Universität Leipzig
Fakultät für Mathematik und Informatik
Wirtschaftswissenschaftliche Fakultät

Ermittlung von Konfigurationsoptionen im Source Code mit Fokus auf Machine Learning Bibliotheken in Python

Bachelorarbeit

Marco Jaeger-Kufel
geb. am: 10.05.1995 in Hannover

Matrikelnummer 3731679

1. Gutachter: Prof. Dr. Norbert Siegmund
2. Gutachter: Prof. Dr. Unknown Yet

Datum der Abgabe: 14. Juni 2022

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Leipzig, 14. Juni 2022

.....
Marco Jaeger-Kufel

Zusammenfassung

A short summary.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit entsteht am Institut für Informatik an der Universität Leipzig in der Abteilung „Softwaresysteme“. Im Folgenden wird die zugrundeliegende Problemstellung und Relevanz erläutert. Anschließend wird dieses Problem auf einen konkreten Anwendungsbereich übertragen und auf die Zielsetzung der Arbeit eingegangen.

1.1 Motivation

Moderne Softwaresysteme ermöglichen den Nutzenden ein breites Spektrum unterschiedlicher Konfigurationsoptionen. Anhand dieser Konfigurationsoptionen sind die Nutzenden in der Lage, viele Aspekte der Ausgestaltung einer Software zu steuern. Konfigurationsoptionen können dabei ganz unterschiedliche Funktionen besitzen, die von den Nutzenden nach den eigenen Bedürfnissen angepasst werden können.

Konfigurationsoptionen können in verschiedenen Teilen eines Softwareprojekts verarbeitet, definiert und beschrieben werden: in der Konfigurationsdatei, im Source Code und in der Dokumentation (Dong et al., 2016, S. 185). Sie werden meist als Key-Value Pair entworfen und gesammelt in einer Konfigurationsdatei gespeichert. Dem Namen der Konfigurationsoption (Key) werden dabei Einstellungsmöglichkeiten beliebigen Typs zugeordnet (Value). Zur Speicherung von Konfigurationsoptionen verwenden einige Systeme, wie das Big Data-Framework Hadoop, strukturierte XML-Formate oder auch JSON-Dateien. Ein einheitliches Schema zur Speicherung als Konfigurationsdatei gibt es jedoch nicht, weshalb sich diese von der Struktur und Syntax unterscheiden. (Rabkin and Katz, 2011, S. 131). Im Source Code können die Konfigurationsoptionen eingelesen und bearbeitet werden (Dong et al., 2016, S. 185).

Während die Vielfältigkeit und Individualisierbarkeit der Software größer wird, erhöht sich auch die Komplexität der Software für Entwickelnde. Vor allem das Warten und Testen gestaltet sich nun umfangreicher. Besonders problematisch wird es, wenn Konfigurationsoptionen auf unvorhergesehene Weise interagieren. Solche Abhängigkeiten sind in einem wachsenden Spielraum möglicher Kombinationen schwer zu entdecken und zu verstehen. Entwickelnde müssen die Konfigurationsoptionen innerhalb der Software verfolgen, um festzustellen, welche Codefragmente von einer Option betroffen sind und wo und wie sie mit ihr interagieren. Des Weiteren kann es vorkommen, dass Entwickelnde Werte von Konfigurationsoptionen nicht aktualisieren, was dazu führen kann, dass über mehrere Module hinweg, unbemerkt mit falschen Werten gearbeitet wird (Dong et al., 2016, S. 185). Die verschiedenen möglichen Ausprägungen einer Konfigurationsoption erschweren hierbei die Nachvollziehbarkeit des Kontrollflusses der Software.

In einer empirischen Studie über Konfigurationsfehler stellen Yin et al. (2011, S. 160) fest, dass in kommerziellen Softwareunternehmen für Speicherlösungen, knapp ein Drittel aller Ursachen von Kundenproblemen auf Konfigurationsfehler zurückzuführen sind (siehe Tabelle 3). Bei Konfigurationsfehler sind Source Code und die Eingabe zwar korrekt, es wird jedoch ein falscher Wert für eine Konfigurationsoption verwendet, sodass sich die Software nicht wie gewünscht verhält (Zhang and Ernst, 2014, S. 152). Solche Fehler können dazu führen, dass die Software abstürzt, eine fehlerhafte Ausgabe erzeugt oder nur unzureichend funktioniert (Zhang and Ernst, 2014, S. 152).

Konfigurationsfehler entstehen zum Beispiel durch die Verwendung unterschiedlicher Versionen einer Software. Im folgendem Codeausschnitt wird die scikit-learn Klasse *LogisticRegression* initialisiert und der Variable *clf* zugewiesen:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
```

Da keine Parameter angegeben werden, wird die Klasse mit den Default-Werten initialisiert, zum Beispiel *max_iter = 100, verbose = 0, n_jobs = 1*. In der Version *scikit-learn 0.21.3* wird für den Parameter *multi_class* als Default noch der Wert *ovr* zugewiesen (2022). Für alle nachfolgenden Versionen ist der Default-Wert für diesen Parameter jedoch *auto* (2022). Folglich führt die Verwendung dieser Klasse ohne explizite Parameterangabe, nach Verwendung

unterschiedlicher Versionen, zu schwer nachvollziehbaren Programmverhalten.

1.2 Anwendungsbereich und Zielsetzung

Das Ziel dieser Arbeit ist es, Konfigurationsoptionen im Source Code zu erkennen und zu extrahieren.

Es existieren bereits einige Forschungsansätze zur Ermittlung und Verarbeitung von Konfigurationsoptionen im Source Code. Viel zitiert wird dabei der Ansatz von Rabkin and Katz, den sie 2011 publizierten. Wie auch Dong et al. oder Lillack et al. entwickelten sie einen Ansatz, mittels statischer Code-Analyse Konfigurationsoptionen im Source Code zu tracken. Dabei fokussierten sie sich auf die Programmiersprache Java, die nach dem TIOBE-Index jahrelang als beliebteste Programmiersprache galt (2022). Der TIOBE-Index misst die Popularität von Programmiersprachen auf der Grundlage von Suchanfragen auf beliebten Websites und in Suchmaschinen (2022). Im Februar 2022 ist in diesem Ranking Python erstmals zur beliebtesten Programmiersprache aufgestiegen (2022). Für Python ist diese Thematik jedoch bislang allerdings noch wenig beleuchtet.

Die Programmiersprache Python ist für ihre Benutzerfreundlichkeit bekannt und obwohl Python eine interpretierte High-Level-Programmiersprache ist, ist sie in der Lage, bei Bedarf die Leistung von Programmiersprachen auf Systemebene zu nutzen (Raschka et al., 2020, S. 2). Insbesondere im Bereich wissenschaftliches Rechnen (Scientific Computing) gewann Python in den letzten Jahren enorm an Popularität, weshalb viele Bibliotheken für maschinelles Lernen auf Python basieren (Raschka et al., 2020, S. 2).

Gegenstand dieser Arbeit werden daher Konfigurationsoptionen sein, die in Python Source Code vorliegen und mittels statischer Code-Analyse erkannt werden. Der Ansatz identifiziert automatisch die Stellen im Source Code, an denen die Optionen gelsesen und ermittelt für jede dieser Stellen den Namen, sowie die übergebenen Werte der Option. Dabei liegt der Fokus auf drei der populärsten Machine Learning Bibliotheken in dieser Sprache (siehe stat): TensorFlow, PyTorch und scikit-learn. Zudem wird die Verwendung von Konfigurationsoptionen der Machine Learning Lifecylce Plattform Mlflow untersucht.

1.3 Aufbau dieser Arbeit und methodisches Vorgehen

Kapitel 2

Hintergrund

2.1 Statische Code-Analyse

Das wichtigste Werkzeug dieser Arbeit ist die statische Code-Analyse, mit der Software-Projekte unabhängig von der Ausführungsumgebung untersucht werden können. Die statische Code-Analyse ist ein Werkzeug, um Fehler in einer Softwareanwendung zu reduzieren (Bardas et al., 2010, S. 99). So ermöglichen sie den Anwendenden, Fehler in einem Programm zu finden, die für den Compiler nicht sichtbar sind (Bardas et al., 2010, S. 99).

Im Gegensatz zur dynamischen Analyse wird die statische Analyse zur Übersetzungszeit durchgeführt und setzt damit bereits vor der tatsächlichen Ausführung des Source Codes an (Gomes et al., 2009, S. 2). Die erzeugten Ergebnisse der statischen Analyse lassen sich besser Verallgemeinern, da sie nicht abhängig von den Eingaben sind, mit denen das Programm während der dynamischen Analyse ausgeführt wurde (Gomes et al., 2009, S. 6).

Für das Aufspüren von Konfigurationsoptionen bietet sich eine statische Code-Analyse daher aus mehreren Gründen an. So kann es viele Optionen geben, die nur in bestimmten Modulen oder als Folge bestimmter Eingaben verwendet werden. Es ist unwahrscheinlich, dass mittels dynamischen Testens alle Verwendungen der gesuchten Konfigurationsoptionen gefunden werden. Dies würde zudem bei größeren Softwareprojekten eine sehr komplexe Test-Suite erfordern, um möglichst alle Fälle abdecken zu können. Mit der statischen Code-Analyse kann eine hohe Abdeckung hingegen deutlich leichter erreicht werden. Gleichzeitig verbergen sich hinter den Methoden und Klassen, der zu untersuchenden Machine Learning Bibliotheken, teils sehr komplexe und rechenintensive Berechnungen, die bis zu mehrere Tage laufen könnten. Aus Kosten-Nutzen-Gründen ist hier eine dynamische Analyse nur bedingt sinnvoll.

2.2 Maschinelles Lernen

Die künstliche Intelligenz (KI) ist ein Teilgebiet der Informatik und befasst sich mit der Entwicklung von Computerprogrammen und Maschinen, die in der Lage sind, Aufgaben auszuführen, die Menschen von Natur aus gut beherrschen (Raschka et al., 2020, S. 1). Dazu gehören zum Beispiel die Verarbeitung von natürlicher Sprache (Natural Language Processing) oder Bilderkennung (Computer Vision). In der Mitte des 20. Jahrhunderts entstand das maschinelle Lernen (ML) als Teilbereich der KI und schlug eine neue Richtung für die Entwicklung von künstlicher Intelligenz ein, inspiriert vom konzeptionellen Verständnis der Funktionsweise des menschlichen Gehirns (Raschka et al., 2020, S. 1).

Pionier Arthur Samuel definiert maschinelles Lernen als ein Fachgebiet, das Computern die Fähigkeit verleiht, zu lernen, ohne ausdrücklich programmiert zu werden (Mahesh, 2020, S. 381). Es befasst sich mit der wissenschaftlichen Untersuchung von Algorithmen und statistischen Modellen, die Computersysteme verwenden, um eine Aufgabe zu lösen (Mahesh, 2020, S. 381). Dabei werden statistische Methoden verwendet, um aus Daten zu lernen und Muster zu erkennen.

2.2.1 Ursprung

Der Ursprung des maschinellen Lernens liegt bereits in der Mitte des 20. Jahrhunderts, als der Psychologoe Frank Rosenblatt von seiner Arbeitsgruppe eine Maschine zur Erkennung von Buchstaben des Alphabets bauen ließ (Fradkov, 2020, S. 1385). Das Konzept der Maschine basiert auf der Verarbeitung von Signalen analog zum menschliche Nervensystem, weshalb sie als Prototyp der modernen künstlichen neuronalen Netze gilt (Fradkov, 2020, S. 1385). Der kommerzielle Durchbruch ließ dennoch bis auf den Anfang 21. Jahrhundert auf sich warten. Nach Fradkov ist dies auf drei Trends zurückzuführen, die zusammen einen spürbaren Synergieeffekt bewirkten.

Unter anderem durch den Erfolg und Möglichkeiten des Internets ist nicht nur das Vorhandensein von Daten enorm gewachsen, auch ihre Heterogenität führt dazu, dass herkömmliche Methode zur Verarbeitung und Informationsgewinnung nicht mehr ausreichen (Fradkov, 2020, S. 1387). Durch das Aufkommen von *Big Data* werden neue Ansätze des maschinellen Lernens nicht nur

aus dem Motiv des wissenschaftlichen Erkenntnisgewinns entwickelt, sondern auch aufgrund der praktischen und kommerziellen Notwendigkeit.

Ein weiterer entscheidender Faktor ist die Senkung der Kosten für parallele Berechnung und Speicher. Dies umfasst die Weiterentwicklung und Verwendung von Grafikprozessoren (vor allem von Nvidia), die zu erheblichen Verbesserungen der Rechenleistung führen (Fradkov, 2020, S. 1387). Gleichzeitig sinken die Kosten für Arbeitsspeicher stetig, was die Verarbeitung großer Datenmengen zusätzlich erleichtert (Fradkov, 2020, S. 1387). Aus Softwaresicht kommt schließlich die MapReduce-Technologie von Google bzw. später auch Hadoop dazu, die es ermöglicht, die komplexe Berechnungen auf mehrere Prozessoren zu verteilen(Fradkov, 2020, S. 1387).

Der dritte Trend ist die Entwicklung künstlicher neuronaler Netze, die um ein Vielzahl an Zwischenschichten erweitert wurden und so noch komplexere Berechnungen ausführen können. In diesem Zusammenhang spricht man auch von *Deep Learning*.

2.2.2 Algorithmische Ansätze

Im Allgemeinen werden die Ansätze des maschinellen Lernens in drei große Kategorien unterteilt, je nach Art des *Signals* oder *Feedbacks*, das dem lernenden System zur Verfügung steht (Fradkov, 2020, S. 2):

- Überwachtes Lernen (supervised learning)
- Unüberwachtes Lernen (unsupervised learning)
- Bestärkendes Lernen (reinforcement learning)

Das *überwachte Lernen* erfordert ein Training mit gelabelten Daten, die Eingaben und gewünschte Ausgaben haben (Qiu et al., 2016, S. 2). So weiß das Modell zum Beispiel, ob auf einem bestimmten Foto ein Hund abgebildet ist oder wie viel ein bestimmtes Haus kostet. Auf dieser Grundlage kann es dann so trainiert werden, dass es neue Hunde erkennt und den Preis für neue Häuser schätzt.

Im Gegensatz dazu erfordert das *unüberwachte Lernen* keine markierten Trainingsdaten und erhält lediglich Eingabedaten (Qiu et al., 2016, S. 2). Die richtige Antwort ist entweder nicht bekannt oder existiert nicht. Stattdessen sucht das Modell nach sinnvollen Strukturen in den Daten, um neue Erkenntnisse über ein Thema zu gewinnen (Mahesh, 2020, S. 383). Unüberwachtes

Lernen kann zum Beispiel dazu verwendet werden, Kunden zu finden, die einen ähnlichen Geschmack haben, und ihnen Artikel empfehlen, die diese Kunden gekauft haben.

Das *bestärkende Lernen* hingegen unterscheidet sich sehr deutlich von den beiden vorherigen Kategorien. Das Modell ist hier ein aktiver Akteur, der mit seiner externen Umgebung interagiert und positives oder negatives Feedback für seine Handlungen erhält (Qiu et al., 2016, S. 2). Aus diesen Rückmeldungen erlernt es optimale Handlungsstrategien für seine Umgebung zu entwickeln (Mahesh, 2020, S. 384). Solche Modelle können zum Beipiel für selbstfahrende Autos verwendet werden oder auch um einer Maschine Gesellschaftsspiele beizubringen.

2.2.3 Python-Bibliotheken

2.2.4 Konfigurationsoptionen

2.3 Web Scraping

2.4 Abstract Syntax Trees

Anhang A

My First Appendix

This was just missing.

Literaturverzeichnis

Alexandru G Bardas et al. Static code analysis. *Journal of Information Systems & Operations Management*, 4(2):99–107, 2010. 2.1

TIOBE Software BV. Tiobe index for march 2022. `https://www.tiobe.com/tiobe-index/`, 2022. Online; accessed 22-March-2022. 1.2

Zhen Dong, Artur Andrzejak, David Lo, and Diego Costa. Orplocator: Identifying read points of configuration options via static analysis. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 185–195, 2016. doi: 10.1109/ISSRE.2016.37. 1.1, 1.2

Alexander L. Fradkov. Early history of machine learning. *IFAC-PapersOnLine*, 53(2):1385–1390, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.1888. URL `https://www.sciencedirect.com/science/article/pii/S2405896320325027`. 21st IFAC World Congress. 2.2.1, 2.2.2

Ivo Gomes, Pedro Morgado, Tiago Gomes, and Rodrigo Moreira. An overview on the static code analysis approach in software development. *Faculdade de Engenharia da Universidade do Porto, Portugal*, 2009. 2.1

Max Lillack, Christian Kästner, and Eric Bodden. Tracking load-time configuration options. *IEEE Transactions on Software Engineering*, 44(12): 1269–1291, 2018. doi: 10.1109/TSE.2017.2756048. 1.2

Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020. 2.2, 2.2.2

Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, 2016. doi: 10.1186/s13634-016-0355-x. URL `https://doi.org/10.1186/s13634-016-0355-x`. 2.2.2

Ariel Rabkin and Randy Katz. Static extraction of program configuration options. In *Proceedings of the 33rd International Conference on Softwa-*

re Engineering, ICSE '11, pages 131–140, New York, NY, USA, 2011. Association for Computing Machinery. doi: 10.1145/1985793.1985812. URL https://doi.org/10.1145/1985793.1985812. 1.1, 1.2

Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, Apr 2020. ISSN 2078-2489. doi: 10.3390/info11040193. URL http://dx.doi.org/10.3390/info11040193. 1.2, 2.2

scikit-learn developers. scikit-learn api. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression, 2022. Online; accessed 25-March-2022. 1.1

Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, page 159–172, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450309776. doi: 10.1145/2043556.2043572. URL https://doi.org/10.1145/2043556.2043572. 1.1

Sai Zhang and Michael D. Ernst. Which configuration option should i change? In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 152–163, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327565. doi: 10.1145/2568225.2568251. URL https://doi.org/10.1145/2568225.2568251. 1.1