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

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

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 Entwickler und Entwicklerinnen. Vor allem das Warten und Testen der Konfigurationsmöglichkeiten 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. Entwickler und Entwicklerinnen 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 Entwickler und Entwicklerinnen 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 Klasse *LogisticRegression* der Machine Learning-Bibliothek *scikit-learn* 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.

Bei den Parametern eines solchen Machine Learning Algorithmus, die vor Beginn des Lernprozesses vom Nutzenden festgelegt werden können, spricht man dabei von sogenannten *Hyperparametern*. Sie werden als Parameter an die jeweilige Klasse übergeben und bieten den Nutzenden Konfigurationsmög-

lichkeiten für das Training des Modells. In dem obigen Fall können so beispielsweise über den Parameter *max_iter* die Anzahl der Trainingsiterationen der Logistischen Regression festgelegt werden und über *n_jobs* die maximale Anzahl der parallel zu verwendenden CPU-Kernen. Durch das Extrahieren dieser Konfigurationswerte, können die Parameter statisch geprüft werden, was für den Nutzenden eine Zeitersparnis bedeutet, wenn sonst ein falscher Konfigurationswert einen langen Batch-Job zum Scheitern bringt oder das Ergebnis nicht den Erwartungen entspricht.

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 Benutzendenfreundlichkeit 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 der zu untersuchenden Bibliotheken gelesen werden und

ermittelt für jede dieser Stellen den Namen der Option. Darauf aufbauend erfolgt eine Datenflussanalyse, um auch für Optionen, die in Form von Variablen als Parameter übergeben werden, die möglichen Werte zu ermitteln. Der Fokus auf drei der populärsten Machine Learning Bibliotheken in Python (siehe Abbildung 2.2):

- TensorFlow
- PyTorch
- 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. 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 Datenflussanalyse

Die Datenflussanalyse ist ein Werkzeug, um Informationen über die möglichen Werte, die an verschiedenen Stellen in einem Codesegment berechnet werden, zu erfassen (Pollock and Soffa, 1989, S. 1537). Es handelt sich um eine statische Analysetechnik mit dem Ziel, das Programmverhalten schon zur Übersetzungszeit, also bevor es ausgeführt wurde, zu bestimmen. Der Datenfluss kann mittels eines Kontrollflussgraphens dargestellt werden und dem Betrachtenden Rückschlüsse über das Verhalten des Programms geben. Der Graph zeigt an, an welchen Stellen eine Variable verwendet wird und welche Werte sie annehmen kann.

Es gibt verschiedene Techniken, die innerhalb der Datenflussanalyse eingesetzt werden, um den Wert von Variablen zu bestimmen. Eine von ihnen ist *Constant Propagation*. Das Ziel von Constant Propagation ist zu bestimmen, an welchen Stellen im Programm, eine Variable einen konstanten Wert besitzt. So kann zum Beispiel toter Code, also redundanter Code, der im weiteren Programmverlauf nicht weiter verarbeitet wird, gefunden werden.

Eine weitere Technik ist *Static Single Assignment*. Bei dieser Methodik werden die Variablen im Verlauf des Übersetzungsprozesses in eine Zwischenform überführt, in der jede Variable genau einmal zugewiesen wird. Die Variablen werden in Versionen aufgeteilt und in der Regel mit einem aufsteigendem Index versehen, sodass jede Definition ihre eigene Version erhält.

Die beide Techniken Static Single Assignment und Constant Propagation werden in dem statischen Analyse-Framework *Scalpel* kombiniert. Aus dem folgenden Codebeispiel geht hervor, dass die Variable a zwei unterschiedliche Werte annehmen kann.

```
c = 10
a = -1
if c > 0:
    a = a + 1
else:
    a = 0
total = c + a
```

Der daraus resultierende Kontrollflussgraph besteht charakteristisch aus Knoten, die die jeweilige Code-Objekte beinhalten und gerichtete Kanten als Übergang zwischen den Knoten, die den Programmablauf darstellen. Mittels

Constant Propagation werden die tatsächlichen Werte der Variablen an den jeweiligen Verwendungszeitpunkten erkannt und über die Φ-Funktion kann durch Static Single Assignment abgeleitet werden, dass es zwei mögliche Rückgabewerte gibt.

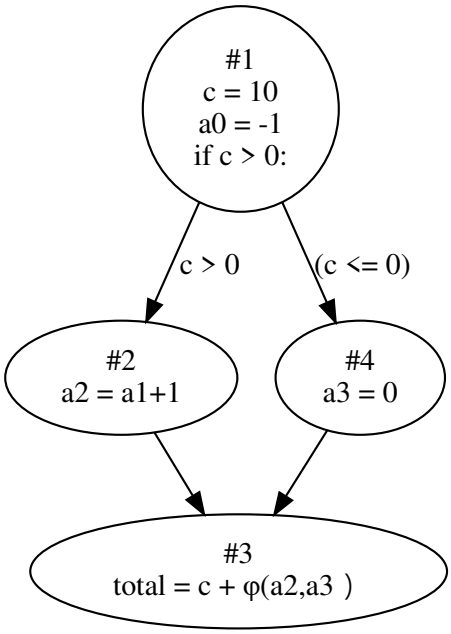

Abbildung 2.1: Kontrollflussgraph (Li et al., 2022)

2.3 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.

2.3.1 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 des 21. Jahrhunderts auf sich warten. Nach Fradkov ist dies auf drei Trends zurückzuführen, die zusammen einen spürbaren Synergieeffekt bewirkten (Fradkov, 2020, S. 1387).

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, die rasant wachsenden generierten Datenmengen leistungsfähig verarbeiten zu können.

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 hinzu, die es ermöglicht, 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*.

2.3.2 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 oder den Preis neuer 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, in einem Online-Store Kunden zu finden, die einen ähnlichen Geschmack haben, und 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, das 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 Beispiel für selbstfahrende Autos verwendet werden oder um einer Maschine Gesellschaftsspiele beizubringen.

2.3.3 Python-Bibliotheken

Fürs maschinelle Lernen gilt Python schon seit langem als erste Wahl für Entwickler und Entwicklerinnen. Eine im Mai 2019 veröffentlichte Umfrage vom Portal KDnuggets ergab, dass Python in der Kategorie „Top Analytics, Data Science, Machine Learning Tools“ von rund 66% der Teilnehmenden verwendet wird und damit die populärste Programmiersprache in diesem Bereich ist KDnuggets (2019).

Python ist eine interpretierte Programmiersprache. Demnach wird während der Ausführung der Python-Code zur Laufzeit interpretiert, wodurch sie im Vergleich mit kompilierten Programmiersprachen wie C / C++ hinsichtlich Leistung und Geschwindigkeit schlechter abschneidet. Ein wichtiger Vorteil von Python ist jedoch die Möglichkeit, Code aus anderen Programmiersprachen relativ einfach einzubinden (Stančin and Jović, 2019, S.977). Viele Lösungen im maschinellen Lernen basieren auf numerischen und vektorisierten Berechnungen mit Bibliotheken wie *NumPy* oder *SciPy* (Stančin and Jović, 2019, S.977). Um diese Berechnungen schnell und effizient auszuführen, werden sogenannte *Wrapper* verwendet, die Algorithmen von kompilierten Programmiersprachen implementieren (Stančin and Jović, 2019, S.977). Die wahrscheinlich am häufigsten verwendete Bibliothek für diesen Zweck ist Cython, die zwar auf Python basiert, aber auch den Aufruf von Funktionen, sowie die Verwendung von Variablen und Klassen aus der Programmiersprache C unterstützt (Stančin and Jović, 2019, S.977). Dadurch können kritische Teile des Codes um ein Vielfaches beschleunigt werden.

Einer der Hauptgründe für die Popularität von Python ist das riesige Ökosystem, das aus einer Vielzahl von umfangreichen und leistungsfähigen Bibliotheken besteht, über die eben angesprochene Algorithmen aufgerufen werden können, wodurch die Benutzendenfreundlichkeit bei gleichzeitiger Effizienz in der Performanz gewahrt bleiben kann (Raschka et al., 2020, S. 2). So sind nach einer jährlich von der Data Science-Plattform Kaggle durchgeführten Umfrage unter rund 25.000 ML-Engineers und Data Scientists, die Python Bibliotheken mit großem Abstand die am meisten genutzten Frameworks für maschinelles Lernen (Kaggle, 2021).

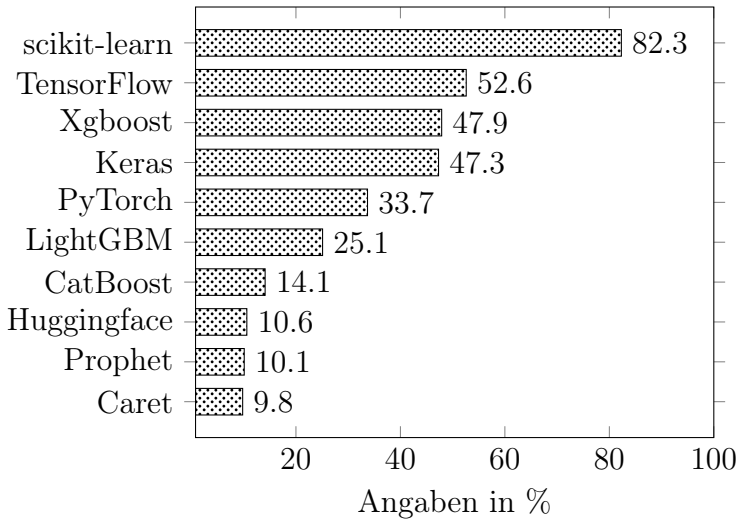

Abbildung 2.2: Nutzung von ML-Frameworks (Kaggle, 2021)

Aus der Grafik lassen sich drei der zu untersuchenden Python Bibliotheken entnehmen. Zum einen *scikit-learn*, das aufgrund der Vielzahl an implementierten Algorithmen wie ein „schweizer Taschenmesser“ für die meisten Projekte eingesetzt werden kann und von über 80% der Befragten verwendet wird (Kaggle, 2021). Die Vorteile von scikit-learn ist die benutzerfreundliche Struktur und Dokumentation mit der maschinelles Lernen auch für unerfahrene oder fachfremde Menschen zugänglich gemacht wird (Varoquaux et al., 2015, S.29).

Bei der zweiten zu untersuchenden Bibliothek handelt es sich um *TensorFlow*. Nachdem es 2015 von Forschern bei Google ursprünglich für interne Zwecke entwickelt wurde, hat sich TensorFlow vor allem im Bereich Deep Learning als populäres Werkzeug bewährt (Pang et al., 2020, S. 227). So bietet TensorFlow Projekte, die viel Customizing erfordern, eine leistungsfähige und flexible Umgebung, die das Trainieren künstlicher neuronaler Netze vereinfacht und beschleunigt (Pang et al., 2020, S. 227).

Die dritte zu untersuchende Bibliothek *PyTorch* wird zwar insgesamt weniger häufig verwendet, verzeichnet jedoch Jahr für Jahr ein starkes Wachstum in der Nutzung (Kaggle, 2021). PyTorch ist vor allem nützlich beim Umgang mit künstlichen neuronalen Netzen, weshalb es 2019 auch die meistgenutzte Deep Learning-Bibliothek auf allen großen Deep Learning-Konferenzen war (Raschka et al., 2020, S. 23).

MLflow ist wie die eben aufgeführten Bibliotheken Open Source und wurde

vom Silicon Valley Startup databricks 2018 ins Leben gerufen (Zaharia et al., 2018, S. 39). Ziel der Plattform ist dabei den Lebenszyklus von ML-Projekten ganzheitlich zu strukturieren (Zaharia et al., 2018, S. 39). Dabei werden über die ML-Algorithmen hinaus, den Anwendenden Werkzeuge mit in die Hand zu geben, um den Herausforderungen von ML-Projekten gegenüber herkömmlicher Softwareentwicklungsprozesse zu begegnen (Zaharia et al., 2018, S. 44).

2.3.4 Konfigurationsoptionen

In objektorientierten Programmiersprachen wie Python sind Klassen einer der grundlegenden Bausteine, die bei der Entwicklung und Anwendung von maschinellem Lernen eingesetzt werden. Sie bieten die Möglichkeit, Daten und Funktionen zu kombinieren und dem Nutzenden von Machine Learning Bibliotheken, bereits vorformulierte Algorithmen aufzurufen. Dabei werden die Konfigurationsoptionen beim Aufruf der jeweiligen Klasse als Parameter übergeben, sodass die Algorithmen an die Bedürfnisse des Anwendenen angepasst werden können. So kann die zielgerichtete Nutzung effizienter und komplexer Algorithmen bei gleichzeitiger Konfigurierbarkeit gewährleistet werden.

Bei den Konfigurationsparametern von Lernalgorithmen handelt es sich um sogenannte *Hyperparameter*. Lernalgorithmen ermöglichen einem Computerprogramm den menschlichen Lernprozess mittels statistischer Methoden zu imitieren. Sie werden zur Mustererkennung, Klassifizierung und Vorhersage verwendet, indem sie aus einem vorhandenen Trainingsdatensatz lernen. Hyperparameter werden festgelegt bevor der Lernprozess beginnt und um den Lernprozess zu steuern (Andonie, 2019, S. 280). Da sich die Algorithmen oft sehr unterschiedlich verhalten, wenn sie mit verschiedenen Hyperparameter instanziiert werden, wurden in den letzten Jahren eine Vielzahl an Hyperparameter-Optimierungsverfahren entwickelt (Hutter et al., 2014, S. 1). Optimierte Hyperparameter können die Leistungsfähigkeit eines Modells oder die Geschwindigkeit und Qualität Lernprozesses stark beeinflussen (Andonie, 2019, S. 280).

Im folgenden Codebeispiel wird der *Gradient Boosting Classifier* aus der scikit-learn Bibliothek betrachtet.

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=20,
        learning_rate=0.05, max_features=2, max_depth=2,
        random_state=0)
```

Es wird eine Instanz dieser Klasse erzeugt und der Variable *gbc* zugewiesen. Die Klasse verfügt über 20 Parameter (Version 1.0.2), die, sofern bei der Instanziierung für den jeweiligen Parameter kein neuer Wert zugewiesen wird, mit einem eigenen Default-Wert initialisiert werden. So werden im Beispiel, den Hyperparametern wie *n_estimators*, *learning_rate* oder *max_depth* Zahlenwerte übergeben, die von den Default-Werten abweichen.

2.4 Web Scraping

Web Scraping ist eine Technik, um Daten aus dem World Wide Web zu extrahieren, um sie später abrufen oder analysieren zu können (Zhao, 2017, S. 1). Dafür werden die Webdaten über das Hypertext Transfer Protocol (HTTP) oder über einen Webbrowser ausgelesen (Zhao, 2017, S. 1). Dies kann entweder manuell durch den jeweiligen Benutzenden oder automatisch durch einen Bot oder Webcrawler erfolgen (Zhao, 2017, S. 1). Ein Web Scraper simuliert das menschliche Browsing-Verhalten im Web, um aus verschiedenen Websites detaillierte Informationen in einer vorgegebenen Struktur zu sammeln (Diouf et al., 2019, S. 6040). Durch die Möglichkeit für eine bestimmte Website-Struktur systematisch ausgerichtet und programmiert zu werden, liegt der Vorteil eines Web Scrapers in seiner Automatisierungsfähigkeit und Geschwindigkeit (Diouf et al., 2019, S. 6040). Mögliche Anwendungsfälle sind zum Beispiel das Überwachen von Preis-Änderungen in Online-Shops oder das Auslesen und Kopieren von Kontaktinformationen.

Ein Web Scraping-Prozess gliedert sich üblicherweise in zwei Schritten (Zhao, 2017, S. 1):

1. Erfassen der Webressourcen
2. Extrahieren der gewünschten Informationen aus den erfassten Daten

Zunächst wird die Kommunikation zur Ziel-Website über das HTTP-Protokoll hergestellt (Glez-Peña et al., 2013, S. 789). Über die HTTP-Anfrage ist der Scraper in der Lage, die Ressourcen der jeweiligen Website zu erfassen (Zhao,

2017, S. 1). Dies erfolgt entweder als URL mit einer GET-Abfrage für Ressourcenanfragen oder als HTTP-Nachricht mit einer POST-Abfrage für die Übermittlung von Formularen (Glez-Peña et al., 2013, S. 789). Nachdem die Anfrage erfolgreich empfangen und von der Ziel-Website verarbeitet wurde, wird die angeforderte Ressource von der Website aufgerufen und an das Web Scraping-Programm zurückgesendet (Zhao, 2017, S. 1). Die Ressource kann in verschiedenen Formaten vorliegen: In Auszeichnungssprachen wie HTML (Hypertext Markup Language) oder XML (Extensible Markup Language), JSON-Format (JavaScript Object Notation) oder in Form von Multimedia-Daten wie Bilder-, Audio oder Videodateien (Zhao, 2017, S. 1).

Im zweiten Schritt folgt der Extraktionsprozess. Die heruntergeladenen Daten können nun geparst werden, um die benötigten Information zu filtern und in ein geeignetes Format umzuwandeln (Glez-Peña et al., 2013, S. 790). Die Daten können nun weiterverarbeitet werden, indem sie beispielsweise analysiert oder in eine gewünschte Struktur organisiert werden.

2.5 Abstract Syntax Trees

Um die Bedeutung von Abstract Syntax Trees (AST) zu verstehen, ist es hilfreich sich zu vergegenwärtigen, welche Prozesse bei der Ausführung eines Python-Skripts im Hintergrund ablaufen. Als eine interpretierte High-Level-Programmiersprache wird der Source Code in Python nicht kompiliert, sondern vom Python Interpreter nach einer bestimmten Abfolge von Schritten in Anweisungen übersetzt. Dabei wird der Python Code in Bytecode umgewandelt, sodass dieser von der Python Virtual Machine übersetzt und ausgeführt werden kann.

Zunächst wird der Code geparst und in sogenannte Tokens unterteilt. Diese Tokens unterliegen einer Reihe von Regeln, damit die verschiedenen Programmierkonstrukte unterschiedlich erkannt und behandelt werden können. Die Liste an Tokens werden dann in eine Baumstruktur, den sogenannten abstrakten Syntaxbaum (Abstract Syntax Tree), umgewandelt. Ein AST ist eine baumartige Darstellung des Codes, bestehend aus einer Sammlung von Knoten, die, basierend auf der Grammatik in Python, miteinander verbunden sind. Der Baum wird in maschinenlesbaren Binärcode umgewandelt und Anweisungen in Bytecode an den Python Interpreter übermittelt. Der Python Interpreter kann den Code nun ausführen und Systemaufrufe an den Kernel starten, um das Programm zu starten.

Während Bytecode eher für Maschinen gemacht ist, sind Abstract Syntax Trees strukturiert aufgebaut und auch für den Menschen lesbar. Im Anhang befindet sich das Codebeispiel aus 2.3.4, nachdem es als AST geparst wurde. Aus der AST-Syntax lassen sich Typ und Struktur einzelner Python Komponenten direkt ablesen. So besteht das Modul aus zwei Codeobjekten vom Typ *ImportFrom* und *Assign*, die sich jeweils in kleinere Objekte verschiedenen Typs unterteilen lassen. Diese Datenstruktur stellt alle relevanten Informationen für die Ausführung des Codes bereit. Jeder Knoten des Baumes kann nun besucht werden, um seine Daten zu verarbeiten und entsprechende Aktionen einzuleiten. Über das *ast*-Modul in Python kann der Code als AST verarbeitet und visualisiert werden, sodass er von Entwicklern und Entwicklerinnen entsprechend seiner AST-Syntax analysiert und manipuliert werden kann.

2.6 Verwandte Arbeiten

Zwar wurden bereits einige Ansätze zum Auffinden von Konfigurationsoptionen publiziert, jedoch ist diese Thematik im Rahmen von maschinellem Lernen kaum beleuchtet. Die meisten Ansätze, wie der von Rabkin and Katz aus dem Jahre 2011, behandeln Anwendungen in der Programmiersprache Java und ermitteln Konfigurationsoptionen mittels statischer Code-Analyse. Der von Rabkin and Katz entwickelte *Confalyzer* ist vermutlich einer der ersten bekannteren Tools, die sich mit der Thematik befassen. Unter der Annahme, das Konfigurationsoptionen als Key-Value Pair vorliegen, betrachtet der Confalyzer Methoden, die mit dem für Java typischen Schlüsselwort *get* in der Konfigurationsklasse beginnen (Rabkin and Katz, 2011, S. 131). Mittels eines Aufrufgraphens (Call Graph) identifiziert der Confalyzer, wo die Methoden im Source Code aufgerufen werden. An den jeweiligen Aufrufstellen kann er dann die Konfigurationsoptionen aus den String-Parametern ablesen (Rabkin and Katz, 2011, S. 131). Der Ansatz beruht auf der Annahme, dass viele Konfigurationsoptionen auf ähnliche Weise verwendet werden und bestimmte Muster ableitbar sind (Rabkin and Katz, 2011, S. 131).

Der von Dong et al. entwickelte *ORPLocator* orientiert sich an dem Confalyzer und vergleicht sich mit diesem (Dong et al., 2016, S. 193). Der ORPLocator untersuchte dieselben Java-Frameworks wie beispielsweise Hadoop und konnte mehr dokumentierte Optionen und dementsprechend auch mehr Verwendungen im Source Code finden (Dong et al., 2016, S. 193). Dies liegt unter anderem daran, dass der gesamte Source Code nach Aufrufstellen von

Konfigurationsschnittstellen durchsucht wird, während der Confalyzer einen Aufrufgraphen erstellt, der manche Optionen nicht erfasst (Dong et al., 2016, S. 193).

Einen anderen Ansatz verfolgten Lillack et al. bei der Entwicklung von *Lotrack*. Lotrack ist ein Tool, das mittels einer erweiterten Taint-Analyse, einer Datenflussanalyse, die externe Daten (Eingabedaten) über den gesamten Kontrollfluss verfolgt, eine Konfigurations-Map erstellt (Lillack et al., 2018, S. 445). Die Konfigurations-Map beschreibt, welche Codefragmente von Konfigurationsoptionen abhängen und hilft dabei die Beziehungen zwischen dem konkreten Programmverhalten und der Konfiguration zu finden (Lillack et al., 2018, S. 447). Der Fokus liegt auf Anwendungen, die auf Android-Systemen laufen (Lillack et al., 2018, S. 445).

Weitere Ansätze wie der *PrefFinder* von Jin et al. verwenden zusätzlich auch dynamische Analysetechniken, um Konfigurationsoptionen nicht nur zu extrahieren, sondern auch in einer Datenbank zur Abfrage und Verwendung zu speichern (Jin et al., 2014, S. 151). Der *Software Configuration Inconsistency Checker* (SCIC) hingegen von Behrang et al. erweitert die Extraktion von Konfigurationsoptionen im Key-Value-Modell des Confalyzer um ein baumstrukturiertes Modell (Behrang et al., 2015, S. 295). Im Gegensatz zu anderen Tools, ist dieses auch in der Lage mehrere Programmiersprachen zu verarbeiten (Behrang et al., 2015, S. 295). Ziel dieses Tools ist Konfigurationsfehler zu identifizieren (Behrang et al., 2015, S. 295).

Kapitel 3

Methodik

Anhang A

Abstract Syntax Tree

```
Module(
    body=[
        ImportFrom(
            module='sklearn.ensemble',
            names=[
                alias(
                    name='GradientBoostingClassifier')],
            level=0),
        Assign(
            targets=[
                Name(
                    id='gbc',
                    ctx=Store())],
            value=Call(
                func=Name(
                    id='GradientBoostingClassifier',
                    ctx=Load()),
                args=[],
                keywords=[
                    keyword(
                        arg='n_estimators',
                        value=Constant(
                            value=20)),
                    keyword(
                        arg='learning_rate',
                        value=Constant(
                            value=0.05)),
                    keyword(
                        arg='max_features',
                        value=Constant(
```

```
                value=2)),
        keyword(
            arg='max_depth',
            value=Constant(
                value=2)),
        keyword(
            arg='random_state',
            value=Constant(
                value=0))]))])
```

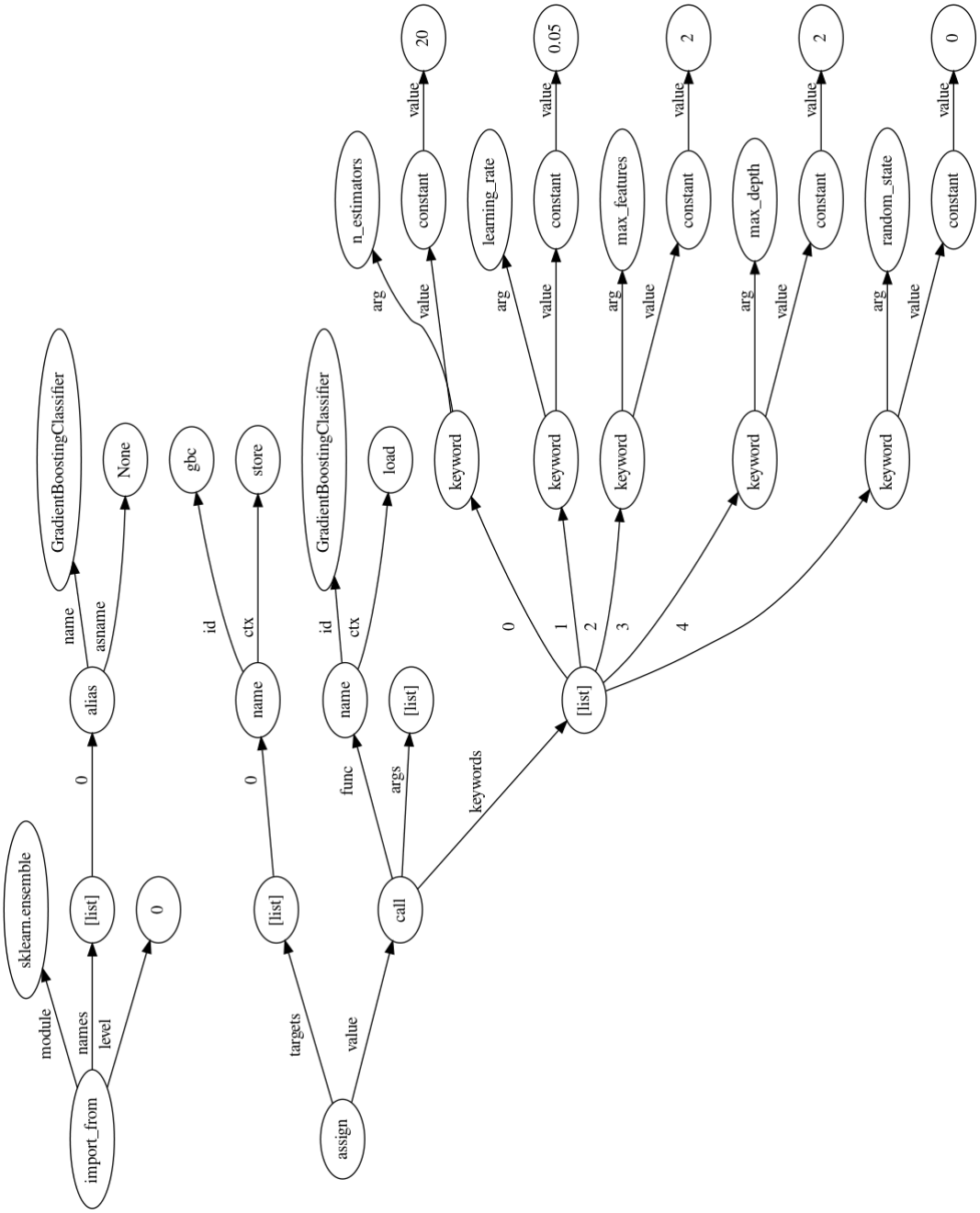

Abbildung A.1: Abstract Syntax Tree

Literaturverzeichnis

Răzvan Andonie. Hyperparameter optimization in learning systems. *Journal of Membrane Computing*, 1(4):279–291, 2019. doi: 10.1007/s41965-019-00023-0. URL https://doi.org/10.1007/s41965-019-00023-0. 2.3.4

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

Farnaz Behrang, Myra B. Cohen, and Alessandro Orso. Users beware: Preference inconsistencies ahead. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 295–306, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336758. doi: 10.1145/2786805.2786869. URL https://doi.org/10.1145/2786805.2786869. 2.6

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

Rabiyatou Diouf, Edouard Ngor Sarr, Ousmane Sall, Babiga Birregah, Mamadou Bousso, and Sény Ndiaye Mbaye. Web scraping: State-of-the-art and areas of application. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 6040–6042, 2019. doi: 10.1109/BigData47090.2019.9005594. 2.4

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, 2.6

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. 2.3.1, 2.3.2

Daniel Glez-Peña, Anália Lourenço, Hugo López-Fernández, Miguel Reboiro-Jato, and Florentino Fdez-Riverola. Web scraping technologies in an API world. *Briefings in Bioinformatics*, 15(5):788–797, 04 2013. ISSN 1467-5463. doi: 10.1093/bib/bbt026. URL `https://doi.org/10.1093/bib/bbt026`. 2.4

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

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Bejing, China, 22–24 Jun 2014. PMLR. URL `https://proceedings.mlr.press/v32/hutter14.html`. 2.3.4

Dongpu Jin, Myra B. Cohen, Xiao Qu, and Brian Robinson. Preffinder: Getting the right preference in configurable software systems. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, page 151–162, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330138. doi: 10.1145/2642937.2643009. URL `https://doi.org/10.1145/2642937.2643009`. 2.6

Kaggle. State of machine learning and data science 2021. `https://www.kaggle.com/kaggle-survey-2021`, 2021. Online; accessed 05-April-2022. (document), 2.3.3, 2.2, 2.3.3

KDnuggets. Python leads the 11 top data science, machine learning platforms: Trends and analysis. `https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html`, 2019. Online; accessed 04-April-2022. 2.3.3

Li Li, Jiawei Wang, and Haowei Quan. Scalpel: The python static analysis framework. *arXiv preprint arXiv:2202.11840*, 2022. (document), 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, 2.6

Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020. 2.3, 2.3.2

Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020. doi: 10.3102/1076998619872761. URL `https://doi.org/10.3102/1076998619872761`. 2.3.3

L.L. Pollock and M.L. Soffa. An incremental version of iterative data flow analysis. *IEEE Transactions on Software Engineering*, 15(12):1537–1549, 1989. doi: 10.1109/32.58766. 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`. 2.3.2

Ariel Rabkin and Randy Katz. Static extraction of program configuration options. In *Proceedings of the 33rd International Conference on Software 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, 2.6

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, 2.3, 2.3.3, 2.3.3

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

I. Stančin and A. Jović. An overview and comparison of free python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 977–982, 2019. doi: 10.23919/MIPRO.2019.8757088. 2.3.3

G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, and A. Mueller. Scikit-learn: Machine learning without learning the machinery. *GetMobile: Mobile Comp. and Comm.*, 19(1):29–33, jun 2015. ISSN 2375-0529. doi: 10.1145/2786984.2786995. URL `https://doi.org/10.1145/2786984.2786995`. 2.3.3

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

Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018. 2.3.3

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

Bo Zhao. Web scraping. *Encyclopedia of big data*, pages 1–3, 2017. 2.4, 2.4