Automatisierte Erkennung von Buganfragen im Kundensupport des Bankwesens

Annie Simon, Martin Prause, Michael Kästl, Enzo Posselt, Matthias Vodel, Manuel Heinzig, Matthias Baumgart, Christian Kastner, Christian Roschke, Marc Ritter

Hochschule Mittweida, Fakultät Angewandte Computer- und Biowissenschaften
Technikumplatz 17, 09648 Mittweida

Abstract

Aktuell werden bereits rund 30 Prozent einer großen Menge an Supportanfragen innerhalb der Sparkassen-App automatisiert verarbeitet, um einerseits den menschlichen Support-Mitarbeiter zu entlasten und um anderseits dem Kunden unabhängig von Auslastung und Tageszeit schnellstmöglich eine Lösung für sein Problem zu liefern. Das vorliegende Paper untersucht dabei eine Möglichkeit, speziell die für Entwickler wichtigen Anfragen zu Bugs und Missverhalten der App ansich herauszufiltern. Hierfür findet ein maschinelles Lernverfahren auf Basis eines Random Forrests Anwendung. Die Trainingsdaten werden zuvor durch synthetische Generierung aufgewertet und auf Grundlage einer Poisson-Verteilung zeitlich an reale Gegebenheiten angenähert. Das entstandene Verfahren liefert eine hohe Präzision bei der gewünschten Klassifikation, die Sensitivität des Ansatzes lässt jedoch noch wesentlichen Spielraum für zukünftige Optimierungen.

1. Einleitung

Aktuell bestehen für Dialogsysteme und Agenten, nicht zuletzt auf Grund der Entwicklungen im Bereich des maschinellen Lernens (ML) sowie des Natural Language Processings (NLP), stark wachsende Möglichkeiten im Bereich der Forschung und wirtschaftsnahen Anwendung. Dieses Forschungsfeld wurde über drei Semester hinweg wissenschaftlich im Rahmen des Lehrkonzeptes *Digital Skills and Products* (Ritter et al. 2019) an der Hochschule Mittweida bearbeitet. Der dabei entstandene Prototyp soll durch die Kombination eines Machine Learning-Models mit einem Wahrscheinlichkeitsmodell auf Basis der Poisson-Verteilung die automatisierte Detektion von Kundenanfragen mit Bezug zu Softwarefehlern (engl. Bugs) ermöglichen.

Eine Veröffentlichung der Firma *Star Finanz* im August 2022 (Krukemeyer, 2022), zeigt die Ausgangslage des Systems auf. Aktuell können etwa 30% aller Kundenanfragen automatisiert beantwortet werden. Dabei handelt es sich inhaltlich um simple, in der Formulierung klar standardisierte Anliegen wie das Zurücksetzen des Passworts oder vergessener Daten zum Onlinebanking. Bei der Erkennung von entwicklungstechnisch wertvollen Bug-Supportanfragen gibt es noch klaren Spielraum nach oben. Als simple Herangehensweise wird das “Bayes Theorem“ angewandt, um die Wahrscheinlichkeit eines Auftretens von Schlüsselworten abzuschätzen. Jedoch schwindet die Nutzbarkeit bei einer mangelnden Anzahl von sich wiederholenden Bug-Supportanfragen, ähnlich zu den Erkenntnissen von (Giger et al., 2012). Das Verfahren ermittelt nahezu keine falsch-positiven Treffer (Jalbert und Weimer, 2008), hat aber in Summe keine hinreichende Zuverlässigkeit.

Die Nutzung der von Giger verwendeten Technisch einer in Reihe geschalteten Machine Learning Algorithmusfolge findet sich aber auch in anderen informatischen Anwendungen wieder und bildet daher die Grundidee für unsere Systemarchitektur.

2. Methodik

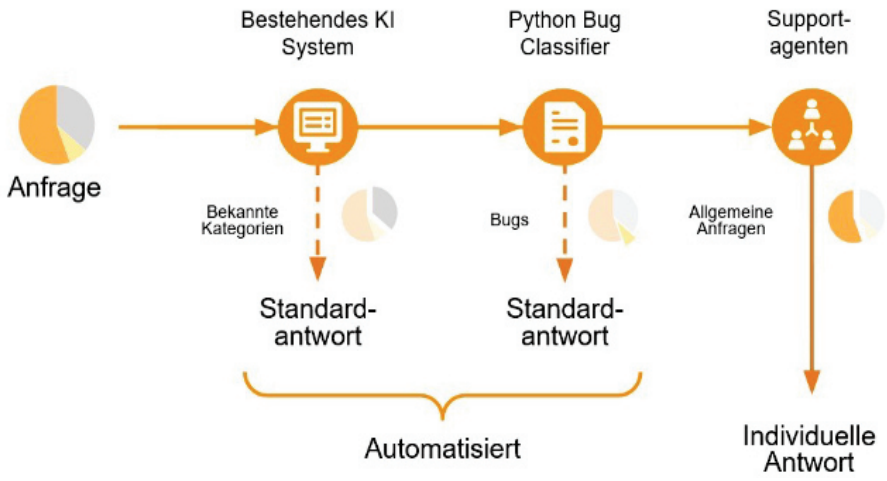

Abbildung 1: Eingliederung des Python Bug Classifiers ins Bestandssystem der Fingentia AG

Grundlage für alle Analysen ist ein Datensatz mit ca. 44.000 anonymisierten Supportanfragen. Diese enthalten neben dem Anfragetext auch Metainformationen wie Einsendedatum und eine intellektuell erstellte Kategoriserung. Der komplette Ablauf der damit stattfindenden Klassifikation ist aus Abbildung 2 ersichtlich.

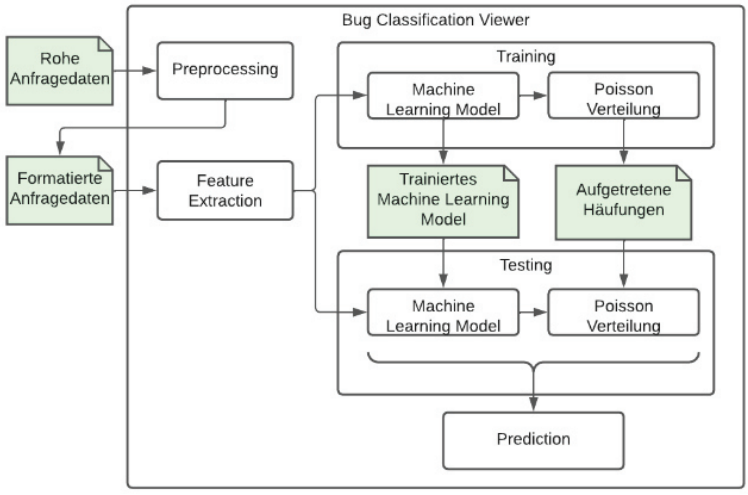

Abbildung 2: Ablauf einer Klassifikation im selbst erstellten Tool „Bug Classification Viewer“

Nach dem Preprocessing (Reduktion auf 28.000 Anfragen) findet die für ML-Prozesse übliche Feature Extraction (hier basierend auf Vorkommenshäufigkeiten mittels Tf-Idf-Vectorizer) statt. Dieser weist jedem in den Anfragetexten vorkommenden Wort eine Relevanz zu, welche später als Feature dient (Beispiel in Abb. 3). Das ML-Model lernt dabei verschiedene Muster in den Kombinationen der Wörter zur späteren Verwendung als Klassifikationsgrundlage.

	Fehler	haben	Tag	ich	Grüße	Pdf	...
Anfrage 1	0,49	0,13	0	0,2	0	0,53	
Anfrage 2	0	0	0.25	0.4	0.19	0	

Abbildung 3: Beispiele für analysierte Anfragen auf Grundlage der vorverarbeiteten Daten

Da in den Ausgangsdaten nur verhältnismäßig wenige echte bugbezogene Anfragen vorhanden sind, wurde ein Algorithmus zur Erstellung syntetischer Anfragen entwickelt, um eine ausführlichere Grundlage für das Training zu schaffen (Beispiele in Abb. 3)

Template	Ich erhalte seit dem letzten Update die Meldung dass ein Fehler in der Datenbank wäre, ich solle die App deinstallieren.
Synthetische Anfrage #1	Guten Tag! Ich erhalte seit dem letztem Aktualisierung die Meldung dass ein Defekt in der Datenlager wäre, ich solle die App deinstallieren. Viele Grüße.
Synthetische Anfrage #2	Hallo Starfinanz-Team, Ich bekomme seit dem gestrigen Aktualisierung die Nachricht dass ein Unbrauchbar in der Datenbank wäre, ich solle die App neu installieren. Vielen Dank!

Abbildung 3: Beispiele für analysierte Anfragen auf Grundlage der vorverarbeiteten Daten

Auf diese Weise wurde die Kategorie „Bug“ um 1.000 Anfragen (28%) vergrößert. Neben dem inhaltlichen Aspekt kann hierbei auch der zeitliche beliebig simuliert werden. Dafür variieren wir den Anfragezeitraum künstlich nach dem Schema einer Poisson-Verteilung mit einem Schwerpunkt auf dem Mittag und dem Nachmittag eines Tages. Die Kombination von Inhalt und Zeitlinie klassifiziert eine Anfrage als „Bug-related“.

Für das Machine Learning Modell wurde der Random-Forest-Klassifikator genutzt. Beim Testen folgen die Daten den Verzweigungen der Bäume bis zu einem Ergebnis. Um die Klassifikation realitätsnaher zu gestalten, wird der Datensatz außerdem in Trainings- und Testdaten unterteilt. Die Klassifikationsergebnisse sind in Abbildung 4 zu sehen.

Aufteilung (Test, Train)	Accuracy	Precision	Recall
50% / 50%	0,871	0,333	0,001
33% / 67%	0,872	0,667	0,001
25% / 75%	0,865	0,125	0,001
20% / 80%	0,854	0,400	0,001

Abbildung 4: Ergebnisse mehrerer Kreuzvalidierungen mit verschiedenen Train-/Test-Aufteilungen

Zur vereinfachten Nutzung aller Prozesse entstand ein Programm mit dem Namen “Bug Classification Viewer”. Es bietet dem nachbereitenden oder forschenden Wissenslogistiker bei der Arbeit mit den Daten die Möglichkeit, Training, Tests und Auswertung via grafischem Interface vorzunehmen.

3. Technische Evaluation

Um das Klassifizierungsverfahren zu testen, wurde mit einer Gruppe von Probanden eine Evaluation durchgeführt. Hierbei wurde der, für das Projekt entwickelte, Bug Classification Viewer (kurz BCV) benutzt, um den Probanden einen Einblick in den Algorithmus zu geben. Auch wenn die Probanden, nicht alle der Gruppe der Wissenslogistiker

zuzuordnen waren, ist das Feedback ähnlicher Zielgruppen für die Weiterentwicklung wichtig. Die ersten Iterationen des algorithmischen Ablaufs bestanden aus drei unterschiedlichen Schritte zur Klassifikation. Durch kontinuierliches, internes Testen der erzielten Ergebnisse wurde das Verfahren stetig verbessert. Die letzte Iteration konnte dadurch eine Treffergenauigkeit von bis zu 86,5% auf den Testdaten erreichen.

Ein Teil der Evaluation betrachtete auch die User-Experience des Classification Viewers. Eine Umfrage mit dem webbasierten Tool *AttrakDiff* ergab, dass das evaluierte Design des Werkzeugs funktional praktisch gut nutzbar, aber optisch noch zu technisch ist.

4. Zusammenfassung

Die in dieser Arbeit vorgestellte Bugerkennungssoftware besitzt die Fähigkeit, Bug-Supportanfragen von allgemeinen Anfragen mit einer hinreichenden Güte zu trennen. Das Machine Learning Modell trifft mit einer oberen Wahrscheinlichkeit von 86,5% die gewünschte Entscheidung. Der mit synthetischen Anfragen erweiterte Trainingsdatensatz hatte zwar den erhofften positiven Effekt, eine eindeutige Erkennung aller realen Anfragen mit Bezug zu Softwarefehlern ist jedoch noch nicht möglich. Eine Supportanfrage wird nur als Bug-Supportanfrage erkannt, wenn der in ihr enthaltene Anfragetext, als auch die Poisson-Verteilung dies bestätigen.

Das Programm eignet sich demnach zur groben Trennung des Datensatzes. Es filtert die sicheren Bug-Supportanfragen heraus, deren inhaltliche und zeitliche Struktur in den Trainingsdaten erfasst wurde. Jedoch muss ein Wissenslogistiker in der Praxis die Anfragen mit unbekannter Struktur und daher unsicherer Klassifikation weiterhin intellektuell und damit ressourcenaufwändig Sichten und Kategorisieren.

Quellen

Giger, E., D'Ambros, M., Pinzger, M. & Gall, H. C. (2012). Method-level bug prediction. *In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, Seite 171-180

Google Cloud. Natural Language AI. *https://cloud.google.com/natural-language*

Heaven, W. D. (31.5.2021). Diese KI lernt, indem sie sich selbst neuen Herausforderungen stellt, *https://www.heise.de/hintergrund/Diese-KI-lernt-indem-sie-sich-selbst-neuen-Herausforderungen-stellt-6056123.html*

Heesen, L., Irnich, U., Knoop, J., Dannenberg, N., Döring, S., Prohaska, E. & Orlowski, C. (2022). Künstliche Intelligenz (KI) als Motor für die Transformation im Kundenservice, *Künstliche Intelligenz erfolgreich umsetzen*, S. 181-204

Jalbert, N. & Weimer, W. (2008). Automated Duplicate Detection for Bug Tracking Systems, *In Proceedings of the International Conference on Dependable Systems & Networks*, Seite 52-61

Krukemeyer, O. (2022). Algorithmen im Kundenservice: Wie trainiert man eine KI? Fallstudie 'App Sparkasse', *https://www.it-finanzmagazin.de/algorithmus-im-kundenservice-wie-trainiert-man-eine-ki-fallstudie-sparkasse-app-144325/*

Natthakul, P, Hideaki, H. & Ken-ichi, M. (2013). Classifying Bug Reports to Bugs and Other Requests Using Topic Modeling, *In Proceedings of the 20th Asia Pacific Software Engineering Conference*, Seite 125-135

Ritter, M., Roschke, C. & Tolkmit, V. (2019). Finanzmars im Kosmos von Blended Learning. *Beiträge des CARF Luzern 2019*, pages 327–344

Sinaris, K. (21. Juli 2016). Natural Language API: Google liest auch zwischen den Zeilen, *https://entwickler.de/api/natural-language-api-google-liest-auch-zwischen-den-zeilen*

User Interface Design GmbH (1998). AttrakDiff, *https://www.attrakdiff.de/*