei dem jedoch über 50% der erkannten Sätze tatsächlich Erklärungsbedarf enthalten.

Kapitel 6

Verwandte Arbeiten

Die Klassifikation von Benutzerfeedback ist ein aktuelles Forschungsthema. Guzman, El-Haliby und Bruegge stellten in ihrer Veröffentlichung *Ensemble Methods for App Review Classification: An Approach for Software Evolution* von 2015 ihre *User Review Taxonomy for Software Evolution* vor, eine Taxonomie von verschiedenen Arten von Nutzerfeedback. Zusätzlich evaluierten sie die Güte von verschiedenen Klassifizierern, die auf maschinellem Lernen basieren, für das Klassifizieren von Reviews in diese Taxonomie. Ihr bester Klassifizierer erreichte einen F-Score von 0.64 bei einer Präzision von 0.74 und einem Recall von 0.59 [GEHB15]. Maalej und Nabil evaluieren in ihrer Veröffentlichung *Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews* von 2015 verschiedene Arten probabilistischer Klassifizierer für die Klassifizierung von Benutzerfeedback in vier von ihnen definierte Klassen. Ihre Klassifizierer erreichen je nach Klasse Präzisions- und Recall-Werte zwischen 0.71 und 0.97 [MN15]. Panichella et al. beleuchteten 2015 in ihrer Veröffentlichung *How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution*, dass Benutzerfeedback eine reichhaltige Quelle von Informationen für App-Anbieter und -Entwickler sein kann, der teils unstrukturierte Schreibstil und variierende Qualität jedoch Herausforderungen darstellen. Sie zeigen drei Methoden, um Benutzerfeedback zu klassifizieren: Natural Language Processing, Textanalyse und Sentimentanalyse (Die Erkennung, ob ein Text eher positiv oder negativ gestimmt ist). Ihr bester Klassifizierer erreicht eine Präzision von 0.75 bei einem Recall von 0.74 [PDG+15].

Bussone, Stumpf und O'Sullivan erläutern in ihrer Veröffentlichung *The Role of Explanations on Trust and Reliance in Clinical Decision Support Systems* die Auswirkungen von Erklärungen auf Nutzervertrauen und Nutzerabhängigkeit. Sie konnten zeigen, dass Erklärungen eine größere Rolle für Vertrauen in ein System spielen als eine Prozentzahl an Gewissheit einer Diagnose. Erklärungen führten jedoch sogar dazu, dass Studienteilnehmer zu viel Vertrauen in eine Diagnose legten [BSO15]. Chazette, Karras

und Schneider haben 2019 für ihre Veröffentlichung *Do End-Users Want Explanations? Analyzing the Role of Explainability as an Emerging Aspect of Non-Functional Requirements* eine Studie durchgeführt, in der Probanden nach Vor- und Nachteilen an in Software eingebettete Erklärungen befragt wurden. Eine klare Mehrzahl der Befragten wünschte sich Erklärungen angesichts einer unerwarteten Situation. Der häufigste Wunsch war nach einer Erklärung über das „warum“ eines Ereignisses, die wenigsten wünschten sich Erklärungen über die inneren Abläufe einer Software [CKS19]. Chazette und Schneider untersuchten 2020 in ihrer Veröffentlichung *Explainability as a non-functional requirement: challenges and recommendations* Erklärungen als Nicht-Funktionale Anforderungen. Es wurden Vor- und Nachteile von in Software eingebetteten Erklärungen sowie ob Erklärungen eine gute Methode sind, um Software Transparenter zu machen. Sie konkludieren, dass die Integration von Erklärungen eine Kosten-Nutzen-Frage ist, die von Anforderungsingenieuren betrachtet werden sollte.

Kapitel 7

Zusammenfassung und Ausblick

Dieses Kapitel bietet zuerst in Abschnitt 7.1 eine Zusammenfassung der Ergebnisse der Arbeit und schließlich in Abschnitt 7.2 einen Ausblick, wie auf den in dieser Arbeit vorgestellten Ergebnissen aufgebaut werden kann.

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurden zunächst eine Kombination aus Web Crawler und Web Scraper entwickelt, die in der Lage sind, menschenlesbare Seiten aus Googles Play Store und Apples App Store herunterzuladen und Reviews und App-Metadaten zu extrahieren. Es wurden 1200 Reviews aus je sechs Apps pro Store analysiert und in insgesamt 7222 Sätze aufgeteilt. Diese Sätze wurden manuell auf Erklärungsbedarf untersucht, um eine Ground Truth aufzubauen. Dabei konnte festgestellt werden, dass nur etwa 5.6% der Sätze Erklärungsbedarf und nur etwa 3.5% der Sätze expliziten Erklärungsbedarf enthalten. Weiterhin wurde festgestellt, dass Sätze aus schlechten Reviews, also Reviews, die mit einem oder zwei Sternen bewertet wurden, häufiger Erklärungsbedarf enthalten als solche mit einer Bewertung von drei oder mehr Sternen. Auch konnte beobachtet werden, dass in der Probe schlechte Reviews länger sind. Ebenfalls konnte gezeigt werden, dass Reviews im Play Store meist auf eine Länge von 500 Zeichen begrenzt sind, was vermutlich an der Maximallänge für Reviews liegt, die Google in der Android-Version des Play Stores erzwingt.

Auf Basis der Ground Truth wurden sechs verschiedene Klassifizierer entwickelt: Eine Heuristik basierend auf 5W1H-Worten (*Who*, *What*, *When*, *Where*, *Why* und *How*), ein naiver Bayes-Klassifizierer, drei verschiedene neuronale Netzwerke und ein Klassifizierer, der nicht Sätze direkt klassifiziert, sondern ihre Skip-Gramme. Der erfolgreichste Klassifizierer war ein Klassifizierer für generellen Erklärungsbedarf, der ein Convolutional Neural

Network nutzt. Er erreichte eine Präzision von 0.41, einen Recall von 0.53 und eine Spezifität von 0.95 und übertraf damit alle getesteten Klassifizierer, insbesondere die Skip-Gramm-basierten Klassifizierer, welche einen höheren Arbeitsaufwand bei der Erstellung der Ground Truth erfordern.

Eine Herausforderung stellte die stark asymmetrische Verteilung von Sätzen mit Erklärungsbedarf in der Ground Truth dar. Viele Klassifizierer konnten keine hohe Spezifität entwickeln und hatten daher eine niedrige Präzision, selbst wenn viele tatsächliche Sätze mit Erklärungsbedarf erkannt wurden.

Unterschätzt wurde der Aufwand, den die manuelle Klassifikation von Sätzen in Anspruch nimmt. Es mussten häufige Pausen eingelegt werden, damit die Qualität der Klassifikation gewährleistet werden konnte.

Die in dieser Arbeit gewonnenen Erkenntnisse könnte ein Ansatz für App-Anbieter und -Entwickler bieten, die sich für Erklärungsbedarf in ihren Reviews interessieren.

7.2 Ausblick

Die gesammelte Datenmenge beschränkt sich auf sechs verschiedenen Apps aus drei Kategorien. Es wäre denkbar, die entwickelten Verfahren mit einer größeren Diversität an Daten zu testen, um zu überprüfen, ob die Modelle generalisierbar sind.

Ebenfalls wäre vorstellbar, einen Datensatz auf Reviews einer einzelnen App zu beschränken, um zu überprüfen, ob sich die Art und Weise, auf die sich Erklärungsbedarf äußert, von App zu App so verschieden ist, dass ein Datensatz einer einzelnen App bessere Ergebnisse liefert. Hier ist jedoch Vorsicht geboten, da vorstellbar ist, dass sich die Art, Erklärungsbedarf zu äußern, über die Zeit ändert, sodass ein Modell selbst für diese eine App nicht generalisierbar bleibt.

Vermutlich würden mehr Beispiel von Sätzen mit Erklärungsbedarf die Klassifizierer-Leistung verbessern. Viele neuronale Netzwerke benutzten Datensets, die ein oder mehrere Größenordnungen mehr Dokumente enthalten.

Gezeigt wurde, dass Sätze mit und ohne Erklärungsbedarf teils ähnliche Wörter und Satzstrukturen besitzen. Hier wäre es interessant, ob Metadaten wie Bewertung oder weitere NLP-Analysen, beispielsweise nach Wortart, Klassifizierer-Leistung verbessern kann.

Literaturverzeichnis

[AGG06] Ben Allison, David Guthrie, and Louise Guthrie. Another look at the data sparsity problem. In *Text, Speech and Dialogue*, pages 327–334, 2006.

[BG18] Toms Bergmanis and Sharon Goldwater. Context sensitive neural lemmatization with Lematus. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1391–1400, 2018.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[BSO15] Adrian Bussone, Simone Stumpf, and Dympna O'Sullivan. The role of explanations on trust and reliance in clinical decision support systems. In *2015 International Conference on Healthcare Informatics*, pages 160–169, 2015.

[CKS19] Larissa Chazette, Oliver Karras, and Kurt Schneider. Do end-users want explanations? analyzing the role of explainability as an emerging aspect of non-functional requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 223–233, 2019.

[CS20] Larissa Chazette and Kurt Schneider. Explainability as a non-functional requirement: challenges and recommendations. *Requirements Engineering*, 2020.

[GAL+06] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. 2006.

[GEHB15] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. Ensemble methods for app review classification: An approach for software evolution (n). pages 771–776, 2015.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–1780, 1997.

[IJ14] Bar Ifrach and Rameh Johari. Pricing a bestseller: Sales and visibility in the marketplace for mobile apps. *SIGMETRICS Perform. Eval. Rev.*, 41(4):51, 2014.

[Inc] Apple Inc. App store logo. `https://en.wikipedia.org/wiki/App_Store_(iOS)#/media/File:App_Store_(iOS).svg` - letzter Zugriff: 2020-07-21.

[Kuha] Marvin Kuhke. Finance-topliste auf iOS -endgerät. Screenshot aus dem Mobilinterface des Apple App Stores - letzter Zugriff: 2020-07-06.

[Kuhb] Marvin Kuhke. Finance-topliste im app store. Screenshot von `https://apps.apple.com/de/genre/ios-finance/id6015` - letzter Zugriff: 2020-07-06.

[Kuhc] Marvin Kuhke. s̈ee all¨-button im app store. Screenshot von `https://apps.apple.com/us/app/google-maps-transit-food/id585027354` - letzter Zugriff: 2020-07-04.

[Kuhd] Marvin Kuhnke. Aggregierte reviews und review mit entwicklerantwort im google play store. Screenshot von `https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=en` - letzter Zugriff: 2020-07-05.

[Liu07] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. 2007.

[LLC] Google LLC. Play store logo. `https://en.wikipedia.org/wiki/Google_Play#/media/File:Google_Play.svg` - letzter Zugriff: 2020-07-21.

[MCCD15] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey A. Dean. Computing numeric representations of words in a high-dimensional space, 2015.

[MN15] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 116–125, 2015.

[MRMN14] Simon Munzert, Christian Rubba, Peter Meissner, and Dominic Nyhuis. Automated data collection with r: A practical guide to web scraping and text mining. 2014.

[MS99] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. 1999.

[Ola15] Christopher Olah. Understanding lstm networks, Aug 2015. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/` - letzer Zugriff: 07.07.2020.

[PDG+15] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 281–290, 2015.

[Rib] Cliff Ribaudo. itunes customer reviews rss feed stops at page 10. Stackoverflow. `https://stackoverflow.com/q/16180191/` - letzter Zugriff: 2020-07-04.

[RK18] Ian Robertson and Phil Kortum. Extraneous factors in usability testing: Evidence of decision fatigue during sequential usability judgments. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 62:1409–1413, 2018.

[Sam15] Ian Sample. Computer says no: why making ais fair, accountable and transparent is crucial. *The Guardian*, 2015. `https://www.theguardian.com/science/2017/nov/05/computer-says-no-why-making-ais-fair-accountable-and-transparent-is-crucial` - letzter Zugriff: 2020-04-20.

[Sch] Holger Schlicht. Review-interface auf android-endgerät. Screenshot aus dem Mobilinterface des Google Play Stores - letzter Zugriff: 2020-07-08.

[Sch14] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[Sid17] Alexander Sidorov. Transitioning entirely to neural machine translation, 2017. `https://engineering.fb.com/ml-applications/transitioning-entirely-to-neural-machine-translation/` - letzer Zugriff: 2020-07-06.

[Tra] Brent Traut. Why does itunes store reviews rss feed sometimes return no results? Stackoverflow. `https://stackoverflow.com/q/42400574` - letzter Zugriff: 2020-07-04.

[VHMN12] Rajesh Vasa, Leonard Hoon, Kon Mouzakis, and Akihiro Noguchi. A preliminary analysis of mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 241–244, 2012.

[VL] Stanford Vision and Learning Lab. Convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks/ - letzter Zugriff: 2020-06-22.

[Vog17] Werner Vogels. Bringing the magic of amazon ai and alexa to apps on aws, 2017. `https://www.allthingsdistributed.com/2016/11/amazon-ai-and-alexa-for-all-aws-apps.html` - letzer Zugriff: 2020-07-06.

[Vog19] Andreas Vogelsang. Explainable software systems. *it - Information Technology*, 61(4):193 – 196, 2019.

[ZFDY16] Chenwei Zhang, Wei Fan, Nan Du, and Philip S. Yu. Mining user intentions from medical queries: A neural network based heterogeneous jointly modeling approach. In *Proceedings of the 25th International Conference on World Wide Web*, page 1373–1384, 2016.