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: Sebastian Simon, M.Sc.

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

Über Konfigurationsoptionen können die Nutzenden eine Software an ihre Bedürfnisse anpassen. Die Wahl der Konfigurationsoptionen hat dabei einen kritischen Einfluss auf die Ausführung des Source Codes. Im Machine Learning (ML) Bereich liegen Konfigurationsoptionen in Form von Hyperparametern vor, mit denen ML-Algorithmen, beispielsweise das Trainieren eines Modells, gesteuert werden können. Es gibt jedoch keine Übersicht darüber, wie ML-Algorithmen konfiguriert werden. Diese Arbeit stellt einen Ansatz vor, bei dem ML-Konfigurationsoptionen in Python Source Code extrahiert werden können. Mittels statischer Code-Analyse lassen sich mit diesem Ansatz die Konfigurationsoptionen populärer ML-Bibliotheken in beliebigen Softwaresystemen lokalisieren. Darüber hinaus können mit Hilfe von Techniken aus der Datenflussanalyse auch bei variablen Hyperparametern mögliche Konfigurationswerte bestimmt werden. Mit diesem Ansatz können so ML-Algorithmen und die Konfigurationsoptionen mit einer sehr hohen Genauigkeit identifiziert werden.

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, die Ausführung einer Software zu steuern, zum Beispiel hinsichtlich der Performance oder Funktionalität [Han et al., 2021]. Konfigurationsoptionen besitzen dabei ganz unterschiedliche Funktionen, die von den Nutzenden nach den eigenen Bedürfnissen angepasst werden können.

Konfigurationsoptionen können in verschiedenen Teilen eines Softwareprojekts verarbeitet, definiert und beschrieben werden, wie beispielsweise in den Konfigurationsdateien, im Source Code und in der Dokumentation [Dong et al., 2016]. Sie werden häufig als Key-Value Pair entworfen und gesammelt in einer Konfigurationsdatei gespeichert. Dem Namen der Konfigurationsoption (Key) werden dabei Einstellungsmöglichkeiten beliebigen Typs zugeordnet (Value). Ein einheitliches Schema zur Speicherung von Konfigurationsdateien gibt es jedoch nicht, weshalb sich diese von der Struktur und Syntax unterscheiden [Rabkin and Katz, 2011].

In Softwaresystemen aus dem Machine Learning Bereich sind Konfigurationsoptionen im Source Code in Form von sogenannten *Hyperparametern* vorhanden. Machine Learning (ML) ist ein Teilgebiet der künstlichen Intelligenz, in der das Lernverhalten von Menschen imitiert wird. Durch den Einsatz statistischer Methoden werden Algorithmen trainiert, um beispielweise Klassifizierungen oder Vorhersagen zu treffen. Bei Hyperparametern handelt es sich um Parameter eines Machine Learning Algorithmus, die vor Beginn des Lernprozesses vom Nutzenden festgelegt werden können. Sie werden als Parameter an die jeweilige Klasse übergeben und bieten den Nutzenden Konfigurationsmöglichkeiten für das Training des Modells [Andonie, 2019].

Die Anzahl der Machine Learning Anwendungen ist in den letzten Jahren stark gestiegen. Unter anderem durch den Erfolg und den Möglichkeiten des Internets ist nicht nur das Volumen an Daten enorm gewachsen, auch die Heterogenität der Daten führt dazu, dass herkömmliche Methoden zur Verarbeitung und Informationsgewinnung nicht mehr ausreichen. Gleichzeitig trägt die technologische Weiterentwicklung der Rechen- und Speicherleistung von Computern in den letzten Jahrzehnten dazu bei, dass durch neue Ansätze des Machine Learnings die rasant wachsenden Datenmengen leistungsfähig verarbeitet werden können [Fradkov, 2020].

Dennoch fehlt es in der Forschung an Ansätzen, um Konfigurationsoptionen von ML Algorithmen zu lokalisieren und extrahieren. Weiterhin gibt es keine Übersicht darüber, wie ML Algorithmen konfiguriert werden. Zudem kann die statische Prüfung der Hyperparameter für den Nutzenden eine Zeitersparnis bedeuten, falls ein falscher Konfigurationswert einen langen Batch-Job zum Scheitern bringt oder das Ergebnis nicht den Erwartungen entspricht.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, Konfigurationsoptionen im Source Code zu erkennen und zu extrahieren. Der Fokus liegt hierbei auf die Konfiguration (Hyperparameter) von ML-Algorithmen, die von bekannten ML Bibliotheken zur Verfügung gestellt werden.

Es existieren bereits einige Forschungsansätze zur Ermittlung und Verarbeitung von Konfigurationsoptionen im Source Code. Viel zitiert wird dabei der von Rabkin and Katz 2011 publizierte Ansatz. 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 Java, die nach dem TIOBE-Index jahrelang als beliebteste Programmiersprache galt. Für die Programmiersprache Python, die im Februar 2022 erstmals zur beliebtesten Sprache in diesem Index aufstieg, ist diese Thematik jedoch bislang allerdings noch wenig beleuchtet [BV, 2022]. Insbesondere im Bereich des wissenschaftlichen Rechnens (Scientific Computing) gewann Python in den letzten Jahren enorm an Popularität, weshalb viele Machine Learning Frameworks auf Python basieren [Raschka et al., 2020].

Gegenstand dieser Arbeit sind daher Konfigurationsoptionen von Machine Learning Algorithmen in Python Source Code, die mittels statischer Code-Analyse erkannt werden. Der Ansatz identifiziert automatisch die Stellen im Source Code, bei denen ML-Algorithmen mit Konfigurationsoptionen aus den zu untersuchenden Bibliotheken gelesen werden. Für jede dieser Stellen werden

die Namen der Optionen ermittelt. Darauf aufbauend erfolgt eine Datenflussanalyse, um auch für Optionen, die in Form von variablen Parametern übergeben werden, die möglichen Werte zu ermitteln. Der Fokus liegt auf drei der populärsten Machine Learning Bibliotheken in Python [Kaggle, 2021]:

- TensorFlow
- PyTorch
- scikit-learn

1.3 Aufbau dieser Arbeit und methodisches Vorgehen

Im ersten Teil dieser Arbeit wird der theoretische Hintergrund vorgestellt. Dieser umfasst die verwendeten Methoden aus der statischen Code- und Datenflussanalyse sowie des Web Scrapings. Zudem erfolgt eine kurze Einführung zu Machine Learning und es wird beschrieben, welche Funktionen Python-Bibliotheken und Konfigurationsoptionen in diesem Zusammenhang erfüllen. Abgerundet wird der erste Teil mit einer Erläuterung zu Abstract Syntax Trees. Nach einem Exkurs zu den verwandten Arbeiten in Kapitel 3. Im Anschluss wird das methodische Vorgehen des entwickelten Ansatzes in seinen einzelnen Teilschritten von der Auswahl der Python-Bibliotheken in Abschnitt 4.1 bis hin zu der Datenflussanalyse in Abschnitt 4.6 beschrieben. Für die Evaluation dieser Arbeit werden die Ergebnisse in Kapitel 5 zusammengetragen und mit einer manuellen Analyse verglichen. Diese Ergebnisse werden diskutiert und die Limitationen des Ansatzes aufgezeigt. Ein Fazit rundet diese Arbeit ab.

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. Bei einer statischen Code-Analyse wird der Quellcode mit computergestützten Methoden untersucht, ohne ihn auszuführen. Dabei werden die einzelnen Statements im Code anhand der jeweiligen Syntax und Regeln der Programmiersprache analysiert [Gomes et al., 2009]. Die statische Code-Analyse ist ein Werkzeug, um Informationen aus dem Source Code abzuleiten oder Fehler in einer Softwareanwendung zu reduzieren. So ermöglichen sie den Anwendenden, alle Pfade des Kontrollflusses der Software zu betrachten oder Fehler in einem Programm zu finden, die für den Compiler nicht sichtbar sind [Bardas et al., 2010].

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

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. Gleichzeitig verbergen sich hinter den Methoden und Klassen, der zu untersuchenden Machine Learning Bibliotheken, teils sehr komplexe und rechenintensive Berechnungen, die bis zu mehreren Tagen laufen können. Aus Kosten-Nutzen-Überlegungen 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. Es handelt sich um eine statische Analysetechnik mit dem Ziel, das Programmverhalten bereits zur Übersetzungszeit, also bevor es ausgeführt wird, zu bestimmen. Der Datenfluss kann mittels eines Kontrollflussgraphens dargestellt werden und Rückschlüsse über das Verhalten des Programms geben. Der Graph zeigt an, an welchen Stellen eine Variable verwendet wird und welche Werte sie annehmen kann [Pollock and Soffa, 1989].

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 es 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 weiterverarbeitet 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 aufsteigenden Index versehen, sodass jede Definition ihre eigene Version erhält [Li et al., 2022]. Die beiden Techniken Static Single Assignment und Constant Propagation werden in dem statischen Analyse-Framework *Scalpel* kombiniert [Li et al., 2022].

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
```

Listing 2.1: Codebeispiel von Scalpel für Datenflussanalyse [Li et al., 2022]

Der daraus resultierende Kontrollflussgraph besteht charakteristisch aus Knoten, die die jeweiligen 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, sodass es zwei mögliche Rückgabewerte gibt.

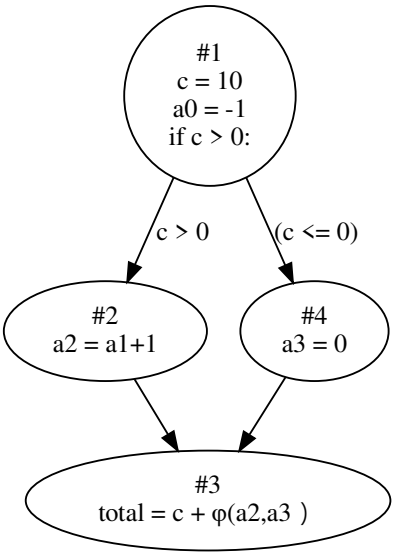


Abbildung 2.1: Kontrollflussgraph [Li et al., 2022]

2.3 Machine Learning

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. Dazu gehören zum Beispiel die Verarbeitung von natürlicher Sprache (Natural Language Processing) oder die Bilderkennung (Computer Vision). In der Mitte des 20. Jahrhunderts entstand Machine Learning (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].

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

2.3.1 Python-Bibliotheken

Für Machine Learning gilt Python schon seit langem als erste Wahl für Entwicklerinnen und Entwickler. 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. Viele Machine Learning Lösungen basieren auf numerischen und vektorisierten Berechnungen mit Bibliotheken wie *NumPy* oder *SciPy*. Um diese Berechnungen schnell und effizient auszuführen, werden sogenannte *Wrapper* verwendet, die Algorithmen von kompilierten Programmiersprachen implementieren. 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]. 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 ML-Algorithmen aufgerufen werden können, wodurch die Benutzendenfreundlichkeit bei gleichzeitiger Effizienz in der Performanz gewahrt bleiben kann [Raschka et al., 2020]. 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 Machine Learning Frameworks [Kaggle, 2021].

Eine Python-Bibliothek entspricht einer Sammlung zusammengehöriger Module, die aus vorkompiliertem Code bestehen. Nach erfolgreicher Installation werden die Funktionalitäten der jeweiligen Bibliothek über die *import*-Anweisung für ein Programm zugänglich gemacht. Entwickler und Entwicklerinnen können die vorprogrammierten Klassen und Methoden aufrufen und beliebig verwenden, ohne die hinterlegten Algorithmen kennen zu müssen.

2.3.2 ML Konfigurationsoptionen

In objektorientierten Programmiersprachen wie Python sind Klassen einer der grundlegenden Bausteine, die bei der Entwicklung und Anwendung von Machine Learning 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 Anwendenden angepasst werden können. Somit 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]. 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]. Optimierte Hyperparameter können die Leistungsfähigkeit eines Modells oder die Geschwindigkeit und Qualität Lernprozesses stark beeinflussen [Andonie, 2019].

Das Codebeispiel in Listing 2.2 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)
```

Listing 2.2: Nutzung der GradientBoostingClassifier-Klasse aus scikit-learn

Es wird eine Instanz dieser Klasse erzeugt und der Variablen *gbc* zugewiesen. Die Klasse verfügt über 20 Parameter (Version 1.1.1), 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, Daten aus dem World Wide Web zu extrahieren, um sie später abrufen oder analysieren zu können. Dafür werden die Webdaten über das Hypertext Transfer Protocol (HTTP) oder über einen Webbrowser ausgelesen. Dies kann entweder manuell durch den jeweiligen Benutzenden oder automatisch durch einen Bot oder Webcrawler erfolgen [Zhao, 2017]. Ein Web Scraper simuliert das menschliche Browsing-Verhalten im Web, um aus verschiedenen Websites detaillierte Informationen in einer vorgegebenen Struktur zu sammeln. Aufgrund der 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]. 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 Schritte [Zhao, 2017]:

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]. Über die HTTP-Anfrage ist der Scraper in der Lage, die Ressourcen der jeweiligen Website zu erfassen [Zhao, 2017]. 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]. 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. 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].

Im zweiten Schritt folgt der Extraktionsprozess. Die heruntergeladenen Daten werden geparst, um die benötigten Informationen zu filtern und in ein geeignetes Format umzuwandeln [Glez-Peña et al., 2013]. 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, die Prozesse, die bei der Ausführung eines Python-Skripts im Hintergrund ablaufen, zu kennen. Als eine interpretierte 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 [Aycock, 1998].

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 wird 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 [Aycock, 1998].

Während Bytecode eher für Maschinen gemacht ist, sind Abstract Syntax Trees strukturiert aufgebaut und auch für den Menschen lesbar. Abbildung A.1 im Anhang zeigt das Codebeispiel aus Listing 2.2, 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.

Kapitel 3

Verwandte Arbeiten

Wenngleich bereits einige Ansätze zum Auffinden von Konfigurationsoptionen publiziert wurden, ist diese Thematik jedoch im Rahmen von Machine Learning 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, der sich mit der Thematik befasst. 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. 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. Der Ansatz beruht auf der Annahme, dass viele Konfigurationsoptionen auf ähnliche Weise verwendet werden und bestimmte Muster ableitbar sind [Rabkin and Katz, 2011].

Der von Dong et al. entwickelte *ORPLocator* orientiert sich an dem Confalyzer und vergleicht sich mit diesem. Der ORPLocator untersuchte dieselben Java-Frameworks wie beispielsweise Hadoop und konnte mehr dokumentierte Optionen und dementsprechend auch mehr Verwendungen im Source Code finden. 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].

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. Diese Konfigurations-Map beschreibt, welche Codefragmente von Konfigurationsoptionen abhängen

und hilft dabei, die Beziehungen zwischen dem konkreten Programmverhalten und der Konfiguration zu finden. Der Fokus liegt auf Anwendungen, die auf Android-Systemen laufen [Lillack et al., 2018].

Einer der neuesten Ansätze ist der von Han et al. entwickelte *ConfProf*. Mittels einer *Performance-Profiling-Technik* fokussierten sie sich darauf, wie Konfigurationsoptionen und ihre Interaktionen die Leistung eines Softwaresystems beeinflussen. Dafür verwenden sie dynamische Code-Analyse, um für die Ausführung eines Programms ein Profil zu erstellen. Mit Machine Learning wird der Leistungseinfluss von Konfigurationsoptionen analysiert und eingeordnet. Untersucht wurden Softwaresysteme mit umfangreichen Konfigurationsmöglichkeiten wie *Apache Server* oder *PostgreSQL* [Han et al., 2021].

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]. Der *Software Configuration Inconsistency Checker* (SCIC) von Behrang et al. hingegen erweitert die Extraktion von Konfigurationsoptionen im Key-Value-Modell des Confalyzer um ein baumstrukturiertes Modell. Im Gegensatz zu anderen Tools ist dieses auch in der Lage, mehrere Programmiersprachen zu verarbeiten. Ziel dieses Tools ist die Identifikation von Konfigurationsfehlern [Behrang et al., 2015].

Zusammenfassend lässt sich daher feststellen, dass die Verwendung und Auswirkungen von Konfigurationsoptionen mit verschiedenen Ansätzen erforscht werden. Allerdings stehen Konfigurationsmöglichkeiten von Machine Learning Algorithmen weniger im Kerninteresse bisheriger Forschungen, was diese Arbeit von vorgestellten Ansätzen unterscheidet.

Kapitel 4

Methodik

In diesem Kapitel wird das Vorgehen zur Ermittlung und Extraktion von Konfigurationsoptionen beschrieben. Zunächst wird in Abschnitt 4.1 auf die Auswahl der Python-Bibliotheken eingegangen. Im nächsten Abschnitt wird der Scraping-Prozess in den Dokumentationen dieser Bibliotheken erläutert. Danach folgt eine Einführung wie die Repositorys von Softwaresystemen in diesem Ansatz geparst werden, um - wie in Abschnitt 4.4 und 4.5 beschrieben - die Klassen und Parameter extrahieren zu können. Zum Schluss gibt es einen Einblick in die Methodik bei der Bestimmung der variablen Parameterwerte. Zu beachten ist, dass der Ansatz auf den Betriebssystemen macOS und Linux Ubuntu entwickelt und getestet wurde.

4.1 Auswahl der Python-Bibliotheken

Bevor auf das Vorgehen zur Extraktion von Konfigurationsoptionen genauer eingegangen wird, erfolgt zunächst eine Beschreibung zu untersuchenden Bibliotheken. Die Auswahl der Bibliotheken erfolgt nach folgenden Kriterien:

- Open Source
- geschrieben in Python
- setzen Fokus auf Machine Learning
- gehören zu den populärsten Machine Learning Bibliotheken

Die Grafik 4.1 zeigt das Ergebnis einer jährlich durchgeführten Umfrage der Online-Community *Kaggle* aus dem Jahr 2021. 25.000 Data Scientists und ML-Ingenieure wurden befragt, welche Frameworks sie im Machine Learning

Bereich verwenden. In der Grafik werden die einzelnen Frameworks der Popularität nach geordnet.

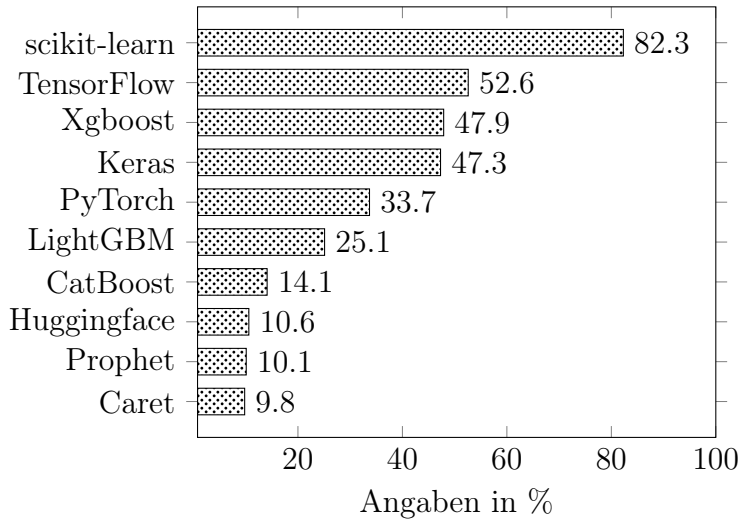


Abbildung 4.1: Nutzung von ML-Frameworks [Kaggle, 2021]

Die Grafik enthält drei der zu untersuchenden Python-Bibliotheken. An erster Stelle steht *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. Scikit-learn ist folglich bei der Auswahl der ML Bibliotheken unverzichtbar. Die Vorteile von scikit-learn sind die benutzerfreundliche Struktur und Dokumentation mit der Machine Learning auch für unerfahrene oder fachfremde Menschen zugänglich gemacht wird [Varoquaux et al., 2015].

An zweiter Stelle steht *TensorFlow*, das von über 50% der Befragten benutzt und daher ebenfalls in dem vorliegenden Ansatz untersucht wird. Nachdem TensorFlow 2015 von Forschern bei Google ursprünglich für interne Zwecke entwickelt wurde, hat sich TensorFlow vor allem im Bereich Deep Learning als populäres Werkzeug bewährt. 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].

Die dritte zu untersuchende Bibliothek *PyTorch* wird insgesamt weniger 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].

4.2 Web Scraping der ML-Klassen

Bevor im Source Code nach den Konfigurationsoptionen gesucht werden kann, ist es erforderlich, sich einen Überblick über die zu untersuchenden Bibliotheken zu verschaffen. Über die Dokumentation der Websites der Bibliotheken erhält man einen Einblick in die jeweilige Modul- und Klassenstruktur. Da die gesuchten Optionen als Parameter von ML-Klassen übergeben werden, zielt dieser Ansatz darauf ab, alle Klassen einschließlich der möglichen Parameter mit Hilfe von Web Scraping zu extrahieren. Aufgrund des systematischen Aufbaus der verschiedenen Websites, kann man an dieser Stelle mit technischer Unterstützung die gesuchten Daten automatisiert erfassen. Die Python-Bibliothek *Beautiful Soup* stellt für diesen Zweck das notwendige Werkzeug bereit. Beautiful Soup ist ein populäres Werkzeug für Web Scraping in Python. So können strukturierte Daten aus HTML- oder XML-Dokumenten als Syntaxbaum geparst werden. Mit einer Reihe verschiedener Methoden kann dieser dann analysiert oder die benötigten Informationen können extrahiert werden [Richardson, 2007].

Der Scraping-Prozess lässt sich für die drei verschiedenen ML-Bibliotheken in folgende Schritte einteilen, die über die Klasse *ClassScraper* aufgerufen werden können.

Je nach Komplexität der Dokumentation werden diese Schritte teilweise in weitere kleinere Schritte unterteilt. Zudem werden die Informationen in einem jeweils unterschiedlichen HTML-Format abgebildet, sodass kein generischer Algorithmus über alle drei Dokumentationen laufen kann. Deshalb wird für jede Bibliothek eine eigene Klasse implementiert, die vom ClassScraper zielführende Methoden erbt und bei einer abweichenden Dokumentationsstruktur Methoden überschreibt.

Listing 4.1 zeigt wie beispielsweise die Modul-URLs aus der Dokumentation von PyTorch extrahiert werden. Um mit dem Scrapen der Daten zu beginnen, wird über das *request*-Modul der Python-Bibliothek *urllib* die URL geöffnet. Die HTML-Datei der URL wird in Beautiful Soup geparst, um auf die Daten der Seite zugreifen zu können. In einem HTML-Dokument werden die Inhalte baumartig gespeichert. Die Knoten dieses Baums werden mit sogenannten *Tags* versehen, die dem Inhalt Form und Struktur verleihen. Um auf den Bereich der Seite zuzugreifen, in dem die PyTorch-Module verlinkt sind, wird der Bereich über die Beautiful Soup-Methoden *find, find_next* und *findAll* weiter eingegrenzt. Zunächst wird ein Paragraph (*p*) vom Typ *caption* gesucht, dessen Text *Python API* ist. Die einzelnen Modul-Links befinden sich in der nächstliegenden Liste, die mit dem Tag *ul* (*unordered list*) versehen ist. Von dem Paragraph ausgehend wird diese Liste mit der *find_next*-Methode erreicht. Die Elemente der Liste sind mit einem *li*-Tag (list item) versehen und werden extrahiert und als Set gespeichert. Durch dieses Set wird iteriert, um für jedes Modul die Hyperlink-Referenz zu speichern.

```
def scrape_module_urls(self):
    link = "https://pytorch.org/docs/stable/index.html"
    html = urllib.urlopen(link)
    soup = bs4.BeautifulSoup(html, "html.parser")

    caption = soup.find("p", {"class": "caption"},
                        text="Python API")
    ul = caption.find_next("ul")
    li = ul.findAll("li", {"class": "toctree-l1"})

    for element in li:
        url = element.find("a").attrs["href"]
        self.module_urls.append(url)
```

Listing 4.1: Web Scraping der Modul-URLs von PyTorch

Im Anschluss daran kann durch die Liste der Modul-URLs iteriert werden, um die Klassen zu extrahieren. Die Hyperlinks der Module werden ebenfalls mit urllib geöffnet und die HTML-Datei in Beautiful Soup geparst. Je nach Layout der Dokumentation werden mit Hilfe eben angesprochenen Beautiful Soup-Methoden schlussendlich alle Klassen inklusive der Parameter gefunden und in einer JSON-Datei gespeichert. In der JSON-Datei wird für jede Klasse der gesamte Pfad als eindeutiger Schlüssel angegeben. Zusätzlich zum Klassen- und zu den Parameternamen werden die Default-Werte der einzelnen Parameter aufgeführt. Die scikit-learn-Klasse *KNeighborsRegressor* wird demnach wie

folgt gespeichert:

```
"sklearn.neighbors.KNeighborsRegressor": {
    "short_name": "KNeighborsRegressor",
    "parameters": {
        "n_neighbors": "5",
        "*": null,
        "weights": "'uniform'",
        "algorithm": "'auto'",
        "leaf_size": "30",
        "p": "2",
        "metric": "'minkowski'",
        "metric_params": "None",
        "n_jobs": "None"
    }
}
```

Listing 4.2: Web Scraping Ergebnis der KNeighborsRegressor-Klasse aus scikit-learn

Im Gegensatz zu den anderen Modulen ist der Web Scraping Vorgang nur einmal je Bibliothek notwendig. Ein Neustart ist somit nicht für jedes Projekt erforderlich. Die am Ende erzeugte JSON-Datei bildet die Basis für das Extrahieren von Konfigurationsoptionen und kann für beliebig viele Projekte verwendet werden. Da bei diesem Vorgang eine Vielzahl an Websites angesteuert werden, handelt es sich hierbei um das Modul mit der längsten Laufzeit. Das Skript ist allerdings anfällig gegenüber Änderungen im Layout der Website. So können beispielsweise Änderungen der HTML-Elemente dazu führen, dass nicht mehr dieselben Ergebnisse erzielt werden.

Das Python-Skript kann sowohl über eine Entwicklungsumgebung als auch über die Kommandozeile ausgeführt werden. Um Python-Dateien über die Kommandozeile auszuführen, muss man lediglich den Befehl *python3* oder bei älteren Versionen *python*, gefolgt vom Namen der Python-Datei, eingeben. Zusätzlich ist es möglich, neben diesem Befehl noch weitere Werte hinzuzufügen. Diese sogenannten Argumente können im Source Code verarbeitet werden, um den Anwendenden eine Gestaltungsspielraum für die Ausführung zu überlassen. In diesem konkreten Fall wird über die Eingabe des Namens der Bibliothek entschieden, von welcher Bibliothek die Klassen extrahiert werden sollen. Damit auch bei unterschiedlichen Schreibweisen einer Bibliothek das Programm ausgeführt wird, sind im Source Code für jede der drei Bibliotheken Tensorflow, PyTorch und scikit-learn mehrere Möglichkeiten hinterlegt. Das Scraping von scikit-learn-Klassen erfolgt zum Beispiel über folgenden Befehl:

```
python3 scraping.py scikit-learn
```

Listing 4.3: Kommandozeilenbefehl für das Web Scraping von scikit-learn

4.3 Parsen der Repositorys

Der erste Schritt, um die Konfigurationsoptionen zu extrahieren, ist eine Umgebung zu schaffen, in der ein zu untersuchendes Git-Repository vorliegt. Über die *GitPython*-Bibliothek wird das Repository geklont, sodass es lokal für die Analyse verfügbar ist. Da sich die Analyse auf Source Code in Python bezieht, werden im nächsten Schritt alle Ordner sukzessiv nach Dateien mit einer *.py*-Endung durchsucht. Diese Python-Dateien werden in einer Liste gespeichert, durch die in der Folge iteriert wird, um die Konfigurationsoptionen auszulesen. Analog zum Web Scraping kann der gesamte Parse- und Extraktionsprozess inklusive der Datenflussanalyse über die Kommandozeile gestartet werden. Die zu übergebenen Argumente sind sowohl der Link des zu untersuchenden Git-Repositorys als auch die ML-Bibliothek, dessen Konfigurationselemente gefunden werden sollen. Der entsprechende Befehl für ein Beispielprojekt mit scikit-learn sieht demnach wie folgt aus:

```
python3 main.py https://github.com/user/project scikit-learn
```

Listing 4.4: Kommandozeilenbefehl für die Extraktion von Konfigurationsoptionen von scikit-learn

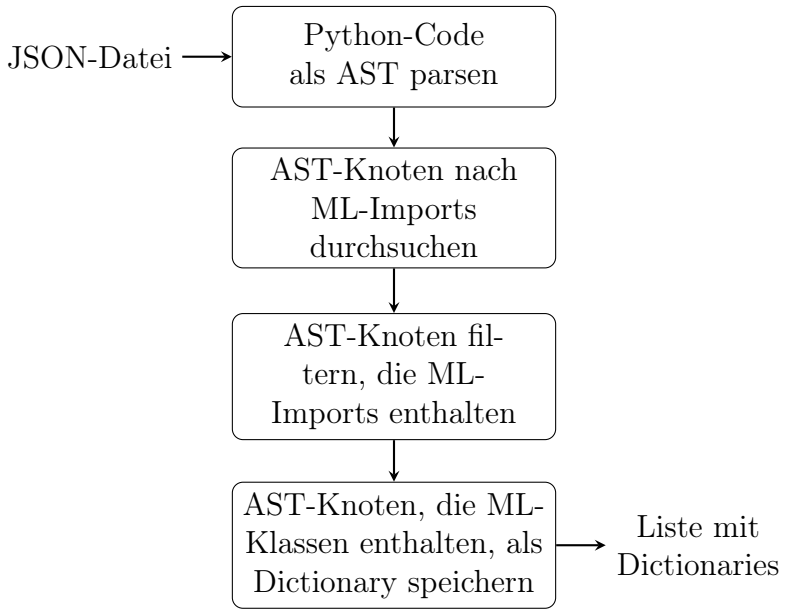
Im Gegensatz zum Web Scraping erfolgt der gesamte Parse- und Extraktionsprozess für die zu untersuchenden Bibliotheken generisch. Die Klasse *ConfigOptions* enthält alle Methoden, um aus den Repositorys die Konfigurationsoptionen auszulesen. Für jede der drei Bibliotheken gibt es eine Klasse, die von *ConfigOptions* abgeleitet ist und ihre Funktionen erbt. Dadurch lässt sich dieses Tool beliebig auf andere Bibliotheken erweitern. Voraussetzung ist jedoch, dass eine JSON-Datei mit den gescrapten Klassen der jeweiligen Bibliothek im entsprechendem Format vorliegt. Möchte man beispielsweise die Funktionalitäten dieses Tools auf die ML-Bibliothek *Keras* anwenden, muss man nach erfolgreichem Scraping, lediglich folgende Codezeilen zur *main.py*-Datei hinzufügen:

```
class KerasOptions(ConfigOptions):
    def __init__(self, repo):
        ConfigOptions.__init__(self, repo)
        self.library = "keras"
```

Listing 4.5: Implementation einer Klasse zur Extraktion von Konfigurationsoptionen einer weiteren ML-Bibliothek

4.4 Extraktion der ML-Klassen

Als Nächstes werden die Klassen der jeweiligen ML-Bibliothek aus dem Source Code extrahiert. Dafür wird durch die Liste mit Python-Dateien iteriert, der Source Code als AST geparst und entlang der AST Struktur nach den entsprechenden Klassen gesucht. Der Extraktion-Prozess der ML-Klassen lässt sich in vier Schritte einteilen, die über die Klasse *MLClasses* aufgerufen werden können.



Zunächst wird die JSON-Datei mit den gescrapten Klassen eingelesen und der Python-Code über die *ast*-Bibliothek in das AST-Format umgewandelt. Die Traversierung des Syntax-Baumes erfolgt nach dem Preoder-Verfahren, sodass jeder Knoten eines Teilbaums vollständig durchlaufen wird, bevor der nächste Knoten auf der gleichen Stufe betrachtet wird. Im ersten Schritt werden alle Import-Objekte herausgefiltert, um festzustellen, ob und wie die ML-Bibliothek verwendet wird. Import-Objekte, die die ML-Bibliothek betreffen,

werden in einer Liste gespeichert und bilden die Basis für die Suche nach den ML-Klassen. Es gibt verschiedene Möglichkeiten wie Elemente einer ML-Bibliothek zugänglich gemacht werden können. Das folgende Codebeispiel zeigt fünf Möglichkeiten, die die Verwendung derselben scikit-learn-Klasse *KNeighborsRegressor* über einen jeweils anderen Pfad im Code ermöglichen.

```
import sklearn                                                    # A
import sklearn as skl                                             # B
from sklearn import neighbors                                     # C
from sklearn.neighbors import KNeighborsRegressor                 # D
from klearn.neighbors import KNeighborsRegressor as KNN           # E

sklearn.neighbors.KNeighborsRegressor()                           # A
skl.neighbors.KNeighborsRegressor()                               # B
neighbors.KNeighborsRegressor()                                   # C
KNeighborsRegressor()                                             # D
KNN()                                                             # E
```

Listing 4.6: Import- und Verwendungsmöglichkeiten der KNeighborsRegressor-Klasse aus scikit-learn

Die Eindeutigkeit des Pfades ist wichtig, um sicherzustellen, dass es sich bei dem jeweiligen Code-Objekt um die Klasse einer ML-Bibliothek handelt. Einige Klassen haben geläufige Bezeichnungen, wie zum Beispiel *Pipeline* aus scikit-learn. Um zu vermeiden, dass man irrtümlicherweise falsche Objekte findet, werden die Code-Objekte aus dem AST auf Basis der Import-Anweisungen der jeweiligen ML-Bibliothek herausgefiltert.

Angenommen das Codebeispiel in Listing 4.7 wird auf Klassen aus der scikit-learn-Bibliothek untersucht.

```
from sklearn.neighbors import KNeighborsRegressor

def knn_func():
    n_neighbors = 6
    knn = KNeighborsRegressor(n_neighbors, leaf_size=25)
    return knn
```

Listing 4.7: Codebeispiel mit KNeighborsRegressor-Klasse

Für das Codebeispiel wird der Syntaxbaum nach Knoten durchsucht, deren Name identisch mit *KNeighborsRegressor* ist und vom AST-Typ *Call* ist. Der AST-Typ versichert, dass ein (Klassen-)Aufruf vorliegt und nicht beispielsweise ein gleichnamiger String ohne Klassenbezug. Ist dies zutreffend, wird geprüft, ob es sich bei diesem Knoten um eine ML-Klasse handelt oder um einen Pfad,

an dessen Ende eine ML-Klasse steht. Ist auch dies gegeben, wird der gesamte Pfad des Objekts beleuchtet, um die Klasse eindeutig zuordnen zu können. Dieser Schritt ist wichtig, da Klassenamen nicht immer eindeutig sind. So verfügt beispielsweise PyTorch über zwei unterschiedliche *profiler*-Klassen. Die eine Klasse ist Teil des *autograd*-Moduls, während die andere zu dem *profiler*-Modul gehört. Beide verfügen über unterschiedliche Konfigurationsoptionen. An dieser Stelle sollte man sich nicht irritieren lassen, dass die beiden Klassen kleingeschrieben sind. Die Namenskonvention, dass Klassen mit Großbuchstaben beginnen, wird von den ML-Bibliotheken nicht immer eingehalten.

Es kann auch der Fall eintreten, dass eine neu implementierte Klasse von einer ML-Klasse erbt. So ist beispielsweise die PyTorch-Klasse *Module* die Basisklasse für künstliche neuronale Netze [Paszke et al., 2019]. Wenn Anwendende ein neuronales Netz implementieren wollen, rufen sie nicht die Module-Klasse auf, sondern erben von ihr. Um solche Fälle zu erfassen, wird beim Traversieren des Syntaxbaumes daher zusätzlich nach Klassendefinition (AST-Typ *ClassDef*) gesucht und geprüft, ob als Basisklasse eine ML-Klasse vorliegt.

Die gefundene Klasse aus Listing 4.7 wird als Listenelement in Form eines Dictionaries gespeichert. Das Dictionary enthält alle benötigten Informationen, um im nächsten Schritt die Konfigurationsoptionen auszulesen. Dazu gehören der vollständige Pfad mit dem Namen der Klasse, der verwendete Pfad inklusive Alias sowie die Datei, indem die Klasse gefunden wurde. Zudem wird das AST-Objekt gespeichert, aus dem sich weitere Informationen wie Parameter und Zeilennummer auslesen lassen. Für das Codebeispiel ergibt sich folgender Listeneintrag:

```
{
    "class": "sklearn.neighbors.KNeighborsRegressor",
    "class_alias": "KNeighborsRegressor",
    "file": "path/file.py",
    "object": <ast.Assign object at 0x106178370>
}
```

Listing 4.8: Dictionary-Eintrag der KNeighborsRegressor-Klasse

4.5 Extraktion und Verarbeitung der Konfigurationsoptionen

Der nächste Vorgang gliedert sich in folgende zwei Teilschritte, bei dem die Ergebnisse aus 4.2 und 4.4 miteinander verbunden werden:

1. Extraktion der Parameterwerte aus AST-Objekt des Dictionaries

2. Zuordnung der Werte zu den gescrapten Parametern aus JSON-Datei

Zunächst wird durch die Liste mit Dictionaries aus 4.4 iteriert. Jedes Dictionary wird zusätzlich mit einem Eintrag zu den gescrapten Parametern der jeweiligen Klasse ergänzt und an die Methode *get_parameters()* der Klasse *MLParameters* übergeben. In dieser Klasse wird der Syntaxbaum des Code-Objekts traversiert, um die übergebenen Parameterwerte in der entsprechenden Reihenfolge zu extrahieren. Die Klasse verfügt über eine Vielzahl an Methoden, die die einzelnen Knoten des Syntaxbaums entsprechend ihres Typs verarbeiten können. Da die gesuchten Werte Parameter von Klassenaufrufen sind, ist die Behandlung von Knoten des AST-Typs *Call* von besonderer Bedeutung. Hinter dem AST-Objekt des Dictionaries aus 4.4 verbirgt sich folgender Syntaxbaum:

```
Assign(
    targets=[
        Name(
            id='knn',
            ctx=Store())],
    value=Call(
        func=Name(
            id='KNeighborsRegressor',
            ctx=Load()),
        args=[
            Name(
                id='n_neighbors',
                ctx=Load())],
        keywords=[
            keyword(
                arg='leaf_size',
                value=Constant(
                    value=25))]))
```

Listing 4.9: AST der KNeighborsRegressor-Klasse

Ein Code-Objekt vom AST-Typ *Call* besteht aus 3 Teilknoten. Der Knoten *func* gibt den Namen des Aufrufs an. Ist an dieser Stelle die *id* des Knoten identisch zur ML-Klasse, werden im nächsten Schritt die Parameter extrahiert. Diese liegen zweigeteilt vor: Im Knoten *args* befinden sich Argumente, die nach ihrer Position übergeben und zugeordnet werden. Im Gegenzug werden im Knoten *keywords* Argumente zusammen mit der Parameterbezeichnung übergeben, sodass direkt zu erkennen ist, um welchen Parameter es sich handelt.

Im gegebenen Beispiel wird die Variable *n_neighbors* als Positionsargument vom AST-Typ *Name* klassifiziert. Für jeden übergebenen Parameter wird der AST-Typ vermerkt, um weitere Informationen für die Analyse von ML-Konfigurationsoptionen bereitzustellen. Wenn Parameter vom AST-Typ *Name* oder *Attribute* vorliegen, werden diese zusätzlich vermerkt, um wie in Kapitel 4.6 beschrieben, mittels Datenflussanalyse mögliche Werte für die Variablen zu ermitteln. Das Keyword-Argument für *leaf_size* kann hingegen direkt zugeordnet werden und ist konstant.

Zusätzlich zur Ermittlung der Parameter wird in der *MLParameters*-Klasse auch geprüft, ob die Instanziierung der Klasse einer Variable zugewiesen wird. Im gegebenen Beispiel wird die Instanz der Klasse (*value*) von der Variablen *knn* (*target*) referenziert. Darüber hinaus kann eine Zuweisung auch über andere Wege als Keyword-Argument erfolgen: bei der Definition einer Klasse oder Funktion und beim Aufruf eines Objekts.

Wie in Kapitel 4.4 wird auch hier der Fall, dass von einer ML-Klasse geerbt wird, gesondert behandelt. Da in diesem Fall keine Klasse aufgerufen wird, sondern auf der Basis einer ML-Klasse eine neue Klasse erstellt wird, werden die Parameter aus der *__init__*()-Methode entnommen. Die __init__()-Methode kann aufgerufen werden, um Attribute einer Klasse zu initialisieren. Über das Schlüsselwort *self* können Werte den Attributen der Klassen zugewiesen werden. Hierbei handelt es sich um die gesuchten Konfigurationsoptionen. Das Listing 4.10 zeigt ein Codebeispiel indem die Klasse *Model* definiert wird und von der PyTorch-Klasse *Module* erbt, um trainierbare Schichten (layers) für ein neuronales Netz zu erstellen. In dem Beispiel handelt es sich bei dem Attribut *self.conv1* um eine Schicht, der die PyTorch-Klasse *Conv2d* zugeordnet wird. Demnach hat die Konfigurationoption *conv1* den Wert *nn.Conv2d(10, 20, 5)*.

```
import torch.nn as nn

class Model(nn.Module):
    def __init__(self):
        self.conv1 = nn.Conv2d(10, 20, 5)
```

Listing 4.10: Vererbung der ML-Klasse Module aus PyTorch

Im zweiten Teil werden die gefundenen Parameter im Code mit den gescrapten aus der JSON-Datei zusammengeführt. Zunächst wird durch die Liste an Positionsargumente iteriert, sodass die Positionsargumente der Reihe nach, den gescrapten Parameter zugeordnet werden können. Für die Keyword-Argumente wird geprüft, ob das Keyword als Parameter vorhanden ist. Bei falschen Keyword-Bezeichnungen, also Keywords, die es als Parameter in der Spezifikation gar

nicht gibt, kann keine Zuordnung zu den gescrapten Parametern erfolgen.

Darüber hinaus gibt es drei Besonderheiten in der Spezifikation, die vom Algorithmus bei der Zuordnung der Parameter beachtet werden müssen: *, **args* und ***kwargs*. Befindet sich in der Spezifikation der Parameter ein Sternchen, dürfen danach keine Positionsargumente mehr übergeben werden. Bei der Spezifikation der Parameter der *KNeighborsRegressor*-Klasse aus Listing 4.2 befindet sich dieses Sternchen an zweiter Stelle. Dementsprechend müssen alle Parameter ab dem zweiten Parameter als Keyword-Argumente übergeben werden. **args* werden verwendet, wenn die Anzahl der Übergabeparameter variabel bleiben soll oder die Parameter in Form eines Tupels oder einer Liste vorliegen. Im ersten Fall kann demnach die Länge der übergebenen Parameter, die Länge der Parameter in der Spezifikation überschreiten. Die *überzähligen* Parameter werden zusammen mit dem letzten übergebenen Parameter als Tupel zusammengefasst. Die Übergabe als Tupel oder als Liste erfolgt mit einem anführenden Sternchen. ***kwargs* hingegen sind eine Möglichkeit, um Dictionaries zu übergeben, die als Schlüssel Parameter der Klasse enthalten können.

Nachdem die Parameter erfolgreich zugeordnet wurden, gibt die *MLParameters*-Klasse ein Dictionary zurück. Es enthält wie das Dictionary aus 4.4 den Namen der Python-Datei, den vollständigen Pfad der Klasse und das AST-Objekt. Ergänzt wird das Dictionary mit Informationen zu den Parametern. Variable Parameter werden zusätzlich gesondert in einem Dictionary aufgeführt. Da die möglichen Werte der variablen Parameter noch unbekannt sind, werden diese mit *None* gekennzeichnet. Sofern vorhanden, wird die Variable, die die Klasse referenziert, ebenfalls aufgeführt, sodass sich für das Codebeispiel aus 4.7 folgendes Dictionary ergibt:

```
{
    "class": "sklearn.neighbors.KNeighborsRegressor",
    "file": "path/file.py",
    "object": <ast.Assign object at 0x106178370>,
    "parameter": {
        "leaf_size": {
            "type": "Constant",
            "value": 25
        },
        "n_neighbors": {
            "type": "Name",
            "value": "n_neighbors"
        }
    },
    "variable": "knn",
    "variable_parameters": {
        "n_neighbors": None
    }
}
```

Listing 4.11: Dictionary-Eintrag der KNeighborsRegressor-Klasse inklusive Parameter

4.6 Ermittlung variabler Konfigurationswerte

Zunächst wird durch die Liste mit Dictionaries aus Kapitel 4.5 iteriert. Die Ermittlung der variablen Parameterwerte erfolgt über die *get_parameter_value()*-Methode der *DataFlowAnalysis*-Klasse. Beim Aufruf der Methode wird für dieses Beispiel das Dictionary zusammen mit der Parametervariable *n_neighbors* übergeben. Die anschließende Datenflussanalyse lässt sich in zwei Bereich einteilen:

1. Sammeln von Informationen: Kontrollflüsse, Aufrufspfade und Funktionsaufrufe
2. Analyse der Informationen mittels SSA, Constant Propagation und AST

Im ersten Schritt wird über die *CFGBuilder*-Klasse des statischen-Analyse-Frameworks *Scalpel* der Kontrollflussgraph für die Python Datei erstellt und in einer Liste gespeichert. Der Kontrollflussgraph stellt alle globalen Pfade dar, die während der Ausführung des Programms durchlaufen werden können. Da Klassen- und Funktionskörper sich jedoch nicht auf dieser Ebene befinden, wird zusätzlich jede Klasse und Funktion der globalen Ebene traversiert

und die Kontrollflussgraphen dieser Objekte ebenfalls in der Liste gespeichert. Dieser Vorgang wiederholt sich für jede weitere Ebene bis schließlich alle Kontrollflussgraphen der Klassen und Funktionen der Liste hinzugefügt wurden.

Im nächsten Schritt werden die möglichen Pfade, mit denen man die einzelnen Klassen und Funktionen erreichen kann, ermittelt. Dieser Schritt ist erforderlich, da die Werte gesuchter Parametervariablen möglicherweise über Klassen und Funktionen hinweg übergeben werden. Der Aufrufspfad einer Funktion oder einer Klasse variiert jedoch je nachdem von wo sie aufgerufen werden. Im folgenden sind vier Beispiele aufgelistet, die zeigen, dass es notwendig ist, die Objektzugehörigkeit und die damit verbundenen Aufrufspfade zu kennen.

- *func()*: ruft globale Funktion auf
- *self.func()*: Funktion befindet in einer Klasse und wird innerhalb dieser Klasse aufgerufen
- *C.func()*: Funktion befindet sich in Klasse C und wird von außerhalb der Klasse aufgerufen
- *C(1)*: ruft die $\_\_init\_\_$-Methode der Klasse C auf

Für jede Klasse und jede Funktion wird daher ein Dictionary erstellt, dessen Werte die möglichen Aufrufspfade sind. Ohne Kenntnisse der Aufrufspfade würde ansonsten bei der Datenflussanalyse beispielsweise der übergebene Wert im vierten Fall nicht der richtigen Methode zugeordnet werden können.

Im Anschluss wird durch die Liste der Kontrollflussgraphen iteriert und alle enthaltenen Anweisungen als AST geparst. Der abstrakte Syntaxbaum wird traversiert, um Funktionsaufrufe herauszufiltern und in einer Liste zu speichern. Nachdem alle Kontrollflussgraphen durchlaufen wurden, enthält die Liste sämtliche Funktionsaufrufe der Datei im AST-Format. Falls nun die gesuchte Variable als Parameter in der Definition einer Funktion steht, können auf Basis der möglichen Pfade der Funktion, die Funktionsaufrufe gefunden werden. Dadurch kann der Wert einer Variable bzw. eines Parameters über den gesamten Kontrollfluss der Datei verfolgt werden.

Nachdem im ersten Teil die notwendigen Informationen gesammelt wurden, werden diese im zweiten Teil analysiert. Für jeden Kontrollflussgraphen der Datei werden über die *compute_SSA()*-Methode der Scalpel-Klasse *SSA* Zuweisungen und Werte der enthaltenen Variablen ermittelt. Dafür werden anhand der Methode die Datenflussanalyse-Techniken *Static Single Assignment* und *Constant Propagation* verwendet. Das Ergebnis ist eine Liste an Dictionaries. Die Schlüssel sind Variablen, die Ziel einer Zuweisung sind, während die

Werte des Dictionaries den jeweiligen Wert der Zuweisung entsprechen. Wird der gesuchten Variable ein Wert zugewiesen, wird dieser in der Liste möglicher Werte gespeichert. Handelt es sich zudem um einen variablen Wert vom AST-Typ *Name* oder *Attribute*, wird im Anschluss auch für diesen Wert der gesamte zweite Teil der Datenflussanalyse ausgeführt.

In diesem Zusammenhang stößt man an mehrere Grenzen des Scalpel-Frameworks, die im folgenden Codebeispiel dargestellt werden:

```
class A:
    def __init__(self):
        a, b = 1, 2                          # issue 1
        c = 3                                # issue 2
        self.cls_variable = c                # issue 3
        c = 4
```

Listing 4.12: Limitierungen der Datenflussanalyse

Die aktuelle Scalpel-Version (vom 02.06.22) kann keine Zuweisungen an Tupel verarbeiten, sodass für die Variablen *a* und *b* von derzeit keine Wertzuweisungen ausgegeben werden (*issue 1*). Zudem wird in der aktuellen Version nur die letzte Zuweisung verarbeitet. Für die Variable *c* beträgt der Wert bei der Ausgabe demnach *4*, obwohl der Variable vorher ein anderer Wert zugewiesen wird (*issue 2*). Das größte Problem ist jedoch, dass in der aktuellen Version von Scalpel Zuweisungen an Objekte des AST-Typs *Attribute* nicht verarbeitet werden und der Wert für *self.cls_variable* nicht ermitteln werden kann (*issue 3*). Da die Lösung dieses Problems essenziell für die Datenflussanalyse ist, wurde der Source Code der *compute_SSA()*-Methode untersucht und so angepasst, dass nun auch Zuweisungen an Instanzvariablen wie *self.cls_variables* verarbeitet werden können. Leider beträgt der Wert für *c* wie in *issue 2* beschrieben zum Zeitpunkt der Zuweisung der SSA-Analyse von Scalpel *4*, obwohl *3* richtig ist. Für diese Probleme wurden jeweils ein Issue, teilweise mit Lösungsansatz, im Git-Repository von Scalpel erstellt. Zum Zeitpunkt der Fertigstellung dieser Arbeit wurden diese Probleme zwar zur Kenntnis genommen, jedoch noch nicht behoben.

Im nächsten Schritt werden die Funktionsdefinitionen der Datei untersucht, um zu überprüfen, ob die gesuchte Variable als Parameter übergeben wird. Ist die gesuchte Variable in der Definition der Funktion vorhanden, müssen einige Besonderheiten bei der Behandlung der Argumente berücksichtigt werden. Die Parameter einer Funktion können fünf unterschiedliche AST-Typen besitzen: *arg*, *vararg*, *kwonlyargs*, *posonlyarg* oder *kwarg*. Ähnlich wie die Argumente des Typs *ast.Call* aus Kapitel 4.5 müssen die AST-Typen unterschiedlich behan-

delt werden, damit die Parametervariable beim Aufruf der Funktion eindeutig zugeordnet werden kann. Im folgenden Codebeispiel werden die verschiedenen Typen mit ihrem Namen kenntlich gemacht:

```
def func(posonlyarg1, posonlyarg2, /, arg1, arg2,
         arg3 = 1, *vararg, kwonlyarg1, kwonlyarg2, **kwarg)
    :
     return

func("posonlyarg1", "posonlyarg2", arg2=2, *["arg1"],
      kwonlyarg1=1, kwonlyarg2=2, **{"arg3": 3, "newarg": 0})
```

Listing 4.13: Parametertypen von Funktionen

Positional-only arguments sind Argumente, die vor der /-Markierung stehen und dürfen nicht mit einem Keyword-Argument übergeben werden. Das Gegenstück dazu sind *Keyword-only arguments*. Sie müssen mit einem Keyword übergeben werden und befinden sich in der Definition immer nach *varargs*. *Args* sind Argumente bei denen nicht festgelegt ist, ob sie mit einem Keyword übergeben werden. Sie werden ansonsten ihrer Position nach zugeordnet. Die Typen *vararg* und *kwarg* referenzieren die übergebenen **args*- und ***kwargs*-Parameter, die in Kapitel 4.5 besprochen wurden. Ist die gesuchte Variable in der Definition der Signatur vorhanden, besteht die Möglichkeit, dass ihr wie bei *arg3* ein Default-Wert zugewiesen wird, der dementsprechend zur Liste möglicher Werte der Parametervariablen hinzugefügt werden muss [Peng et al., 2021].

Im letzten Schritt der Datenflussanalyse wird, sofern die gesuchte Variable ein Funktionsparameter ist, durch die Liste mit Funktionsaufrufen iteriert. Jeder Funktionsaufruf wird mit den möglichen Werten aus der Liste an Dictionaries mit möglichen Aufrufspfaden verglichen und einer Funktion zugeordnet. Für jeden Aufruf der Funktion wird je nach Argumenttyp und Position der gesuchten Parametervariable der übergebene Wert ausgelesen. Handelt es sich bei einem übergebenen Wert erneut um eine Variable, wird im Anschluss auch für diesen Wert der gesamte zweite Teil der Datenflussanalyse wiederholt.

Nach erfolgreichem Durchlauf gibt die *get_parameter_value()*-Methode der *DataFlowAnalysis*-Klasse eine Liste mit möglichen Werten zurück. Die Liste wird im Anschluss in ein Dictionary umgewandelt, dessen Schlüssel aufsteigende Zahlen sind. Schlussendlich wird eine JSON-Datei erstellt, die alle gesuchten Konfigurationsoptionen des Git-Repositories enthält. Für das *KNeighborsRegressor*-Beispiel sieht der Eintrag wie folgt aus:

```
{
    "file": "path/file.py",
    "class": "sklearn.neighbors.KNeighborsRegressor",
    "parameter": {
        "leaf_size": {
            "type": "Constant",
            "value": 25
        },
        "n_neighbors": {
            "type": "Name",
            "value": "n_neighbors"
        }
    },
    "variable_parameters": {
        "n_neighbors": {
            "type": "Constant",
            "value": 26
        }
    },
    "variable": "knn",
    "line_no": 4
}
```

Listing 4.14: JSON-Eintrag der KNeighborsRegressor-Klasse mit möglichen Parameterwerten

Kapitel 5

Evaluation

In diesem Kapitel werden die Ergebnisse aus Kapitel 4 evaluiert. Zunächst werden die Resultate des Web Scraping-Moduls beleuchtet, bevor im Anschluss der vorgestellte Ansatz als Ganzes evaluiert wird. Die Verwendung von Konfigurationsoptionen von ML-Bibliotheken im Source Code ist jedoch nicht umfassend erforscht. Daher gibt es keine Datensätze oder Vergleichswerte, um die Genauigkeit beurteilen zu können. Dementsprechend wird der Ansatz auf zwei verschiedene Arten überprüft:

1. Tests
2. Analyse von Projekten

Tests sind ein gängiges Mittel in der Softwareentwicklung. Sie dienen der Qualitätssicherung, um die Funktionalität des Systems zu gewährleisten und zu untersuchen, inwieweit die gestellten Anforderungen an eine Software auch erfüllt werden. In der *tests.py*-Datei des Git-Repositorys zu dieser Arbeit befinden sich mehrere Codebeispiele mit Konfigurationsoptionen, die aus verschiedenen Softwareprojekten entnommen wurden [Jaeger, 2022]. Zudem werden auch weitere ausgedachte Codebeispiele getestet, um zu prüfen, ob der entwickelte Ansatz auch komplexere Herausforderungen bewältigen kann. Ziel ist es zu prüfen, ob die verwendeten ML-Klassen inklusive der Parameter richtig erkannt und zugeordnet werden. Zudem wird getestet, ob die Werte variabler Parameter, die teilweise auch über Klassen und Funktionen hinweg weitergegeben werden, bestimmt werden können. Mit Hilfe des *pytest*-Frameworks werden die gefundenen Konfigurationsoptionen des entwickelten Ansatzes, mit dem tatsächlichen Ergebnis verglichen. Falls sich die Ergebnisse unterscheiden, gibt pytest eine Fehlermeldung aus.

Um darüber hinaus eine aussagekräftige Bewertung abzugeben, wird sich am Evaluationsansatz von Rabkin and Katz orientiert. Dafür werden verschiedene Softwareprojekte mit dem vorgestellten Ansatz analysiert und das Er-

gebnis mit einer manuellen Analyse verglichen. Für die manuelle Analyse wird der Source Code auf die Verwendung von Konfigurationsoptionen der zu untersuchenden ML-Bibliothek händisch untersucht.

5.1 Ergebnisse Web Scraping

ML-Bibliothek	Version	Laufzeit (min)	Klassen	Parameter	P/K
TensorFlow	2.9.1	27:15	844	3728	4,4
PyTorch	1.11.0	0:37	408	1340	3,3
scikit-learn	1.1.1	0:31	262	2081	7,9

Tabelle 5.1: Scraping Ergebnisse der ML-Dokumentation

In der Tabelle 5.1 wird für jede ML-Bibliothek die Anzahl an Klassen und Parameter sowie die durchschnittliche Anzahl der Parameter je Klasse (P/K) für die jeweilige Version dargestellt. Zudem wird die Laufzeit des Web-Scraping-Vorgangs gemessen. Die angegebenen Zeiten sind Mittelwerte aus fünf Durchläufen der Anwendung und können je nach Internetverbindung und Rechnerleistung stark variieren. Sie wurden mit einer LAN-Verbindung des Universitätsnetzwerks in Leipzig und einem MacBook Pro mit M1 Pro Chip, 10-Core CPU und 16 Gigabyte Arbeitsspeicher erzielt. Die Anzahl der Klassen steht dabei in Relation zur Dauer des Scraping-Prozesses. Während das Web Scraping der Klassen aus der scikit-learn-Dokumentation die geringste Laufzeit hatte, dauerte der Prozess bei TensorFlow mit Abstand am längsten. Weitere Einflussfaktoren auf die Laufzeit sind zudem die Dokumentations- und die Modulstruktur. Während in TensorFlow und scikit-learn für jede Klasse eine neue Website aufgerufen werden muss, um die Parameter zu extrahieren, sind bei PyTorch teilweise mehrere Klassen inklusive der Parameter auf einer Seite aufgelistet. TensorFlow und PyTorch weisen zudem eine hohe Zahl an Modulen auf und oft müssen mehrere Module durchlaufen werden, um zu einer Klasse zu gelangen. Sciki-learn hingegen verfügt über eine eher flache Modultiefe, sodass die Klassen meist über einen kürzeren Pfad erreicht werden können. Des Weiteren weisen die Klassen von scikit-learn im Schnitt eine deutlich höhere Zahl an Parametern und somit mehr Konfigurationsmöglichkeiten auf.

Zu beachten ist, dass Erfolg und Reproduzierbarkeit des Scraping-Moduls stark mit der Website-Struktur der Dokumentation zusammenhängen, da das Modul nicht Robust gegenüber Änderungen ist. Ändert sich das Layout indem beispielsweise HTML-Elemente anders strukturiert werden, so ist es möglich,

dass der Scraping-Algorithmus nicht mehr die gewünschten Ergebnisse liefert. Gleichzeitig müssen insgesamt drei TensorFlow- und sechs PyTorch-Klassen einzeln zum Scraping-Ergebnis hinzugefügt werden, da sie nicht wie die anderen Klassen in die Dokumentation eingebettet und mit den gleichen HTML-Tags versehen sind.

5.2 Auswahl Subject Systems

Die Auswahl der zu untersuchenden Softwaresysteme erfolgt nach folgenden Kriterien:

- Open Source Git Repository mit mindestens 1.000 Sternen
- Verwendung von mindestens zehn Klassen einer ML-Bibliothek
- möglichst Verwendung von variablen Parametern beim Aufruf der Klassen
- Code ist syntaktisch fehlerfrei

Ein vergebener Stern für ein Git-Repository kann sowohl als Lesezeichen als auch als Zeichen der Wertschätzung für das Projekt dienen. Aus der Anzahl an Sternen lassen sich somit Aussagen über die Popularität eines Projektes ableiten. Daher eignet sie sich als Auswahlkriterium für die Erstellung einer repräsentativen Liste an Softwareprojekten, die die ML-Bibliotheken benutzen. Um bei der Analyse auch möglichst viele verschiedene Fälle von ML-Konfigurationsoptionen abzudecken, müssen mindestens zehn Klassen einer ML-Bibliothek verwendet werden. Im besten Fall werden den ML-Klassen auch variable Parameter übergeben, um die Genauigkeit des Datenflussanalyse-Moduls untersuchen zu können. Damit der Code als AST geparst werden kann, muss er zudem syntaktisch fehlerfrei sein.

Für jede der drei ML-Bibliotheken werden jeweils fünf Softwareprojekte untersucht, die in der Tabelle 5.2 aufgelistet werden.

Projekt	ML-Bibliothek	letzter Commit	Sterne
DeepSpeech		17.11.21	19.7k
Fast Style Transfer		18.09.21	10.5k
Darkflow	TensorFlow	25.02.20	6k
Text Classification		21.07.18	5.5k
Tensorforce		10.02.22	3.1k
Voice Cloning		01.03.22	35k
YOLOv3		04.04.22	6.7k
Image Captioning	PyTorch	03.06.20	2k
BatchNorm		08.04.21	1.4k
TorchIO		25.05.22	1.4k
AutoGluon		03.06.22	4.5k
igel		06.02.22	3k
MLJAR	scikit-learn	14.04.22	1.9k
KServe		03.06.22	1.5k
sklearn-porter		23.05.22	1.2k

Tabelle 5.2: Untersuchte Softwareprojekte

Tensorforce ist ein *Deep Reinforcement Learning*-Framework, das auf Googles TensorFlow beruht [Kuhnle et al., 2017]. *CNN for Text Classification* und *Fast Style Transfer* basieren ebenfalls auf TensorFlow und verwenden sich faltende künstliche neuronale Netzwerke (*Convolutional Neural Networks*) zur Textklassifizierung [Kim, 2014] und für eine schnelle Stilübertragung von berühmten Gemälden auf beliebige Fotos [Engstrom, 2016]. Mit *DeepSpeech* von Mozilla lässt sich Sprache zu Text auf einem Raspberry Pi verarbeiten [Mozilla, 2022] und mit *Darkflow* Objekte in Echtzeit erkennen und klassifizieren [Trieu, 2018].

Bei *Real-Time Voice Cloning* handelt es sich um ein Deep Learning Framework und ist das erste von fünf analysierten PyTorch-Projekten [Jemine, 2022]. In *YOLOv3* ist der YOLO-Algorithmus zur Bilderkennung in Echtzeit implementiert [Redmon and Farhadi, 2018]. *PyTorch Tutorial to Image Captioning* ist eine Anleitung zur Bildbeschriftung [Vinodababu, 2020]. *TorchIO* bietet eine Reihe von Deep Learning Werkzeugen zur medizinischen Bildgebung [Pérez-García et al., 2021] und *Synchronized Batch Normalization* befasst sich mit synchronisierter Batch Normalisierung zur Objekterkennung in PyTorch [Tete Xiao, 2021].

AutoGluon verwendet scikit-learn und automatisiert ML-Aufgaben, um ML-Anwendungen mit geringerem Aufwand entwickeln zu können [Erickson et al., 2020]. *Igel* ist ein ML-Tool mit man Modelle trainiert und verwendet, ohne Code schreiben zu müssen [Baccouri, 2022]. Bei *MLJAR* handelt

es sich um ein Python-Paket für *AutoML* auf tabellarische Daten [Płońska and Płoński, 2021]. *KServe* stellt eine benutzerdefinierte Ressourcendefinition für ML-Modelle beliebiger Frameworks auf *Kubernetes* bereit [Authors, 2022]. Mit *sklearn-porter* werden Schätzer von scikit-learn in andere Programmiersprachen übersetzt [Morawiec, 2022].

5.3 Analyse Subject Systems

Für die Evaluation wird für jedes der 15 Softwareprojekte auf das Auftreten von Konfigurationsoptionen der enthaltenen ML-Bibiliothek einmal manuell und mit dem vorgestellten Ansatz untersucht. Das Ergebnis beider Analysen wird am Ende verglichen, um die Genauigkeit des entwickelten Ansatzes beurteilen zu können.

Im Rahmen der manuellen Analyse wird in jeder Python-Datei des Softwareprojekts untersucht, ob und welche Elemente der ML-Bibliothek importiert werden. Sind Elemente der ML-Bibliothek importiert, wird in der jeweiligen Python-Datei nach ihnen gesucht. Bei jedem gefundenen Objekt wird geprüft, ob es sich um eine Klasse handelt. Dies erfolgt durch einen Abgleich der Beschreibung des Objekts in der Dokumentation oder über die Definition des Objekts im Source Code der ML-Bibliothek. Die gefundenen Klassen werden inklusive der Parameter in einer separaten Textdatei gespeichert. Werden variable Parameter übergeben, erfolgt zusätzlich eine Suche nach möglichen Wertzuweisungen für die Variablen in der Python-Datei. Bei der Suche wird berücksichtigt, dass die Variablen auch über Aufrufe von Klassen und Funktionen übergeben werden können oder ihnen ebenfalls ein variabler Wert zugewiesen werden kann. Für diese Fälle werden die übergebenen Werte der jeweiligen Aufrufe betrachtet oder nach Zuweisungen für die neue Variable gesucht. Die gefundenen Zuweisungen werden ebenfalls in separaten Textdatei gespeichert.

Die Ergebnisse werden in der Tabelle 5.3 zusammengefasst. Für jedes der fünfzehn Softwareprojekte ist die Anzahl der gefundenen Klassen und Anzahl der (variablen) Parameter angegeben. Darüber hinaus sind für die variablen Parameter in der letzten Spalte die Anzahl der möglichen Werte anhand der Wertezuweisungen aufgeführt. An dieser Stelle sollte man sich nicht irritieren lassen, dass, wie bei Fast Style Transfer, die Anzahl der variablen Parameter die Anzahl der Wertzuweisungen übersteigt. Dies liegt daran, dass viele variable Konfigurationswerte bei der Initialisierung von Klassen oder Aufruf von Funktionen übergeben werden. Diese Aufrufe liegen jedoch nicht immer in derselben Python-Datei oder werden teilweise erst zur Laufzeit ermittelt. Sie sind daher für die Datenflussanalyse in diesem Ansatz außerhalb des Betrachtungsrahmens.

ML-Projekt	Klassen	Parameter	variable Parameter	
			gesamt	Zuweisungen
DeepSpeech	34	48	24	31
Fast Style Transfer	18	10	4	3
Darkflow	15	10	7	9
Text Classification	21	30	11	4
Tensorforce	61	123	58	96
Voice Cloning	107	274	100	99
YOLOv3	31	67	28	35
Image Captioning	32	77	34	26
BatchNorm	14	26	4	3
TorchIO	30	50	36	31
AutoGluon	81	168	52	31
igel	12	32	34	26
MLJAR	33	151	22	29
KServe	21	21	-	-
sklearn-porter	34	34	23	123

Tabelle 5.3: Ergebnis der manuellen Analyse

Im zweiten Schritt wird der entwickelte Ansatz auf die fünfzehn Softwareprojekte angewendet. Dies erfolgt indem in der Kommandozeile der Befehl aus Listing 4.4 angepasst für das jeweilige Repository und die jeweilige ML-Bibliothek ausgeführt wird. Zuvor wurden bereits über den Kommandozeilenbefehl aus Listing 4.3 die Klassen und Parameter der ML-Bibliotheken aus den Dokumentationen extrahiert. Die im Ausgabeergebnis erzeugte JSON-Datei wird mit den gefundenen Klassen, Parametern und Zuweisungen der manuellen Analyse verglichen. Für jedes gefundene Objekt der manuellen Analyse erfolgt eine Prüfung, ob es auch vom entwickelten Ansatz gefunden wird und das Ergebnis in der Tabelle 5.4 zusammengefasst. Die Tabelle folgt demselben Schema wie die Tabelle 5.3. Sie gibt jedoch eine Prozentzahl an, die die Genauigkeit des entwickelten Ansatzes zeigt. Die Prozentzahl berechnet sich dabei aus dem Anteil der gefundenen Objekte im Vergleich zur manuellen Analyse. Darüber hinaus werden die Laufzeiten der Analysen angegeben. Die erste Spalte gibt die Dauer der Analyse an, wenn das Repository noch aus GitHub geklont werden muss und die zweite Spalte gibt die Dauer für den Fall an, dass das Repository bereits lokal verfügbar ist.

ML-Projekt	Klassen	Parameter	variable Parameter		Laufzeit	
			gesamt	Zuweisungen	online	lokal
DeepSpeech	100%	98%	100%	100%	6,0 s	1,9 s
Fast Style Transfer	100%	100%	100%	100%	1,2 s	0,2 s
Darkflow	100%	100%	100%	100%	2,7 s	0,5 s
Text Classification	100%	100%	100%	100%	0,8 s	0,3 s
Tensorforce	100%	100%	100%	79%	6,9 s	4,3 s
Voice Cloning	100%	100%	100%	97%	19,6 s	4,2 s
YOLOv3	100%	100%	100%	100%	2,0 s	0,7 s
Image Captioning	100%	100%	100%	96%	1,6 s	0,6 s
BatchNorm	100%	100%	100%	100%	0,5 s	0,1 s
TorchIO	100%	100%	100%	100%	3,8 s	1,2 s
AutoGluon	100%	100%	100%	100%	5,5 s	4,0 s
igel	100%	100%	100%	100%	2,1 s	0,5 s
MLJAR	100%	100%	100%	93%	2,0 s	1,1 s
KServe	100%	100%	-	-	8,3 s	1,3 s
sklearn-porter	100%	100%	100%	100%	2,6 s	1,8 s

Tabelle 5.4: Ergebnis des entwickelten Ansatzes: Anteil gefundener Objekte gegenüber manueller Analyse und Laufzeit

Der Vergleich der beiden Analysen zeigt, dass der Algorithmus alle Klassen und fast alle Parameter erkennen kann. Schwächen hat das Programm lediglich bei der Erkennung von Zuweisungen. Dies hat zwei Ursachen, die in Listing 4.12 dargestellt sind. Die aktuelle Version von Scalpel (vom 02.06.22) kann im Rahmen des *Static Single Assignment* Zuweisungen an Tupel nicht verarbeiten (*issue 1*). Bei Tensorforce ist dies elf- und bei Real-Time Voice Cloning einmal der Fall. Die zweite Ursache ist die Verarbeitung mehrerer Zuweisungen an die gleiche Variable innerhalb eines Funktionskörpers (*issue 2*). Scalpel gibt aktuell nur den letzten Zuweisungswert als möglichen Wert der Variable aus. Die vorherigen Zuweisungen des Funktionskörpers werden nicht ausgegeben. Neun weitere Zuweisungen von Tensorforce, zwei von Real-Time Voice Cloning, zwei von MLJAR und eine Zuweisung vom PyTorch Tutorial to Image Captioning werden dadurch nicht erkannt. Die beiden Probleme wurden bereits den Entwicklern gemeldet und werden derzeit bearbeitet. Darüber hinaus wird in DeepSpeech einer Instanz der TensorFlow-Klasse *SparseTensor* erzeugt und der Parameter **transcript* übergeben. Dass als Übergabeparameter eine Liste als **args* übergeben werden darf, ist aus der Dokumentation nicht ersichtlich und wurde vom entwickelten Ansatz daher nicht erkannt und verarbeitet.

5.4 Diskussion

Die Analyse der Softwareprojekte zeigt, dass der entwickelte Ansatz die Verwendung von ML-Klassen und Parameter mit einer sehr hohen Genauigkeit erkennt. Die Kombination aus den extrahierten Daten aus den Dokumentationen der ML-Bibliotheken und der Verarbeitung des Source Codes als AST harmoniert sehr gut, um die verwendeten Konfigurationsoptionen zu identifizieren.

Hinsichtlich der Wertebestimmung variabler Parameter werden ebenfalls viele verschiedene Fälle abdeckt. Gleichwohl unterliegt diese Wertebestimmung einigen Limitierungen, die auch das Ergebnis der manuellen Analyse beeinträchtigen. Es gibt jedoch Möglichkeiten, den Algorithmus zu verbessern an denen teilweise, wie beim Analyse-Frameworks Scalpel, bereits gearbeitet wird. So werden nur Zuweisungen betrachtet, weshalb die Zahl möglicher Werte der Variablen deutlich höher sein kann. Jede weitere Operation, die auf eine Variable angewendet wird, wird in diesem Ansatz nicht weiter erfasst. Verbirgt sich hinter der Variable zum Beispiel eine Liste oder ein Dictionary, so wird nur der Wert der Variablen bei der Initialisierung bestimmt. Weitere Operationen, wie das Hinzufügen neuer Elemente an die Liste oder Einträge in das Dictionary, werden nicht verarbeitet.

Parametern, die nicht vom AST-Typ *Name* oder *Attribute* sind, können zudem keine Werte zugewiesen werden. Sie werden im Rahmen der Datenflussanalyse grundsätzlich nicht weiter verfolgt oder ausgewertet. Des Weiteren werden nur Klassen- und Funktionsaufrufe betrachtet, die sich innerhalb derselben Python-Datei befinden wie die Parametervariable. Wenn sich also die gesuchte Variable in der Definition einer Klasse oder einer Funktion befindet und der Wert beim Aufruf dieser Objekte von außerhalb der Datei übergeben wird, liegt dies außerhalb des Betrachtungsrahmens. An dieser Stelle gibt es für den Ansatz Verbesserungspotenzial. Ebenfalls nicht behandelt werden Parametervariablen, deren Wert erst durch die Eingabe während der Ausführung eines Programms oder im Dictionary eines Funktionsdekorators bestimmt wird. Da alle gleichnamigen Variablen erfasst werden, kann es jedoch auch vorkommen, dass die Datenflussanalyse Zuweisungen ermittelt, die für die gesuchte Parametervariable keine Bedeutung haben. So kann es sein, dass sich eine solche Variable zum Beispiel nur lokal in einer separaten Funktionen befindet.

Verglichen mit anderen Ansätzen, die sich mit Konfigurationsoptionen in Softwaresystemen beschäftigen, bewegt sich dieser Ansatz in einem noch unerforschtem Terrain. Dies kann auf die Tatsache zurückzuführen sein, dass die Popularität von Python und die Nachfrage nach ML-Anwendungen erst in den letzten 10-15 Jahren stark gestiegen ist [Srinath, 2017]. Nachteilig ist, dass es für den entwickelten Ansatz daher keine Benchmarks für die Evaluierung gibt.

Eine manuellen Analyse als Vergleichswert kann nur eine Notlösung sein, da stets das Risiko von Flüchtigkeitsfehlern besteht. Gleichzeitig überrascht es, dass es im Bereich der Datenflussanalyse in Python keine große Auswahl umfangreicher Frameworks gibt. Der Fokus der Frameworks liegt überwiegend auf die Kontrollflussgraphen und nicht auf die Wertebestimmung von Variablen. Scalpel scheint noch eines der ausgereifteren Werkzeuge zu sein, auch wenn hier die angesprochenen Limitierungen greifen. Viele Wertezuweisungen werden nur gefunden, weil zusätzlich Workarounds geschrieben wurden. Dementsprechend gibt es noch Potenzial für die Entwicklung von Datenflussanalyse-Frameworks in Python, die sich insbesondere mit Constant Propagation und Static Single Assignment beschäftigen und verschiedene Fälle abdecken, ohne das Nutzende selber noch Methoden ergänzen müssen. Grundsätzlich konnte dieses Vorgehen jedoch viele Fälle zur Bestimmung variabler Konfigurationswerte abdecken.

Hinsichtlich der Laufzeit des entwickelten Ansatz fällt vor allem die Größe des Git-Repositorys ins Gewicht. Um den Prozess zu beschleunigen, besteht zum Beispiel die Möglichkeit, dass aus dem Git-Repository nur die Python-Dateien geklont werden, da der Rest für diese Analyse nicht relevant ist. Sobald jedoch das Git-Repository lokal geklont ist, beträgt die Laufzeit für ein untersuchtes Softwareprojekt nur wenige Sekunden. Verbunden mit der unkomplizierten Bedienung über die Kommandozeile bietet der vorgestellte Ansatz eine kostengünstige Analyse von ML-Projekten.

Mit diesem Ansatz kann man in kurzer Zeit eine große Anzahl von Softwaresystemen im Hinblick auf die Verwendung von ML-Klassen und Hyperparametern in kurzer Zeit untersuchen. Die Ergebnisse können genutzt werden, um die Konfiguration von ML-Algorithmen grundlegend zu analysieren.

Kapitel 6

Fazit

Konfigurationsoptionen haben einen entscheidenden Einfluss auf die Ausführung des Source Codes und bieten Nutzenden die Möglichkeit, eine Software an ihre Bedürfnisse anzupassen. Sie führen in der Softwareentwicklung jedoch auch zu einer Reihe von Problemen. So bleiben Konfigurationsfehler oft unentdeckt oder sind schwer nachvollziehbar. In Python und vor allem im Machine Learning ist die Verwendung von Konfigurationsoptionen allerdings noch wenig erforscht. In der vorliegenden Arbeit wurde ein Ansatz entwickelt, der es ermöglicht, Konfigurationsoptionen von ML-Bibliotheken aus Python Source Code zu extrahieren. Der Ansatz zeigt, dass mittels statischer Code-Analyse, Konfigurationsoptionen mit einer hohen Genauigkeit identifiziert werden können.

Konfigurationsoptionen liegen bei maschinellen Lernalgorithmen in Form von sogenannten Hyperparametern vor, um beispielsweise das Trainieren von Modellen zu steuern. Um sie in vielen möglichen Softwaresystemen zu identifizieren, werden dafür Informationen zu Klassen und Parametern der ML-Bibliotheken TensorFlow, PyTorch und scikit-learn aus den Dokumentationen mittels Web Scraping extrahiert. Nachdem der Source Code von ML-Projekten als Abstract Syntax Tree geparst wird, werden durch die Traversierung des Syntaxbaumes die ML-Algorithmen und Optionen erkannt. Darüber hinaus können mit Hilfe von Techniken aus der Datenflussanalyse auch bei variablen Hyperparametern mögliche Konfigurationswerte bestimmt werden. Hilfreich ist hierfür die Kombination aus AST und den Analysemethoden Static Single Assignment und Constant Propagation des Scalpel-Frameworks. Allerdings unterliegt der Ansatz insbesondere bei der Ermittlung variabler Konfigurationswerte einigen Limitierungen. Obwohl viele verschiedene Fälle erfasst werden, zeigt der Ansatz auch Verbesserungsmöglichkeiten, um ein noch detaillierteres Bild möglicher Konfigurationswerte zu erhalten.

Während der Web Scraping-Prozess für die spezifische Struktur der Do-

kumentation einer ML-Bibliothek angepasst werden muss, erfolgt die Code- und Datenflussanalyse generisch. So lässt sich der Ansatz auf weitere ML-Bibliotheken mit nur geringem Aufwand übertragen. Gleichzeitig weist der Ansatz eine geringe Laufzeit auf. Das Web Scraping dauert je nach Dokumentationsstruktur unterschiedlich lang. Die Dauer der statischen Analyse und des Extraktionsprozesses bewegt sich jedoch nur im einstelligen Sekundenbereich.

Abschließend lässt sich festhalten, dass mit dem vorgestellten Ansatz nützliche Informationen bei der Verwendung von ML-Bibliotheken in Python mit geringem Aufwand extrahiert werden können. Aus diesen Informationen lassen sich bei der Analyse von ML-Projekten weitere Erkenntnisse gewinnen. Dieser Ansatz ermöglicht einen Einblick darüber, wie Machine Learning Algorithmen konfiguriert sind.

Anhang

A.1 Abstract Syntax Tree

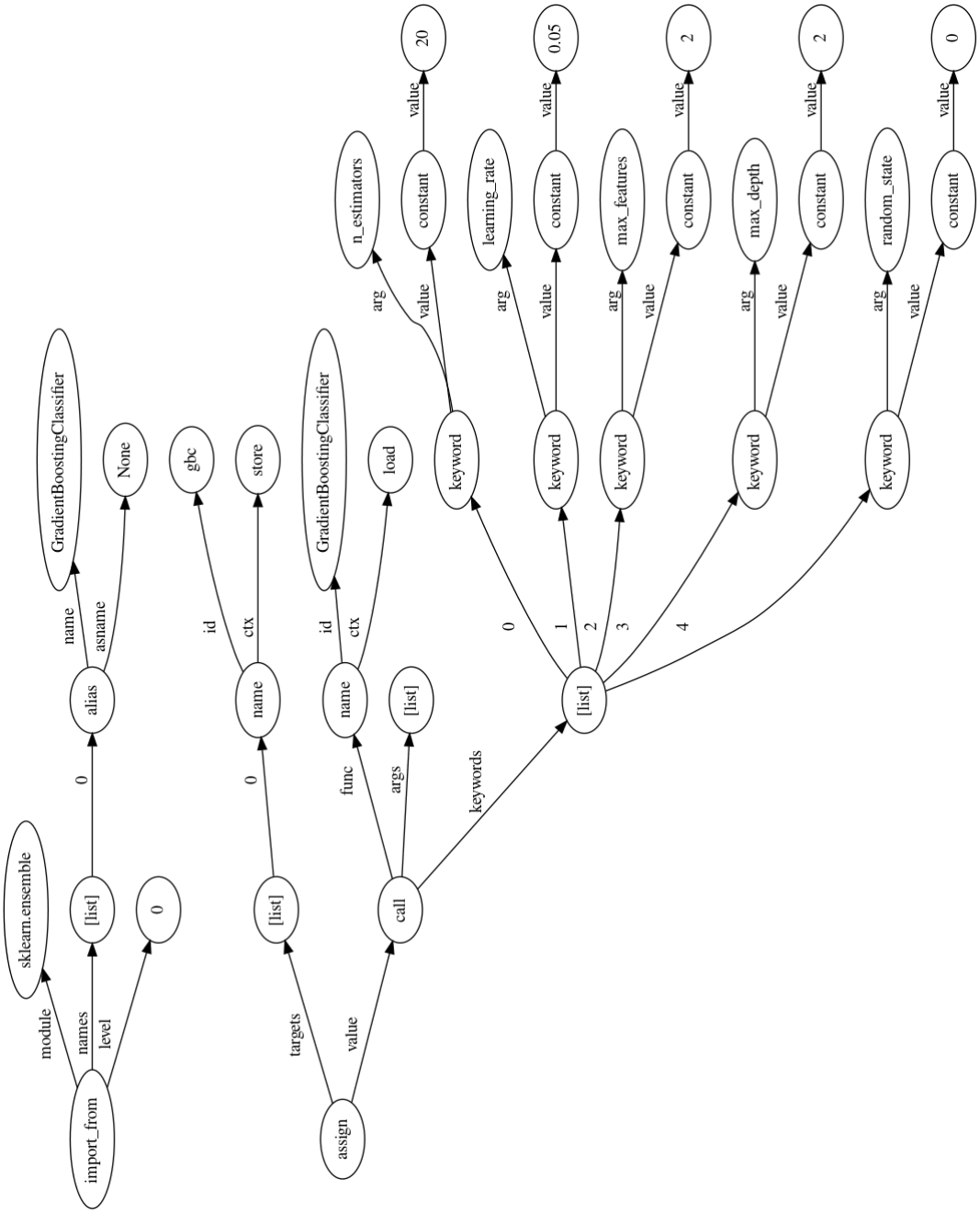


Abbildung A.1: AST der GradientBoostingClassifier-Klasse aus scikit-learn

Literaturverzeichnis

Răzvan Andonie. Hyperparameter optimization in learning systems. *Journal of Membrane Computing*, 279–291, 2019. doi: 10.1007/s41965-019-00023-0. https://doi.org/10.1007/s41965-019-00023-0. 1.1, 2.3.2

The KServe Authors. Kserve. 2022. https://github.com/kserve/kserve. Online; accessed 05-June-2022. 5.2

John Aycock. Compiling little languages in python. In *Proceedings of the 7th International Python Conference*, pages 69–77, 1998. http://pages.cpsc.ucalgary.ca/~aycock/papers/ipc7-compiler.pdf. 2.5

Nidhal Baccouri. igel. 2022. https://github.com/nidhaloff/igel. Online; accessed 05-June-2022. 5.2

Alexandru G Bardas et al. Static code analysis. *Journal of Information Systems & Operations Management*, 4(2):99–107, 2010. http://www.rebe.rau.ro/RePEc/rau/jisomg/WI10/JISOM-WI10-A10.pdf. 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. https://doi.org/10.1145/2786805.2786869. 3

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, 3

Logan Engstrom. Fast style transfer. `https://github.com/lengstrom/fast-style-transfer/`, 2016. Online; accessed 05-June-2022. 5.2

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. `https://github.com/awslabs/autogluon`, 2020. Online; accessed 05-June-2022. 5.2

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. 1.1

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. `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

Xue Han, Tingting Yu, and Michael Pradel. Confprof: White-box performance profiling of configuration options. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '21, page 1–8, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450381949. doi: 10.1145/3427921.3450255. `https://doi.org/10.1145/3427921.3450255`. 1.1, 3

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. `https://proceedings.mlr.press/v32/hutter14.html`. 2.3.2

Marco Jaeger. ML configuration options. `https://github.com/mj-support/ml-config-options`, 2022. Online; accessed 10-June-2022. 5

Corentin Jemine. Real-time-voice-cloning. `https://github.com/CorentinJ/Real-Time-Voice-Cloning`, 2022. Online; accessed 05-June-2022. 5.2

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. `https://doi.org/10.1145/2642937.2643009`. 3

Kaggle. State of machine learning and data science 2021. `https://www.kaggle.com/kaggle-survey-2021`, 2021. Online; accessed 05-April-2022. (document), 1.2, 2.3.1, 4.1

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. 2.3.1

Yoon Kim. Convolutional neural networks for sentence classification, 2014. `https://arxiv.org/abs/1408.5882`. 5.2

Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. `https://github.com/tensorforce/tensorforce`. Online; accessed 05-June-2022. 5.2

Li Li, Jiawei Wang, and Haowei Quan. Scalpel: The python static analysis framework. *arXiv preprint arXiv:2202.11840*, 2022. `https://github.com/SMAT-Lab/Scalpel`. Online; accessed 08-June-2022. (document), 2.2, 9, 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, 3

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

Darius Morawiec. sklearn-porter. Transpile trained scikit-learn estimators to C, Java, JavaScript and others, 2022. `https://github.com/nok/sklearn-porter`. Online; accessed 05-June-2022. 5.2

Mozilla. Deepspeech. `https://github.com/mozilla/DeepSpeech`, 2022. Online; accessed 05-June-2022. 5.2

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. `https://doi.org/10.3102/1076998619872761`. 4.1

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. `https://arxiv.org/abs/1912.01703`. 4.4

Yun Peng, Yu Zhang, and Mingzhe Hu. An empirical study for common language features used in python projects. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 24–35, 2021. doi: 10.1109/SANER50967.2021.00012. 4.6

Fernando Pérez-García, Rachel Sparks, and Sébastien Ourselin. Torchio: a python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning. *Computer Methods and Programs in Biomedicine*, page 106236, 2021. ISSN 0169-2607. doi: https://doi.org/10.1016/j.cmpb.2021.106236. `https://www.sciencedirect.com/science/article/pii/S0169260721003102`. 5.2

Aleksandra Płońska and Piotr Płoński. Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3, 2021. `https://github.com/mljar/mljar-supervised`. Online; accessed 05-June-2022. 5.2

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

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. `https://doi.org/10.1145/1985793.1985812`. 1.1, 1.2, 3, 5

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. `http://dx.doi.org/10.3390/info11040193`. 1.2, 2.3, 2.3.1, 4.1

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. `https://github.com/eriklindernoren/PyTorch-YOLOv3`, 2018. Online; accessed 05-June-2022. 5.2

Leonard Richardson. Beautiful soup documentation. `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`, 2018. Online; accessed 05-June-2022. 4.2

KR Srinath. Python–the fastest growing programming language. *International Research Journal of Engineering and Technology (IRJET)*, 4(12):354–357, 2017. 5.4

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. 2.3.1

DTennant Tete Xiao. Synchronized batch normalization. `https://github.com/vacancy/Synchronized-BatchNorm-PyTorch`, 2021. Online; accessed 05-June-2022. 5.2

Trinh Hoang Trieu. Darkflow. `https://github.com/thtrieu/darkflow`, 2018. Online; accessed 05-June-2022. 5.2

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. `https://doi.org/10.1145/2786984.2786995`. 4.1

Sagar Vinodababu. a pytorch tutorial to image captioning. `https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning`, 2020. Online; accessed 05-June-2022. 5.2

Bo Zhao. Web scraping. *Encyclopedia of big data*, pages 1–3, 2017. doi: 10.1007/978-3-319-32001-4_483-1. 2.4, 2.4