Exposé zur Bachelorarbeit:

Softwarequalitätskontrolle – Risikoabschätzung von Softwareanforderungen

Qiqi Chen

Juli 2021

Inhaltsverzeichnis

1 Einführung

Heutzutage wird Software von großen Entwicklerteams entwickelt. Zu der Unterstützung des Prozesses gibt es die Webanwendung Jira, deren Funktionalität Fehlerverwaltung, Problembehandlung und operatives Projektmanagement umfasst. Jede Software-anforderung bzw. Änderung von Software wird darin als Issue protokolliert. Ein Issue besteht aus mehreren strukturierten Feldern – sowohl kontrollierten Feldern als auch Freitextfeldern. Felder werden auch als Attribute bezeichnet.

Komplexe Software haben meistens nicht nur eine Komponente und sind auch schwerer als einfache Software zu verwalten. Eine Änderung im Code kann andere Änderungen nach sich ziehen. Einige Änderungen im Code können dazu führen, dass mehrere Softwarekomponenten geändert werden müssen. Darum ist es hilfreich für Software-entwickler, das Risiko einer Codeänderung im Vorfeld zu erkennen, damit die erforderlichen Tests, die ein wichtiger Schritt im Softwareengineering sind, an den Softwarekomponenten rechtzeitig implementiert werden können.

Ein Risiko ist die Erwartung eines Verlustes, ein potenzielles Problem, das in der Zukunft auftreten könnte. Die Möglichkeit, im Softwareentwicklungsprozess einen Verlust zu erleiden, wird als Software-Risiko bezeichnet. Ein Verlust kann alles Mögliche sein, z. B. ein Anstieg der Produktionskosten, die Entwicklung von Software schlechter Qualität oder die Unfähigkeit, das Projekt rechtzeitig abzuschließen. Das Software-Risiko besteht, weil die Zukunft ungewiss ist und es viele bekannte und unbekannte Dinge gibt, die nicht in den Projektplan aufgenommen werden können.

Das Risiko kann als ein Attribut mit beispielsweise drei Kontrollwerten „High“, ,,Medium“ und „Low“ zu einem Issue annotiert werden. Komplexe Software mit vielen beteiligten Entwicklern macht eine Risikoabschätzung jedoch schwierig. Um zu vermeiden, dass die Risikostufe willkürlich nach dem Bauchgefühl der Softwareentwickler festgelegt wird, braucht man bestimmte Kriterien bzw. für softwarequalitätsrelevante Metriken, um eine plausible Risikostufe für ein Issue festzulegen. Solche klaren Kriterien sind schwer zu definieren, und ihre Ausprägungen in der Praxis noch schwerer abzuschätzen. Daher soll hier ein anderer Weg gegangen werden: die automatische Klassifizierung des Risikos ist eine mögliche Technik, um Probleme bei der Änderung, deren Risiko abgeschätzt werden muss, automatisch einschätzen zu können.

2 Ziel

Ziel dieser Bachelorarbeit ist es, ein Klassifizierungsmodell zu implementieren, um die Softwareanforderungen automatisch in drei Risikostufen zu klassifizieren.

3 Methodik

Um das Ziel zu erreichen, sind folgende Schritte geplant.

Datensammlung. Die Datenquelle sind Issues eines laufenden Softwareprojekt. 581 von insgesamt 16763 Issues sind bereits von den Entwicklern manuell mit 5 Risikostufen (Very high, High, Medium, Low, Very low) gekennzeichnet. Sie werden aus Jira exportiert.

Datenreinigung. Die Issues in Jira haben mehrere Attribute, welche in einer Tabelle als verschiedene Spalten dargestellt werden können. Alle leeren Spalten werden gelöscht. Zusammen mit den Entwicklern des Projekts wird entschieden, welchen Attribute relevant für die Risikoklassifizierung sind und welche bei der Erstellung eines Issues schon eingetragen werden. Diese werden letztendlich beibehalten. Da das Projekt eine internationale Zusammenarbeit ist, werden die Issues teilweise auf Deutsch und Englisch geschrieben. Die Sprache der gelabelter Issues wird mithilfe der Python Bibliothek langdetect erkannt, 496 Issues sind Deutsch. Deswegen werden in dieser Arbeit nur deutsche Issues berücksichtigt. Außerdem werden die Daten mit Risikolabel „Very high“ und „Very low“ jeweils zu „High“ und „Low“ geändert, da nur sehr wenige Issues mit „Very high“ und „Very low“ gekennzeichnet werden.

Feature-Engineering. Da die Anzahl der Issues mit Risikolabel sehr gering ist, sollen die restlichen Issues auch mit Risikolabel nachgetragen werden. Hierzu werden einige für das Risiko relevante DevOps Metriken durch die Attribute definiert und für jedes Issue berechnet. Mit diesen wird ein Risikolabel auf die Issues projiziert. Denn bei abgeschlossenen Issues kann man z. B. sehen, wie viele Fehler (Defects) bei der Implementierung gamacht wurden, aber bei neuen Issues muss man das vorhersagen und schätzen. Und durch DevOps Metriken kann man das Risiko eines abgeschlossenen Issues retrospektiv nachtragen.

Darüber hinaus wird Software-spezifische NER (Named-entity Recognition) [1],[2] auf die Freitextfelder, wie zum Beispiel „Zusammenfassung“ und „Beschreibung“ angewendet, um weitere für das Risiko wichtige Informationen aus dem Text zu extrahieren.

Da die Werte der Attribute keine Zahlen sind, sondern Texte, wie zum Beispiel Name, Nomen, Issue ID usw., können gängige Klassifikationsalgorithmen die Daten nicht direkt bearbeiten. Deswegen muss man die Werte codieren. Es gibt sowohl ordinale Daten als auch nominale Daten [12]. Für ordinale Daten wird das Label Encoding verwendet, während für nominale Daten das One-hot Encoding eingesetzt wird [13]. Beide Codierungstechniken sind die einfachsten, was allerdings auch Nachteile haben. Es gibt noch mehr Codierungstechniken, die jeweils für verschiedene Daten geeigneter sind. Dazu wird experimentiert, welche Codierungstechniken unseren Daten am besten passen.

Experimente. Wir werden die Daten auf zwei verschiedener Weise behandeln.

Auf einem Weg trainieren wir verschiedene maschinelles Lernverfahren wie zum Beispiel Multi-Layer Perceptron, logistische Regression oder Random Forests und evaluieren diese Methoden mit Train- und Testdaten, die mit verschiedenen Codierungstechniken codiert werden, um das beste Modell zu finden. Eingaben (x) sind die codierten Daten, Ausgaben (y) sind die 3 Klassen-Risikostufen „High“, „Medium“ und „Low“. Multi-Layer Perceptron kombiniert mit One-hot Encoding wird die Baseline liefern.

Auf dem anderen Weg trainieren wir die nicht codierten Daten als Textdaten mit Sprachmodellen zur Textklassifikation. Die Ausgaben sind hier auch die drei Risikostufen. Dazu wird Bert [14] die Baseline liefern. Zusätzlich werden wir noch das State-of-the-Art Modell BIGBIRD [8] anwenden.

Am Ende wollen wir vergleichen, welches Modell zum besten Ergebnis führt.

Evaluierung. Da die Daten unausgewogen sind, werden sie durch Geschichtete Zufallsstichprobe schichtweise zufällig in 80% Training Set und 20% Testset geteilt. Und wir führen eine fünffache Kreuzvalidierung durch. Für die Evaluierung, und zum Vergleich mit ähnlichen Forschungen, werden Accuracy, Precision, Recall und F1-Score berechnet.

4 Verwandte Arbeiten

Wegen der Digitalisierung ist die Datenmenge im Unternehmen immer größer geworden. Daher sind traditionelle Datenanalyse-Methoden immer teurer und schwerer einsetzbar. Aus diesem Grund wurden KI-Methoden in den letzten Jahren immer mehr in Softwareentwicklungsbereichen verwendet. Moreno et al. haben ein ähnliches Verfahren für die Softwareanforderungsqualität verwendet [10]. Sie haben 18 quantitative Metriken mit RQA Tool berechnet, und haben dann 1035 textuelle Softwareanforderungen mit diesen Metriken in einer binären Klassifikation realisiert, indem sie regelbasierte Klassifikatoren mit Entscheidungsbäumen bauten. Auch Gramajo et al. haben 4 binäre Klassifikationsmodelle mit LSTM und GRU für 1000 textuelle Softwareanforderungen vom PURE (PUblic REquirements dataset) implementiert [11].

Named Entity Recognition (NER) wird hauptsächlich durch neuronale Netze realisiert. In den letzten Jahrzehnten entwickelten sich die Modelle immer weiter. Im Jahr 2008 wurde das Convolutional Neuronales Netz erstmals für NER eingesetzt [3]. Danach sind Recurrent Neural Network, Long Short Term Memory (LSTM) und Gated Recurrent Unit (GRUs) zum Einsatz gekommen. RNN-Varianten mit Gating-Mechanismus, die in der NER am häufigsten verwendet werden, sind zurzeit Long Short Term Memory und Gated Recurrent Units [4]. Das Neue am LSTM [5] ist, dass es in der Lage ist, lange Zeitintervalle

zu überbrücken (z. B. Langzeitgedächtnis) und gleichzeitig die letzten Eingaben zu bewahren (z. B. Kurzzeitgedächtnis). Darüber hinaus gewährleistet die LSTM-Architektur eine konstante Neugewichtung, wodurch die Explosion oder das Verschwinden des Fehlers durch die versteckten Zustände vermieden wird.

Da die meisten maschinellen Lernmodelle für numerische Daten entworfen werden, spielt Feature Engineering eine wichtige Rolle für die industrielle Daten, welche meistens nicht nur aus Zahlen bestehen. Im Feature Engineering muss man die textuellen Daten codieren, damit die ML-Algorithmen einsetzbar sind. Eine Deep-Learned Einbettungstechnik für die Codierung kategorialer Features wird in [12] beschrieben. Es wurde mit 3 Datenbeständen mit über 10.000 bis 50.000 Samples experimentiert. Im Vergleich zu anderen Encodingtechnik ergibt sie die höchste Genauigkeit und ist sowohl für nominale Daten als auch für ordinale Daten geeignet. Sie generiert weniger neue Features und braucht weniger Speicherplatz als das One-hot Encoding. Auch in diesem Jahr haben Pargent et al. ein groß angelegtes Benchmark-Experiment durchgeführt, um die Auswirkungen verschiedener kategorialer Codierungstechniken, unabhängig vom nachfolgenden maschinellen Lernmodell, auf Features mit hoher Kardinalität zu bewerten [15]. Ihre Ergebnisse zeigen, dass die regularisierte Zielcodierung in allen Datensätzen überlegen oder zumindest wettbewerbsfähig ist. Besonders effektiv war das glmm-Encoding, welches bei allen ML-Algorithmen außer SVM den ersten Platz belegte.

Bisher hat Semi-Supervised Learning mit Adversarial Attack [6] den größten Erfolg in der Text-Klassifikation erlangt. Beim adversen Training wird ein Modell so trainiert, dass es sowohl unveränderte Beispiele als auch adverse Beispiele korrekt klassifiziert. Sie verwendeten ein einfaches LSTM Modell und eine bidirektionale LSTM Architektur [7] auf fünf verschiedene Datensätze und lieferten ein besseres Ergebnis als ihre Basislinie. Eine andere Methode ist die Verwendung von Transformer. BIGBIRD [8] ist eine Entwicklung der Standardmodelle wie BERT und RoBERTa, indem es statt dem Mechanismus der vollen Aufmerksamkeit den Mechanismus der spärlichen Aufmerksamkeit benutzt, sodass es besonders gut für Lang Text geeignet ist.

5 Referenzen

[1] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li and N. Kapre, "Software-Specific Named Entity Recognition in Software Engineering Social Content," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2016, pp. 90-101, doi: 10.1109/SANER.2016.10.

[2] Tabassum J, Maddela M, Xu W, et al. Code and named entity recognition in stackoverflow[J]. arXiv preprint arXiv:2005.01634, 2020.

[3] Collobert, Ronan and J. Weston. “A unified architecture for natural language processing: deep neural networks with multitask learning.” ICML '08 (2008).

[4] Roy A. Recent Trends in Named Entity Recognition (NER)[J]. arXiv preprint arXiv:2101.11420, 2021.

[5] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[6] Miyato T, Dai A M, Goodfellow I. Adversarial training methods for semi-supervised text classification[J]. arXiv preprint arXiv:1605.07725, 2016.

[7] Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures[J]. Neural networks, 2005, 18(5-6): 602-610.

[8] Zaheer M, Guruganesh G, Dubey K A, et al. Big Bird: Transformers for Longer Sequences[C]//NeurIPS. 2020.

[9] Aschenbruck R, Szepannek G. Einsatz von KI zur Qualitätssicherung[J]. Digitale Daten, 47.

[10] Moreno V, Génova G, Parra E, et al. Application of machine learning techniques to the flexible assessment and improvement of requirements quality[J]. Software Quality Journal, 2020, 28(4): 1645-1674.

[11] Gramajo M G, Ballejos L C, Ale M. Hacia la Evaluación Automática de la Calidad de los Requerimientos de Software usando Redes Neuronales Long Short Term Memory[C]//WER. 2020.

[12] M. K. Dahouda and I. Joe, "A Deep-Learned Embedding Technique for Categorical Features Encoding," in IEEE Access, vol. 9, pp. 114381-114391, 2021, doi: 10.1109/ACCESS.2021.3104357.

[13] A. Sethi. One-Hot Encoding vs. Label Encoding Using Scikit-Learn. Accessed: 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/

[14] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[15] Pargent F, Pfisterer F, Thomas J, et al. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features[J]. arXiv preprint arXiv:2104.00629, 2021.