FAKULTÄT FÜR INFORMATIK
Faculty of Informatics

Source-Code-Gruppierung mittels Machine Learning für die automatische Identifizierung einer Software-Architektur

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Sebastian Störchle
Matrikelnummer 0825711

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuer: Thomas Grechenig
Mitwirkung: Johann Grabner

Wien, 29.01.2020

(Unterschrift Verfasser) (Unterschrift Betreuer)

Source Code Grouping using Machine Learning for the automatic Identification of a Software Architecture

Master's Thesis

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Sebastian Störchle
Registration Number 0825711

elaborated at the
Institute of Information Systems Engineering
Research Group for Industrial Software
to the Faculty of Informatics
at TU Wien

Advisor: Thomas Grechenig
Assistance: Johann Grabner

Vienna, 29.01.2020

Technische Universität Wien, Forschungsgruppe INSO
A-1040 Wien ▪ Wiedner Hauptstr. 76/2/2 ▪ Tel. +43-1-587 21 97 ▪ www.inso.tuwien.ac.at

Eidesstattliche Erklärung

Sebastian Störchle
Sigmundgasse 10, 3121 Karlstetten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

I hereby declare that I am the sole author of this thesis, that I have completely indicated all sources and help used, and that all parts of this work – including tables, maps and figures – if taken from other works or from the internet, whether copied literally or by sense, have been labelled including a citation of the source.

(Ort, Datum) (Unterschrift Verfasser)

Kurzfassung

Das Einarbeiten in Softwareprojekte ist mit sehr hohem Aufwand verbunden. Einen wichtigen Aspekt, um sich in einem Softwareprojekt zu Recht zu finden, bildet die projektbezogene Softwarearchitektur.

Die Motivation, diese Diplomarbeit zu verfassen, war ein Hilfswerkzeug anzubieten, welches Aufschlüsse über die projektbezogene Softwarearchitektur bereitstellt, um so das Verständnis über die projektbezogene Softwarearchitektur zu fördern. Dies soll die Einarbeitungsphase in ein Softwareprojekt effizienter gestalten.

In dieser Arbeit wird ein Programm vorgestellt, welches automatisiert durch die Angaben der Quelltextdateien und der Programmiersprache des Softwareprojektes, die projektbezogene Softwarearchitektur ableitet. Dabei befasst sich diese Arbeit konkret mit dem Entwurf, der Implementierung und der Evaluierung der Ergebnisse dieses Programmes.

Auf Basis einer Literaturrecherche werden mittels einer Anforderungsanalyse die Anforderungen an die Applikation definiert. Die Implementierung dieses Programmes wurde mit einem iterativen Entwicklungsmodell durchgeführt. Dabei wurde schrittweise die Entwicklung der Applikation optimiert und erweitert, bis das Programm alle Anforderungen der Zielsetzung erfüllen konnte.

Die Evaluierung wurde mittels einer Befragung von fachkundigen Personen aus dem Bereich des Software Engineering durchgeführt. Bei der Evaluierung konnte sowohl die Sinnhaftigkeit des Ansatzes dieser Diplomarbeit validiert werden, als auch Evidenz geliefert werden, dass der Cluster-Algorithmus die Quelltextdateien ähnlich zuordnet wie fachkundige Personen aus dem Bereich des Software Engineering. Durch die Resultate der prototypischen Implementierung dieser Diplomarbeit kann das Verständnis der projektbezogene Softwarearchitektur verbessert werden.

Schlüsselwörter

Software maintenance, Software architecture, Clustering methods

Abstract

The incorporation of software developers in software projects takes a lot of effort. An important aspect of working on a software project is the knowledge about the project-related software architecture.

The motivation to write this thesis was to provide a tool for software developers, which learns the project-related software architecture and helps developers to get started more efficiently.

This work presents a program which uses the source code files and the programming language as an input and derives the project-related software architecture.

This work deals specifically with the design, the implementation and the evaluation of the results of this application.

A requirement analysis based on a literature search is used to define and analyze the requirements of this program. The whole application was developed with an iterative development model.

The development of the program was gradually optimized and expanded until the program fulfilled all the requirements of the predefined goals.

The evaluation took place as a survey of experts. The meaningfulness of the approach of this thesis got validated as well as the evidence that the cluster algorithm assigns the source code files in a similar way as the experts. Due to the results of the prototypical implementation of this thesis the comprehension of the project-related software architecture can be improved.

Keywords

Software maintenance, Software architecture, Clustering methods

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Problemstellung . . . 1
  - 1.1.1 Features aus Quelltext auslesen . . . 3
  - 1.1.2 Cluster und Anzahl der Cluster erkennen . . . 4
  - 1.1.3 Bewertung der Resultate . . . 4
- 1.2 Motivation . . . 4
- 1.3 Zielsetzung . . . 5
- 1.4 Methodik . . . 6
- 1.5 Aufbau der Arbeit . . . 6

2 Grundlagen **7**
- 2.1 Künstliche Intelligenz . . . 7
- 2.2 Maschinelles Lernen . . . 8
- 2.3 Überwachtes Lernen . . . 8
  - 2.3.1 Lineare Regression . . . 9
- 2.4 Unüberwachtes Lernen . . . 9
- 2.5 Clustering . . . 10
  - 2.5.1 K-Means . . . 12
  - 2.5.2 Density-Based Spatial Clustering of Applications with Noise . . . 15
  - 2.5.3 Kennzahlen eines Clusters . . . 17
- 2.6 Dimensionsreduktion . . . 19
  - 2.6.1 Hauptkomponentenanalyse (engl. PCA (Principal Component Analysis)) 19

3 Stand der Technik **22**
- 3.1 Clustering durch semantische Ähnlichkeit mit Wikipedia . . . 22
- 3.2 Clustering durch Identifizieren von Themen . . . 23
- 3.3 Weiter wissenschaftliche relevante Dokumente . . . 26
  - 3.3.1 SDVisu . . . 26
  - 3.3.2 Software Features Extraction von objektorientierten Quelltext mittels Clustering . . . 27
  - 3.3.3 Segmentierung von Software Komponenten mittels Feature Vector . . . 28

4 Anforderungsanalyse **30**
- 4.1 Anforderungen an die Applikation . . . 30
- 4.2 Eigenschaften einer Quelltextdatei . . . 32
  - 4.2.1 Metadaten . . . 33
  - 4.2.2 Text . . . 33
  - 4.2.3 Abstrakter Syntaxbaum . . . 35
  - 4.2.4 Formatierung . . . 37
- 4.3 Preprocessing und Feature engineering . . . 38

5 Implementierung **40**
- 5.1 Iteratives Entwicklungsmodell . . . 40
- 5.2 Initiale Planung . . . 41

5.3 Iteration 1: Bag of word, tf-idf und k-Means mit Clusteranzahl 42
5.3.1 Requirementanalyse der ersten Iteration 43
5.3.2 Entwurf der ersten Iteration 43
5.3.3 Implementierung der ersten Iteration 46
5.3.4 Evaluierung und Test der ersten Iteration 48
5.4 Iteration 2: Visualisierung der Cluster 50
5.4.1 Requirementanalyse der zweiten Iteration 50
5.4.2 Entwurf der zweiten Iteration 51
5.4.3 Implementierung der zweiten Iteration 52
5.4.4 Evaluierung und Test der zweiten Iteration 54
5.5 Iteration 3: Automatisches Erkennen der Clusteranzahl 58
5.5.1 Requirementanalyse der dritten Iteration 59
5.5.2 Entwurf der dritten Iteration 59
5.5.3 Implementierung der dritten Iteration 60
5.5.4 Evaluierung und Test der dritten Iteration 63
5.6 Iteration 4: Optimiertes Feature engineering und preprocessing 63
5.6.1 Requirementanalyse der vierten Iteration 63
5.6.2 Entwurf der vierten Iteration 64
5.6.3 Implementierung der vierten Iteration 67
5.6.4 Evaluierung und Test der vierten Iteration 69

6 Evaluierung **71**
6.1 Entwurf der Studie zur Evaluierung der Zuordnung 71
6.1.1 Entwurf der Studie zur Bedarfsanalyse 71
6.1.2 Entwurf der Studie zur Evaluierung der Ergebnisse der Kommandozeilen-applikation 74
6.2 Ausführung der Studie zur Evaluierung der Zuordnung 77
6.3 Analyse der Ergebnisse der Studie zur Evaluierung der Zuordnung 78
6.3.1 Ergebnisse der Studie zur Bedarfsanalyse 78
6.3.2 Ergebnisse der Studie zur Evaluierung der Resultate der Applikation . . . 83
6.3.3 Forschungsvalidität 84
6.3.4 Fallbeispiel eines Softwareprojektes 84

7 Zusammenfassung und Ausblick **87**
7.1 Ausblick 87
7.1.1 Unterstützung mehrerer Programmiersprachen 87
7.1.2 Integration 88
7.1.3 Visualisierung 88
7.1.4 Features der Feature Extraction erweitern 88
7.1.5 Verbesserungsvorschläge in der Softwarearchitektur 89
7.1.6 Unterstützung bei Quelltext Restrukturierung 89

Literatur **90**
Wissenschaftliche Literatur 90
Online-Referenzen 93

Abbildungsverzeichnis

1.1 Populärsten zwanzig Programmiersprachen nach dem RedMonk Ranking von September 2012 bis Juni 2019 [72]. . . . 2
1.2 Populärsten zehn Programmiersprachen nach dem TIOBE Index von 2002 bis 2019 [92]. . . . 2
1.3 Überblick über den Prozess „Preprocessing und Feature engineering“ . . . 3
1.4 Interesse des Begriffes „Machine Learning“ seit 2004. (Data source: Google Trends [31]) . . . 5
1.5 Überblick über das Kommandozeilenprogramm, welches das Resultat dieser Diplomarbeit ist. . . . 5

2.1 Einordnung wichtige Begriffe in das Themengebiet Künstliche Intelligenz (KI) . . . 7
2.2 Beispielgraph einer Linearen Regression . . . 9
2.3 Vergleich verschiedener Algorithmen für Clustering. Abbildung von [78] . . . 11
2.4 K-Means Clustering Beispiel mit Stufen für das Finden der Cluster. [10] . . . 15
2.5 Anwendung des DBSCAN-Algorithmus anhand eines Beispiels. . . . 16
2.6 Abbildungen zur Veranschaulichung einer Dimensionsreduktion mittels PCA von 3 Dimensionen auf 2. . . . 21

3.1 Übersicht der auszuführenden Schritte des Segmentierungsverfahrens nach Kuhn, Ducasse und Girba [49] . . . 23
3.2 Umwandlung einer Matrix in einem Vektorraum mittels LSI. [49] . . . 24
3.3 Beispiel von „correlation matrices“ um das Verfahren von Kuhn, Ducasse und Girba zu erklären. Von links nach rechts: ungeordnete „correlation matrix“ sortiert nach Ähnlichkeit, gruppiert nach Cluster und mit „semantic links“ [49] . . . 25
3.4 Illustration eines „semantic links“. Ein Dokument in Cluster A ist ähnlicher zu Cluster B, als seine Nachbarn in Cluster A zu Cluster B. [49] . . . 25
3.5 Semantische Cluster mit top 7 Wörter des jeweiligen Cluster des Softwareprojektes JEdit. [49] . . . 26
3.6 Screenshot der Applikation SDVisu [71] . . . 27
3.7 Prozess des Verfahrens nach Araar et al. [5] . . . 28

4.1 Design der Baumstruktur Visualisierung eines fiktiven Software Projektes . . . 31
4.2 Design der Liste Visualisierung eines fiktiven Software Projektes . . . 31
4.3 Mindmap für Eigenschaften eines Softwareprojektes . . . 32
4.4 Kategorisierung von Dateien in Text- und Binärdateien . . . 34
4.5 Lexikalische Analyse anhand eines Beispiels [2] . . . 35
4.6 Lexikalische Analyse anhand eines Beispiels [2] . . . 36
4.7 Übersicht und Zusammenhänge der „Preprocessing und Feature engineering“ Prozesse 39

5.1 Phasen des iteratives Entwicklungsmodells . . . 40
5.2 Top 10 „Machine Learning“ Programmiersprachen, Bibliotheken und Projekte auf der Plattform GitHub [28] im Jahr 2018 [91] . . . 42
5.3 Beispiel des „Bag of words“-Verfahrens, angewandt auf drei JavaScript Quelltextdateien 44
5.4 Entwurf der ersten Iteration um Quelltextdateien Clustern zuzuordnen. . . . 46

5.5 Design der Visualisierung der Zuordnungen von Quelltextdateien zu Clustern im 2-dimensionalen und 3-dimensionalen Raum 51
5.6 Design der Visualisierung der Zuordnungen von Quelltextdateien zu Clustern im 2-dimensionalen und 3-dimensionalen Raum 56
5.7 Visualisierung der Zuordnung von Quelltextdateien zu Clustern in der Listenansicht . 57
5.8 Visualisierung der Zuordnung von Quelltextdateien zu Clustern in der Baumansicht . 58
5.9 Anforderungen an die dritte Iteration im iterativen Entwicklungsmodells. 59
5.10 Darstellung der Eigenschaften, welche in „Preprocessing und Feature engineering“ in der vierten Iteration berücksichtigt werden. 64
5.11 Darstellung des „Preprocessing und Feature engineering“ der Struktur des Quelltextes anhand eines Beispiels. 66
5.12 Ergebnis der Zuordnung des Java-Softwareprojektes Jenkins [42] in einem 3-dimensionalen und 2-dimensionalen Raum. 70

6.1 Screenshot der Umfrage zur Bedarfsanalyse. 73
6.2 Screenshot der Umfrage zur Evaluierung des Ergebnisses der Applikation. 75
6.3 Screenshot der Umfrage zur Evaluierung des Ergebnisses der Applikation, mit bereits zugewiesenen und eingefärbten Clustern 76
6.4 Ergebnisse der ersten Frage der Umfrage zur Bedarfsanalyse 79
6.5 Ergebnisse der zweiten Frage der Umfrage zur Bedarfsanalyse 79
6.6 Ergebnisse der dritten Frage der Umfrage zur Bedarfsanalyse 80
6.7 Ergebnisse der vierten Frage der Umfrage zur Bedarfsanalyse 80
6.8 Ergebnisse der fünften Frage der Umfrage zur Bedarfsanalyse 81
6.9 Boxplot zu den Antworten der Bedarfsanalyse. 82
6.10 Anzahl der gleichen Zuweisung wie die Applikation. 83
6.11 Ergebnisse der Umfrage nach identen Zuweisungen von Quelltextdateien zu Clustern der Befragten mit der Applikation 84
6.12 Listenansicht des Ergebnisses des Fallbeispiels pinstagram [65]. 86
6.13 Ergebnisses des Fallbeispiels pinstagram [65] abgebildet in einem 3-dimensionalen Raum. 86

Tabellenverzeichnis

1.1 Populärsten zwanzig Programmiersprachen nach dem PYPL Ranking von Juni 2019 [66]. 3

3.1 Component Attribute Matrix [85] 28
3.2 Beispiel einer „Similarity Matrix“ für die „Feature Vectors“ aus Tabelle 3.1. [85] . . 29

6.1 Tabelle zur Erklärung der Punktevergabe des Bewertungssystems nach Wahrscheinlichkeiten. 77
6.2 Tabelle zur Berechnung von Beispiel Zuweisungen nach dem Bewertungssystems nach Wahrscheinlichkeiten. 77
6.3 Auswahl der Softwareprojekte, die für die Umfrage zur Evaluierung der Ergebnisse der Kommandozeilenapplikation herangezogen wurden. 78
6.4 Auswahl der Softwareprojekte, die für die Umfrage zur Evaluierung der Ergebnisse der Kommandozeilenapplikation herangezogen wurden. 82

Liste der Listings

2.1 DBSCAN Algorithmus Implementierung in der Programmiersprache Python 17

3.1 Beispiel einen Quelltextes als Veranschaulichung des Preprocessing des Segmentierungsverfahrens nach Kuhn, Ducasse und Girba. [49] 23
3.2 Beispiel einer resultierenden Liste nach dem Preprocessing des Segmentierungsverfahren nach Kuhn, Ducasse und Girba. [49] 23

4.1 Beispiel eines Quelltextes zur Veranschaulichung eines abstrakten Syntaxbaumes in der Programmiersprache JavaScript. 36
4.2 Unformatierter Beispiel-Quelltext in JavaScript 37
4.3 Formatierter Beispiel-Quelltext in JavaScript 38

5.1 Implementierung der ersten Iteration: Bag of words mit Tf-idf und k-Means 48
5.2 Beispielausgabe des Kommandozeilenprogrammes, das aus dieser Iteration resultiert. 50
5.3 Änderungen der $main.py$ Datei um die Visualisierungen aus dem $cluster\_output$ Modul anzuzeigen. 52
5.4 Auszug der Methode $plot\_2d\_3d\_graph$ aus dem Quelltext des Python Modules $cluster\_output$ 54
5.5 Auszug des Python Modules $kmeans\_auto\_k$, welches automatisch die Anzahl der Cluster ableitet. 61
5.6 Quelltext der Elbow-Methode aus dem Modul $kmeans\_auto\_k$. 62
5.7 Ausschnitt aus der $main.py$ Quelltextdatei um die Änderungen zu den vorherigen Iterationen darzustellen. 62
5.8 Quelltext des Modul $preprocess\_structure$ um die Struktur des Quelltextes in Eigenschaften abzubilden. 68
5.9 Ausschnitt des Quelltext der $main.py$ Datei um die Änderungen zu den vorherigen Iterationen hervorzuheben. 69

Abkürzungen

DBSCAN Density-Based Spatial Clustering of Applications with Noise

KI Künstliche Intelligenz

KNN Künstliche neuronale Netze

LSI Latent Semantic Indexing

NMI Normalized mutual information

PCA Principal Component Analysis

PYPL PopularitY of Programming Language

VCS Version Control System

1 Einleitung

Untersuchungen zeigen, dass Software Wartung durchschnittlich 60 Prozent der Kosten eines Softwareprojektes verursacht [29]. Für die Wartung eines Softwareprojektes benötigt man jedoch Wissen über das gesamte Projekt. Unter anderem braucht man Kenntnisse über die Softwarearchitektur. Bei dieser gibt es jedoch von Projekt zu Projekt große Unterschiede, selbst wenn sie die selben Programmiersprachen und „Frameworks “ verwenden. Dieses erschwert auch die Einarbeitungszeit in das Softwareprojekt für neue Programmierer. Detaillierte und aktuelle Dokumentation über das Softwareprojekt würde diese Probleme verringern, aber diese Dokumentation muss ebenfalls gewartet und geschrieben werden. Diese Diplomarbeit beschäftigt sich mit dem automatisierten Erlernen der projektspezifischen Softwarearchitektur. Im Rahmen dieser Diplomarbeit wurde ein Programm geschrieben, das als Eingabe Quelltextdateien eines Software Projektes bekommt und als Resultat Cluster dieser Quelltextdateien zurückliefert. Diese Cluster geben Aufschluss über die projektspezifische Softwarearchitektur. Dadurch soll eine automatisierte Generierung von Visualisierungen und Dokumentation der projektspezifischen Softwarearchitektur ermöglicht werden.

1.1 Problemstellung

Das Ziel dieser Diplomarbeit ist es, aus bestehendem Quelltext eines Softwareprojektes, die projektspezifische Softwarearchitektur automatisiert abzuleiten. Dieser Vorgang soll für Softwareprojekte verschiedener Programmiersprachen ermöglicht werden. Dazu soll im Rahmen dieser Diplomarbeit ein Prototyp entwickelt werden. Dieser Prototyp soll als Kommandozeilenprogramm realisiert werden. Als Argumente soll der Prototyp die Programmiersprache und die Quelltextdateien des Softwareprojektes erhalten. Als Resultat wird eine Datei erwartet, in der jeder Quelltextdatei ein Cluster zugewiesen ist. Cluster sind Gruppen von Quelltextdateien, die eine gewisse Ähnlichkeit zueinander aufweisen. Quelltextdateien eines Clusters sind sich demnach ähnlicher, als zu Quelltextdateien eines anderen Clusters. Dadurch kann ein Programm in Softwarekomponenten aufgeteilt und die Softwarearchitektur abgeleitet werden. Folgende drei Programmiersprachen sollen von dem Prototypen akzeptiert werden:

- Java
- Python
- JavaScript

Diese Auswahl der Programmiersprachen wurde anhand der drei populärsten Programmiersprachen aus dem PopularitY of Programming Language (PYPL) [66] und „RedMonk Programming Language Rankings “ [72] getroffen. Abbildung 1.1 zeigt den Verlauf der Popularität der populärsten 20 Programmiersprachen nach dem RedMonk Rankings von September 2012 bis Juni 2019. Dabei kombiniert das RedMonk Ranking die Rankings der Programmiersprachen von den zwei bekannten Webseiten GitHub [28] und StackOverflow [86]. Tabelle 1.1 listet die zwanzig populärsten Programmiersprachen nach dem PYPL Ranking von September 2019. PYPL misst die Popularität der Programmiersprachen nach Suchanfragen von Tutorials auf Google [30]. Ein weiterer Index, der die Popularität von Programmiersprachen darstellt ist der TIOBE Index [92]. Dabei wird die Suche der Namen der Programmiersprache als Popularität gemessen. Es wird die

Suche von jenen Webseiten verwendet, welche gemäß Alexa Internet [4] die wichtigsten Internetseiten mit Suchfunktion repräsentieren. Abbildung 1.2 zeigt den Verlauf des Indexes von 2002 bis 2019 an. Auch bei diesem Index sind zwei der drei ausgewählten Programmiersprachen in den ersten drei Plätzen für das Jahr 2019 vertreten. Da dieser Index umstritten ist, wurde dieser nur indirekt bei der Wahl der unterstützenden Programmiersprachen miteinbezogen.

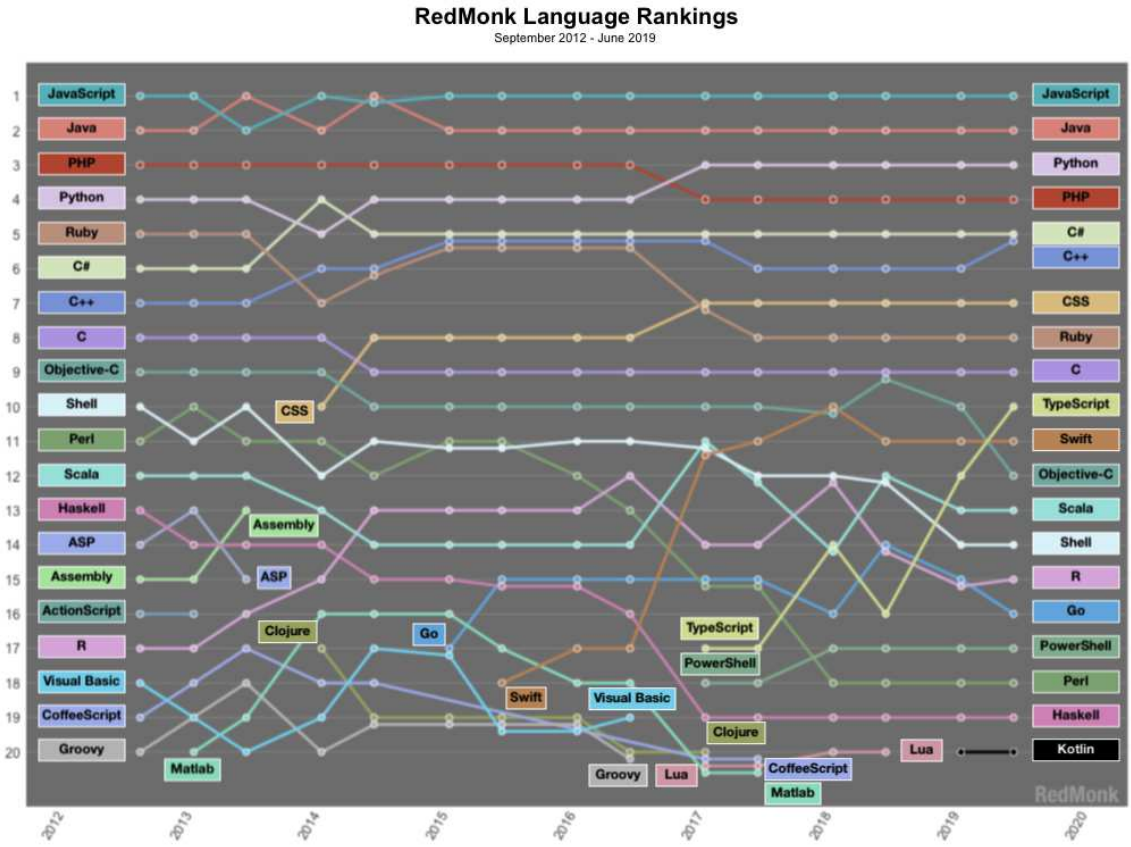


Abbildung 1.1: Populärsten zwanzig Programmiersprachen nach dem RedMonk Ranking von September 2012 bis Juni 2019 [72].

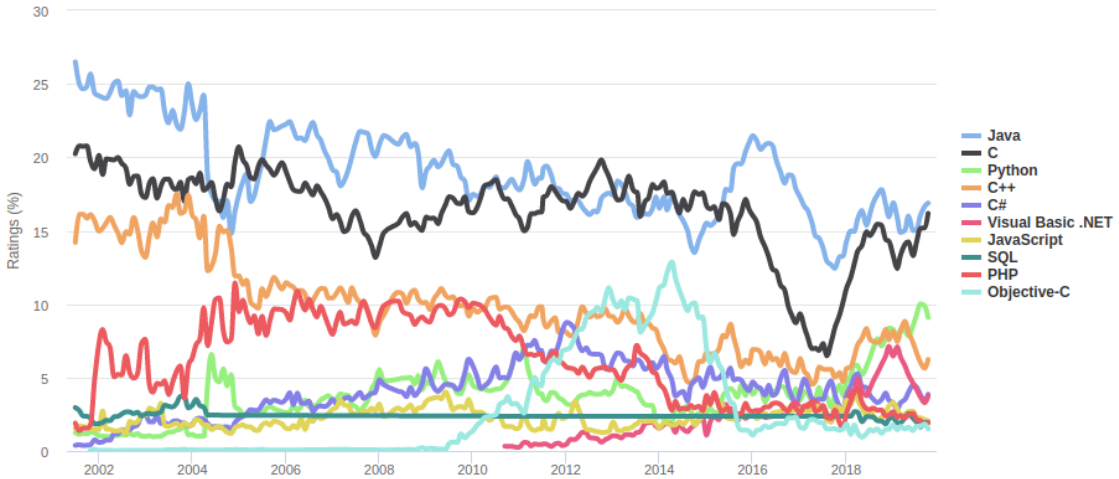


Abbildung 1.2: Populärsten zehn Programmiersprachen nach dem TIOBE Index von 2002 bis 2019 [92].

Rank	Language	Share
1	Python	29.21%
2	Java	19.9%
3	JavaScript	8.39%
4	C#	7.23%
5	PHP	6.69%
6	C/C++	5.8%
7	R	3.91%
8	Objective-C	2.63%
9	Swift	2.46%
10	Matlab	1.82%
11	TypeScript	1.77%
12	Kotlin	1.55%
13	VBA	1.44%
14	Ruby	1.4%
15	Go	1.2%
16	Sala	1.14%
17	Visual Basic	1.05%
18	Rust	0.66%
19	Perl	0.53%
20	Lua	0.37%

Tabelle 1.1: Populärsten zwanzig Programmiersprachen nach dem PYPL Ranking von Juni 2019 [66].

Um diesen Prototypen erfolgreich entwickeln zu können, muss im Zuge dieser Diplomarbeit eine Reihe von Problemen gelöst werden. Dieses Kapitel beschreibt die Probleme, die für eine erfolgreiche Realisierung gelöst werden müssen.

1.1.1 Features aus Quelltext auslesen

Bevor ein Verfahren aus dem Bereich „Machine Learning“ eingesetzt wird, müssen die Daten in ein bestimmtes, für die Maschine lesbares Format, umgewandelt werden. Dieser wichtige Schritt wird als „Preprocessing und Feature engineering“ bezeichnet und in Abbildung 1.3 dargestellt.

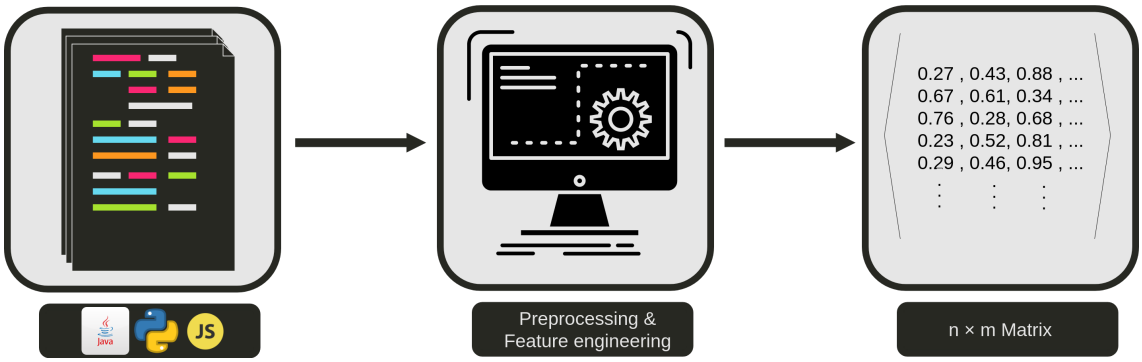


Abbildung 1.3: Überblick über den Prozess „Preprocessing und Feature engineering“

Das für die Maschine lesbare Format ist eine Matrix von Zahlen. Diese Matrix wird rechts auf der Abbildung 1.3 symbolisiert. Welche Dimensionen die Matrix hat, hängt von der Anzahl der Quelltextdateien, die zu dem Cluster zugeordnet werden sollen und von der Anzahl der Features

pro Quelltextdatei ab. Dabei ist zu beachten, dass die Matrix nicht zu groß wird und keine redundanten Daten beinhaltet, um Überanpassung und falsche Prioritäten beim Clustering zu vermeiden [34].

Im Rahmen dieser Diplomarbeit muss ein Algorithmus gefunden werden, der aus n Quelltextdateien eine $n \times m$ Matrix ableiten kann, wobei m die Anzahl der gefundenen Features darstellt. Jeder Wert a_{ij} dieser Matrix entspricht einer Zahl. Viele Verfahren aus dem Bereich des „Machine Learning“ erwarten auch normalisierte Daten, um Abstände korrekt zu berechnen. Die Quelltextdateien eines Projektes, für welches die projektspezifische Softwarearchitektur gefunden werden soll, werden links auf der Abbildung 1.3 dargestellt. Der Algorithmus des „Preprocessing und Feature engineering“, welcher im Rahmen dieser Diplomarbeit entwickelt werden muss, ist in der Mitte in Abbildung 1.3 dargestellt.

1.1.2 Cluster und Anzahl der Cluster erkennen

Nachdem m Features der n Quelltextdateien in einer $n \times m$ Matrix abgebildet sind, können Verfahren angewandt werden, um jede Quelltextdatei $Q_1, Q_2, ..., Q_n$ einem Cluster $C_1, C_2, ..., C_k$ zuzuordnen. Dabei muss nicht nur ein passendes Verfahren für das Clustering gefunden werden, sondern auch der Wert k, der die Anzahl der Cluster definiert.

1.1.3 Bewertung der Resultate

Nachdem die Cluster und die dazugehörige Anzahl der Cluster gefunden worden sind, muss ein Weg gefunden werden, diese Ergebnisse wissenschaftlich zu evaluieren. Im Rahmen dieser Diplomarbeit werden die Schritte dokumentiert, die zu einer wissenschaftlichen Evaluierung führen. Darüber hinaus werden die Ergebnisse analysiert und beschrieben.

1.2 Motivation

In der heutigen Zeit gibt es so viele Aufzeichnungen von Daten wie noch nie zuvor. Mit Technologien wie Smartphones und Tablets hat die Menschheit Möglichkeiten geschaffen, rund um die Uhr Daten aufzeichnen. Dadurch und durch die verbesserten CPUs und GPUs sind Neurale Netzwerke und der Begriff des „Machine Learning“ wieder ins Rampenlicht gerückt. Abbildung 1.4 veranschaulicht diesen Trend, indem es das Interesse des Begriffes seit 2004 darstellt. Durch diesen Trend sind viele Werkzeuge und Bibliotheken für Programmierer entwickelt worden, die helfen sollen, aus gesammelten Daten, Resultate abzuleiten.

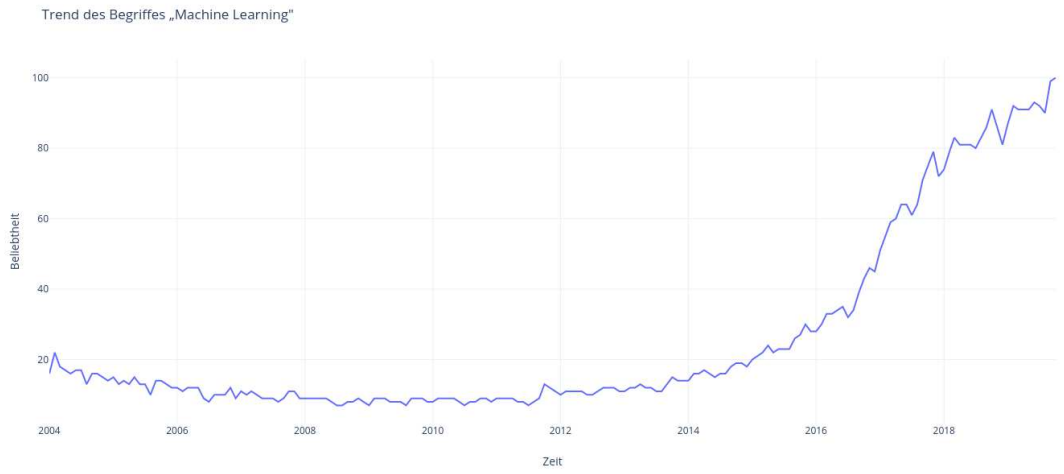


Abbildung 1.4: Interesse des Begriffes „Machine Learning“ seit 2004. (Data source: Google Trends [31])

Diese Algorithmen lernen aus Daten, im Gegensatz zum menschlichen Gehirn, welches Rückschlüsse aus Erfahrungen zieht. Die projektspezifische Softwarearchitektur eines Softwareprojektes zu kennen, erleichtert Programmierern den Einstieg in das Projekt und gibt einen guten Überblick über die Komponenten, die in dem Software Projekt verwendet werden. Diese Dokumentation muss jedoch gewartet werden. Im Zuge dieser Diplomarbeit wird ein Programm entwickelt, dass automatisiert die projektspezifische Softwarearchitektur eines Softwareprojektes erkennt und veranschaulicht. Dazu sollen die passenden Werkzeuge und Bibliotheken, die ein Ergebnis des Trends sind, verwendet werden. Die Kosten für die Wartung der Dokumentation bezüglich der Softwarearchitektur sollen sich durch das Resultat dieser Diplomarbeit verringern. Dadurch soll der Einstieg in das Softwareprojekt vereinfacht und beschleunigt werden.

1.3 Zielsetzung

Im Rahmen dieser Diplomarbeit soll ein Kommandozeilenprogramm entwickelt werden. Dieses Programm soll die Quelltextdateien eines Softwareprojektes und die Programmiersprache des Softwareprojektes als Argument bekommen.

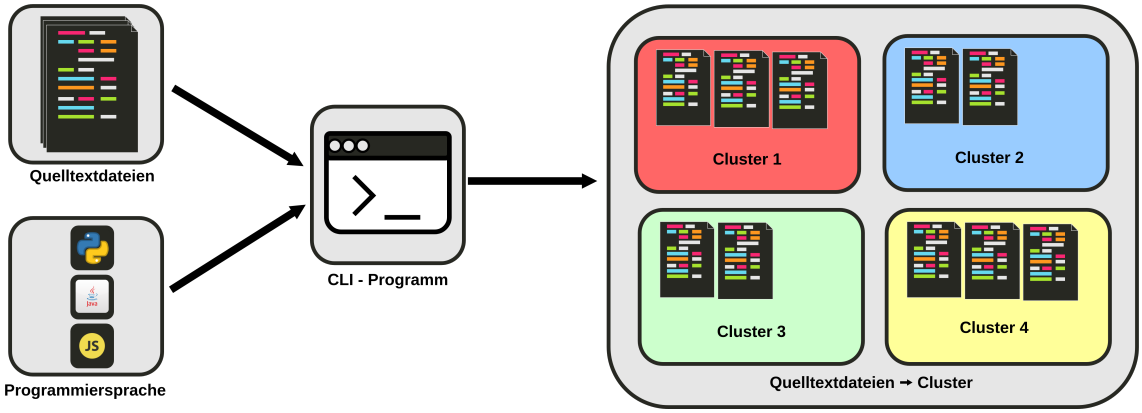


Abbildung 1.5: Überblick über das Kommandozeilenprogramm, welches das Resultat dieser Diplomarbeit ist.

Diese Argumente werden links in der Abbildung 1.5 dargestellt. Das Resultat dieser Applikation ist definiert als eine Zuordnung von Quelltextdateien zu Clustern. Dieses Ergebnis wird rechts auf der Abbildung 1.5 dargestellt. Aus dieser Zuordnung ist es möglich, die projektbezogene Software Architektur abzuleiten. Außerdem sollen diese Zuordnungen wissenschaftlich evaluiert und analysiert werden.

1.4 Methodik

Mit einer Literaturrecherche wird der Stand der Technik zu dem Thema Clustering von Quelltextdateien in einem Softwareprojekt ermittelt. Aufbauend auf diese Ergebnisse wird eine Anforderungsanalyse durchgeführt, welche die fachlichen und technischen Anforderungen an die Applikation, welche im Rahmen dieser Diplomarbeit entwickelt wird, ermittelt.

Anschließend wird mit einem iterativen Entwicklungsmodel die Implementierung des Programmes schrittweise optimiert und erweitert, bis das erwünschte Resultat erreicht wurde. Dieses erwünschte Resultat bezieht sich auf die Vorstellung des Autors für private selbst entwickelte Softwareprojekte. Diese wurden herangezogen, da der Autor bei diesen Projekten das notwendige Verständnis über die Struktur und Architektur der Softwareprojekte besitzt.

Diese Entwicklungsmethode eignet sich für die Problemstellung dieser Diplomarbeit, da der Hauptfokus der Implementierung auf das Entwickeln des „Preprocessing und Feature engineering“ Prozesses liegt, welche somit effizient erweitert und optimiert werden kann.

Nachdem die prototypische Applikation fertiggestellt ist, wird eine Evaluierung des Programmes durchgeführt. Diese soll Evidenz über den Bedarf und die Resultate des entwickelten Programmes liefern. Hier muss eine Lösung gefunden werden um die Zuweisungen von Quelltextdateien zu Clustern des Programmes zu evaluieren.

Dabei sollen Experten helfen die Ergebnisse des entwickelten Programmes an bekannten Softwareprojekten zu evaluieren.

1.5 Aufbau der Arbeit

Kapitel 2 behandelt sowohl Grundlagen als auch Definitionen und bietet einen Überblick über das Thema „Machine Learning “.

Danach wird der Stand der Technik zu dem Thema Clustering von Quelltextdateien in Kapitel 3 beschrieben.

In Kapitel 4 werden die Anforderungen aufgelistet und analysiert. Diese Anforderungen werden für die Implementierung des Kommandozeilenprogrammes benötigt.

Diese Implementierung wird in Kapitel 5 erörtert. Um die Resultate des Kommandozeilenprogrammes zu bewerten, wird in Kapitel 6 eine wissenschaftliche Evaluierung der Resultate vorgestellt.

Abschließend fasst Kapitel 7 die wesentlichen Erkenntnisse zusammen und gibt einen Ausblick in die Zukunft.

2 Grundlagen

In diesem Kapitel werden die Grundlagen erklärt, welche für das weitere Verständnis dieser Arbeit benötigt werden. Dazu werden in den Kapiteln 2.1 und 2.2 die Begriffe „Künstliche Intelligenz“ und „Maschinelles Lernen“ beschrieben. Danach werden die Teilgebiete dieser Forschungsfelder in Kapitel 2.3 und 2.4 erklärt. In dem Abschnitt 2.5 wird detailliert auf das Thema „Clustering“ eingegangen. Zum Abschluss widmet sich Kapitel 2.6 der Dimensionsreduktion.

2.1 Künstliche Intelligenz

Das Themengebiet KI (engl. artificial intelligence oder AI) ist laut [20] ein Teilgebiet der Informatik. Mit Hilfe von KI soll es nach [20] möglich sein, einen intelligenten Agenten zu entwickeln. Dabei wird der Begriff KI verwendet, wenn der Agent kognitive Fähigkeiten des Menschen, wie etwa komplexes Problemlösen, nachahmt. Beispiele solcher Fähigkeiten sind das Verständnis der menschlichen Sprache, selbst fahrende Autos oder eigenständiges Spielen von komplexen Strategiespielen wie etwa Schach oder Go, aber auch das Erkennen und Erlernen von komplexen Mustern. Wie in Abbildung 2.1 veranschaulicht, ist Maschinelles Lernen ein spezielles Themengebiet in der KI.

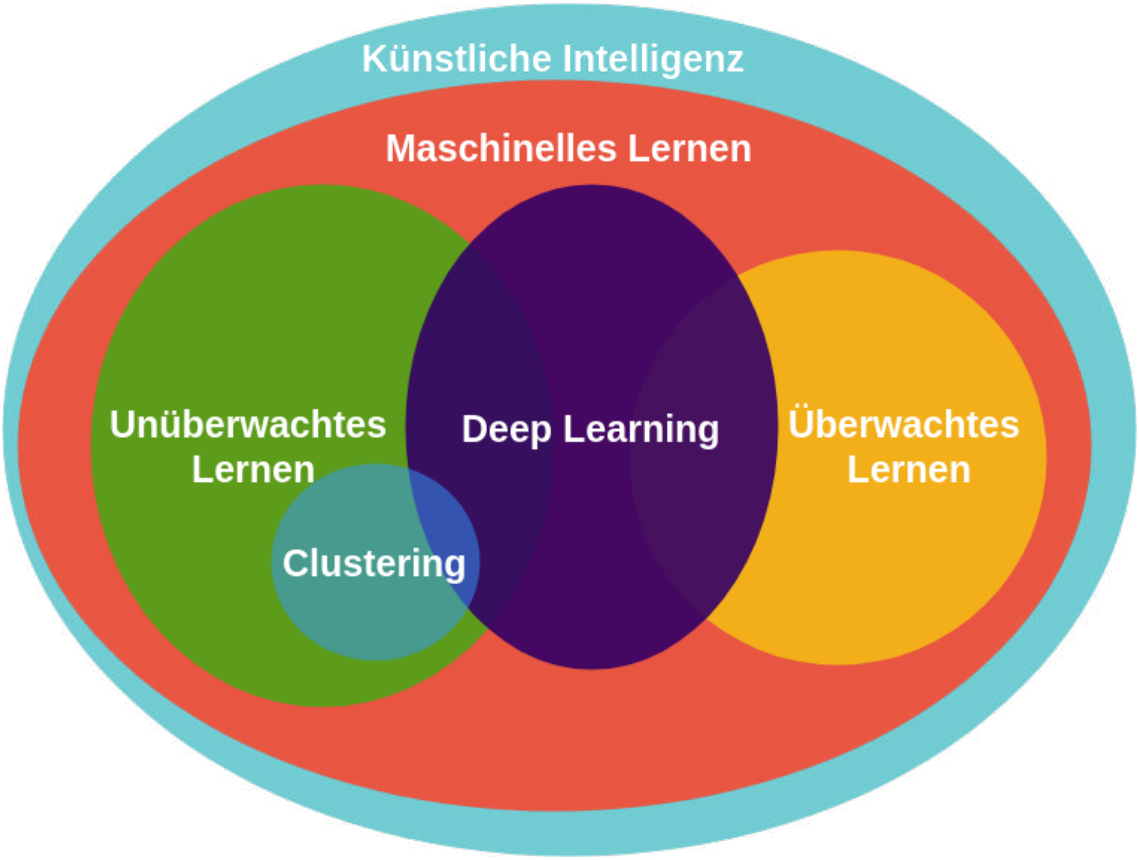


Abbildung 2.1: Einordnung wichtige Begriffe in das Themengebiet KI

2.2 Maschinelles Lernen

Wie schon in Kapitel 2.1 erwähnt, wird maschinelles Lernen dem Themengebiet KI zugeordnet. Arthur Samuel beschreibt maschinelles Lernen in [74] als Forschungsfeld, welches Computern die Möglichkeit geben soll zu lernen, ohne explizit programmiert worden zu sein.

Maschinelles Lernen wird laut [62] als eng verwandt mit Statistical Computing definiert, mit dem es auch möglich ist, Vorhersagen aufgrund von statistischen Daten mit Hilfe eines Computers zu treffen. Bei maschinellem Lernen wird mit statistischen Methoden ein Muster in Daten erkannt, die es ermöglichen, für weitere unbekannte Werte Vorhersagen zu treffen.

Dabei wird laut [20] zwischen drei unterschiedlichen Methoden des Maschinellen Lernens unterschieden:

- Überwachtes Lernen (engl. supervised learning) (siehe Abschnitt 2.3)
- Unüberwachtes Lernen (engl. unsupervised learning) (siehe Abschnitt 2.4)
- Bestärkendes Lernen (engl. reinforcement learning) [88]

2.3 Überwachtes Lernen

Überwachtes Lernen wird nach [62] dem Themengebiet des maschinellem Lernen zugeordnet. Bei dieser Methode wir laut [20] ein Algorithmus angewandt, der aus gegebenen Paaren von Ein- und Ausgabewerten eine Funktion findet, welche die Werte gut approximiert. Diese Funktion wird dann auf unbekannte Daten angewandt, um eine Vorhersage zu treffen. Dabei spiegelt das Ergebnis der Funktion die Vorhersage wieder. Diese Anwendung des Überwachten Lernens wird in [20] und [62] als Regression bezeichnet. Mit überwachtem Lernen ist es laut [20] auch möglich, Daten in vorher definierten Klassen einzuordnen. Dabei fungiert die gefundene Funktion als Trennung zwischen den verschiedenen Klassen. Diese Art der Anwendung von überwachtem Lernen wird Klassifikation genannt. Anwendungsbeispiele solcher Klassifikationen sind Erkennung von Handschriften oder auch Objekterkennung in Fotos.

Es gibt verschiedene Arten, wie der Algorithmus anhand der Ein- und Ausgabewerte die Funktion berechnet, welche die Vorhersagen für die unbekannten Werte treffen soll. Einige bekannte Vertreter werden in der folgenden Auflistung erwähnt:

- **Lineare Regression**

  Die Lineare Regression ist ein statistisches Verfahren, welches in Kapitel 2.3.1 genauer erklärt wird [79].

- **Logistische Regression**

  Die Logistische Regression ist eine Regressionsanalyse. Sie modelliert die Verteilung abhängiger diskreter Variablen [48].

- **Bayes-Klassifikator**

  Der Bayes-Klassifikator ist ein Klassifikator, welcher aus dem Satz von Bayes hergeleitet wurde [55].

- **Nächste-Nachbarn-Klassifikator**

  Der Nächste-Nachbarn-Klassifikator ist ein Klassifikator, welcher zur Schätzung der Wahrscheinlichkeitsdichte verwendet wird [45].

- **Klassifizierung mittels einem Künstlichen Neuronalen Netzes**

  Künstliche neuronale Netze (KNN) bieten die Möglichkeit Klassifizierungen durch eine Auswahl an Beispieldatensätze zu erlernen [15].

2.3.1 Lineare Regression

Lineare Regression ist laut [51] eine Methode des überwachten Lernens. Es ermöglicht nach [18] das Vorhersagen eines unbekannten Zielwertes anhand eines Eingabeparameters. Hierfür wird jedoch eine ausreichend große Anzahl an bereits bestehenden Eingabeparametern und dazugehörigen Zielwerten benötigt. Dabei wird eine Gerade durch die Punktwolke aus Eingabeparametern x und Zielwerten y gelegt, sodass der Abstand zwischen jedem Punkt und der Geraden minimiert wird. Ein Beispiel einer Linearen Regression wird in der Abbildung 2.2 veranschaulicht.

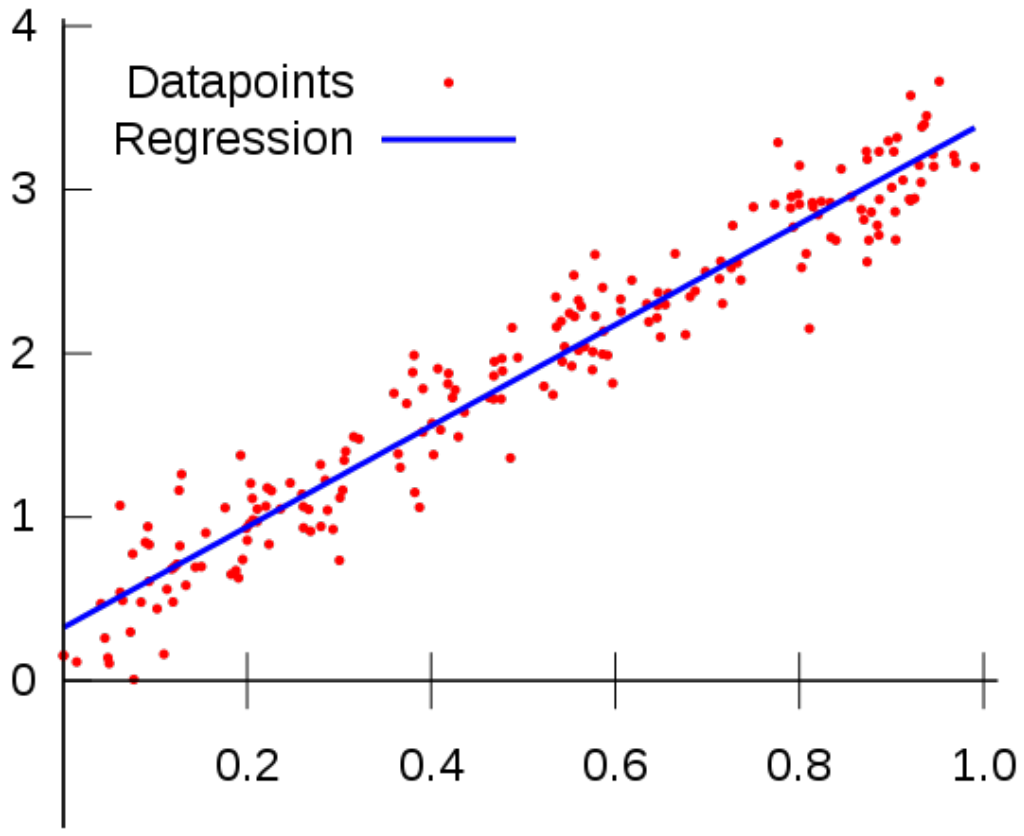


Abbildung 2.2: Beispielgraph einer Linearen Regression

2.4 Unüberwachtes Lernen

Unüberwachtes Lernen wird laut [62] ebenso wie überwachtes Lernen als ein Teilgebiet von maschinellem Lernen eingegliedert. Es gibt nach [20] bei dieser Art des Maschinellen Lernens einen Trainings-Datensatz mit Parametern, aber keine Ergebnisse zu diesen Parametern. Die Maschine versucht im Trainings-Datensatz ein Muster zu erkennen. Mit Hilfe dieser Technik kann die Maschine laut [51] eine Segmentierung oder eine Komprimierung von Daten zur Dimensionsreduktion durchführen.

Es gibt verschiedenste Methoden und Algorithmen, um Unüberwachtes Lernen durchzuführen. Einige Vertreter solcher Methoden und Algorithmen, die für das Verständnis dieser Arbeit benötigt werden, sind in den Unterkapiteln beschrieben, andere werden für die Vollständigkeit in der Auflistung benannt.

- **K-Means** [87]

  Dieses Clusteringverfahren findet mittels Schwerpunkte Cluster und wird im Kapitel 2.5.1 detailliert erklärt.

- **Density-Based Spatial Clustering of Applications with Noise**

  Ein Clustering Algorithmus, welcher auf der Dichteverbundenheit der Objekte basiert. Mit diesem Clusteringverfahren ist es nicht nur möglich Punkte zu Clustern zuzuordnen, sondern auch Punkte als Rauschpunkte zu ermitteln [77]. Auf diesen Algorithmus wird in Kapitel 2.5.2 näher eingegangen.

- **Hierarchische Clusteranalyse**

  Unter hierarchische Clusteranalyse werden Clusterverfahren bezeichnet, welche distanzbasiert Cluster ermitteln [43].

- **EM-Algorithmus**

  Der Erwartungs-Maximierungs-Algorithmus (engl. Expectation-Maximization-Algorithmus) ist ein Clustering Algorithmus, welcher mit einen zufälligen Model startet und iterativ Teile des Modells und die Parameter des Modells optimiert [54].

- **Selbstorganisierende Karte**

  Unter einer selbstorganisierenden Karte bezeichnet man eine Art eines künstlichen neuronalen Netzes, welche in der Clusteranalyse zum Einsatz kommt [23].

- **Autoencoder**

  Ein Autoencoder ist ein künstliches neuronales Netz, welches effiziente Codierung lernt. Damit kann dieses Verfahren zur Dimensionsreduktion eingesetzt werden [50].

2.5 Clustering

Clustering, auch Segmentierung genannt, beschreibt laut [8] ein Verfahren, dass zu jedem Datenpunkt $x \in M$ in einer Menge von Datenpunkten $M = \{x_0, ..., x_{N-1}\}$ der Größe N einen Cluster $C_1, ..., C_k$ zuweist. Die Wert k spiegelt dabei die Anzahl der Cluster wieder.

Es gibt verschiedene Algorithmen, um Datenpunkte zu segmentieren. In den weiteren Unterkapiteln 2.5.1 und 2.5.2 werden die für diese Diplomarbeit relevanten Clustering Algorithmen beschrieben und erklärt.

Abbildung 2.3 veranschaulicht die verschiedenen Algorithmen des Clustering. In dieser Abbildung werden die Algorithmen aus einem bekannten Python Moduls [78] für maschinelles Lernen gezeigt. In jeder Zeile wird ein anderes Set von Datenpunkten in einem zweidimensionalen Raum verwendet. Die Spalten zeigen das Clustering der verschiedenen Algorithmen. Die Farbe der Datenpunkte zeigt die verschiedenen gefundenen Cluster.

In dem Unterkapitel 2.6.1 wird die Hauptkomponentenanalyse erklärt, um Dimensionen zu reduzieren und somit die Daten besser zu veranschaulichen und das Clusteringverfahren zu beschleunigen.

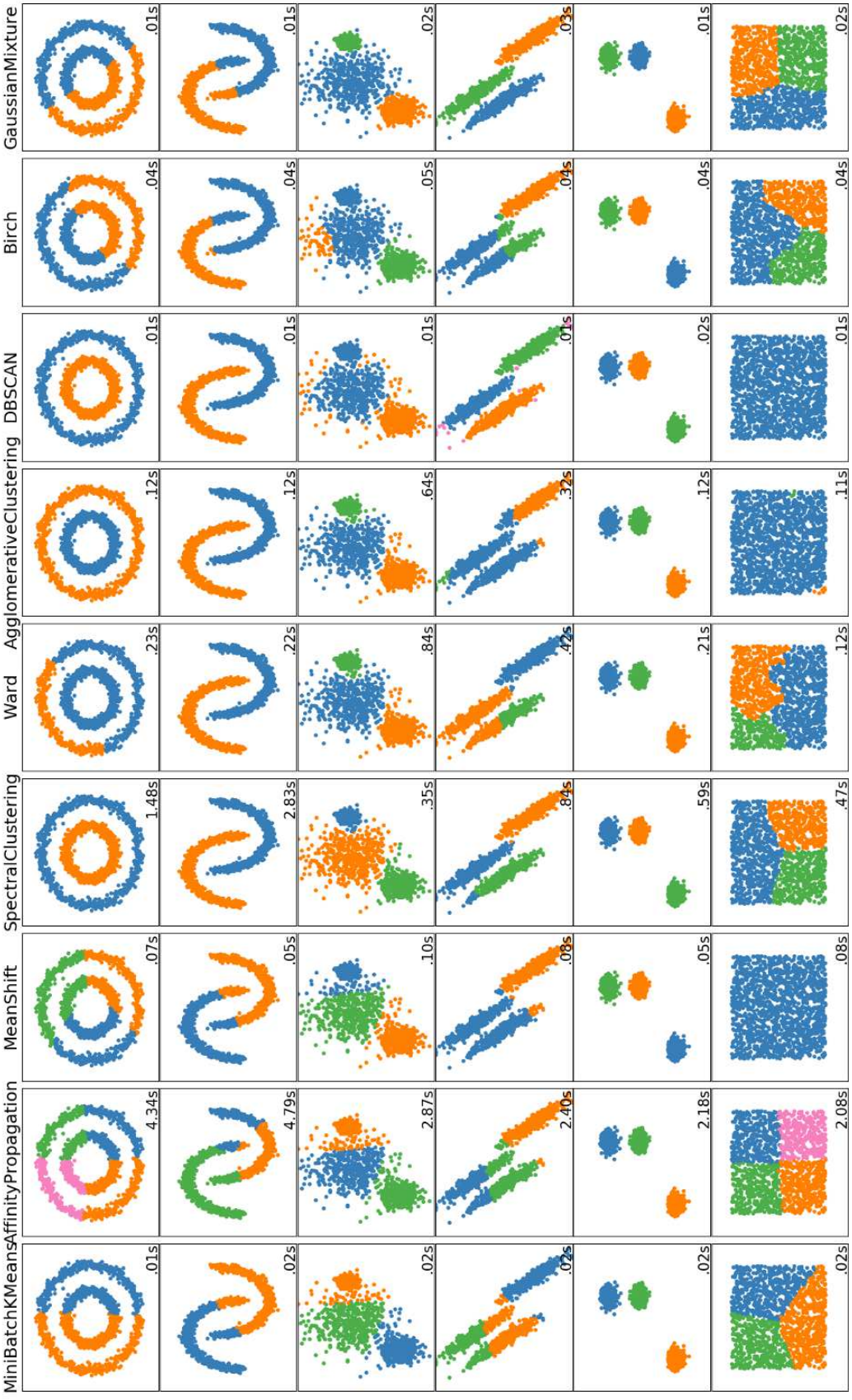


Abbildung 2.3: Vergleich verschiedener Algorithmen für Clustering. Abbildung von [78]

2.5.1 K-Means

K-Means ist laut [20] ein Algorithmus des unüberwachten Lernens und des Clustering. Es sind also Eingabeparameter vorhanden, aber keine Zielwerte. Mittels K-Means ist es nach [1] möglich, diese Eingabeparameter in K Cluster (Segmente) einzuteilen.

Die mathematische Formel für den K-Means-Algorithmus ist laut [98] in der Formel 2.1 definiert:

$$\arg\min_S \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2 = \arg\min_S \sum_{i=1}^{k} |S_i| Var S_i \tag{2.1}$$

Wobei k die Anzahl der zu findenden Cluster ist. Die Datenpunkte sind mit x_j definiert, wobei jeder Datenpunkt einen d-dimensionalen realen Vektor repräsentiert. Die Mittelwerte sind in der Formel als μ_i gegeben.

Der K-Means Algorithmus kann nach [89] in drei Schritten beschrieben werden:

1. **Initialisierung**

   In diesem Schritt werden k Punkte zufällig gewählt. Diese ausgewählten Punkte werden als Mittelwerte (engl. means) im ersten Durchlauf $m_1^{(1)}, ..., m_k^{(1)}$ initialisiert.

2. **Zuordnung**

   In diesem Schritt wird jeder Datenpunkt x_j einem Cluster $S_i^{(t)}$ zugeordnet. Wobei $i \in 1, ..., k$ und t die Zahl des Durchlaufes ist. Es wird der Cluster zu dem Datenpunkt zugeordnet, welcher die geringste quadrierte euklidische Distanz von seinem Mittelwert $m^{(t)}$ zu dem Datenpunkt hat. Dieser Schritt ist in der Formel 2.2 definiert.

$$S_i^{(t)} = x_j : \|x_j - m_i^{(t)}\|^2 \leq \|x_j - m_j^{(t)}\|^2 \forall j, 1 \leq j \leq k \tag{2.2}$$

3. **Aktualisierung**

   In diesem Schritt werden die Mittelwerte $m_i^{(t)}$ für jeden Cluster $S^{(t)}$ neu berechnet. Dabei berechnen sich die neuen Mittelwerte $m_i^{(t+1)}$ nach der Formel 2.3.

$$m_i^{t+1} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2.3}$$

Die Schritte 2 und 3 werden solange wiederholt, bis sich die Zuordnung von den Mittelwerten nicht mehr ändert. Es werden die Schritte solange wiederholt, solange $\exists(m_i^{(t)} \neq m_i^{(t+1)})$.

In Abbildung 2.4 wird der K-Means Algorithmus in den verschieden Schritten dargestellt, die in der folgenden Auflistung 2.5.1 genauer erklärt wird.

(a) Die grünen Punkte sind die Datenpunkte x_j. Für die Anzahl der Cluster wird $k = 2$ gewählt. Für jedes Cluster wird ein Mittelwert $m_i^{(1)}$ zufällig ausgewählt. Das blaue X-Symbol kennzeichnet den Mittelwert $m_1^{(1)}$ des Clusters $S_1^{(t)}$. Das rote X-Symbol kennzeichnet den

Mittelwert $m_2^{(1)}$ des Clusters $S_2^{(t)}$. Diese Abbildung (a) spiegelt das Ergebnis des Schrittes 1 wieder.

(b) Alle grünen Punkte aus (a) wurden in den Farben blau oder rot umgefärbt. Alle Punkte der Farbe Blau bilden dan Cluster $S_1^{(t)}$. Alle Punkte der Farbe Rot bilden den Cluster $S_2^{(t)}$. Blau eingefärbte Punkte verdeutlichen, dass der jeweilige Datenpunkt x_j zu dem Cluster $S_1^{(1)}$ zugeordnet wurde. Rot eingefärbte Punkte verdeutlichen, dass der jeweilige Datenpunkt x_j zu dem Cluster $S_2^{(1)}$ zugeordnet wurde. Die rote Linien gibt die Trennung der 2 Cluster $S_1^{(t)}$ und $S_2^{(t)}$ an. Dieses Darstellung (b) spiegelt das Ergebnis des Schrittes 2 wieder. Die Mittelwerte bleiben zu dem vorherigen Bild (a) unverändert.

(c) Alle blauen und roten Punkte bleiben unverändert zu dem vorherigen Schritt (b). Die Cluster bleiben demnach bestehen, wie sie im vorherigen Schritt waren. Jedoch werden die Mittelwerte $m_1^{(2)}$(blaues Kreuz Symbol) und $m_2^{(2)}$(rotes Kreuz Symbol) der Cluster $S_1^{(1)}$(blaue Punkte) und $S_2^{(1)}$(rote Punkte) neu berechnet. Dieses Bild (c) zeigt das Ergebnis aus Schritt 3.

(d) In dieser Abbildung werden die Datenpunkte x_j erneut zu den Clustern $S_1^{(2)}$(blaue Punkte) und $S_2^{(2)}$(rote Punkte) zugewiesen. Dabei ändern einige Punkte ihren Cluster, was man an der roten Linie, welche die Trennung der 2 Cluster abbildet, erkennen kann. Dieses Abbildung (d) zeigt das Ergebnis aus Schritt 2 im 2. Durchlauf.

(e) In diesem Bild werden die Mittelwerte $m_1^{(3)}$(blaues Kreuz Symbol) und $m_2^{(3)}$(rotes Kreuz Symbol) der Cluster $S_1^{(2)}$(blaue Punkte) und $S_2^{(2)}$(rote Punkte) auf Basis der neuen Cluster berechnet. Das Bild (e) spiegelt das Ergebnis aus Schritt 3 im 2. Durchlauf wieder.

(f) In diesem Schritt werden wieder die Datenpunkte x_j zu den Clustern $S_1^{(3)}$(blaue Punkte) und $S_2^{(3)}$(rote Punkte) auf Basis der in Bild (e) berechneten Mittelwerte $m_1^{(3)}$(blaues Kreuz Symbol) und $m_2^{(3)}$(rotes Kreuz Symbol) zugewiesen. Dabei ändern einige Punkte ihren Cluster, was man an der roten Linie, welche die Trennung der 2 Cluster abbildet, erkennen kann. Diese Darstellung (f) zeigt das Ergebnis aus Schritt 2 im 3. Durchlauf.

(g) In diesem Bild werden die Mittelwerte $m_1^{(4)}$(blaues Kreuz Symbol) und $m_2^{(4)}$(rotes Kreuz Symbol) der Cluster $S_1^{(3)}$(blaue Punkte) unf $S_2^{(3)}$(rote Punkte) auf Basis der neuen Cluster berechnet. Das Bild (g) spiegelt das Ergebnis aus Schritt 3 im 3. Durchlauf wieder.

(h) In diesem Schritt werden wieder die Datenpunkte x_j zu den Clustern $S_1^{(4)}$(blaue Punkte) und $S_2^{(4)}$(rote Punkte) auf Basis der in Bild (e) berechneten Mittelwerte $m_1^{(4)}$(blaues Kreuz Symbol) und $m_2^{(4)}$(rotes Kreuz Symbol) zugewiesen. Dabei ändern einige Punkte ihren Cluster, was man an der roten Linie, welche die Trennung der 2 Cluster abbildet, erkennen kann. Dieses Bild (h) zeigt das Ergebnis aus Schritt 2 im 4. Durchlauf.

(i) In diesem Bild werden die Mittelwerte $m_1^{(5)}$(blaues Kreuz Symbol), $m_2^{(5)}$(rotes Kreuz Symbol) der Cluster $S_1^{(4)}$(blaue Punkte), $S_2^{(4)}$(rote Punkte) auf Basis der neuen Cluster berechnet. Da $m_1^{(5)} = m_1^{(4)}$ und $m_2^{(5)} = m_2^{(4)}$ ist, terminiert der Algorithmus und hat die beiden Cluster $S_1^{(4)}$(blaue Punkte) und $S_2^{(4)}$(rote Punkte) gefunden. Diese Abbildung (i) spiegelt das Ergebnis aus Schritt 3 im 4. Durchlauf wieder und ist das Ergebnis des K-Means Algorithmus.

K-Means++

Es gibt einige Erweiterungen und Variationen des K-Means Algorithmus. Eine der bekanntesten Erweiterungen ist der K-Means++ Algorithmus. Bei dieser Erweiterung wird nach [52] der Schritt 1(Initialisierung) ersetzt. Im Gegensatz zum klassischen K-Means-Algorithmus werden hier nicht alle Mittelwerte $m_1^{(1)}, ..., m_k^{(1)}$ zufällig gewählt, sondern nach folgender Methode:

1. Der erste Mittelwert $m_1^{(1)}$ des ersten Clusters $S_1^{(1)}$ wird zufällig aus allen Datenpunkten x_j ausgewählt und zugewiesen.
2. Für alle Datenpunkte x_j wird der Abstand $D(x)$ zu $m_1^{(1)}$ berechnet.
3. Der Mittelwert des nächsten Clusters $S_2^{(1)}$ wird zufällig aus allen Datenpunkten x_j ausgewählt und zugewiesen. Es wird aber ein Zufallsprinzip gewählt, sodass ein Datenpunkt eine höhere Wahrscheinlichkeit hat, als Mittelwert $S_2^{(1)}$ ausgewählt zu werden, umso weiter der Datenpunkt von dem bereits gewählten Mittelwerten entfernt ist. Somit ist die Wahrscheinlichkeit eines Datenpunktes x als Mittelwert $S_2^{(1)}$ ausgewählt zu werden proportional zu $D^2(x)$.

Die Schritte 2 und 3 werden solange wiederholt, bis k Mittelwerte $S_1^{(1)}, S_1^{(1)}, ..., S_k^{(1)}$ zugewiesen wurden. Durch diese Optimierung kann die Laufzeit der Segmentierung gegenüber K-Means reduziert werden, trotz Zusatzberechnungen beim initialisieren der ersten Mittelwerte. In [6] wird gezeigt, dass der K-Means++ Algorithmus eine Laufzeit von $O(\log k)$ aufweist.

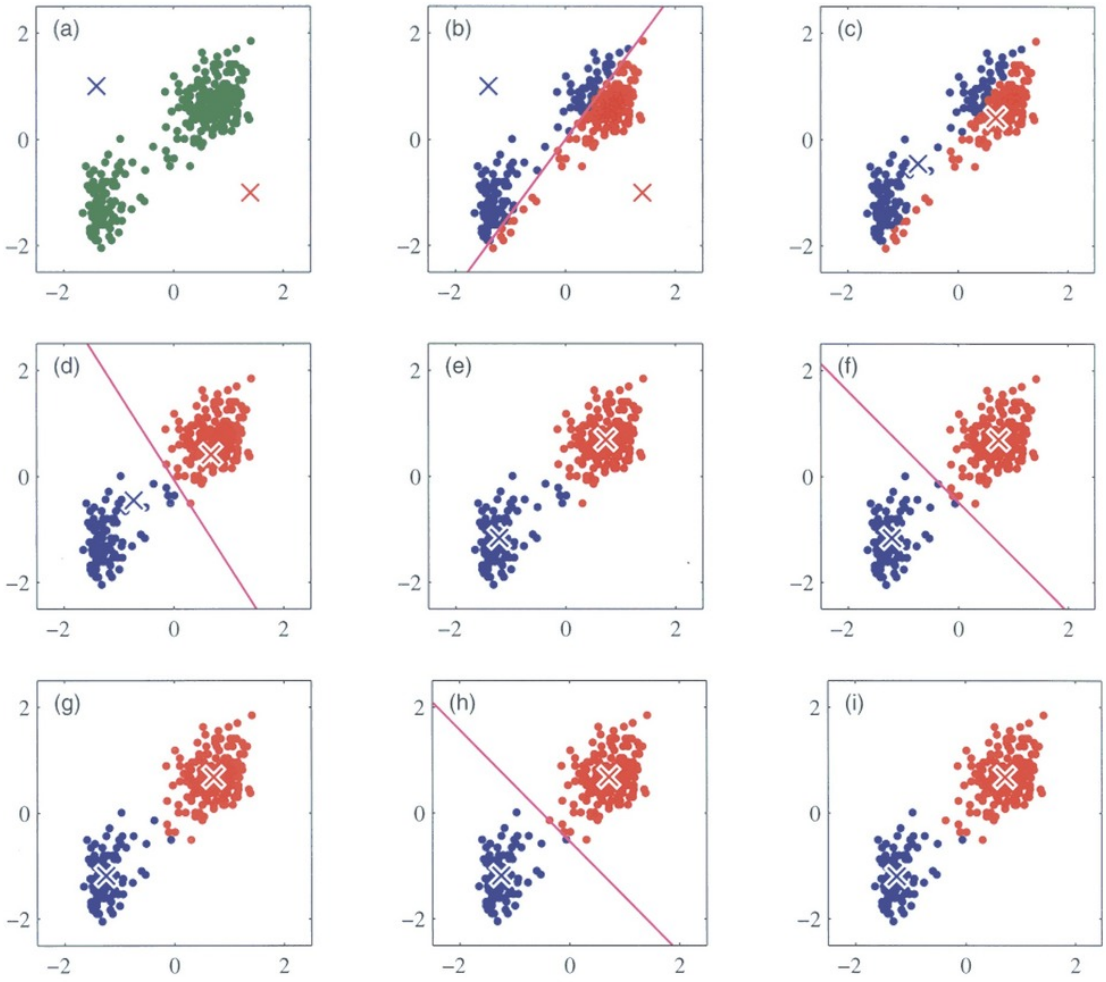


Abbildung 2.4: K-Means Clustering Beispiel mit Stufen für das Finden der Cluster. [10]

2.5.2 Density-Based Spatial Clustering of Applications with Noise

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) ist laut [22] ein Algorithmus des unüberwachten Lernens und des Clustering. Mit DBSCAN ist es nicht nur möglich Punkte in Cluster zu unterteilen, sondern auch ein Rauschen in Form von Ausreißer zu finden. Bei DBSCAN muss man im Gegensatz zu dem K-Means Algorithmus nicht die Anzahl der zu findenden Cluster als Parameter eingeben. Dafür benötigt diese Variante des Clustering zwei Parameter zum finden der Cluster:

- **Minimale Punkte eines Kernpunktes** $minPts$

  $minPts$ definiert die Bedingung, ob ein Punkt x_i in DBSCAN auch ein Kernpunkt cp_j darstellt. $minPts$ ist eine positive ganze Zahl. Wenn mindestens $minPts$ Punkte existieren, die vom Punkt x_i eine maximale Entfernung von ϵ haben, dann ist x_i ein Kernpunkt cp_j

- **Die Nachbarschaftslänge** ϵ

  Die Nachbarschaftslänge ϵ gibt an, wie weit ein Punkt x_i von einem Kernpunkt cp_j entfernt liegen darf, damit x_i und cp_j mittels DBSCAN zum selben Cluster zugewiesen werden. Der Punkt x_i darf dabei entweder zu dem Kernpunkt cp_j eine maximale Entfernung von ϵ aufweisen oder es gibt einen weiteren Kernpunkt cp_k im selben Cluster wie cp_j zu dem der Punkt x_i eine maximale Entfernung von ϵ hat.

In Abbildung 2.5 wird der DBSCAN Algorithmus veranschaulicht. Gegeben sind die 7 Punkte $[A, B, C, D, E, F, G]$. Diese Punkte bilden den Mittelpunkt der farbigen Kreise. Ein Kreis symbolisiert den Abstand ϵ von den jeweiligen Punkt. In dieser Abbildung wird DBSCAN mit $minPts = 2$ ausgeführt. Somit findet der Algorithmus 4 Kernpunkte in 2 Cluster. Die Cluster werden in den Farben grün und violett dargestellt. Die Kernpunkte in dem grüne Cluster sind $[C, A]$, die Kernpunkte in dem violetten Cluster sind $[E, F]$. Der Punkt D ist kein Kernpunkt, aber innerhalb der Nachbarschaftslänge ϵ von C und gehört somit auch zu dem grünen Cluster. Genauso ist der Punkt B kein Kernpunkt aber innerhalb der Nachbarschaftslänge ϵ von A und wird auch dem grünen Cluster zugeordnet. Der Punkt G ist weder ein Kernpunkt, noch innerhalb einer Nachbarschaftslänge ϵ eines Kernpunktes und wird somit als Ausreißer markiert und in der Abbildung mit der Farbe gelb gekennzeichnet.

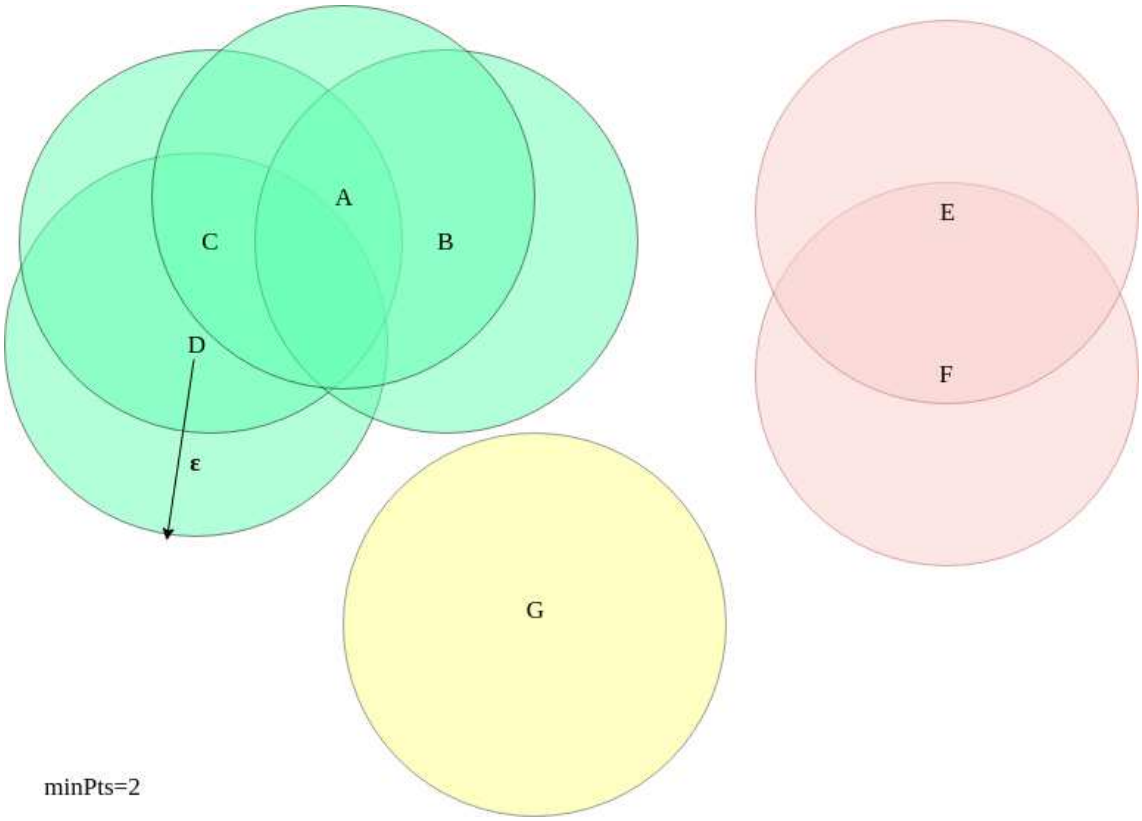


Abbildung 2.5: Anwendung des DBSCAN-Algorithmus anhand eines Beispiels.

```python
import matplotlib.pyplot as plt
from numpy.random import rand
from numpy import square, sqrt

def regionQuery(P, eps, D):
  neighbourPts = []
  for point in D:
    if sqrt(square(P[1] - point[1]) + square(P[2] - point[2]))<eps:
      neighbourPts.append(point)
  return neighbourPts

def DBSCAN(D, eps, MinPts):
  noise = []
  visited = []
  C = []
  c_n = -1
  for point in D:
    visited.append(point) #marking point as visited
    neighbourPts = regionQuery(point, eps, D)
    if len(neighbourPts) < MinPts:
      noise.append(point)
    else:
      C.append([])
      c_n+=1
      expandCluster(point, neighbourPts, C, c_n,eps, MinPts, D, visited)
  print "no. of clusters: " , len(C)
  print "length of noise:", len(noise)
  for cluster in C:
    col =[rand(1),rand(1),rand(1)]
    print cluster
    plt.scatter([i[1] for i in cluster],[i[2] for i in cluster],color=col)
  plt.show()

def expandCluster(P, neighbourPts, C, c_n,eps, MinPts, D, visited):
  C[c_n].append(P)
  for point in neighbourPts:
    if point not in visited:
      visited.append(point)
      neighbourPts_2 = regionQuery(point, eps, D)
      if len(neighbourPts_2) >= MinPts:
        neighbourPts += neighbourPts_2
    if point not in (i for i in C):
    C[c_n].append(point)
```

Listing 2.1: DBSCAN Algorithmus Implementierung in der Programmiersprache Python

Ein Beispiel des DBSCAN Algorithmus implementiert in der Programmiersprache Python ist im Quelltext 2.1 zu finden.

2.5.3 Kennzahlen eines Clusters

Silhouettenkoeffizient

Der Silhouettenkoeffizient s_D wird laut [73] als das arithmetische Mittel alle Silhouetten n_D des Datensatzes D beschrieben. Der Datensatz kann dabei ein Cluster oder auch der gesamte Datensatz aller Punkte sein. Er wird ,wie in der Formel 2.4 angegeben, berechnet.

$$s_D = \frac{1}{n_D} \sum_{x \in D} s(x) \tag{2.4}$$

Die Silhouetten $s(x)$ sind in der Formel 2.5 beschrieben.

$$s(x) = \begin{cases} 0 & \text{für } d(A,x) = d(B,x) \\ \frac{d(B,x)-d(A,x)}{max(d(A,x),d(B,x))} & \text{sonst} \end{cases} \tag{2.5}$$

Die Distanz $d(C,x)$ eines Punktes x zu einem Cluster C wird in der Formel 2.6 definiert.

$$d(C,x) = \frac{1}{n_C} \sum_{c \in C} dist(c,x) \tag{2.6}$$

Die Anzahl der Punkte im Cluster C wird in der Formel als n_C angegeben. Die Distanz $d(A,x)$ ist demnach also der Mittelwert der Distanz aller Punkte im Cluster A zu dem Punkt x. Die Distanz $d(B,x)$ zum nächstgelegenen Cluster B wird als die minimale durchschnittliche Distanz berechnet. Diese Berechnung ist in Formel 2.7 dargestellt.

$$d(B,x) = min_{C \neq A}(d(C,x)) = min_{C \neq A}(\frac{1}{n_C} \sum_{c \in C} dist(c,x)) \tag{2.7}$$

Dementsprechend deutet ein Silhouettenkoeffizient mit einer negativen Zahl an, dass Punkte zu einem falschen Cluster zugewiesen wurden, da ein anderer Cluster näher zu diesen Punkten steht. Ein Silhouettenkoeffizient mit dem Wert 0 deutet darauf hin, dass es Punkte gibt, wo die Distanz zu verschiedenen Clustern gleich ist und es deswegen überlappende Cluster gibt. Ein Silhouettenkoeffizient mit einem positiven Wert deutet auf ein gutes Clustering hin, da die Punkte am nächsten dem zugewiesenen Cluster sind. Der Silhouettenkoeffizient ist eine Zahl zwischen [-1;1]. [33] beschreibt, wie man mithilfe des Silhouettenkoeffizienten zu einer guten Anzahl an Clustern für den K-Means Algorithmus kommt.

Elbow-Method

Die Elbow Methode wird angewandt um heuristisch die Anzahl der Cluster für ein Clusteringverfahren zu bestimmen. Dabei wird das Clusteringverfahren an einer Reihe von Clusteranzahlen $1, 2, ..., i$ angewandt. Für jedes k dieser Werte wird die Abweichungsquadratsumme berechnet. Danach werden Punkte in ein Koordinatensystem eingetragen, wobei auf der x-Achse die Anzahl der Cluster und auf der y-Achse die Abweichungsquadratsumme zu sehen sind. Diese Punkte werden miteinander verbunden. Wenn die resultierende Line einem menschlichen Arm ähnlich sieht, so befindet sich die optimalste Anzahl an Cluster k an dem Ellbogen des Armes [46].

Calinski-Harabasz Index

Calinski-Harabasz Index ist ein Index zur Bewertung des Ergebnisses eines Clusteringverfahrens, ohne dass man eine „Ground truth“ benötigt.

Der Index für k Cluster errechnet sich wie in der Formel 2.8 dargestellt [13].

$$s(k) = \frac{T_r(B_k)}{T_r(W_k)} \times \frac{N-k}{k-1} W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T B_k = \sum_{q} n_q (c_q - c)(c_q - c)^T \tag{2.8}$$

Die Variable N ist dabei die Anzahl der Punkte in dem Datensatz. C_q ist das Set der Punkte in Cluster q. Die Mitte des Clusters q ist mit c_q definiert. Die Anzahl der Punkte in Cluster q entspricht den Wert n_q. Die Mitte aller Punkte wird von dem Wert c repräsentiert.

Davies-Bouldin Index

Der Davies-Bouldin Index ist ein weiterer Index um Cluster-Algorithmen zu evaluieren. Der Index ist definiert als die durchschnittliche Ähnlichkeit zwischen jedem Cluster C_i für jedes $i = 1, ..., k$ und deren ähnlichstem Cluster C_j [16].

Normalisierte Transinformation (engl: normalized mutual information)

Normalisierte Transinformation (eng: normalized mutual information Normalized mutual information (NMI)) ist laut [27] eine Kennzahl, um die Übereinstimmung zweier Cluster zu beschreiben. [19] definiert NMI mit der Formel 2.9.

$$s(k) = \frac{T_r(B_k)}{T_r(W_k)} \times \frac{N-k}{k-1} I_{norm}(X, Y) = \frac{2I(X, Y)}{\sqrt{H(X)H(Y)}} \tag{2.9}$$

Das Ergebnis ist eine Zahl in dem Bereich [0,1]. Wobei der Wert 0 definiert, dass die beiden Cluster X und Y keine Gemeinsamkeiten haben und der Wert 1 definiert, dass die beiden Cluster X und Y alle Punkte gleich segmentiert haben.

2.6 Dimensionsreduktion

In diesem Kapitel wird das Verfahren der Dimensionsreduktion beschrieben. Die Dimensionsreduktion dient dazu, Punkte in einem p-dimensionalen Raum $\mathbb{R}^p$ zu Punkten in einem q-dimensionalen Raum $\mathbb{R}^q$ zu konvergieren. Dabei is p größer als q, es kommt dadurch zu einer Reduktion der Dimension. Das hat zur Folge, dass Information durch die Reduktion der Dimension verloren geht. Dieser Informationsverlust soll minimal gehalten werden. Die Dimensionsreduktion wird unter anderem verwendet um Datenpunkte eines m-dimensionalen Raumes, wobei $m > 2$ ist, auf einen 2-dimensionalen Raum abzubilden, um die Punkte grafisch veranschaulichen zu lassen. Außerdem hat man mit der Dimensionsreduktion die Möglichkeit vorhandene Datensätze zu verkleinern und somit das Verfahren, dass auf die Punkte angewendet werden soll, zu beschleunigen.

2.6.1 Hauptkomponentenanalyse (engl. PCA (Principal Component Analysis))

Die Hauptkomponentenanalyse [81] ist ein Verfahren, welches in Kapitel 2.6 als Dimensionsreduktion beschrieben wird. Dadurch werden n Punkten $x_1, x_2, x_3, ..., x_n$ in einem p-dimensionalen Raum $\mathbb{R}^p$ zu n Punkten $y_1, y_2, y_3, ..., y_n$ in einem q-dimensionalen Raum projiziert. Dieses Verfahren sucht in einem p-dimensionalen Raum $\mathbb{R}^p$ p Hauptkomponente.

Die erste Hauptkomponente F_1 ist eine Gerade, die durch den Mittelpunkt m_x aller Punkte $x_1, x_2, x_3, ..., x_n$ verläuft und die die Summe der euklidischen Abstände zwischen der Gerade

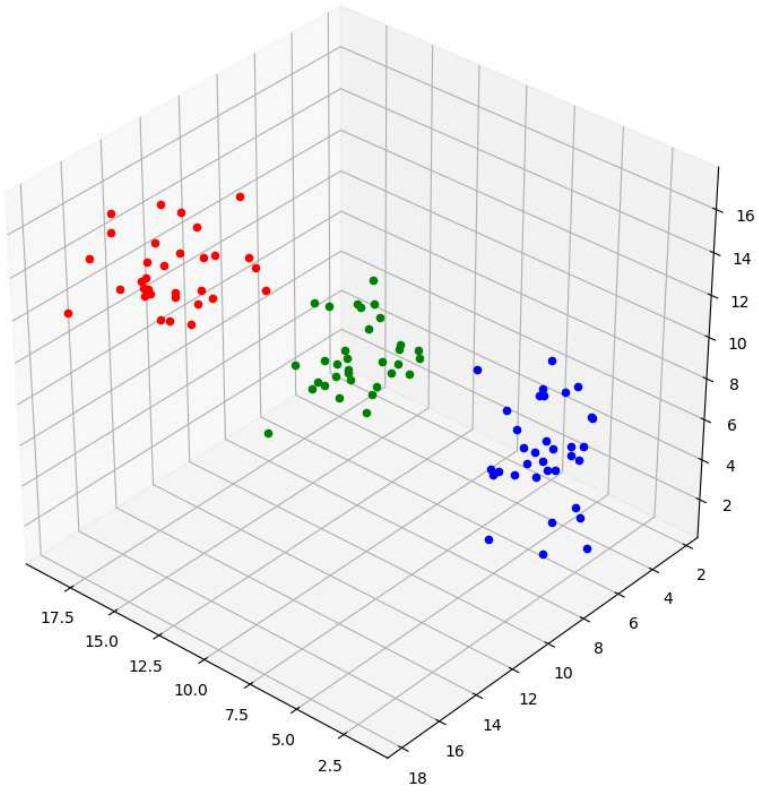
F_1 und allen Punkte $x_1, x_2, x_3, ..., x_n$ quadriert minimiert. Jeder Punkt $x_1, x_2, x_3, ..., x_n$ spannt zusammen mit dem Mittelpunkt m_x und den Normalen $N_1, N_2, N_3, ..., N_n$ von der Geraden F_1 zu den Punkten $x_1, x_2, x_3, ..., x_n$ ein rechtwinkeliges Dreieck. Durch den Satz von Pythagoras kann man die Hauptkomponente dementsprechend auch als Maximierung der Distanz auf einer Geraden F_1 von Mittelpunkt m_x zu den Schnittpunkten zwischen der Normalen der Punkte $x_1, x_2, x_3, ..., x_n$ und der Geraden F_1 berechnen.

Die zweite Hauptkomponente F_2 ist eine Gerade, die wie die erste Hauptkomponente F_1, durch den Mittelpunkt m_x verläuft und senkrecht zur ersten Hauptkomponente F_1 steht. Die folgenden Hauptkomponenten verlaufen alle durch den Mittelpunkt m_x und sind senkrecht zu allen vorherigen Hauptkomponenten.

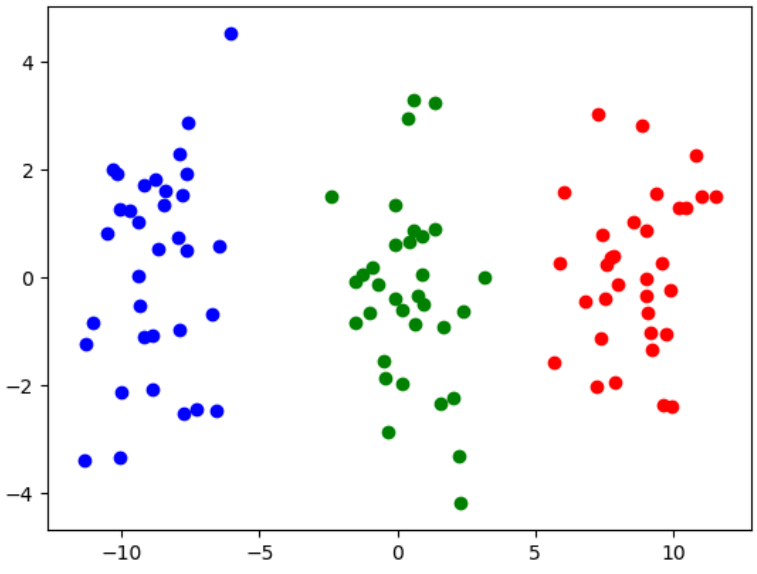
Somit enthält die erste Hauptkomponente F_1 den größten Anteil an der totalen Varianz. Die zweite Hauptkomponente F_2 den zweitgrößten Anteil an der totalen Varianz und die Hauptkomponente F_n den n größten Anteil an der totalen Varianz.

Das erhoffte Ergebnis einer Hauptkomponentenanalyse ist, dass die ersten q von p Hauptkomponenten einen sehr großen Anteil an der totalen Varianz ausmachen und man somit die Dimensionen mit den niedrigen Anteilen an der totalen Varianz weglassen kann, ohne das Ergebnis eines Clusteringverfahrens erheblich zu beeinflussen.

In Abbildung 2.6a werden Punktwolken in einem 3-dimensionalen Raum gezeigt, die zu 3 verschiedenen Clustern zugeordnet wurden. Abbildung 2.6b zeigt die mittels PCA reduzierten Punkte aus Abbildung 2.6a in einem 2-dimensionalen Raum.



(a) 3 Cluster Punktwolken in einem 3-dimensionalen Raum



(b) 3 Cluster Punktwolken reduziert auf einen 2-dimensionalen Raum mittels PCA

Abbildung 2.6: Abbildungen zur Veranschaulichung einer Dimensionsreduktion mittels PCA von 3 Dimensionen auf 2.

3 Stand der Technik

In diesem Kapitel werden relevante wissenschaftliche Dokumente zu dem Thema Clustering in Softwareprojekten aufgelistet und beschrieben. Daraus soll der Stand der Technik zu dem Thema, mit dem sich auch diese Diplomarbeit beschäftigt, abgeleitet werden.

Außerdem beschäftigt sich dieses Kapitel mit den Gemeinsamkeiten und Unterschieden zwischen dieser Diplomarbeit und den angeführten wissenschaftlichen Dokumenten.

Dieses Kapitel widmet sich zu Beginn zwei wissenschaftlichen Dokumenten die große Überschneidungen mit dieser Diplomarbeit haben und geht im Detail auf diese zwei Dokumente ein. Danach werden noch weitere wissenschaftliche Dokumente aufgelistet und grob beschrieben, die auch relevant für diese Diplomarbeit sind.

3.1 Clustering durch semantische Ähnlichkeit mit Wikipedia

Schindler, Fox und Rausch beschreiben in [76], wie sie Quelltextelemente nach lexikalischen Ähnlichkeiten segmentieren.

Dabei wenden sie zuerst das Prinzip des „camel case splitting“ auf Klassen-, Interface-, Variablen- und Parameternamen an. Dieses ist ein übliches Verfahren des „Textmining“ von Quelltext. Bei dem Verfahren des „camel case splitting“ werden zusammen geschriebene Wörter ohne Leerzeichen durch Groß- und Kleinschreibung wieder auseinander genommen. Diese Art wird sehr oft verwendet, da Leerzeichen nicht in Variablennamen enthalten sein dürfen. In 3.1 werden angewandte Bespiele von „camel case splitting“ aufgezeigt.

$$\begin{aligned} CamelSplit('= WLANConnectionManager') &= \{'WLAN','Connection','Manager'\} \\ CamelSplit('ProfileImageFilePath') &= \{'Profile','Image','File','Path'\} \end{aligned} \tag{3.1}$$

Dabei einsteht eine Liste von Namen für das jeweilige Quelltextelement. Für jeden dieser Namen wird danach eine Wikipedia-Suchabfrage gestartet und die gefundenen Artikel als Text konkateniert.

Danach wenden sie eine „stop word list“ [68] derselben Sprachen an, in der auch die Wikipedia Artikel geschrieben sind. Daraus werden die n häufig vorkommenden Wörter der Artikel für das jeweilige Quelltextelement ausgewählt. Aufgrund ihre Experimente haben sich die Autoren für $n = 150$ entschieden. Die resultierende Liste an Wörtern für das jeweilige Quelltextelement e wird Wikipedia-Repräsentation X_e genannt. Die semantische Ähnlichkeit zwischen den beiden Quelltextelementen e_1 und e_2 ist in der Formel 3.2 definiert.

$$SemanticSimilarity(e_1, e_2) = \frac{2|X_{e1} \cap X_{e2}|}{|X_{e1}| + |X_{e2}|} \in [0, 1] \tag{3.2}$$

Danach wurde „Spectral Clustering“ [94] angewendet, um die Quelltextelemente zu segmentieren.

Wie auch in dieser Diplomarbeit beschrieben, finden Schindler, Fox und Rausch Cluster von Quelltextelementen. Ein Unterschied zu dieser Diplomarbeit sind der Algorithmus der „Feature Extraction“ und das Clusteringverfahren.

3.2 Clustering durch Identifizieren von Themen

Einen anderen Ansatz wählen Kuhn, Ducasse und Girba. In [49] beschreiben diese Autoren ihre Herangehensweise, Quelltextelemente zu segmentieren. Die Abbildung 3.1 fasst diese Schritte grob zusammen, um mit ihrem Ansatz zu einer Segmentierung zu gelangen.

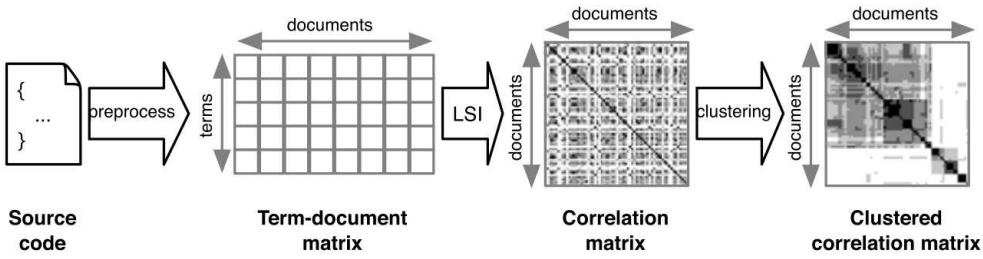


Abbildung 3.1: Übersicht der auszuführenden Schritte des Segmentierungsverfahrens nach Kuhn, Ducasse und Girba [49]

Als ersten Schritt werden Quelltextelemente in Listen von Wörtern umgewandelt. Quelltextelemente können eine beliebige Granularität haben. Sie können Packages, Klassen oder auch Methoden sein. Diese Wörter sind Namen, die im Quelltextelement verwendet worden sind. Dabei wird, wie auch schon in [94], „camel case splitting“ verwendet. Aus einer Quelltextdatei, wie im Quelltext 3.1, wird dementsprechend eine Liste extrahiert. Der Beispieltext 3.2 zeigt eine solche Liste von Wörtern, die aus dem Quelltext 3.1 resultiert.

```
1 public boolean isMorning(int hours, int minutes, int seconds) {
2   if(! isDate(hours, minutes, seconds))
3     throw Exception("Invalid input: not a time value.");
4   return hours < 12 && minutes < 60 && seconds < 60;
5 }
```

Listing 3.1: Beispiel einen Quelltextes als Veranschaulichung des Preprocessing des Segmentierungsverfahrens nach Kuhn, Ducasse und Girba. [49]

```
is morning hours minutes seconds
is date hours minutes seconds
invalid time value
hours 12 minutes 60 seconds 60
```

Listing 3.2: Beispiel einer resultierenden Liste nach dem Preprocessing des Segmentierungsverfahren nach Kuhn, Ducasse und Girba. [49]

Als nächsten Schritt wenden diese Autoren eine Art „bag-of-words“ [101], [47] Algorithmus auf ihre Liste von Wörtern an. Eine Liste von Wörtern wird dabei Dokument genannt. Daraus resultiert eine $n \times m$ Matrix, wobei n die Anzahl der Dokumente und m die Anzahl aller vorhandenen

Wörter in allen Dokumenten ist. Der Wert a_{ij} gibt an, wie oft das Wort w_i in dem Dokument d_j vorkommt.

Auf diese Matrix wird danach ein Verfahren mit dem Namen Latent Semantic Indexing (LSI) [17], [35] angewandt. LSI verwendet Konzepte aus Principal Component Analysis (PCA) und führt eine Singulärwertzerlegung (engl. Singular Value Decomposition) [95], [80] durch. Die Matrix wird dabei in einem Vektorraum interpretiert. Abbildung 3.2 veranschaulicht dieses Verfahren.

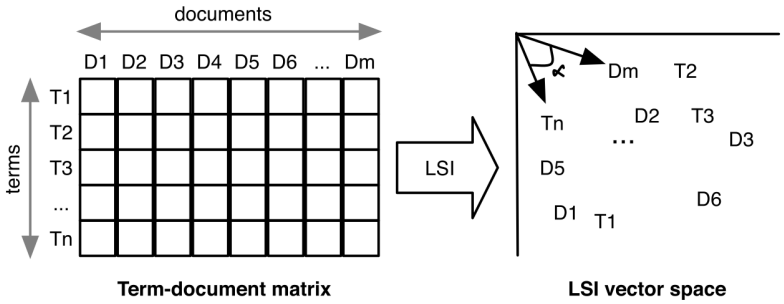


Abbildung 3.2: Umwandlung einer Matrix in einem Vektorraum mittels LSI. [49]

Aus dieser $n \times m$ Matrix, ergeben sich dadurch n Vektoren. Jeder Vektor stellt ein Dokument d_i dar. Mit der Kosinus-Ähnlichkeit [38], [99] werden die Ähnlichkeiten von jedem Dokument zueinander berechnet. Geometrisch zeigen 2 Vektoren $\vec{d_1}$ und $\vec{d_2}$ in die selbe Richtung, wenn sie eine Kosinus-Ähnlichkeit von 1 haben und in die entgegengesetzte Richtung, wenn sie einen Wert von -1 haben. Dabei kann die Kosinus-Ähnlichkeit einen Wert von -1 bis 1 annehmen. In der Formel 3.3 wird die Berechnung der Kosinus-Ähnlichkeit $\cos\theta$ definiert [38], wobei n die Anzahl der Dimension der Vektoren ist und θ der Winkel zwischen den Vektoren $\vec{a}$ und $\vec{b}$.

$$\cos\theta = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2} \cdot \sqrt{\sum_{i=1}^{n}(b_i)^2}} n \tag{3.3}$$

Es wird für jeden Vektor $\vec{d_i}$ die Kosinus-Ähnlichkeit zu allen anderen Vektoren $\vec{d_1}, \vec{d_2}, ..., \vec{d_n}$ berechnet und in eine neue $n \times n$ Matrix eingetragen. Somit ist der Wert a_{ij} die Kosinus-Ähnlichkeit der Dokumente d_i und d_j.

Zur Veranschaulichung haben Kuhn, Ducasse und Girba in [49] eine „correlation matrix“ erstellt. Die Werte sind hier als graue Quadrate definiert. Ist das Quadrat an der stelle a_{ij} dunkel, so repräsentiert es eine hohe Kosinus-Ähnlichkeit zwischen den Dokumenten d_i und d_j. Umso heller das Quadrat an der Stelle a_{ij} ist, umso weniger ähnlich sind sich die Dokumente d_i und d_j. In Abbildung 3.3 sind 4 solcher „correlation matrices“ abgebildet.

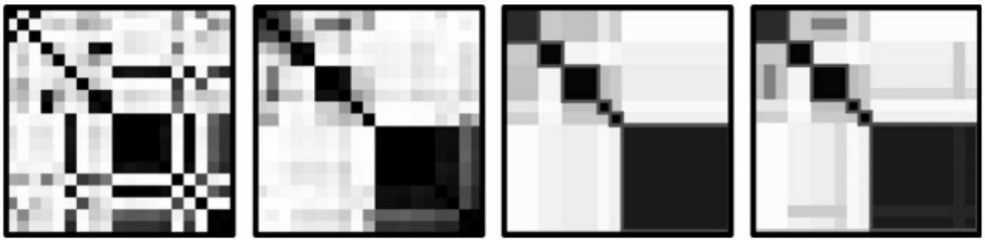


Abbildung 3.3: Beispiel von „correlation matrices“ um das Verfahren von Kuhn, Ducasse und Girba zu erklären. Von links nach rechts: ungeordnete „correlation matrix“ sortiert nach Ähnlichkeit, gruppiert nach Cluster und mit „semantic links“ [49]

Die erste „correlation matrix“ von links ist die ungeordnete Matrix des vorherigen Schrittes. Diese wird auch als „correlation matrix“ dargestellt. Es sind noch keine Muster zu erkennen. Das Einzige was heraussticht sind die Quadrate an den Stellen a_{ij} für $i = j$. Diese Quadrate verlaufen auf der Diagonale fallend von links oben bis rechts unten. Sie werden schwarz dargestellt, da das Dokument $d_i = d_j$ ist.

Der nächste Schritt wird in der Abbildung 3.3 in der zweiten „correlation matrix“ von links dargestellt. Es ist dieselbe „correlation matrix“ wie in der ersten „correlation matrix“ von links, nur ist diese geordnet. Es wurde also die Reihenfolge der Dokumente so verändert, damit ähnliche Dokumente nah beieinander liegen und nicht ähnliche Dokumente weit voneinander entfernt sind. Jedes Quadrat auf der Diagonale repräsentiert hier ein Cluster.

In der dritten „correlation matrix“ von links, wird der weitere Schritt des Verfahrens nach Kuhn, Ducasse und Girba abgebildet. Es wird der Fokus des Clusterings von der Ähnlichkeit der Dokumente zu der Ähnlichkeit der Cluster gelegt. Die Größe der Quadrate gibt an, wieviele Dokumente dem jeweiligen Cluster zugewiesen sind.

Die letzte „correlation matrix“ in Abbildung 3.3 zeigt den letzten Schritt dieses Verfahrens. Hier werden „semantic links“ in der „correlation matrix“ eingefügt. Wenn der Unterschied der Kosinus-Ähnlichkeit eines Dokumentes d_n von einem Cluster A zu einem Durchschnitt eines anderen Cluster B größer ist als ein vorher definierter Schwellenwert, dann wurde ein „semantic link“ gefunden. Er wird grafisch auf der Höhe des Dokumentes d_n und $d_i, d_i + 1, ..., d_j$ eingetragen. Wobei $d_i, d_i + 1, ..., d_j$ die Dokumente des Clusters B sind und d_n ein Dokument des Clusters A. Abbildung 3.4 zeigt dieses Methode grafisch veranschaulicht.

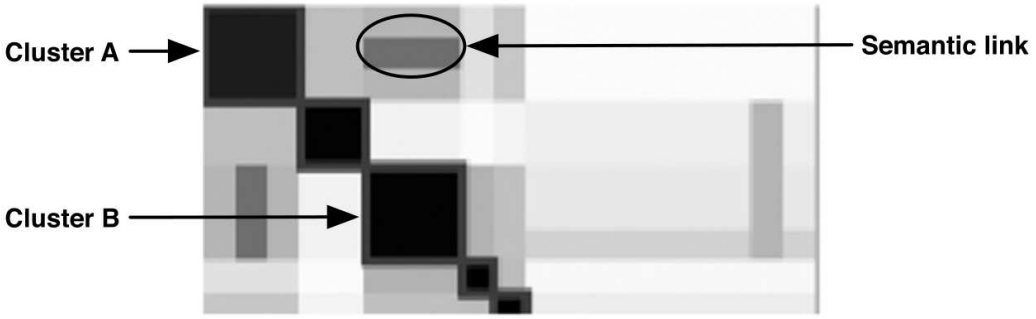


Abbildung 3.4: Illustration eines „semantic links“. Ein Dokument in Cluster A ist ähnlicher zu Cluster B, als seine Nachbarn in Cluster A zu Cluster B. [49]

Als Themen der Cluster werden in [49] die 7 Wörter ausgewählt, die in dem Cluster am häufigsten vorkommen. Ein solches Ergebnis mit den jeweiligen Themen wird in der Abbildung 3.5 für das Softwareprojekt JEdit [41] dargestellt.

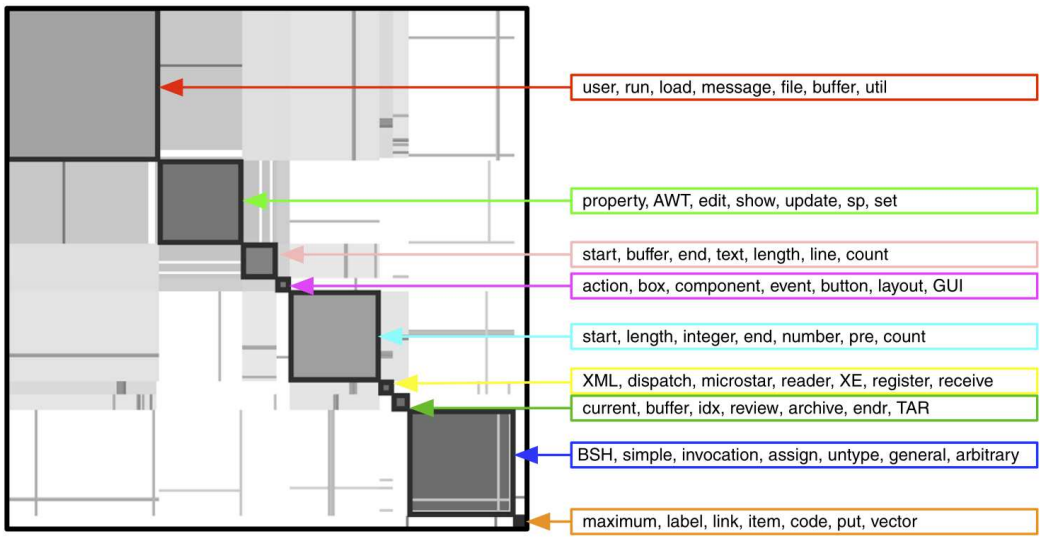


Abbildung 3.5: Semantische Cluster mit top 7 Wörter des jeweiligen Cluster des Softwareprojektes JEdit. [49]

Das Verfahren, welches Kuhn, Ducasse und Girba in [49] vorstellen, liefert Cluster zu Quelltextelementen. Im Gegensatz zu dieser Diplomarbeit, können diese Quelltextelemente eine beliebige Granularität haben. Damit wäre es mit dem Verfahren von Kuhn, Ducasse und Girba möglich, einzelne Funktionen und Methoden in Cluster zu unterteilen. Auch mit dem Verfahren, welches in dieser Diplomarbeit beschrieben wird, wäre es möglich einzelne Funktionen und Methoden in Cluster zu unterteilen. Es wurde jedoch der Fokus auf das Finden von Clustern von Quelltextdateien gelegt, da diese die Softwarearchitektur besser veranschaulichen. Anders als in dieser Diplomarbeit versuchen Kuhn, Ducasse und Girba durch „bag of words “eine „correlation matrix“ abzuleiten und Cluster zu erkennen.

3.3 Weiter wissenschaftliche relevante Dokumente

3.3.1 SDVisu

In [71] beschäftigen sich Reddivari, Kotapalli und Niu mit der Visualisierung der Cluster von Quelltextdateien.

Abbildung 3.6 zeigt einen Screenshot der Software SDVisu (static-dependency-based visual clustering), die im Laufe der Arbeit zu [71] entwickelt worden ist.

Das Programm erwartet einen „dependency graph“ als Input. Dieser „dependency graph“ soll von dem Softwareprojekt stammen, von welchem eine Visualisierung gewünscht wird. Der „dependency graph“ ist ein Graph, in dem die Knoten Quelltextdateien und die Kanten „static dependencies“ zwischen den Quelltextdateien sind. Unter „static dependencies“ werden Abhängigkeiten zu anderen Quelltextdateien verstanden, die vor der Laufzeit des Programmes erkannt werden. Um aus diesen „dependency graph“ Cluster abzuleiten verwenden Reddivari, Kotapalli und Niu das Verfahren „energy value metric“ [57].

Die 4 gelb markierten Punkte in Abbildung 3.6, die mit Buchstaben beschriftet sind, zeigen die verschiedenen Ansichten der Applikation. Die Ansicht, die mit dem Buchstaben 'A' beschriftet ist, soll einen globalen Überblick über den „dependency graph" geben. Die Ansicht mit dem Buchstaben 'D' ist eine vergrößerte Ansicht von der Ansicht die mit dem Buchstaben 'A' beschriftet ist, um den Graphen besser zu erkennen. Die Buchstaben 'B' und 'C' beschriften das „control panel" der Applikation, über welches die Applikation gesteuert wird, wobei 'B' eine vereinfachte Ansicht und 'C' eine erweiterte Ansicht darstellt.

Abbildung 3.6: Screenshot der Applikation SDVisu [71]

Reddivari, Kotapalli und Niu versuchen mit einen „dependency graph" und der „energy value metric" [57] Cluster zu finden. Die Abhängigkeiten der Quelltextdateien fliessen zwar in dieser Diplomarbeit auch in das Cluster Verfahren ein, aber es werden noch weitere Features aus dem Quelltext extrahiert. Außerdem ist der Fokus bei dieser Diplomarbeit nicht so sehr auf die Darstellung der Ergebnisse gerichtet sondern richtet sich mehr auf die Methodik.

3.3.2 Software Features Extraction von objektorientierten Quelltext mittels Clustering

Araar et al. [5] präsentieren eine Lösung, um das „Feature Model" eines Softwareprojektes aus dessen Quelltext herzuleiten. In Abbildung 3.7 sind die notwendigen Schritte des Verfahrens abgebildet.

Als ersten Schritt verwenden Araar et al. das Kommandozeilenprogramm „DependencyExtractor", um von Byte Code Dateien einen „dependency graph" zu bekommen. Dieses Programm ist ein Teil von „JDependencyFinder" [40].

Danach wird die „Similarity Matrix" nach dem Verfahren von Floyd [26] and Warshall [96] berechnet und diese mit dem „OClustR" [63] Algorithmus in Cluster aufgeteilt.

Nach dem Aussortieren von irrelevanten Clustern kommen Araar et al. dann auf das gewünschte „Feature Model" Dabei definieren sie irrelevante Cluster als Cluster, in denen keine Klassen vorkommen.

Genauso wie diese Diplomarbeit hat [5] das Ziel, aus bestehendem Quelltext Informationen abzuleiten, um die Dokumentation des Projektes zu verbessern. Dadurch soll die Einarbeitung in das

-	PL	Platform	S/W Type	Plan	Modern Practice	Dev Model	Priority	Risk	Release Type
P_1	C	PC	Application	No	Yes	Waterfall	Low	Less	1.0
P_2	C++	PC	Application	No	Yes	Iterative	Low	Less	1.0
P_3	Java	Mobile	Embedded	No	Yes	Agile	High	Medium	1.0
P_4	C	Mobile	Embedded	No	Yes	Spiral	Low	Less	1.0
P_5	C	Mobile	Embedded	No	Yes	Prototype	Low	Less	1.0
P_6	Cobol	Mainframe	Application	Yes	No	Iterative	Medium	Medium	2.0
P_7	Java	Mobile	Embedded	No	Yes	Spiral	Low	Less	2.0
P_8	C#	Mobile	Embedded	No	Yes	Agil	Low	Less	2.0
P_9	Java	Mobile	Embedded	No	Yes	Spiral	Low	Less	1.0

Tabelle 3.1: Component Attribute Matrix [85]

Projekt einfacher gestaltet werden. Dabei fokussieren sich Araar et al. auf die Programmiersprache Java und bieten keine Lösung für mehrere Programmiersprachen an, im Gegensatz zu dieser Diplomarbeit, die sich mit den Sprachen Java, JavaScript und Python beschäftigt. Als Resultat liefern Araar et al. ein automatisch generiertes „Feature Model“, das mittels Clustering von „static dependencies“ erzeugt wurde. Mit einem solchen „Feature Model“ bekommt man in kürzester Zeit einen Überblick über die Features des Softwareprojektes. Dadurch unterscheidet sich das Resultat von Araar et al. von dem Resultat dieser Diplomarbeit, welches Cluster von Quelltextdateien darstellt, um die Softwarearchitektur des Projektes abzuleiten.

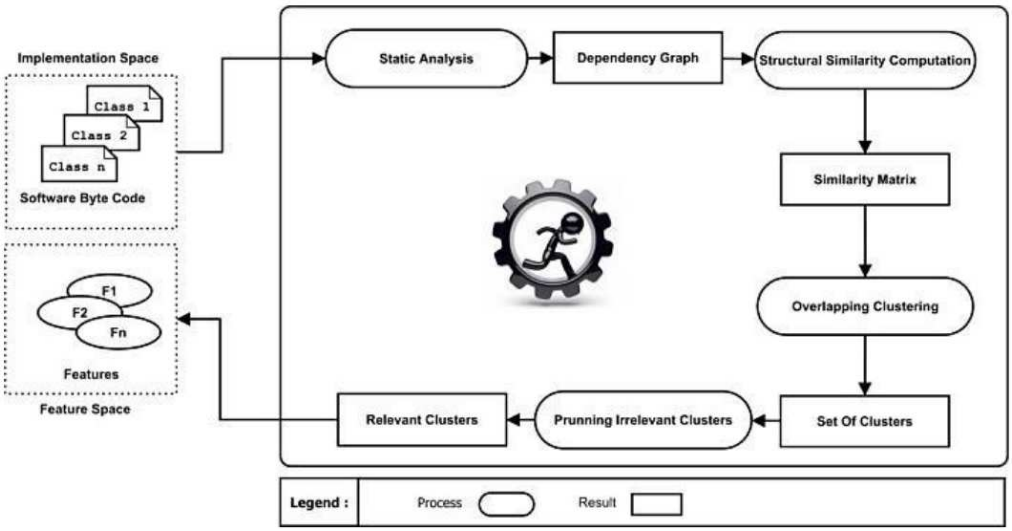


Abbildung 3.7: Prozess des Verfahrens nach Araar et al. [5]

3.3.3 Segmentierung von Software Komponenten mittels Feature Vector

Srinivas und Rao bilden Cluster aus Softwarekomponenten mit der Hilfe von „Feature Vectors“ [84]. „Feature Vectors“ sind Vektoren der Größe k, wobei k die Anzahl von Attributen ist, welche die Softwarekomponente definiert. Für alle n Softwarekomponenten $P_0, P_1, ..., P_n$ wird ein solcher „Feature Vector“ gebildet. In Tabelle 3.1 wird ein Beispiel eines solchen „Feature Vectors“ in einer „Component Attribute Matrix“ dargestellt. Jede Zeile der Matrix stellt einen „Feature Vector“ dar.

Sie verwenden eine Mapping Funktion $\Phi(P_{ik}, P_{jk})$, welche zwei Softwarekomponenten P_i und P_j einen Wert zwischen 0 und 1 zuweist. Dieser Wert repräsentiert die Ähnlichkeit dieser zwei

-	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
P_1	0.8289	0.6623	0.5511	0.8289	0.8289	0.5521	0.7178	0.7178	0.7734
P_2	-	0.6074	0.6629	0.7185	0.7185	0.6629	0.6629	0.6629	0.7815
P_3	-	-	0.5504	0.7727	0.7727	0.5504	0.7727	0.7727	0.8283
P_4	-	-	-	0.5525	0.9383	0.4939	0.8828	0.8828	0.9383
P_5	-	-	-	-	0.8282	0.4949	0.8282	0.8282	0.8838
P_6	-	-	-	-	-	0.8289	0.5525	0.5525	0.4969
P_7	-	-	-	-	-	-	0.8289	0.7291	0.9383
P_8	-	-	-	-	-	-	-	0.8289	0.8272
P_9	-	-	-	-	-	-	-	-	0.9213

Tabelle 3.2: Beispiel einer „Similarity Matrix“ für die „Feature Vectors“ aus Tabelle 3.1. [85]

Softwarekomponenten, wobei der Wert 1 bedeutet, dass P_{ik} und P_{jk} gleich sind und der Wert 0 bedeutet, dass P_{ik} und P_{jk} verschieden sind.

Aus diesen n „Feature Vectors“ und einer Mapping Funktion $\Phi(P_{ik}, P_{jk})$ bilden Srinivas und Rao eine „Similarity Matrix“ Eine „Similarity Matrix“ ist eine $n \times n$ große Matrix, wobei n die Anzahl der „Feature Vectors“ ist. Tabelle 3.2 zeigt ein Beispiel einer solchen Matrix mit erfundenen Werten zur Veranschaulichung.

Durch Anwenden eines auf k-Means 2.5.1 basierenden Clusterverfahrens, kommen sie dann auf die Cluster der Softwarekomponenten.

4 Anforderungsanalyse

Dieses Kapitel beschäftigt sich mit den Anforderungen dieser Diplomarbeit. Dazu wird in Kapitel 4.1 auf die fachlichen Probleme und deren Anforderungen eingegangen. Abschnitt 4.2 betrachtet die Eigenschaften einer Quelltextdatei. Zum Abschluss beschreibt das Unterkapitel 4.3 das „Preprocessing und Feature engineering“.

4.1 Anforderungen an die Applikation

Das Resultat der Applikation, welche im Rahmen dieser Diplomarbeit verfasst wurde, ist ein Dokument, welches verschiedene Komponenten eines Softwareprojektes auflistet. Diese Softwarekomponenten werden nach ihren Ähnlichkeiten gegliedert und geben so einen Überblick über die projektspezifische Softwarearchitektur. Dieses Dokument wird automatisch erzeugt und verursacht keinen zusätzlichen Aufwand. Dadurch ist die Wartung des Dokumentes auch mit keinem zusätzlichen Aufwand verbunden.

Mit Hilfe des Dokumentes erhalten Personen, die sich gerade in das Softwareprojekt einarbeiten, einen schnellen Überblick über Gemeinsamkeiten verschiedener Softwarekomponenten im Softwareprojekt, dadurch sind sie in der Lage, sich in der Architektur dieses Softwareprojektes schneller zurecht zu finden.

Dieser Überblick wird in Form von zwei verschiedenen Visualisierungen dargestellt. Beide Visualisierungen sind in dem resultierenden Dokument erhalten. Eine Visualisierung gibt die Ordnerstruktur des Softwareprojektes aus, wobei die Knoten mit Farben eingefärbt werden. Diese Farben symbolisieren die verschiedenen Cluster. In Abbildung 4.1 wird der Designvorschlag einer solchen Visualisierung abgebildet. Der Designvorschlag der zweiten Visualisierung, die in dieser Diplomarbeit umgesetzt wird, ist in Abbildung 4.2 dargestellt. Bei dieser Visualisierung werden alle gefundenen Cluster als Listen visualisiert. Auch bei dieser Visualisierung sind alle Cluster eingefärbt. Die Farben aus beiden Visualisierungen sollen ident sein. Dementsprechend soll jeder Cluster in beiden Visualisierungen die gleiche Farbe besitzen. Die Baumstruktur wurde gewählt, da Softwareprojekte bereits mit einer Ordnerstruktur gegliedert werden und so Unterschiede und Gemeinsamkeiten von Quelltextdateien aus verschiedenen Ordnern zur Geltung kommen. Die Listenansicht gibt einen Überblick von allen Quelltextdateien der jeweiligen Cluster. Hier muss nicht in Ordnern gesucht und verglichen werden. Die Baumansicht ergänzt die Listenansicht, um einen besseren Überblick über die Architektur der Softwareprojektes zu erhalten.

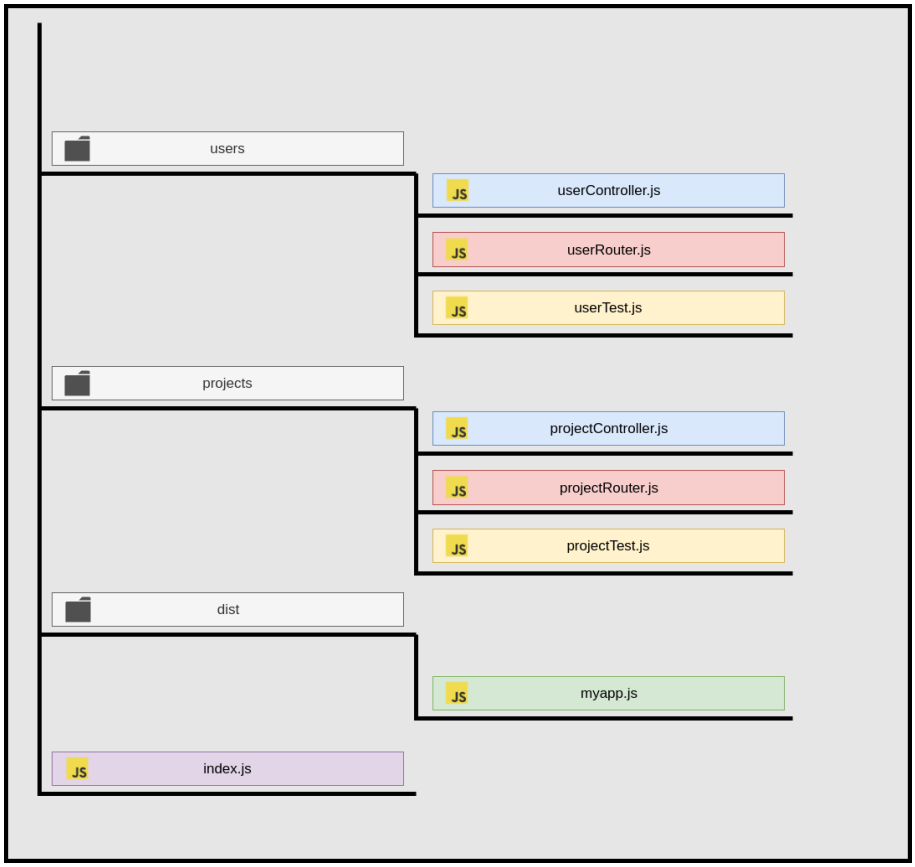


Abbildung 4.1: Design der Baumstruktur Visualisierung eines fiktiven Software Projektes

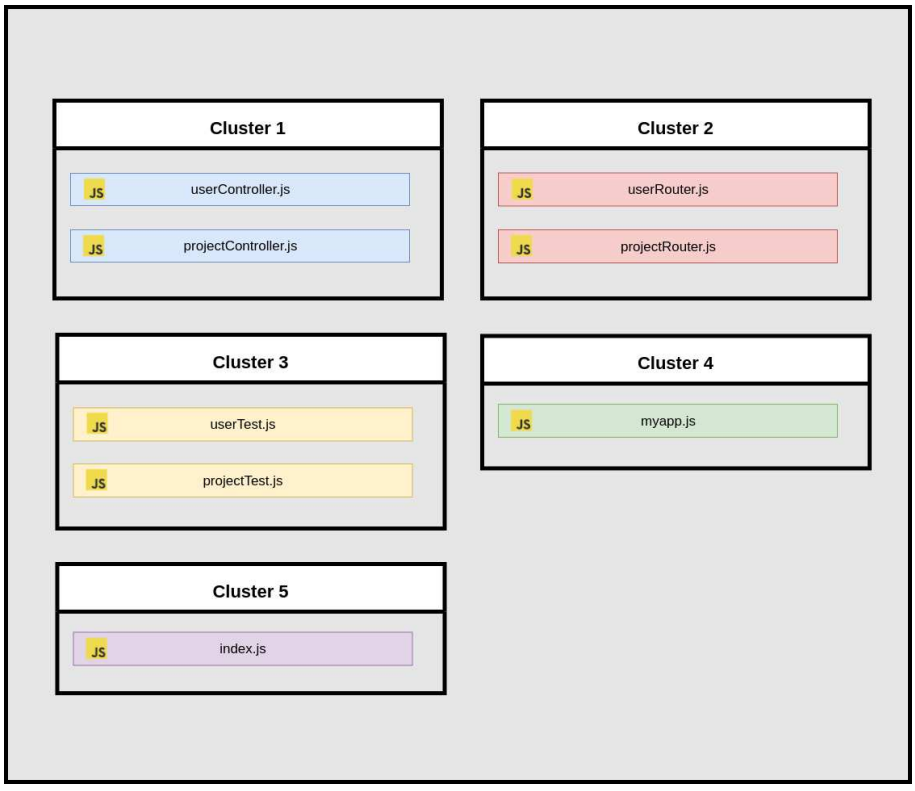


Abbildung 4.2: Design der Liste Visualisierung eines fiktiven Software Projektes

Auf diesem Dokument aufbauend, könnte man in Zukunft noch weitere Visualisierungen entwickeln, mit denen sich das Verständnis über die projektbezogene Softwarearchitektur noch effektiver fördern ließe. Darüber hinaus könnte man aufbauend auf die Ergebnisse dieser Diplomarbeit auch das Verfahren des Clustering und die Verfahren „Preprocessing und Feature engineering“ optimieren, indem man die Anzahl der Eigenschaften aus dem Quelltext des Softwareprojektes anpasst.

4.2 Eigenschaften einer Quelltextdatei

Ein Problem, dass im Rahmen dieser Diplomarbeit gelöst werden muss, ist Quelltextdateien eines Softwareprojektes als Eingabe Parameter eines Clusteringverfahrens zu verwenden. Dafür muss aus den Quelltextdateien eine $n \times m$ Matrix abgeleitet werden. Der Wert n ist die Anzahl der Quelltextdateien in dem Softwareprojekt und der Wert m ist die Anzahl der „Features“ der Quelltextdateien. Diese „Features“ sind Eigenschaften, die aus dem Quelltext extrahiert werden können. Dieses Kapitel beschreibt verschiedene Eigenschaften einer Quelltextdatei. Anhand der Erkenntnisse, die in diesem Kapitel erklärt werden, wird in Kapitel 5 ein Algorithmus für das „Preprocessing und Feature engineering“ entwickelt.

Um die Eigenschaften eines Softwareprojektes darzustellen, wurde im Rahmen dieser Diplomarbeit eine Mindmap [12] erstellt. Diese Mindmap veranschaulicht welche Eigenschaften aus einem Softwareprojekt ausgelesen werden können und ist in der Abbildung 4.3 ersichtlich.

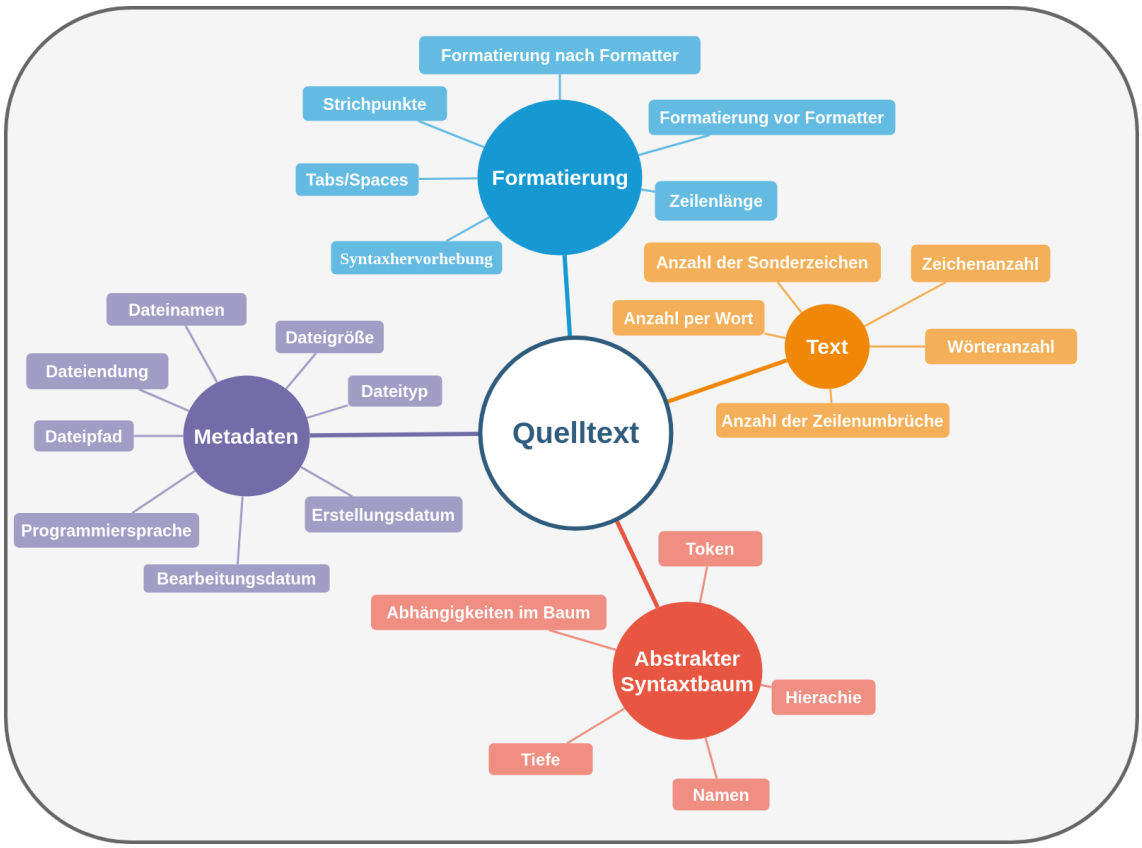


Abbildung 4.3: Mindmap für Eigenschaften eines Softwareprojektes

Die Mindmap, die in Abbildung 4.3 dargestellt ist, hat eine Baumstruktur. In dieser Baumstruktur werden Begriffe als Knoten und Kanten als Zusammenhänge der Begriffe dargestellt. Der Wurzelknoten bestimmt das Themengebiet und ist in der Abbildung 4.3 in der Mitte dargestellt und

mit dem Text „Quelltext“ beschriftet. Es wurden vier Themengebiete erkannt, aus denen Eigenschaften gefunden werden können und die sich dem Thema „Quelltext“ unterordnen. Diese vier Themengebiete sind in der folgenden Auflistung definiert und werden in den jeweiligen Unterkapiteln genauer beschrieben.

- (Metadaten (4.2.1)
- (Text 4.2.2)
- (Abstrakter Syntaxbaum 4.2.3)
- (Formatierung 4.2.4)

4.2.1 Metadaten

Jede gespeicherte Datei besitzt zusätzlich zur ihrem eigentlichen Inhalt auch eine Reihe an Metadaten. Diese werden beim Anlegen oder Bearbeiten der Datei automatisch gespeichert. Einige Metadaten können auch von dem Benutzer manuell eingegeben und bearbeitet werden. Es gibt zu unterschiedlichen Dateiformaten unterschiedliche Metadaten. Zum Beispiel haben Fotos Metadaten über die Kameraeinstellung wie Blende, Belichtungszeit und ISO Wert. In Abbildung 4.3 sind die Metadaten auf der linken Seite mit der Farbe violett gekennzeichnet.

Im Rahmen der Diplomarbeit haben sich folgende Metadaten als nützliche Eigenschaften, um Quelltextdateien in Cluster zu gruppieren, ergeben:

- Dateiname
- Dateigröße
- Dateityp
- Erstellungsdatum
- Bearbeitungsdatum
- Programmiersprache
- Dateipfad
- Dateiendung

4.2.2 Text

Wie in Abbildung 4.4 ersichtlich, kann jede Datei unterschiedliche Kategorien eingeordnet werden.

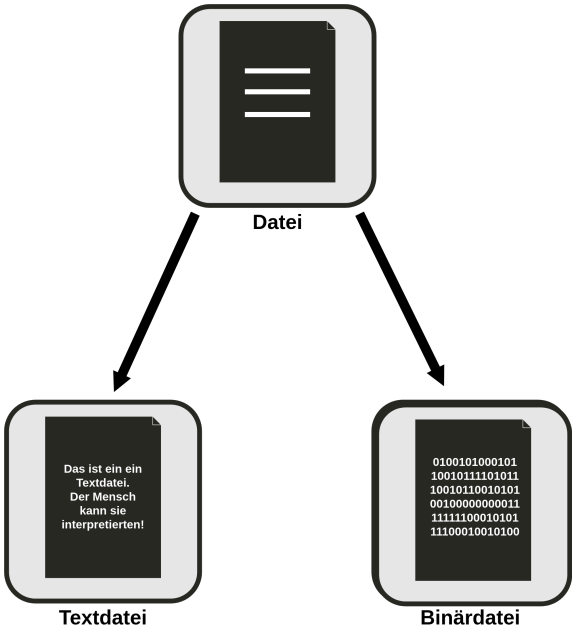


Abbildung 4.4: Kategorisierung von Dateien in Text- und Binärdateien

Diese Kategorien sind Binärdateien und Textdateien. Binärdateien sind dabei Daten, in denen deren Inhalt als Binärmuster gespeichert ist. Dadurch sind Binärdateien für den Menschen kaum lesbar, weil der Kontext hinter dem Binärmuster verstanden werden muss, um die Datei interpretieren zu können. Dazu gibt es zu den verschiedenen Dateiformaten unterschiedliche Programme, um diese Dateien zu interpretieren und deren Inhalt für den Menschen in verständlicher Form anzeigen zu können. Die folgende Auflistung gibt Beispiele für weit verbreitete Binärdateien und deren Formate:

- **Bilder**: jpg, png, gif, bmp, tiff, psd, ...
- **Videos**: mp4, mkv, avi, mov, mpg, vob, ...
- **Audio**: mp3, aac, wav, flac, ogg, mka, wma, ...
- **Dokumente**: pdf, doc, xls, ppt, docx, odt, ...
- **Archive**: zip, rar, 7z, tar, iso, ...
- **Datenbanken**: mdb, accde, frm, sqlite, ...
- **Executable**: exe, dll, so, class, ...

Textdateien sind im Gegensatz zu Binärdateien für den Menschen interpretierbar. Es sind Wörter in Textdateien enthalten, die für den Menschen lesbar sind und woraus der Kontext der Datei ableitbar ist. Eine Textdatei kann dabei aber auch Sonderzeichen enthalten. Die folgende Auflistung gibt Beispiele für weit verbreitete Textdateien und deren Formate:

- **Web Standards**: html, xml, css, svg, json, ...
- **Quelltext**: c, cpp, h, cs, js, py, java, rb, pl, php, sh, ts, ...
- **Dokumente**: txt, tex, md, rtf, ps, ...

- **Konfiguration**: config, ini, cfg, rc, ...
- **Datenbanken**: mdb, accde, frm, sqlite, ...
- **Tabellen**: csv, tsv, ...

Da es sich bei Quelltextdateien um Textdateien handelt, kann der Mensch diese interpretieren. Dazu wird der Text in den Dateien gelesen. Der Compiler oder Interpreter verwendet für die Interpretation der Daten einen abstrakten Syntaxbaum. Dieser wird in Kapitel 4.2.3 erklärt. Aus dem Text einer Textdatei lassen sich folgende Eigenschaften extrahieren:

- Anzahl der Sonderzeichen
- Zeichenanzahl
- Wörteranzahl
- Anzahl per Wort
- Anzahl der Zeilenumbrüche

Wie in der Auflistung ersichtlich, werden die Eigenschaften durch quantitative Messungen im Text extrahiert. Dabei können diese Messungen auf die ganze Datei bezogen werden oder auf Abschnitte oder Zeilen in der Datei. So kann beispielsweise nicht nur die Anzahl der Sonderzeichen in der gesamten Datei errechnet werden, sondern auch die Anzahl der Sonderzeichen in der ersten Hälfte der Datei oder in der ersten Zeile.

4.2.3 Abstrakter Syntaxbaum

Ein abstrakter Syntaxbaum wird verwendet, damit ein Compiler oder Interpreter eine Quelltextdatei interpretieren kann. Dazu wird eine lexikalische Analyse des Quelltextes durchgeführt. Diese lexikalische Analyse teilt den Quelltext der Datei in Folgen von logisch zusammengehörigen Einheiten auf. Solche Einheiten werden als Tokens bezeichnet [97].

Abbildung 4.5 zeigt eine lexikalische Analyse anhand eines Beispiels. Links neben dem „Lexical Analyzer“ in Abbildung 4.5 wird der Quelltext definiert, der mithilfe der lexikalischen Analyse, Tokens auflistet. Ebenfalls wird rechts auf der Abbildung, neben dem „Lexical Analyzer“, die Liste der erkannten Tokens angeführt. Zusätzlich ist auf der Abbildung auch die Option dargestellt, dass die lexikalische Analyse im Quelltext einen Syntax-Fehler erkennt.

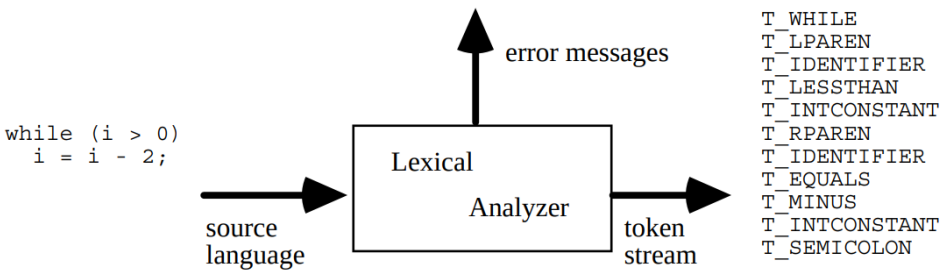


Abbildung 4.5: Lexikalische Analyse anhand eines Beispiels [2]

Ein abstrakter Syntaxbaum ist ein Baum, in dem die Knoten Programmkonstrukte und die Kanten die Zusammenhänge zwischen den Programmkonstrukten darstellen. Dadurch bildet der Baum die Grammatik des Quelltextes ab. Abbildung 4.6 zeigt einen abstrakten Syntaxbaum [14].

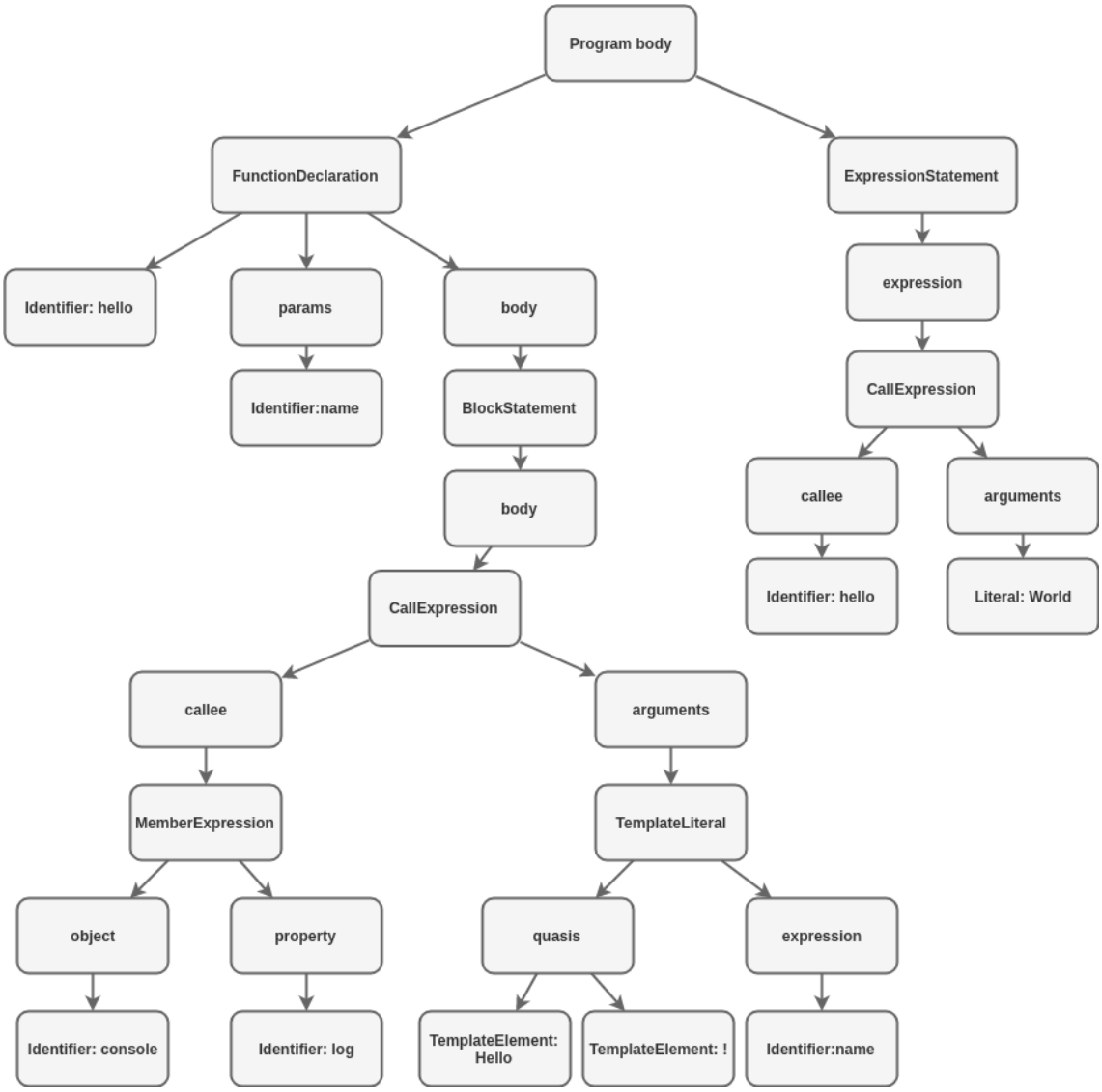


Abbildung 4.6: Lexikalische Analyse anhand eines Beispiels [2]

Dieser abstrakte Syntaxbaum wurde aus dem Quelltext 4.1 abgeleitet und mit dem Werkzeug esprima [21] erzeugt. Der Beispiel-Quelltext ist ein einfaches „Hello World“ Programm in der Programmiersprache JavaScript.

```
1 function hello( name) {
2   console.log(`Hello ${name}!`);
3 }
4
5 hello('World');
```

Listing 4.1: Beispiel eines Quelltextes zur Veranschaulichung eines abstrakten Syntaxbaumes in der Programmiersprache JavaScript.

Wie in der Abbildung 4.6 ersichtlich, stellt die Repräsentation eines Quelltextes in Form eines Abstrakten Syntaxbaums mehr Informationen zur Verfügung als nur der Quelltext selbst, weil auch die Grammatik der Programmiersprache in den abstrakten Syntaxbaum miteinfließt.

Diese Informationen und die Informationen der lexikalischen Analyse bieten eine Reihe von Eigenschaften eines Quelltextes, die für eine Segmentierung in Betracht gezogen werden können. Diese Informationen sind in der folgenden Auflistung ersichtlich:

- Token
- Hierachie
- Namen
- Tiefe
- Abhängigkeiten im Baum

4.2.4 Formatierung

Die Formatierung des Quelltextes hat erhebliche Auswirkungen auf die Verständlichkeit des Programmes für den Menschen [53].

Durch ein einheitliches Format des Quelltextes im gesamten Softwareprojekt, schafft man durch das bessere Verständnis auch eine bessere Vergleichbarkeit. Zur besseren Veranschaulichung zeigen die Quelltextbeispiele 4.2 und 4.3 denselben Quelltext unterschiedlich formatiert.

```
function HelloWorld({greeting = "hello", greeted = '"World"', silent =
false, onMouseOver,}) {

  if(!greeting){return null};

     // TODO: Don't use random in render let num = Math.floor
(Math.random() * 1E+7).toString().replace(/\.\d+/ig, "")

  return <div className='HelloWorld' title={`You are visitor number ${
num }`} onMouseOver={onMouseOver}>

    <strong>{ greeting.slice( 0, 1 ).toUpperCase() +
greeting.slice(1).toLowerCase() }</strong> {greeting.endsWith(",") ? "
" : <span style={{color: '\grey'}}>", "</span> } <em> { greeted }
</em> { (silent) ? "." : "!"}

    </div>;
```

Listing 4.2: Unformatierter Beispiel-Quelltext in JavaScript

Die Eigenschaften des formatierten Quelltextes lassen sich aus den Quelltextdateien extrahieren, um ein Clustering von Quelltextdateien zu ermöglichen. Folgende Eigenschaften bezüglich der Formatierung des Quelltextes können extrahiert werden:

- Formatierung nach Formatter
- Formatierung vor Formatter
- Zeilenlänge

```
function HelloWorld({
   greeting = "hello",
   greeted = '"World"',
   silent = false,
   onMouseOver
}) {
   if (!greeting) {
      return null;
   }

   // TODO: Don't use random in render
   let num = Math.floor(Math.random() * 1e7)
      .toString()
      .replace(/\.\d+/gi, "");

   return (
      <div
         className="HelloWorld"
         title={`You are visitor number ${num}`}
         onMouseOver={onMouseOver}>
         <strong>
            {greeting.slice(0, 1).toUpperCase() +
               greeting.slice(1).toLowerCase()}
         </strong>
         {greeting.endsWith(",") ? (
            " "
         ) : (
            <span style={{ color: "grey" }}>", "</span>
         )}
         <em>{greeted}</em>
         {silent ? "." : "!"}
      </div>
   );
}
```

Listing 4.3: Formatierter Beispiel-Quelltext in JavaScript

- Syntaxhervorhebung
- Leerzeichen
- Strichpunkte und sonstige relevanten Sonderzeichen

4.3 Preprocessing und Feature engineering

Im Kapitel 4.2 sind die Eigenschaften einer Datei aufgelistet und beschrieben worden. Damit eine Datei als Eingabeparameter für einen Algorithmus im Bereich des maschinellen Lernens einsetzbar wird, muss aus den gefundenen Eigenschaften der Datei ein m-dimensionaler Vektor gemacht werden. Wobei m die Anzahl der Eigenschaften der Datei ist. Außerdem muss jeder Wert in diesem m-dimensionalen Vektor mit einer Zahl repräsentiert werden [61][67][100].

Dieser Prozess besteht aus einer Reihe von Teilprozessen und wird unter den Begriffen „Data preprocessing“ und „Feature engineering“ zusammengefasst. Abbildung 4.7 zeigt die Zusammenhänge dieser zwei Begriffe über den gesamten Prozess.

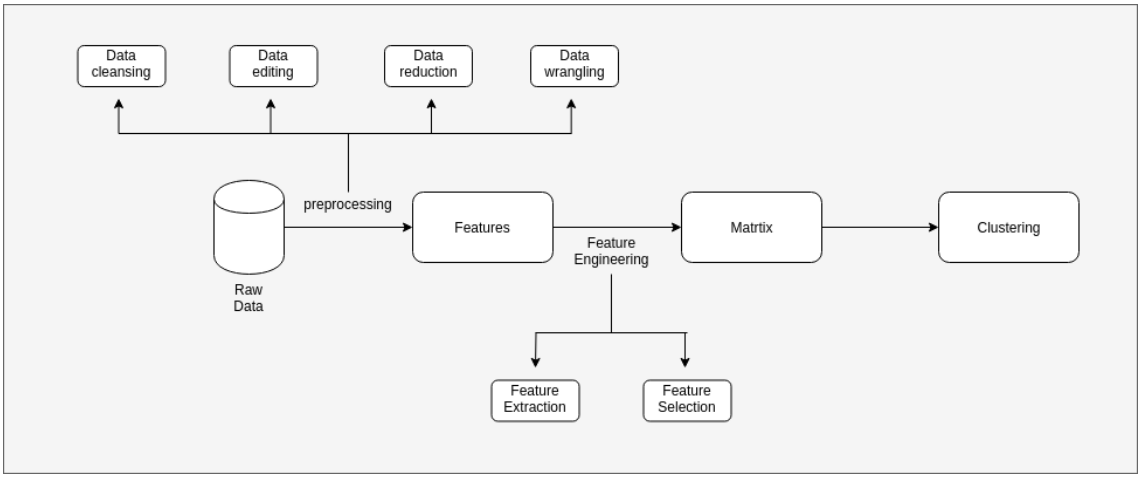


Abbildung 4.7: Übersicht und Zusammenhänge der „Preprocessing und Feature engineering“ Prozesse

Neben dem gesamten Prozess bildet die Grafik 4.7 auch die detailierten Prozesse für „Data preprocessing“ and „Feature engineering“ ab. Diese Prozesse sind in der folgenden Auflistung erklärt.

- **Data cleansing**

  Unter „Data cleansing“ wird das Entfernen korrupter und falscher Daten aus den rohen Daten verstanden.

- **Data editing**

  „Data editing“ ist das Editieren der gesammelten rohen Daten um eine bessere Respräsentation für den „Machine Learning“ Algorithmus bereitstellen zu können.

- **Data reduction**

  „Data reduction“ bezeichnet das Verfahren, die rohen Daten in eine geordnete simplifizierte Form zu transferieren.

- **Data wrangling**

  „Data wrangling“ wird als die Transformation der rohen Daten zu einer, von dem „Machine Learning“-Algorithmus erwarteten, Repräsentation der Daten bezeichnet.

- **Feature extraction**

  Der Prozess „Feature extraction“ extrahiert zusätzlich Information aus den bestehenden Daten. Dazu werden Methoden verwendet, um aus den bestehenden Daten Information abzuleiten. Diese Methoden sind von den rohen Daten abhängig.

- **Feature selection**

  Unter „Feature selection“ versteht man den Prozess, aus bestehenden Daten die Teilmenge zu selektieren, mit welcher der „Machine Learning“ Algortihmus, das optimalste Ergebnis liefert.

Diese Aufteilung der Teilprozesse ist sehr granular und in der Praxis schwer von einander zu trennen. Wie in Abbildung 4.7 abgebildet, ist das Ergebnis des „Preprocessing und Feature engineering“ eine $n \times m$ Matrix, in der n die Anzahl an Elemente ist und m die Anzahl an Eigenschaften, die im Zuge dieses Prozesses, abgeleitet wurden.

5 Implementierung

In diesem Kapitel wird die Implementierung einer Applikation beschrieben. Diese Applikation soll es ermöglichen, aus Quelltextdateien eines Softwareprojektes, die projektbezogene Softwarearchitektur dieses Projektes abzuleiten. Dazu werden Algorithmen und Verfahren verwendet, die schon in Kapitel 2 beschrieben wurden. Die Entwicklung der Applikation orientiert sich an dem iterativen Entwicklungsmodel [11]. Diese Methode der Softwareentwicklung wird in Kapitel 5.1 erklärt. Die restlichen Kapitel beschreiben die iterativen Phasen, die sich aus diesem Entwicklungsmodels ergeben haben und sind chronologisch nach den Iterationen gegliedert.

5.1 Iteratives Entwicklungsmodell

Für die Entwicklung des Programmes ist ein iterativer Ansatz gewählt worden, um regelmäßig Feedback zu bekommen. Dadurch können Adaptionen zur Verbesserung vorgenommen werden oder Fehler zum frühest möglichen Zeitpunkt rückgängig gemacht werden [9].

Bei der iterativen Softwareentwicklung wird das Softwareprojekt kontinuierlich durch permanente Wiederholungen optimiert. Nach einer Iteration wird das Ergebnis mit dem erwarteten Ergebnis verglichen und entschieden, welche Änderungen und Optimierungen in die nächste Wiederholung einfliessen sollen. [93]

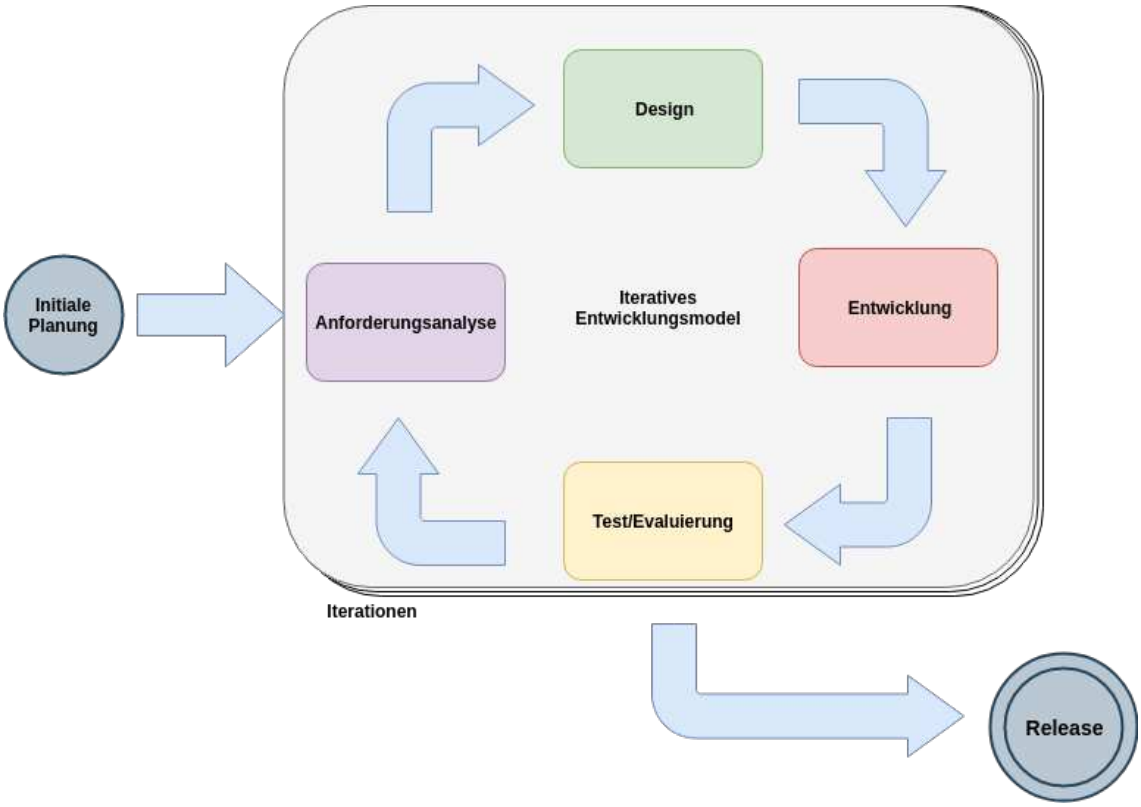


Abbildung 5.1: Phasen des iteratives Entwicklungsmodells

Abbildung 5.1 veranschaulicht dieses Software-Entwicklungsmodell. In der folgenden Auflistung sind alle Phasen des iterativen Entwicklungsmodells beschrieben.

- **Initiale Planung**

  In der Phase der initialen Planung werden technische und organisatorische Spezifikationen definiert, die für das Softwareprojekt relevant sind.

- **Anforderungsanalyse**

  In der Anforderungsanalyse werden die Anforderungen an die jeweilige Iteration definiert. Es wird beschrieben, welche Optimierung diese Iteration an dem Programm vornehmen soll. Detailliert wird darauf in jeder Iteration eingegangen, die in den Kapiteln 5.3 bis 5.6 erklärt werden.

- **Design**

  In der Design Phase wird nach einer Lösung gesucht, um die Anforderungen, die in der Anforderungsanalyse definiert wurden, zu lösen.

- **Entwicklung**

  In dieser Phase findet die Implementierung des Designs statt. Das Resultat dieser Phase ist ein Programm, dass in der nächsten Phase getestet und evaluiert werden kann.

- **Test/Evaluierung**

  Das Programm, dass in der vorherigen Phase entwickelt worden ist, wird in diesem Schritt getestet. Außerdem findet eine Evaluierung statt, ob alle Anforderungen der Anforderungsanalyse mit der Implementierung gedeckt sind. In dieser Phase wird entschieden, ob weitere Iterationen notwendig sind, um das Softwareprojekt zu optimieren. Es kann auch zu dem Resultat kommen, dass die Implementierung dieser Phase verworfen wird und ein anderer Ansatz im nächsten Schritt gewählt wird.

- **Release**

  Ergibt sich aus der Evaluierung, dass alle Anforderungen gedeckt sind, die das Softwareprojekt für eine Veröffentlichung benötigt, so wird dieses veröffentlicht.

5.2 Initiale Planung

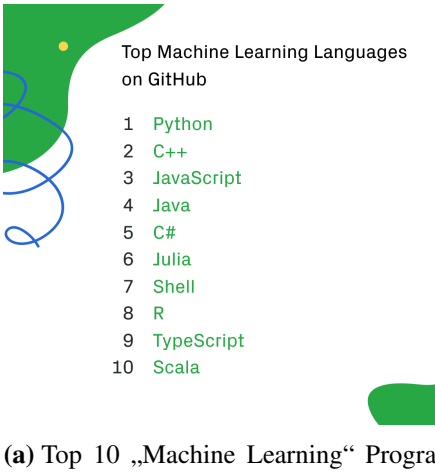
Am Beginn des Softwareprojektes steht, im iterativen Entwicklungsmodell, die Phase der initialen Planung. Hier werden Entscheidungen getroffen, die das Softwareprojekt für die iterative Entwicklung benötigt. Dabei werden organisatorische und technische Spezifikationen für das Softwareprojekt definiert. Diese Spezifikationen, die sich aus der initialen Planung ergeben haben sind in diesem Kapitel erklärt.

Die Kommandozeilenapplikation, die im Rahmen dieser Diplomarbeit entwickelt wurde, ist in der Programmiersprache Python verfasst. Abbildung 5.12a zeigt die Top 10 Programmiersprachen im Jahr 2018 auf der Seite GitHub [28]. Diese Resultate sind Teil des „Octoverse“ [59] Reports, der jedes Jahr veröffentlicht wird. Grundlage dieses Reports sind Softwareprojekte, welche den Quelltext auf dieser Platform verwalten. GitHub ist die Quelltext-Hosting Platform mit den meisten Seitenaufrufen und den meisten Projekten im Jahr 2018 [4] [82] [28].

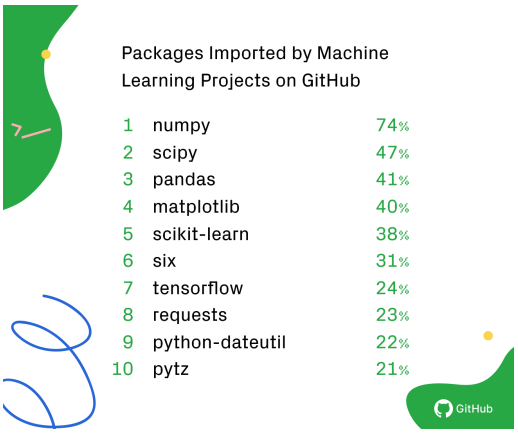
Wie in der Abbildung 5.12a zu erkennen ist, ist die Programmiersprache Python die Programmiersprache, mit der 2018 die meisten Projekte geschrieben wurden, die mit dem Tag „Machine

Learning“ gekennzeichnet wurden. Zusätzlich bildet die Abbildung 5.12b die Top 10 Bibliotheken ab, die in Python Projekten verwendet wurden und die mit dem Tag „Machine Learning“ gekennzeichnet sind. Daraus ergibt sich, dass die Programmiersprache Python ein optimales Ökosystem für Projekte anbietet, die sich mit dem Thema „Machine Learning“ beschäftigt. Ebenfalls in diesem Report vorhanden, waren die Top 10 Softwareprojekte, die sich dem Thema „Machine Learning“ widmen. Diese Liste ist in Abbildung 5.12d dargestellt. Auch hier sind die ersten zwei Projekte in der Sprache Python verfasst. Der erste Platz in diesem Ranking, tensorflow [90], ist eine Bibliothek, die sich auf das Teilgebiet „Deep Learning“ fokussiert hat. Der zweite Platz scikit-learn [78], bietet Module und Funktionen für Klassifikation, Regression und Clustering an. Da sich diese Diplomarbeit mit dem Teilgebiet Clustering beschäftigt, wird die Bibliothek scikit-learn verwendet um den Prototypen zu entwickeln.

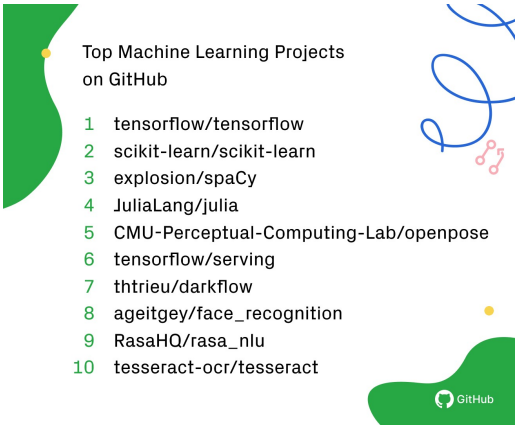
Organisatorisch wird in dem Projekt mit einem iterativen Entwicklungsmodel gearbeitet. Die jeweiligen Iterationen sind in den folgenden Kapiteln beschrieben.



(a) Top 10 „Machine Learning“ Programmiersprachen



(b) Top 10 „Machine Learning“ Bibliotheken



(c) Top 10 „Machine Learning“ Projekte

Abbildung 5.2: Top 10 „Machine Learning“ Programmiersprachen, Bibliotheken und Projekte auf der Plattform GitHub [28] im Jahr 2018 [91]

5.3 Iteration 1: Bag of word, tf-idf und k-Means mit Clusteranzahl

Dieses Kapitel beschreibt, wie das Softwareprojekt dieser Diplomarbeit, die erste Iteration des iterativen Entwicklungsmodells durchläuft. In den Unterkapiteln sind die jeweiligen Phasen des Entwicklungsmodells in dieser Iteration und deren Resultate beschrieben.

5.3.1 Requirementanalyse der ersten Iteration

Die erste Iteration der Applikation, welche im Rahmen dieser Diplomarbeit entwickelt wurde, hatte als Anforderung, aus einem Softwareprojekt Cluster abzuleiten und diese Quelltextdateien zuzuordnen. Das Resultat dieser Iteration ist ein „Proof of Concept“, welches in den weiteren Phasen optimiert werden soll. Da in dieser Iteration ein „Proof of Concept“ erzeugt werden soll, ist die Anzahl der zu unterstützenden Programmiersprachen der zu segmentierenden Softwareprojekte, auf eine Programmiersprache beschränkt. Diese Anzahl der zu unterstützenden Programmiersprachen soll in weiteren Iterationen, wie in Kapitel 1.3 beschrieben, auf die drei Programmiersprachen JavaScript, Python und Java erweitert werden. Die Evaluierung und die Tests dieser Iteration werden sich dementsprechend nur auf die Programmiersprache JavaScript beschränken.

Dieses „Proof of Concept“ soll als Kommandozeilenprogramm entwickelt werden und als Parameter die Anzahl der Cluster und alle Quelltextdateien eines Softwareprojektes erhalten, welche zu Clustern zugeordnet werden sollen. Das Resultat dieses Kommandozeilenprogrammes soll eine Liste von Quelltextdateien mit den zugeordneten Clustern ergeben.

Dafür werden zwei Algorithmen benötigt. Ein Algorithmus der das „Preprocessing und Feature engineering“ der Quelltextdateien durchführt und ein Algorithmus, der die verarbeiteten Daten einer bestimmten Anzahl an Clustern zuordnet.

Angewandt auf verschiedene JavaScript Softwareprojekte, soll das Kommandozeilenprogramm eine Liste von Quelltextdateien und einer Zuordnung zu Clustern ausgeben. Diese Liste mit Zuordnungen dient zur Evaluierung dieser Iteration.

5.3.2 Entwurf der ersten Iteration

Um die Anforderungen, welche in Kapitel 5.3.1 beschrieben wurden, erfüllen zu können, wird in diesem Kapitel ein Lösungsvorschlag beschrieben, der in Kapitel 5.3.3 implementiert und in Kapitel 5.3.4 getestet und evaluiert wird.

Der Entwurf der ersten Iteration ist in Abbildung 5.4 dargestellt. Das Kommandozeilenprogramm bekommt die JavaScript Quelltextdateien und die Anzahl der Cluster als Eingangsparameter. Die Applikation führt auf Basis der Quelltextdateien ein „Preprocessing und Feature engineering“ durch. Danach werden die verarbeiteten Daten in einen Cluster Algorithmus eingesetzt, aus dem eine Zuordnung von Quelltextdateien zu Clustern resultiert.

Bei dieser Iteration werden Eigenschaften des Quelltextes nur von dessen Text abgeleitet, da bereits Lösungen auf diesem Wissensgebiet existieren. Der Cluster-Algorithmus, der in dieser Iteration angewandt wird, ist k-Means. Dieser Cluster-Algorithmus ist detailliert in Kapitel 2.5.1 erklärt. Er erwartet als Parameter die Anzahl der Cluster und eine Matrix und erfüllt damit alle Anforderungen, die diese Iteration mit sich bringt.

In der „Preprocessing und Feature Engineering“ Phase werden die Verfahren „Bag of words“ und Tf-idf [69][3] eingesetzt. Unter „Bag of words“ versteht man ein Verfahren, welches Text in eine für die Maschine simplifizierte Repräsentation bringt. Dabei werden alle m Wörter,die in allen n Quelltextdateien $Q_1, Q_2, ..., Q_n$ vorkommen, als eindeutige Tokens $T_1, T_2, ..., T_m$ definiert. Jede Quelltextdatei wird als m-dimensionaler Vektor repräsentiert, in dem m die Anzahl der Tokens ist. Aus diesen n Quelltextdateien ergibt sich eine $n \times m$ Matrix. In dieser Matrix ist der Wert a_{ij} die Anzahl der Vorkommnisse des Wortes T_i in der Quelltextdatei Q_j. Bei der Umwandlung von der Quelltextdatei in einen Vektor, werden Sonderzeichen ignoriert. Abbildung 5.3 zeigt ein Beispiel, in dem das „Bag of words“-Verfahren auf 3 Quelltextdateien angewandt wurde.

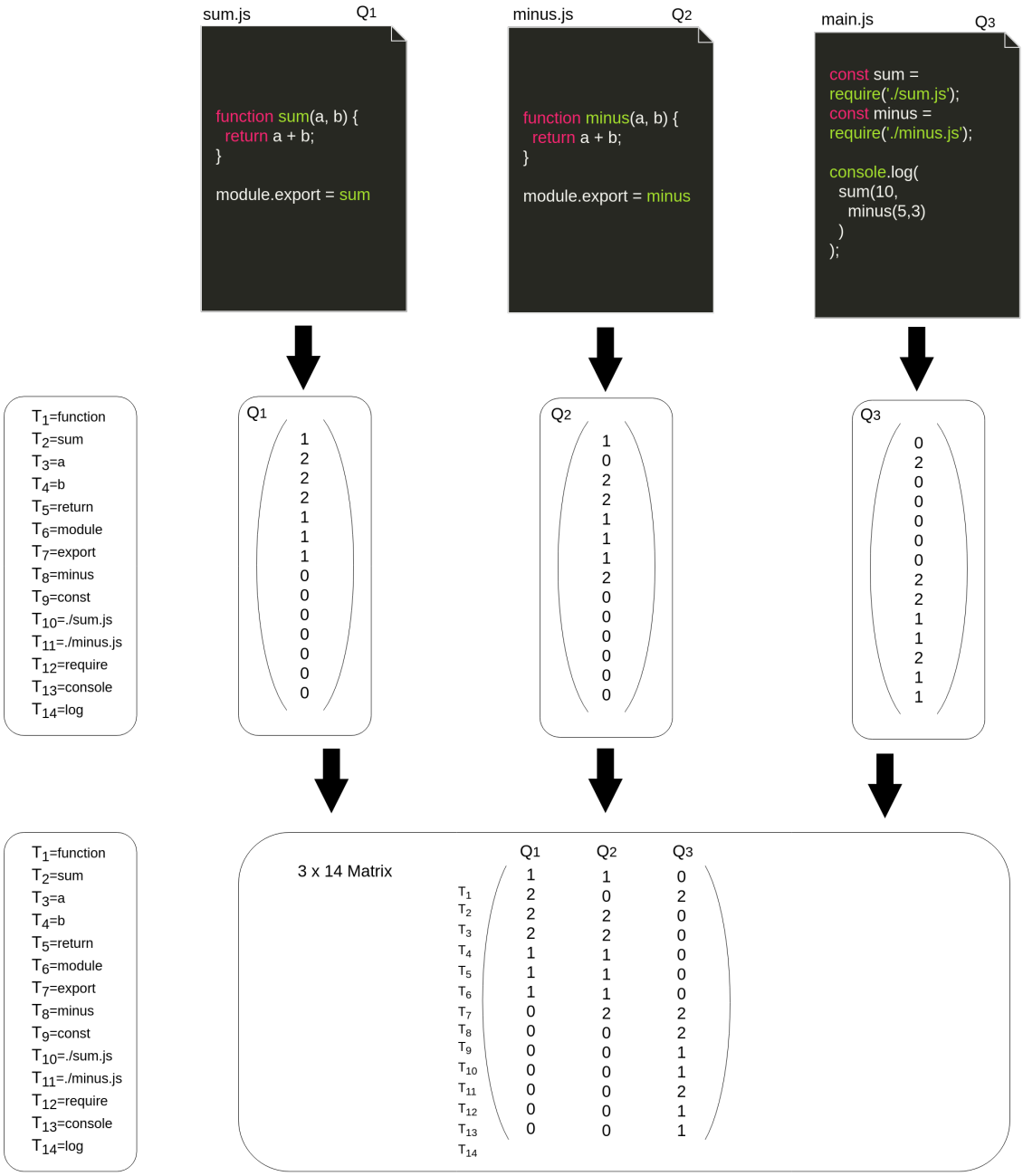


Abbildung 5.3: Beispiel des „Bag of words“-Verfahrens, angewandt auf drei JavaScript Quelltextdateien

Um mit dieser $n \times m$ Matrix aussagekräftigere Ergebnisse mit dem Clusteringverfahren zu erhalten und die Werte der Matrix zu normalisieren, wird in dieser Matrix noch das Tf-idf Verfahren eingesetzt. Dieses Verfahren gewichtet und normalisiert jeden Wert der Matrix. Es wird für jeden Wert a_{ij} der Matrix ein neuer Wert w_{ij} gebildet. Dabei wird der Wert w_{ij} nach der Formel 5.1 berechnet.

$$w_{ij} = tf(T_i, Q_j) \cdot idf(T_i, D) = \frac{a_{ij}}{\sum_{k=1}^{m} a_{kj}} \cdot \log \frac{n}{\sum_{Q:T_i \in D} 1} \tag{5.1}$$

Der Wert w_{ij} ist das Produkt von $tf(T_i, Q_j)$ und $idf(T_i, D)$. Der erste Faktor dieser Multiplikation ist der „Term frequency“ Kennwert. Dieser Wert gibt an, wie häufig ein Token T_i in einer Quelltextdatei Q_j vorkommt. Dabei wird der Wert a_{ij} durch die Summe aller Tokens $T_1, T_2, ..., T_m$ in Q_j dividiert. Mit dieser Vorkommenshäufigkeit wird eine Verzerrung des Ergebnisses in langen Quelltextdateien verhindert und der Wert normalisiert. In der Formel 5.2 wird der Kennwert definiert.

$$tf(T_i, Q_j) = \frac{a_{ij}}{\sum_{k=1}^{m} a_{kj}} \tag{5.2}$$

Der zweite Faktor der Multiplikation, um den Wert w_{ij} zu kalkulieren, ist die „inverse document frequency“ idf. Dieser Wert gibt die Spezifität eines Tokens für die Gesamtmenge aller Quelltextdateien an und wird nach der Formel 5.3 berechnet.

$$idf(T_i) = \log \frac{n}{\sum_{Q:T_i \in D} 1} \tag{5.3}$$

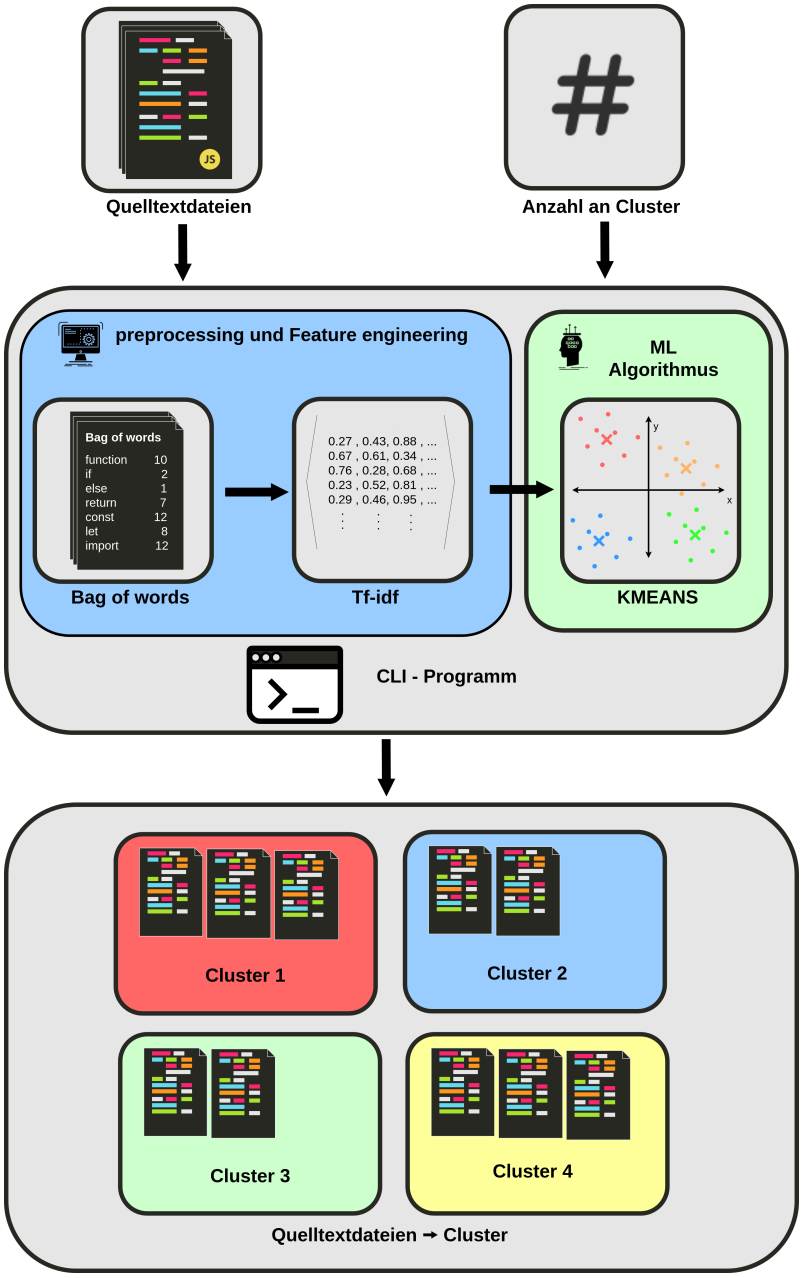


Abbildung 5.4: Entwurf der ersten Iteration um Quelltextdateien Clustern zuzuordnen.

5.3.3 Implementierung der ersten Iteration

In dem Quelltext 5.1 ist der Quelltext definiert, der den Entwurf von 5.3.2 implementiert. Wie schon in Kapitel 5.2 beschrieben, verwendet das Programm die Python Bibliothek scikit-learn. Aus dieser Bibliothek werden Methoden verwendet, um eine Matrix mit tf-idf gewichteten Werten aus einer Liste von Texten zu erhalten und diese dann mittels k-Means zu segmentieren. Dabei ist die Applikation in 4 Methoden aufgeteilt:

- *getDocuments*

  Diese Methode verlangt einen Parameter *path*, der den Dateipfad zu dem Verzeichnis des JavaScript Softwareprojektes im Betriebssystem als *String* definiert. Zusätzlich hat diese Methode zwei optionale Parameter, welche für den rekursiven Aufruf der Methode benö-

tigt werden, um alle relativen Dateipfade und Dateinamen der Quelltextdateien in diesem Projekt zu speichern. Der Rückgabewert dieser Funktion ist ein Tupel mit zwei Werten. Der erste Wert ist eine Liste von Quelltexten und der zweite Wert ist eine Liste von Dateinamen mit den relativen Pfaden innerhalb des Softwareprojektes. Dabei filtert diese Methode alle Dateien, sodass nur JavaScript Dateien in dem Tupel gelistet sind. Diese Methode wird verwendet, um aus einem Dateipfad zu einem Verzeichnis im Betriebssystem eine Liste von Quelltexten mit deren Inhalten abzuleiten. Dabei werden alle Dateien in allen Unterverzeichnissen berücksichtigt, die eine $.js$ Dateiendung im Dateinamen enthalten.

- $kmeans\_bow\_tfidf\_clustering$

  Ziel dieser Methode ist ein Tupel von zwei Listen mit derselben Anzahl an Elementen zurückzugeben. In der erste Liste sind alle Dateinamen mit deren relativen Dateipfaden innerhalb des Softwareprojektes, welches unter dem Dateipfad dir_path zu finden ist. Die zweite Liste sind Zahlen zwischen 0 und $k\_cluster - 1$, welche die zugewiesenen Cluster der Quelltextdateien beschreiben. Dabei stellt der Index der Listen jeweils die Zuordnung der Quelltextdatei und des zugewiesenen Clusters dar. Als Parameter bekommt diese Methode den dir_path, der den Dateipfad des Softwareprojektverzeichnisses repräsentiert und die Anzahl der Cluster $k\_cluster$. Diese Methode verwendet die $getDocuments$ Methode, um aus dem dir_path die Quelltextdateien des Projektes zu erhalten und ruft dann Methoden der Bibliothek scikit-learn auf, um das k-Means Clustering durchzuführen.

- $main$

  In dieser Methode werden die 2 Argumente gelesen, die das Kommandozeilenprogramm für die korrekte Ausführung benötigt. Diese Methode ruft zuerst $kmeans\_bow\_tfidf\_clustering$ auf, um die Quelltextdateien des gewünschten Softwareprojektes Clustern zuzuordnen und danach die Methode $printResult$, um die Liste der Zuweisungen auszugeben.

- $printResult$

  Diese Methode gibt alle Quelltextdateien und deren zugewiesenen Cluster aus. Dabei bekommt sie die beiden Werte als selbes Format wie der Rückgabewert der Methode $kmeans\_bow\_tfidf\_clus$

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
import argparse
import os

def getDocuments( path,rel_path='./', file_names = [] ):
    document_list = []
    for filename in os.listdir(path + '/' + rel_path ):
        if os.path.isdir(path + '/' + rel_path + '/' + filename):
            document_list_dir, file_names_dir = getDocuments( path,rel_path = rel_path+'/'+
            document_list = document_list + document_list_dir
            file_names =  file_names_dir
        else:
            _, ext = os.path.splitext(filename)
            if ext in ['.js']:
                file_names.append((rel_path+'/'+filename)[2:])
                with open(path+'/'+rel_path+'/'+filename, 'r') as file:
                    pathStr = path.split('/')
                    dataString=file.read().replace('\n', '')
                    for s in pathStr:
                        dataString = dataString + s
                    document_list.append(dataString)
    return document_list, file_names

def kmeans_bow_tfidf_clustering(dir_path, k_cluster):
    documents, filenames = getDocuments(dir_path)
    vectorizer = TfidfVectorizer(analyzer='word')
    X = vectorizer.fit_transform(documents)
    model = KMeans( n_clusters=k_cluster )
    model.fit(X)
    labels = model.labels_
    return filenames, labels

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("dir")
    parser.add_argument("clustercount")
    args = parser.parse_args()
    (dir_path, cluster_k)= (args.dir, int(args.clustercount))
    filenames, labels = kmeans_bow_tfidf_clustering(dir_path,cluster_k)
    printResult(filenames, labels)

def printResult(filenames, labels):
    for i,name in enumerate(filenames):
        print(str(labels[i]), name )

if __name__== "__main__":
    main()
```

Listing 5.1: Implementierung der ersten Iteration: Bag of words mit Tf-idf und k-Means

Der gesamte Quelltext dieser Iteration ist in einer Quelltextdatei *main.py* gespeichert.

5.3.4 Evaluierung und Test der ersten Iteration

Angewandt auf für den Autor bekannte JavaScript Projekte hat das Kommandozeilenprogramm, welches in Kapitel 5.3.3 beschrieben wurde, eine korrekte Liste von Quelltextdateien und eine Zuweisung der Quelltextdateien zu Clustern ausgegeben. Dabei wurde kontrolliert, ob die Quelltextdateien die aufgelistet wurden, ident mit den Quelltextdateien in dem Softwareprojekt waren.

Dabei wurde das Clustering an sich sich selbst nicht bewertet, sondern nur die Funktionalität des Programmes. Durch Experimente mit unterschiedlichen JavaScript-Projekten und unterschiedlichen Cluster Anzahlen wurden folgende Optimierungsmöglichkeiten für die nächsten Iterationen bemerkt, um dem gewünschten Endergebnis dieser Diplomarbeit näher zu kommen:

- **Bessere Darstellung der Zuweisung von Clustern zu Quelltextdateien in einer Baumansicht**

  Durch die Zuweisung in Form einer Liste von Quelltextdateien und deren Clustern ist es möglich, die Zuweisung nachzuvollziehen. Da dies aber mit hohem Aufwand verbunden ist und die Zielsetzung, welche in Kapitel 1.3 definiert worden ist, unter anderem auch eine Baumansicht der Zuweisung verlangt, würde sich eine Optimierung der Darstellung als Baumansicht anbieten.

- **Bessere Darstellung der Zuweisung von Clustern zu Quelltextdateien in einer Listenansicht**

  Die Zielsetzung für diese Diplomarbeit verlangt zusätzlich zu der Baumansicht auch eine Listenansicht der Cluster. Da diese zwei Ansichten von den Anforderungen und der Logik des Programmes sehr ähnlich sind, würde sich eine Implementierung der beiden Ansichten in derselben Iteration als nützlich erweisen.

- **Veranschaulichung der Cluster auf ein Koordinatensystem**

  Um nähere Einblicke in den Cluster-Algorithmus zu bekommen, wäre eine visuelle Darstellung der Cluster in einem Koordinatensystem hilfreich. Dadurch können bessere Rückschlüsse und Optimierungsmöglichkeiten bei weiteren Experimenten erkannt werden.

In dem Beispieltext 5.2 ist die Ausgabe des Kommandozeilenprogrammes, angewandt auf ein Beispielprojekt [64] mit der Clusteranzahl 4, als Beispiel eines Experimentes veranschaulicht.

```
1 /public/javascript/main.js
1 /public/javascript/modules/modalRemove.js
1 /public/javascript/modules/handleLikes.js
1 /public/javascript/modules/handleFollow.js
1 /public/javascript/modules/handleNotifications.js
1 /public/javascript/modules/handleComments.js
1 /public/javascript/modules/uploadImage.js
1 /public/javascript/modules/modal.js
1 /public/javascript/modules/shortDom.js
2 /routes/api.js
2 /routes/auth.js
2 /routes/routes.js
3 /handlers/errors.js
3 /handlers/passport.js
3 /handlers/mail.js
0 /models/User.js
0 /models/Image.js
0 /models/Comment.js
3 /controllers/notificationControllers.js
3 /controllers/authControllers.js
3 /controllers/userControllers.js
3 /controllers/commentControllers.js
3 /controllers/imageControllers.js
3 /webpack.config.js
1 /helpers.js
3 /server.js
```

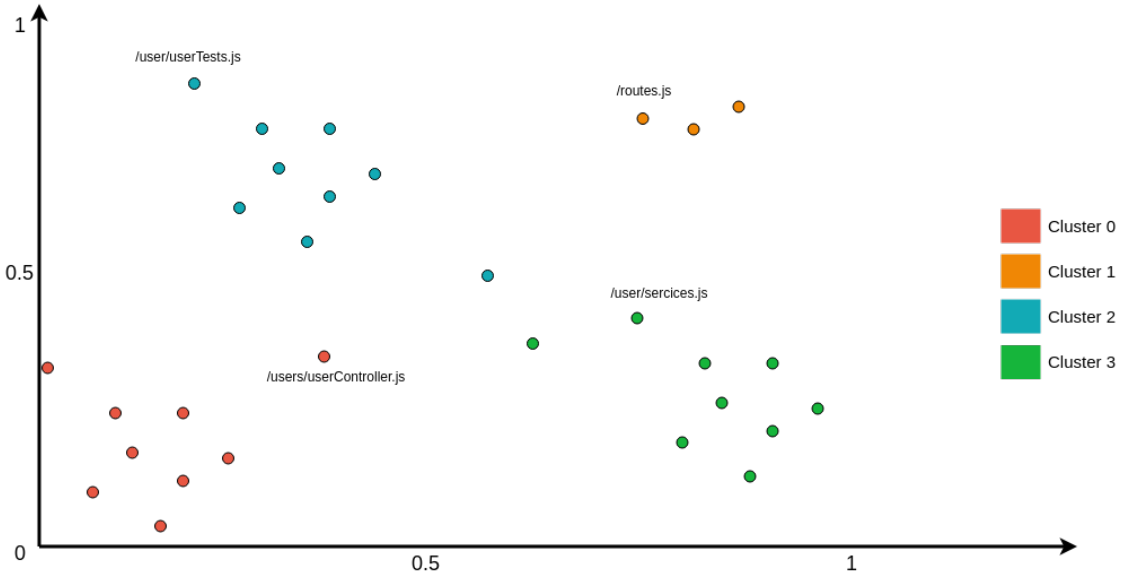
Listing 5.2: Beispielausgabe des Kommandozeilenprogrammes, das aus dieser Iteration resultiert.

5.4 Iteration 2: Visualisierung der Cluster

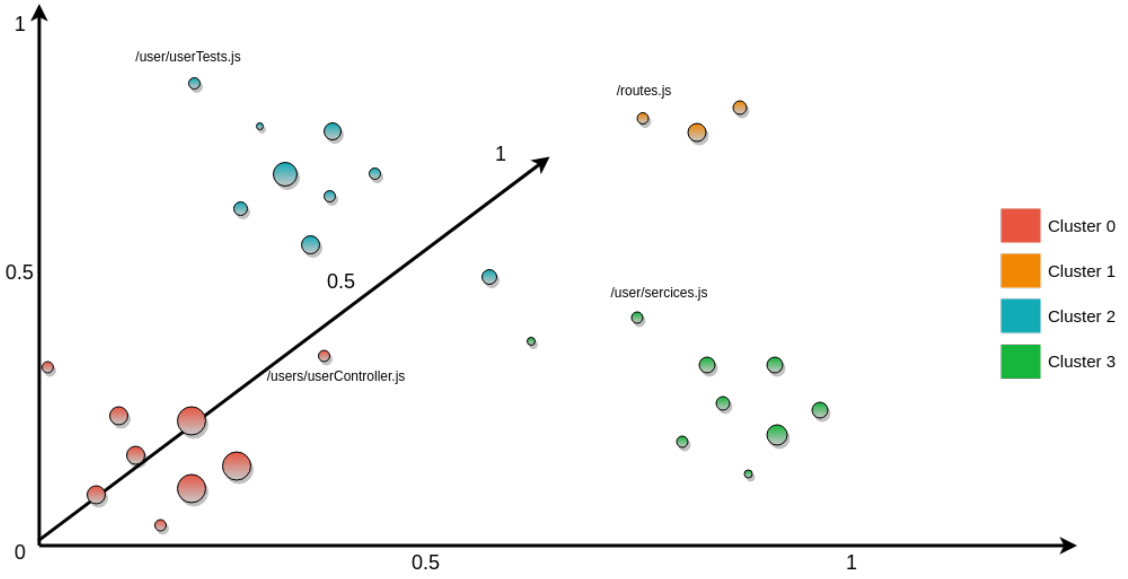
Dieses Kapitel beschreibt, wie das Softwareprojekt dieser Diplomarbeit, die zweite Iteration des iterativen Entwicklungsmodell durchläuft. Diese Iteration baut auf den Resultaten der ersten Iteration auf, welche in Kapitel 5.3 beschrieben wurde. In den Unterkapiteln sind die jeweiligen Phasen des Entwicklungsmodells für diese Iteration und deren Resultate beschrieben.

5.4.1 Requirementanalyse der zweiten Iteration

Aus den Zielsetzungen und den Ergebnissen der ersten Iteration haben sich folgende Anforderungen für diese Iteration ergeben, welche detailliert in diesem Kapitel beschrieben sind. Es sollen drei unterschiedliche Visualisierungen in dieser Iteration realisiert werden. Alle drei Visualisierungen beziehen sich auf das Resultat des Kommandozeilenprogrammes, welches Zuordnungen von Quelltextdateien zu Clustern sind. Dabei wurde in Kapitel 4.1 bereits das User Interface Design von zwei dieser Visualisierungen vorgestellt. Zusätzlich zu diesen beiden Visualisierungen soll auch eine Visualisierung generiert werden, welche diese Zuordnungen in einen geometrischen Raum abbildet. Um diese Zuordnungen besser analysieren zu können, sollen diese Zuordnungen in einem 2-dimensionalen und 3-dimensionalen Raum veranschaulicht werden. Dabei sollen Punkte in diesem geometrischen Raum Quelltextdateien repräsentieren. Diese Punkte sollen eingefärbt werden. Quelltextdateien, die zu demselben Cluster zugewiesen wurden, sollen die gleiche Farbe erhalten. Jeder Cluster soll mit einer anderen Farbe dargestellt werden. In Abbildung 5.5a ist das Design für den 2-dimensionalen Raum abgebildet. Abbildung 5.5b zeigt das Design für den 3-dimensionalen Raum. Dabei bildet die Größe der Punkte im Design den Wert der z-Achse wieder.



(a) Design der Visualisierung im 2-dimensionalen Raum.



(b) Design der Visualisierung im 3-dimensionalen Raum.

Abbildung 5.5: Design der Visualisierung der Zuordnungen von Quelltextdateien zu Clustern im 2-dimensionalen und 3-dimensionalen Raum

5.4.2 Entwurf der zweiten Iteration

Da alle Designs der Visualisierungen in den Anforderungen der Iteration, welche in Kapitel 5.4.1 bereits beschrieben wurden, definiert sind und nur diese Visualisierungen in dieser Phase zu implementieren sind, ist die Entwurf-Phase vergleichsweise zu Entwurf-Phasen anderer Iterationen relativ kurz.

Die Visualisierung der Listen- und der Baumansicht, sollte durch die Ergebnisse der ersten Iteration bereit für die Implementierung sein. Dabei wird eine HTML-Datei gespeichert, welche die Listen und Baumstruktur inkludiert. Der Dateipfad dieser generierten Datei soll per Kommandozeilenargument übergeben werden. Für die Visualisierungen in den geometrischen Räumen muss eine Dimensionsreduktion, wie in Kapitel 2.6 beschrieben, vorgenommen werden.

5.4.3 Implementierung der zweiten Iteration

Aufbauend auf den Quelltext der ersten Iteration wird in diesem Kapitel die Implementierung der zweiten Iteration vorgestellt.

Dabei ist $cluster\_output$, ein Python Modul, entwickelt worden, das zuständig für die Visualisierungen dieser Diplomarbeit ist. Im Quelltext 5.4 ist ein Auszug des Moduls dargestellt. Dieser Auszug betrifft die Methode $plot\_2d\_3d\_graph$, welche für die Darstellung der Zuordnung der Cluster im geometrischen Raum zuständig ist. Für die Visualisierungen im geometrischen Raum wurde das Python-Modul $matplotlib$ verwendet. Die Listen- und Baumansicht ist eine Erweiterung der $printResult$ Methode aus der ersten Iteration. Dabei werden die Zuordnungen nicht auf der Kommandozeile ausgegeben sondern in einer HTML-Datei.

Dabei musste die Quelltextdatei $main.py$ nur minimal geändert werden um das Visualisierung Modul $cluster\_output$ zu integrieren. Die Änderungen sind im Quelltext 5.3 beschrieben.

```
1  from cluster_output import html_cluster_output,LAYOUT_LIST_GROUP,LAYOUT_FILE_TREE,plot_2d_3
2
3  def kmeans_bow_tfidf_clustering(dir_path, k_cluster):
4      documents, filenames = getDocuments(dir_path)
5      vectorizer = TfidfVectorizer(analyzer='word')
6      X = vectorizer.fit_transform(documents)
7      model = KMeans( n_clusters=k_cluster )
8      model.fit(X)
9      labels = model.labels_
10     return filenames, labels, X
11
12 def main():
13     parser = argparse.ArgumentParser()
14     parser.add_argument("dir")
15     parser.add_argument("clustercount")
16     parser.add_argument("html")
17     args = parser.parse_args()
18     (dir_path, cluster_k, output)= (args.dir, int(args.clustercount),args.html)
19     filenames, labels, X= kmeans_bow_tfidf_clustering(dir_path,cluster_k)
20     html_cluster_output(filenames, labels, output, [LAYOUT_LIST_GROUP,LAYOUT_FILE_TREE])
21     plot_2d_3d_graph(filenames, labels, X)
```

Listing 5.3: Änderungen der $main.py$ Datei um die Visualisierungen aus dem $cluster\_output$ Modul anzuzeigen.

Folgende Änderungen mussten in $main.py$ vorgenommen werden, um das Modul $cluster\_output$ zu verwenden:

- Zusätzlicher Rückgabewert X in der Methode $kmeans\_bow\_tfidf\_clustering$

  Der Wert X, welcher eine Matrix der Eigenschaften der Quelltextdateien ist, wurde als Rückgabewert in das Tupel hinzugefügt. Grund dafür war, dass die Visualisierung im geometrischen Raum die komplette Matrix benötigt, um eine Dimensionsreduktion durchzuführen und die Punkte in einem geometrischen Raum abzubilden.

- Zusätzliches Kommandozeilenargument, dass den Speicherort der Listen- und Baumansicht angibt

  Da bei der Listen- und Baumansicht der Zuordnungen von Quelltextdateien zu Clustern eine HTML-Datei erzeugt werden muss, wird ein zusätzliches Kommandozeilenargument benötigt, um den Dateipfad zu der zu generierenden HTML-Datei zu erhalten. Dieser Dateipfad wird in der Variable $output$ gespeichert und dem Modul $cluster\_output$, welches für die Visualisierungen zuständig ist, mit übergeben.

- Aufruf der Methoden zur Visualisierung

  Anstatt der Methode $printResult$, werden in der $main$ Methode in $main.py$ die Methoden aus dem Modul $cluster\_output$ aufgerufen, welche die Visualisierungen darstellen.

```
import matplotlib.pyplot as plt
from mpldatacursor import datacursor
from sklearn.decomposition import PCA

HEX_COLORS = [
'#1ABC9C',
'#3498DB',
'#F1C40F',
'#34495E',
'#EA4C88',
'#E74C3C',
'#8E44AD',
'#F39C12',
'#CA2C68'
]
MARKER = ['.','x','d']

def plot_2d_3d_graph( filenames, y, X ):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    fig2 = plt.figure()
    ax2d = fig2.add_subplot(111)
    pca3d = PCA(n_components=3)
    pca3d.fit(X.todense())
    X3D = pca3d.transform(X.todense())
    for i in range(0, X3D.shape[0]):
            colorIndex = y[i]%len(HEX_COLORS)
            markerIndex=int(y[i]/len(HEX_COLORS))
            ax.scatter(X3D[i,0],X3D[i,1],X3D[i,2],c=HEX_COLORS[colorIndex],
                    marker=MARKER[markerIndex],label=filenames[i])

    if len(X3D) <= 40:
        ax.legend(loc='lower center', shadow=True, fontsize='small',
            ncol=6)

    pca2d = PCA(n_components=2)
    pca2d.fit(X.todense())
    X2D = pca2d.transform(X.todense())
    for i in range(0, X2D.shape[0]):
            colorIndex = y[i]%len(HEX_COLORS)
            markerIndex=int(y[i]/len(HEX_COLORS))
            ax2d.scatter(X2D[i,0],X2D[i,1],c=HEX_COLORS[colorIndex],marker=MARKER[markerInd

    if len(X2D) <= 40:
        ax2d.legend(loc='lower center', shadow=True, fontsize='small',
            ncol=6)

    datacursor(formatter='{label}'.format,
            display='multiple',draggable=True,bbox=dict(fc='white'))
    plt.show()
```

Listing 5.4: Auszug der Methode *plot_2d_3d_graph* aus dem Quelltext des Python Modules *cluster_output*

5.4.4 Evaluierung und Test der zweiten Iteration

Nach dem Ausführen des resultierenden Programmes dieser Iteration werden folgende Ansichten korrekt angezeigt:

- Listenansicht (5.7)

  Die Listenansicht der Zuordnung von Quelltextdateien zu Clustern ist Teil der HTML-Datei, welche durch den Aufruf der Kommandozeilenapplikation generiert wurde.

- Baumansicht (5.8)

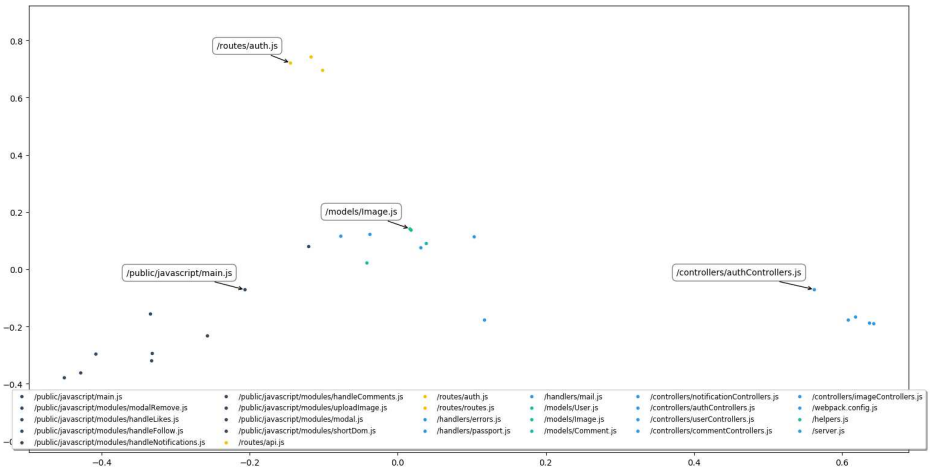
  Ebenfalls Teil dieser HTML-Datei ist auch die Visualisierung eines Baumes, welcher die Zuordnungen der Quelltextdateien zu Clustern in den jeweiligen Dateipfaden darstellt.

- Quelltextdateien und Cluster in einem 2-dimensionalen Raum (5.6a)

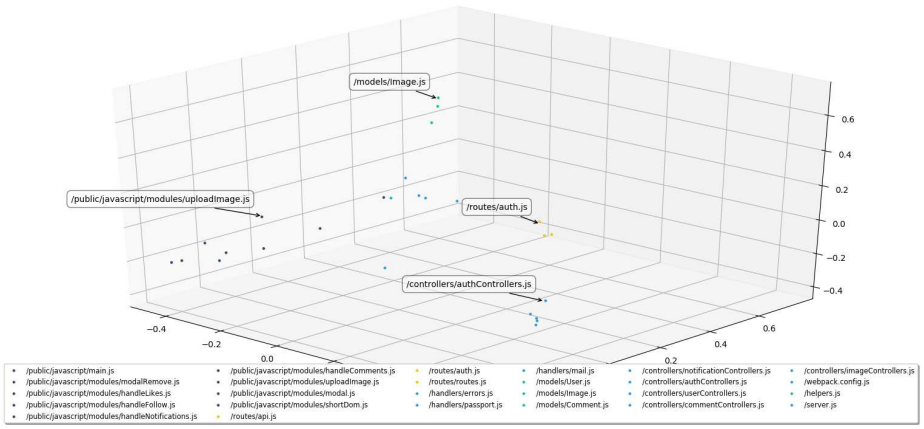
  Diese Visualisierung wird mit dem Kommandozeilenprogramm geöffnet. Es ermöglicht das navigieren und zoomen innerhalb des Koordinatensystems. Außerdem gibt es die Option, den Graphen als Bilddatei abzuspeichern. Dieses Koordinatensystem ist in dieser Visualisierung in einem 2-dimensionalen Raum.

- Quelltextdateien und Cluster in einem 3-dimensionalen Raum (5.6b)

  Diese Visualisierung wird ebenfalls mit dem Kommandozeilenprogramm geöffnet. Es ermöglicht dieselben Optionen, wie die Visualisierung im 2-dimensionalen Raum. Einziger Unterschied ist, dass diese Visualisierung in einem 3-dimensionalen Raum dargestellt ist.



(a) Design der Visualisierung im 2-dimensionalen Raum.



(b) Design der Visualisierung im 3-dimensionalen Raum.

Abbildung 5.6: Design der Visualisierung der Zuordnungen von Quelltextdateien zu Clustern im 2-dimensionalen und 3-dimensionalen Raum

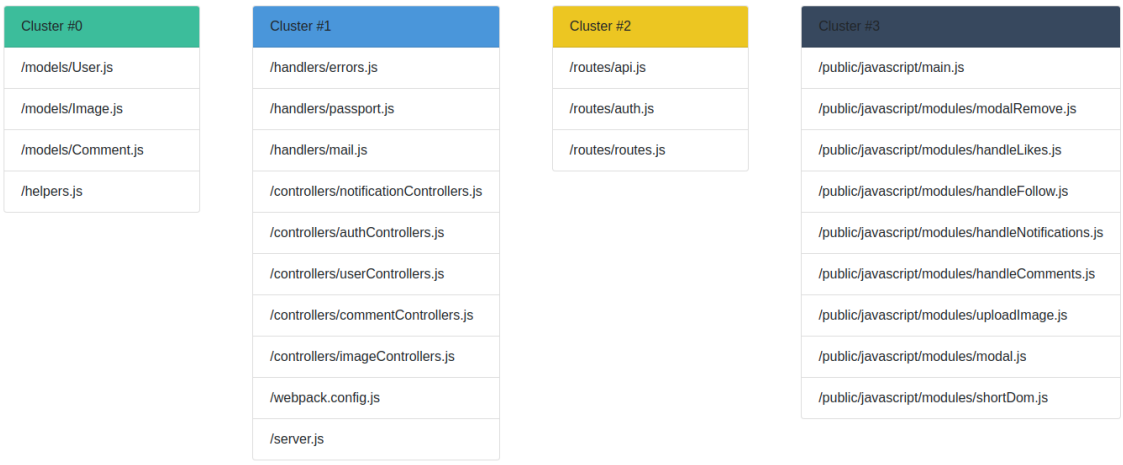


Abbildung 5.7: Visualisierung der Zuordnung von Quelltextdateien zu Clustern in der Listenansicht

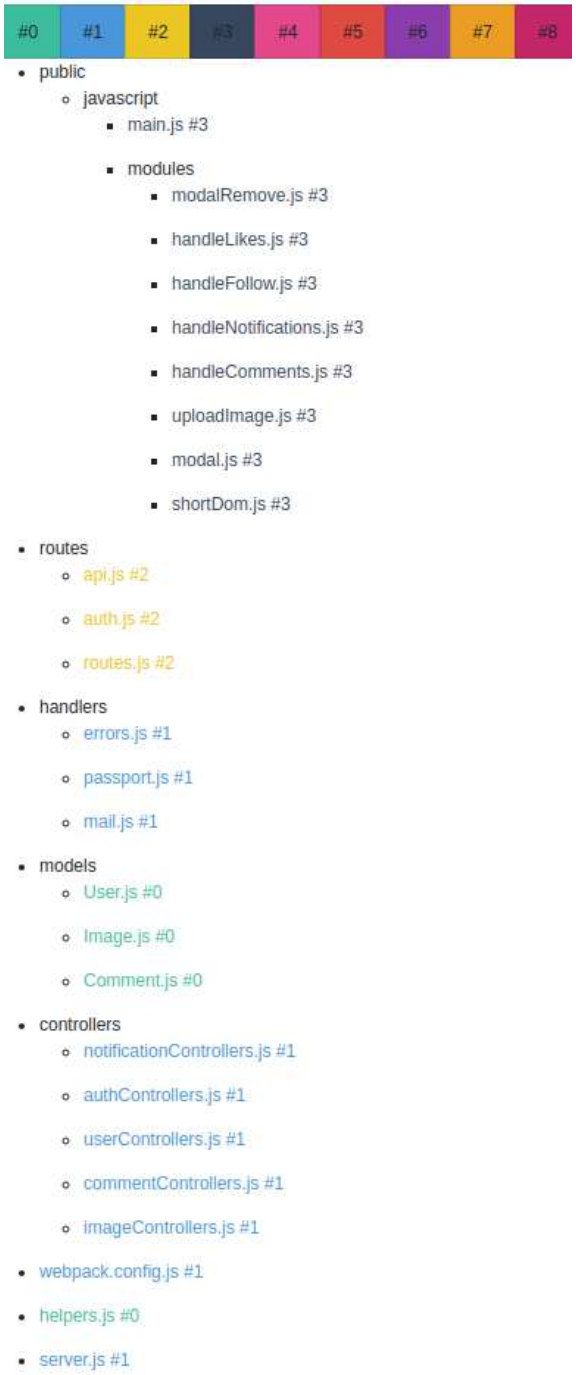


Abbildung 5.8: Visualisierung der Zuordnung von Quelltextdateien zu Clustern in der Baumansicht

Damit sind alle angeforderten Visualisierungen erfolgreich in dieser Iteration implementiert worden. Um die restlichen Anforderungen realisieren zu können wird sich die nächste Iteration mit dem automatischen Erkennen der Clusteranzahl beschäftigen.

5.5 Iteration 3: Automatisches Erkennen der Clusteranzahl

Dieses Kapitel beschreibt, wie das Softwareprojekt dieser Diplomarbeit, die dritte Iteration des iterativen Entwicklungsmodells durchläuft. Diese Iteration baut auf den Resultaten der ersten beiden

Iterationen auf, welche in Kapitel 5.3 und 5.4 beschrieben wurden. In den Unterkapiteln sind die jeweiligen Phasen des Entwicklungsmodells für diese Iteration und deren Resultate beschrieben.

5.5.1 Requirementanalyse der dritten Iteration

Die resultierenden Kommandozeilenprogramme aus den ersten beiden Iterationen, welche in Kapitel 5.3 und 5.4 beschrieben wurden, haben als verpflichtendes Kommandozeilenargument die Anzahl der Cluster, welche die Applikation in den ausgewählten Softwareprojekt erkennen sollte. Ziel dieser Iteration ist es, automatisch diese Anzahl der Cluster abzuleiten.

Demnach soll das Kommandozeilenprogramm, welches in dieser Iteration optimiert wird, zur Ausführung 2 Argumente bekommen. Den Dateipfad des Verzeichnisses des zu clusternden Softwareprojektes und den Dateipfad zu der zu genierenden HTML-Datei.

Abbildung 5.9 stellt dabei die Anforderungen dieser Iteration graphisch dar. Dabei wird der Algorithmus, der die Anzahl der Cluster automatisch ableiten soll, in der Abbildung 5.9 rot markiert. Diesen Algorithmus gilt es, in der nächsten Phase zu entwerfen und im Anschluss zu implementieren und zu testen.

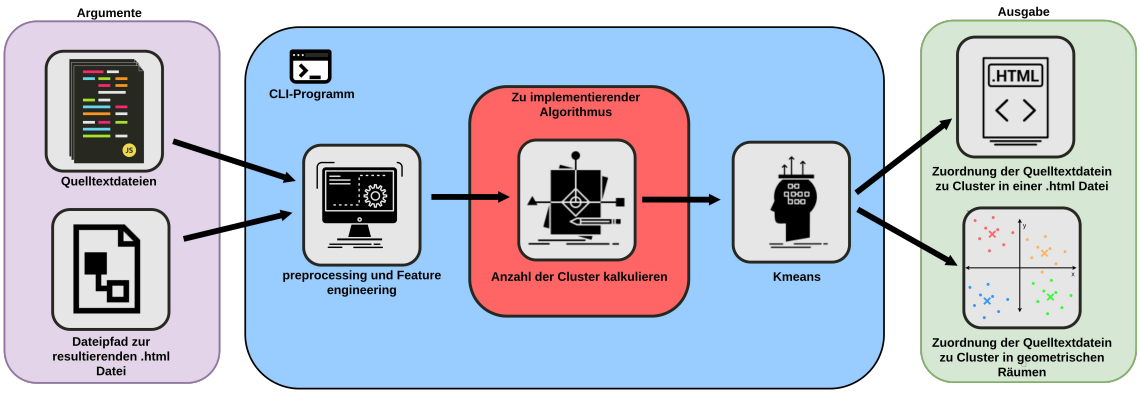


Abbildung 5.9: Anforderungen an die dritte Iteration im iterativen Entwicklungsmodells.

5.5.2 Entwurf der dritten Iteration

In diesem Kapitel wird die Lösung beschrieben, um die Anforderungen, welche im Kapitel 5.5.1 erläutert wurden, zu erfüllen. Zuerst wurde nach einem Cluster-Algorithmus gesucht, welcher anders als der in den ersten beiden Iterationen verwendeten k-Means-Algorithmus, keine Anzahl der Cluster als Parameter erwartet. Gefunden wurde der Cluster-Algorithmus DBSCAN, der in Kapitel 2.5.2 erklärt wird. Allerdings benötigt dieser Algorithmus andere Parameter, die ebenfalls automatisch abgeleitet werden müssen.

Danach wurden Kennzahlen ermittelt, die nach dem Clustering Informationen über die Cluster preisgeben. Diese ermöglichen es, Cluster mit verschiedenen Clusteranzahlen zu bewerten und miteinander zu vergleichen. Um die optimierte Clusteranzahl zu erhalten, werden verschiedene Anzahlen an Cluster zu denselben Quelltextdateien kalkuliert und bewertet. Die Clusteranzahl mit der besten Bewertung wird dann als Anzahl der Cluster in dem Kommandozeilenprogramm verwendet, ohne dass der Benutzer eine Anzahl an Cluster per Argument vorgibt. Als Cluster-Algorithmus wird hier, wie auch in den ersten beiden Iterationen, k-Means verwendet.

Folgende Kennzahlen werden für die Bewertung der Cluster herangezogen:

- Silhouettenkoeffizient (2.5.3)

- Elbow-Method (2.5.3)
- Calinski-Harabasz Index (2.5.3)
- Davies-Bouldin Index (2.5.3)

Alle Kennzahlen wurden in dem Kapitel 2.5.3 genauer erklärt. Für die Bewertung werden alle Cluster mit Anzahlen an Cluster von 2 bis k berechnet. Für jedes Ergebnis C_i werden die oben angeführten Kennzahlen ermittelt. Danach wird für jede Kennzahl die Anzahl an Cluster ausgewählt, welche die beste Bewertung hat. Von diesen Werten wird die höchste Anzahl der Cluster i verwendet um die Visualisierungen darzustellen. Dementsprechend ist C_i die Zuordnung von Quelltextdateien zu i Clustern.

5.5.3 Implementierung der dritten Iteration

Um die Anzahl der Cluster automatisch aus den Quelltextdateien abzuleiten, wurde der Algorithmus, welcher in Kapitel 5.5.2 beschrieben wurde, als eigenes Python-Modul $kmeans\_auto\_k$ programmiert. Ein Auszug des Quelltextes dieses Modules ist in 5.5 ersichtlich. Dabei wurden alle Kennzahlen, bis auf die Elbow-Methode, von dem Python-Modul scikit-learn verwendet. Die Elbow-Methode wurde als eigene Methode innerhalb des Modules programmiert und ist im Quelltext 5.6 dargestellt.

```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score,calinski_harabasz_score,davies_bouldin_score
import numpy as np
import numpy.matlib as matlib

def kmeans_auto_k( X ):
    sum_of_squared_distances = []
    print('Finding optimal clustersize ....')
    true_k = 2
    K = range(2,20)
    max_k_cal_score = 1
    max_k_silhouette = 1
    max_k_davies = 1
    max_silhoutte = 0
    max_cal_score = 0
    max_davies_score = 0
    labels_i =dict()
    for true_k in K:
        try:
            model = KMeans(n_clusters=true_k)
            model.fit(X)
            label = model.labels_
            labels_i[true_k]=label
            sil_coeff = silhouette_score(X, label, metric='euclidean')
            cal_score = calinski_harabasz_score(X.toarray(), label)
            davies_score = davies_bouldin_score( X.toarray(), label )
            sum_of_squared_distances.append(model.inertia_)
            print('''For n_clusters={}
                            The Silhouette Coefficient is {}
                            cal-score is {}
                            davies-bouldin-score is {}'''.format(true_k,
                        sil_coeff, cal_score, davies_score))
            if cal_score > max_cal_score:
                max_cal_score = cal_score
                max_k_cal_score = true_k
            if sil_coeff > max_silhoutte:
                max_silhoutte = sil_coeff
                max_k_silhouette = true_k
            if max_davies_score is None or davies_score < max_davies_score:
                max_davies_score = davies_score
                max_k_davies = true_k
        except:
            print('Clustering not possible with {} clusers'.format(true_k))

    elbow_k = elbow( sum_of_squared_distances )
    print( 'Best clustersize with elbow mothod ', str(elbow_k) )

    true_k = max(max_k_silhouette, max_k_cal_score, max_k_davies, elbow_k)

    print("Bester Clustersize ={}".format(true_k))
    return true_k,labels_i[true_k]
```

Listing 5.5: Auszug des Python Modules $kmeans\_auto\_k$, welches automatisch die Anzahl der Cluster ableitet.

```python
def elbow(curve):
    nPoints = len(curve)
    allCoord = np.vstack((range(nPoints), curve)).T
    np.array([range(nPoints), curve])
    firstPoint = allCoord[0]
    lineVec = allCoord[-1] - allCoord[0]
    lineVecNorm = lineVec / np.sqrt(np.sum(lineVec**2))
    vecFromFirst = allCoord - firstPoint
    scalarProduct = np.sum(vecFromFirst * matlib.repmat(lineVecNorm, nPoints, 1), axis=1)
    vecFromFirstParallel = np.outer(scalarProduct, lineVecNorm)
    vecToLine = vecFromFirst - vecFromFirstParallel
    distToLine = np.sqrt(np.sum(vecToLine ** 2, axis=1))
    idxOfBestPoint = np.argmax(distToLine)
    return idxOfBestPoint
```

Listing 5.6: Quelltext der Elbow-Methode aus dem Modul *kmeans_auto_k*.

Um das Modul *kmeans_auto_k* in der Kommandozeilenapplikation verwenden zu können, wurden minimale Änderungen an dem *main.py* Quelltext vorgenommen. Diese Änderungen sind im Quelltext 5.7 veranschaulicht.

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import argparse
import os
from cluster_output import html_cluster_output,LAYOUT_LIST_GROUP,LAYOUT_FILE_TREE,plot_2d_3d
from kmeans_auto_k import kmeans_auto_k


from sklearn.cluster import OPTICS, cluster_optics_dbscan

def kmeans_bow_tfidf_clustering(dir_path):
    documents, filenames = getDocuments(dir_path)
    vectorizer = TfidfVectorizer(analyzer='word')
    X = vectorizer.fit_transform(documents)
    k_cluster, labels = kmeans_auto_k( X ) # kmeans_auto_k instead of KMEANS
    return filenames, labels, X

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("dir")
    parser.add_argument("html")
    args = parser.parse_args()
    (dir_path, output)= (args.dir, args.html) # remove of k_cluster
    filenames, labels, X= kmeans_bow_tfidf_clustering(dir_path)
    html_cluster_output(filenames, labels, output, [LAYOUT_LIST_GROUP,LAYOUT_FILE_TREE])
    plot_2d_3d_graph(filenames, labels, X)
```

Listing 5.7: Ausschnitt aus der *main.py* Quelltextdatei um die Änderungen zu den vorherigen Iterationen darzustellen.

Dabei wurde das Kommandozeilenargument, welches die Anzahl der Cluster bestimmt, entfernt und der Aufruf der Clustering-Methode geändert, sodass nun die Clustering-Methode aus dem implementierten Python-Modul *kmeans_auto_k* verwendet wird.

5.5.4 Evaluierung und Test der dritten Iteration

Nach dem Ausführen des resultierenden Programmes dieser Iteration werden Zuordnungen von Quelltextdateien zu Clustern visuell dargestellt. Dabei ist es nicht mehr notwendig, die Anzahl der Cluster als Kommandozeilenargument an die Applikation zu übergeben. Experimente auf verschiedene JavaScript-Projekte haben folgende Punkte evaluiert, die im Rahmen der nächsten Iteration realisiert werden müssen, um den Zielsetzungen aus Kapitel 1.3 näher zu kommen:

- **Verbesserte Algorithmen für „Preprocessing und Feature engineering“**

  Aus Experimenten mit verschiedenen JavaScript-Softwareprojekten waren folgende Punkte auffällig. Für die Hälfte der Projekte waren die Ergebnisse bereits sehr zufriedenstellend. Bei der anderen Hälfte der Projekte war das „Preprocessing und Feature engineering“ irreführend. Zum Beispiel wurden Kommentare in Quelltextdateien nicht berücksichtigt und werden genauso bewertet und gewichtet wie Quelltext. Deswegen soll die nächste Iteration Features miteinbeziehen, die in Kapitel 4.2 beschrieben wurden.

- **Unterstützung der 3 definierten Programmiersprachen aus den Zielsetzungen**

  In dieser und den vorherigen Iterationen wurden nur Quelltextdateien der Programmiersprache JavaScript betrachtet. Da sich die nächste Iteration unter anderem auch mit der Verbesserung des „Preprocessing und Feature engineering“ beschäftigt und diese Komponenten relevant sind bei der Integration von mehreren Programmiersprachen, soll es nach der nächsten Iteration auch möglich sein, Cluster zu den Quelltextdateien der Programmiersprachen JavaScript, Python und Java zuzuordnen.

5.6 Iteration 4: Optimiertes Feature engineering und preprocessing

Dieses Kapitel beschreibt, wie das Softwareprojekt dieser Diplomarbeit, die vierte Iteration des iterativen Entwicklungsmodells durchläuft. Diese Iteration baut auf den Resultaten der ersten drei Iterationen auf, welche in Kapitel 5.3 bis 5.5 beschrieben wurden. In den Unterkapiteln sind die jeweiligen Phasen des Entwicklungsmodells dieser Iteration und deren Resultate beschrieben.

5.6.1 Requirementanalyse der vierten Iteration

Aus den Zielsetzungen und den vorherigen Iterationen haben sich folgende Anforderungen für diese Iteration ergeben, welche detailliert in diesem Kapitel beschrieben sind. In dieser Iteration soll das Kommandozeilenprogramm erweitert werden, sodass die Applikation Softwareprojekte der drei Programmiersprachen JavaScript, Python und Java akzeptiert. Außerdem sollen das „Preprocessing und Feature engineering“ optimiert werden, sodass Eigenschaften miteinbezogen werden, die in Kapitel 4.2 beschrieben wurden. Dadurch sollen in der Evaluierung bessere Ergebnisse als in den vorherigen Iterationen erzielt werden. So soll der Clustering-Algorithmus den Kontext des Quelltextes verstehen und so Texte aus Kommentaren anders bewerten oder gewichten als reinen Quelltext.

Das Resultat dieser Iteration ist das optimierte Kommandozeilenprogramm, welches folgende Kommandozeilenargumente von dem Benutzer erwartet:

- Dateipfad für die HTML-Datei, welche die Zuordnung von Quelltextdateien zu Clustern visualisiert
- Dateiendung der Programmiersprache des Softwareprojektes. ($.js$, $.py$, $.java$)

Das Ergebnis des Aufrufs dieser Kommandozeilenapplikation ist eine HTML-Datei, welche die Zuordnungen von Quelltextdateien der ausgewählten Programmiersprache zu Clustern zuordnet. Dadurch soll die projektbezogene Softwarearchitektur abgeleitet werden.

5.6.2 Entwurf der vierten Iteration

Abbildung 5.10 zeigt, welche Eigenschaften aus dem Quelltext für das Clustering in dieser Iteration in Betracht gezogen werden.

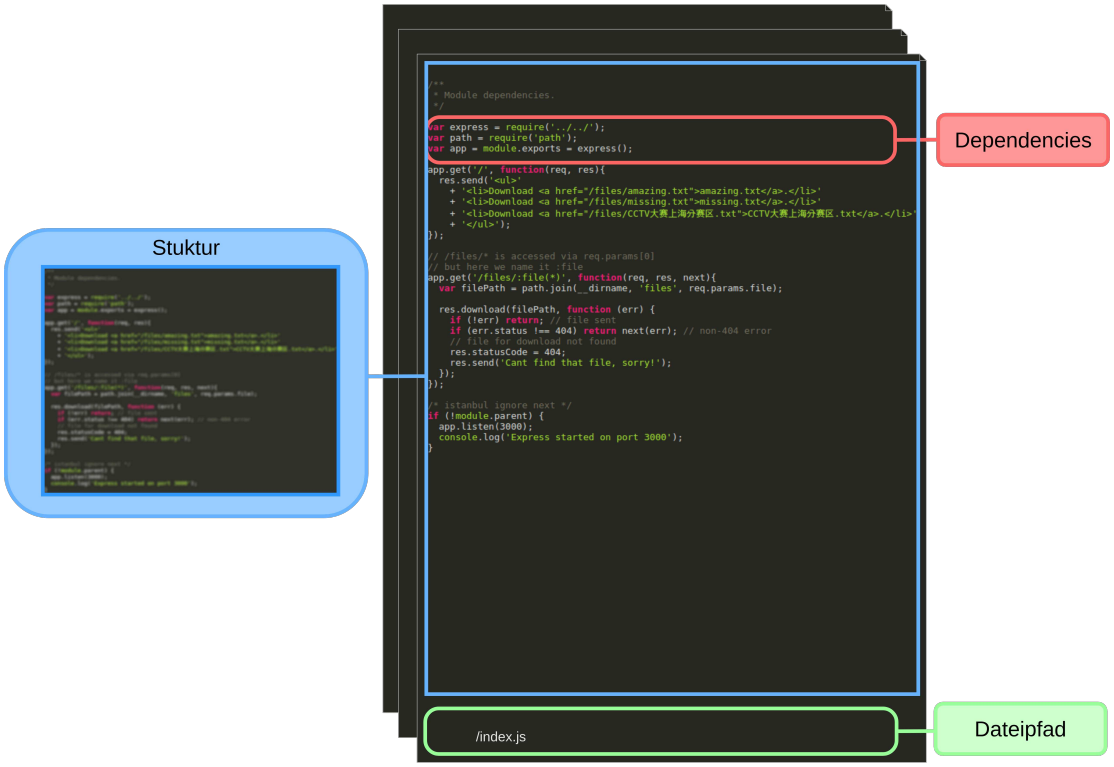


Abbildung 5.10: Darstellung der Eigenschaften, welche in „Preprocessing und Feature engineering“ in der vierten Iteration berücksichtigt werden.

Um aus Quelltextdateien von Softwareprojekten, die in einer von drei verschiedenen Programmiersprachen verfasst wurden, Eigenschaften im Rahmen des „Preprocessing und Feature engineering“ abzuleiten, werden die folgenden drei Aspekte des Quelltextes analysiert:

- **Struktur**

  Da die Struktur der Quelltextdateien Auswirkungen auf die Vergleichbarkeit der Quelltextdateien hat [75], widmen wir einen Aspekt des optimierten „Preprocessing und Feature engineering“ der Struktur des Quelltextes. Aus der lexikalischen Analyse, welche in Kapitel 4.2.3 beschrieben wurde, lassen sich die Tokens des Quelltextes ableiten. Dieses Verfahren wird bei „Syntax Highlighter“ angewandt, um den Quelltext farblich darzustellen und Schlüsselwörter hervorzuheben. Mit der Struktur des Quelltextes sind die Vorkommnisse unterschiedlicher Tokens in Teilen des Quelltextes gemeint. Dabei wird zuerst jede Quelltextdatei Q_i als eine Liste von Tokens repräsentiert. Alle Tokens $T_1, T_2, ..., T_k$, welche in mindestens einer dieser Quelltextdateien vorkommen, bilden einen Vektor. Danach wird jede Liste der Quelltextdateien Q_i in l gleich große Listen $P_1, P_2, ..., P_l$ unterteilt. Für jede solcher Liste P_j werden die Anteile aller Tokens $T_1, T_2, ..., T_k$ in der Liste berechnet. Somit

ergibt sich eine $k \times l$ Matrix für jede Quelltextdatei Q_i. Der Wert a_{tp} gibt dabei den Anteil des Tokens T_t in der Teilliste P_p an. Somit erhält jede Quelltextdatei Q_i eine gleich große $k \times l$ Matrix. Jede einzelne dieser Matrix wird als Liste repräsentiert, wobei die Länge jeder dieser Listen $m = k * l$ ist. Dabei ist m die Anzahl der Eigenschaften, die aus der Struktur des Quelltextes gewonnen wurden. Abbildung 5.11 veranschaulicht dieses Verfahren.

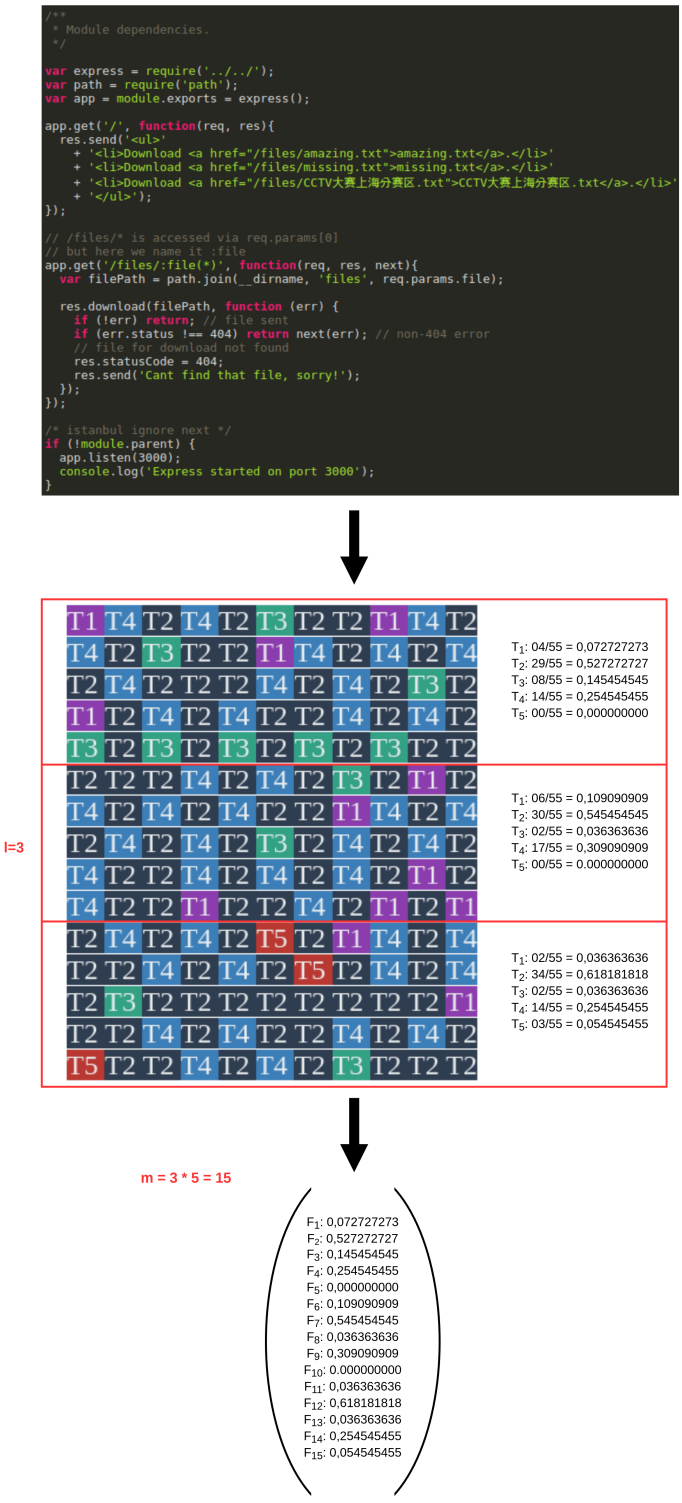


Abbildung 5.11: Darstellung des „Preprocessing und Feature engineering“ der Struktur des Quelltextes anhand eines Beispiels.

- **Dependencies**

  Dependencies von Quelltextdateien spielen eine große Rolle in der Vergleichbarkeit verschiedener Quelltextdateien [24].

Dadurch erzeugen wir eine Liste aller Dependencies $D_1, D_2, .., D_m$ die im gesamten Softwareprojekt vorkommen. Für jede Quelltextdatei Q_i wird die Anzahl der Dependencies gezählt und anschliessend normalisiert.

- **Dateipfad**

  Da der Dateipfad eine wichtige Rolle im Bereich der Softwarearchitektur spielt, wurde dieser ebenfalls in das „Preprocessing und Feature engineering“ Verfahren inkludiert. Dabei wird jedes Verzeichnis $V_1, V_2, ..., V_m$, das im gesamten Softwareprojekt existiert, in eine Liste eingetragen. Für jede Quelltextdatei Q_i wird eine Liste mit m Elementen erstellt. Jedem Elemente Q_{ij} dieser Liste kann der Wert 0 oder 1 zugewiesen werden, wobei der Wert 0 besagt, dass das Verzeichnis V_j nicht Teil des Dateipfades der Quelltextdatei Q_i ist. Dementsprechend würde der Wert 1 aussagen, dass das Verzeichnis V_j teil des Dateipfades der Quelltextdatei Q_i ist.

Das Resultat jedes dieser 3 Verfahren ist eine Liste pro Quelltextdatei. Um diese 3 Listen zu kombinieren, wurden sie konkateniert, um so eine korrekte Matrix für das Clusteringverfahren zu erhalten.

5.6.3 Implementierung der vierten Iteration

In dieser Iteration wurden drei neue Python-Module implementiert. Jedes Modul beschäftigt sich mit einem der drei Aspekte, die in Kapitel 5.6.2 beschrieben wurden. Die Module haben folgende Dateinamen:

- $preprocess\_dependencies$

  In diesem Modul wird mit einer Suche im Quelltext nach Abhängigkeiten zu andern Bibliotheken oder Modulen gesucht. Dabei wird nicht nur das resultierende Modul, sondern auch die Verzeichnisstruktur der Abhängigkeit betrachtet.

- $preprocess\_paths$

  In diesem Modul wird jeder Pfad der Quelltextdateien als Feature abgeleitet. Danach werden mittels „OneHotEncoder “ die Listen in eine geeignete Matrix umgewandelt.

- $preprocess\_structure$

  Dabei wird das Python-Modul $bblfsh$ verwendet, um aus einer Quelltextdatei mittels einer lexikalischen Analyse Tokens zu generieren. Bei diesem Modul werden alle drei Programmiersprachen aus den Zielsetzungen unterstützt. Dabei wurde nach Experimenten der Wert $l = 10$, der in 5.6.2 erklärt wurde, definiert. In Quelltext 5.8 wird die Methode $tokens\_to\_parts\_dict$ dieses Modules beschrieben. In dieser Methode werden alle Tokens der Quelltextdatei in ein Python-Dictionary umgewandelt. Die Methode entspricht also dem letzten Schritt in der Abbildung 5.11, wobei l dem Argument $parts$ entspricht.

Darüber hinaus musste die $main.py$ Datei in dieser Iteration geändert werden, sodass sie das zusätzliche Kommandozeilenargument für die Programmiersprache des Softwareprojektes akzeptiert. Zusätzlich werden die Methoden aus den drei neuen Python-Modulen aufgerufen, um das optimierte „Preprocessing und Feature engineering“ auszuführen. Die Änderungen des Quelltextes der $main.py$ Datei sind in Quelltext 5.9 veranschaulicht.

```python
def tokens_to_parts_dict(tokens, parts):
    total = len(tokens)
    part = int(round(total/parts))
    token_parts = []
    for idx in range(parts):
        if idx == parts-1:
            token_parts.append( tokens[idx*part:] )
        else:
            token_parts.append( tokens[idx:(idx+1)*part] )
    return_dict = {}
    token_types = py_(tokens).map('type').uniq().value()
    for token_type in token_types:
        types_count_total = len(
            py_(tokens)
            .map('type')
            .filter(
                lambda c: c == token_type
            )
            .value())
        return_dict[token_type] = types_count_total / total
        for i,token_part in enumerate(token_parts):
            length = len(token_part) if len(token_part)>0 else 1
            types_count = len(
                py_(token_part)
                .map('type')
                .filter(
                    lambda c: c == token_type
                )
                .value())
            return_dict[token_type+str(i)] = types_count/length
    return return_dict
```

Listing 5.8: Quelltext des Modul *preprocess_structure* um die Struktur des Quelltextes in Eigenschaften abzubilden.

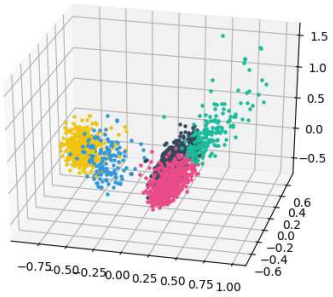
```
def features_dependencies( dir_path, extensions,debug=False ):
        filenames, documents = preprocess_dependencies.convert_dir_to_depedenci
            extensions, debug)
        vectorizer = TfidfVectorizer(
                stop_words=None,
                analyzer='word'
                )
        X = vectorizer.fit_transform(documents)
        return X,filenames

def features_structure( dir_path, extensions,debug=False ):
        filenames, documents = preprocess_structure.convert_dir_to_token_dict(
            extensions, debug)
        vectorizer = DictVectorizer(sparse=True)
        X = vectorizer.fit_transform(documents)
        return X,filenames

def features_filepath( dir_path, extensions,debug=False ):
        d , filenames = preprocess_paths.convert_dir_to_token_and_fill_filename
        enc = OneHotEncoder()
        X_path_unnormalized = enc.fit_transform(d)
        X = normalize(X_path_unnormalized, norm='l2')
        return X,filenames

def kmeans_clustering_combined(dir_path, extensions):
    X_dependencies, filenames = features_dependencies(dir_path, extensions)
    X_structure, filenames = features_structure(dir_path, extensions)
    X_filepath, filenames = features_filepath(dir_path, extensions)
    X = hstack([X_structure, X_dependencies,X_filepath])
    k_cluster, labels = kmeans_auto_k( X )
    return filenames, labels, X

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("dir")
    parser.add_argument("html")
    parser.add_argument("language")
    args = parser.parse_args()
    (dir_path, output, language)= (args.dir, args.html, args.language)
    filenames, labels, X= kmeans_clustering_combined(dir_path,[language])
    html_cluster_output(filenames, labels, output, [LAYOUT_LIST_GROUP,LAYOUT_FI
    plot_2d_3d_graph(filenames, labels, X)

if __name__== "__main__":
    main()
```

Listing 5.9: Ausschnitt des Quelltext der *main.py* Datei um die Änderungen zu den vorherigen Iterationen hervorzuheben.

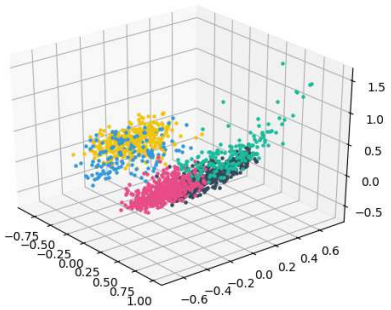
5.6.4 Evaluierung und Test der vierten Iteration

Nach dem Ausführen des resultierenden Programmes dieser Iteration werden Zuordnungen von Quelltextdateien zu Clustern für Softwareprojekte der Programmiersprachen JavaScript, Python und Java angezeigt. Experimente mit unterschiedlichen Projekten dieser drei Programmiersprachen, in welchen der Autor, das notwendige Grundwissen besitzt, deckten sich mit den Vorstellungen der Cluster des Autoren. Diese Experimente zeigen, dass die Optimierung erfolgreich war. Dadurch ist keine weitere Iteration mehr notwendig und die Implementierungsphase abgeschlos-

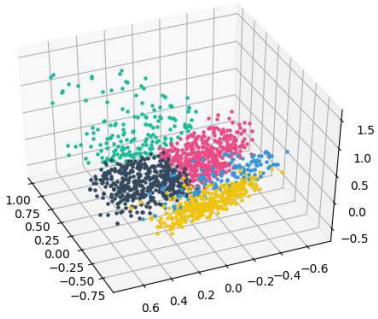
sen. Abbildung 5.12 zeigt die Ergebnisse eines solchen Experimentes in einem 3-dimensionalen Raum an dem Open-Source-Java-Programm Jenkins [42].



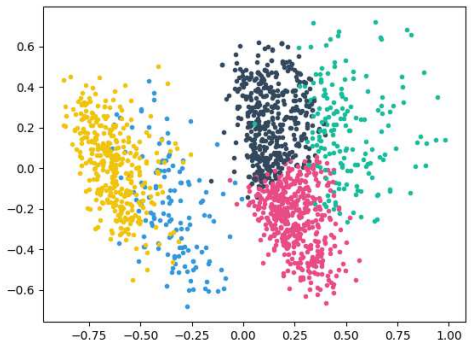
(a) Ansicht des 3-dimensionalen geometrischen Raums von links.



(b) Ansicht des 3-dimensionalen geometrischen Raums von der Mitte.



(c) Ansicht des 3-dimensionalen geometrischen Raums von rechts.



(d) Ansicht des 2-dimensionalen geometrischen Raums.

Abbildung 5.12: Ergebnis der Zuordnung des Java-Softwareprojektes Jenkins [42] in einem 3-dimensionalen und 2-dimensionalen Raum.

6 Evaluierung

In diesem Kapitel wird die Evaluierung der Ergebnisse der Kommandozeilenapplikation beschrieben. Dieses Programm liefert Zuordnungen von Quelltextdateien zu Clustern, wie bereits in Kapitel 5.6.4 erfolgreich getestet wurde. Dieses Kapitel beschäftigt sich mit der Evaluierung dieser Zuordnungen.

Dabei soll mit statistischen Methoden eine Evidenz geliefert werden, dass die Ergebnisse des Programmes relevant für Personen im Bereich der Softwareentwicklung sind und sich die abgeleitete projektspezifische Softwarearchitektur mit den Vorstellungen von erfahrenen Personen im Bereich der Softwareentwicklung deckt. In diesem Kapitel wird beschrieben, wie die Studie chronologisch entworfen, ausgeführt und analysiert wurde.

Dabei widmet sich Kapitel 6.1 dem Entwurf der Studie.

In Kapitel 6.2 wird anschließend erklärt, wie die Ausführung der Studie verlaufen ist. Danach wird in Abschnitt 6.3 die Analyse und Auswertung der Ergebnisse dieser Studie beschrieben. Das Kapitel 6.3.3 beschäftigt sich mit der Validierung der Umfragen.

6.1 Entwurf der Studie zur Evaluierung der Zuordnung

In diesem Kapitel wird der Entwurf der Studie zur Evaluierung der Ergebnisse des Kommandozeilenprogrammes beschrieben. Dabei wird die Studie in 3 Teile aufgeteilt:

- Umfrage über Zweck und Bedarf eines Programmes, welches die projektspezifische Softwarearchitektur eines Softwareprojektes automatisch erlernt und visualisiert.
- Umfrage über die Evaluierung der Zuordnungen von Quelltextdateien zu Clustern.
- Fallbeispiel eines Softwareprojektes

Das Kapitel 6.1.1 beschäftigt sich dabei mit der Erstellung des Fragebogens für eine Umfrage, die den Bedarf eines Programmes analysieren soll, welches automatisch visuelle Darstellungen über die projektspezifische Softwarearchitektur erzeugt. Das anschließende Kapitel 6.1.2 erklärt den Entwurf einer Umfrage zur Evaluierung der Ergebnisse des Kommandozeilenprogrammes. Im Kapitel 6.3.4 wird das Ergebnis eines Softwareprojektes beschrieben und als Fallbeispiel erklärt.

6.1.1 Entwurf der Studie zur Bedarfsanalyse

In diesem Kapitel wird ein Fragebogen entworfen, der den Bedarf nach einer Applikation analysieren soll, die automatisch projektspezifische Softwarearchitektur eines beliebigen Softwareprojektes erkennt und visuell darstellt. Dazu sollen die Ergebnisse der Umfrage folgende Forschungsfrage beantworten:

Forschungsfrage Bedarfsanalyse
Gibt es einen momentanen oder zukünftigen Bedarf nach einem Programm, so wie jenes, das im Rahmen dieser Diplomarbeit entwickelt wurde?

Dazu verwenden wir folgende Hypothese, die mit Hilfe der Antworten aus der Umfrage bestätigt bzw. widerlegt werden soll:

Hypothese Bedarfsanalyse
Eine solche Applikation, welche die projektspezifische Softwarearchitektur eines Softwareprojektes automatisch erlernt und visuell darstellt, erleichtert die Einarbeitungsphase in ein Softwareprojekt.

Um die Forschungsfrage zu beantworten und die Hypothese zu bestätigen oder zu widerlegen, wollen wir folgende Variablen in der Umfrage messen:

1. Potential von KI in Softwareentwicklung.
2. Relevanz von Gruppierungen ähnlicher Quelltextdateien in der Einarbeitungsphase eines Softwareprojektes.
3. Relevanz der projektspezifischen Softwarearchitektur in der Einarbeitsphase eines Softwareprojektes.
4. Relevanz des Wartungsaufwandes von Dokumentation in Softwareprojekten.
5. Bedarf eines Programmes, welches automatisiert Dokumentation über die projektspezifische Softwarearchitektur generiert.

Um diese Variablen im Rahmen der Analyse und Auswertung der Umfrage statistisch berechnen zu können, sind folgende Fragen für diese Umfrage definiert worden:

1. Für wie wichtig schätzen sie das Potential von künstlicher Intelligenz in Softwareentwicklung ein? (1: sehr wichtig, 5: von keiner Bedeutung)
2. Wie sehr hilft eine Gruppierung von ähnlichen Quelltextdateien beim Einarbeiten in ein neues Softwareprojekt? (1: hilft sehr, 5: hilft überhaupt nicht)
3. Wie wichtig ist das Erlernen einer projektspezifischen Softwarearchitektur für das Einarbeiten in ein Softwareprojekt? (1: sehr wichtig, 5: von keiner Bedeutung)
4. Wie wichtig ist das Warten der Dokumentation eines Softwareprojektes? (1: sehr wichtig, 5: von keiner Bedeutung)
5. Angenommen, es existiert ein Tool, welches automatisch ein Dokument erstellt, das die projektspezifische Softwarearchitektur beschreibt. Wie wahrscheinlich ist es, dass Sie ein solches Tool einsetzen würden? (1: wahrscheinlich, 5: unwahrscheinlich)

Dabei ist für jede Frage eine Werteskala von 1 bis 5 als Antwort definiert. Die Reihenfolge der Fragen wird zufällig für jeden Teilnehmer sortiert. Ein Screenshot des Fragebogens ist in Abbildung 6.1 dargestellt.

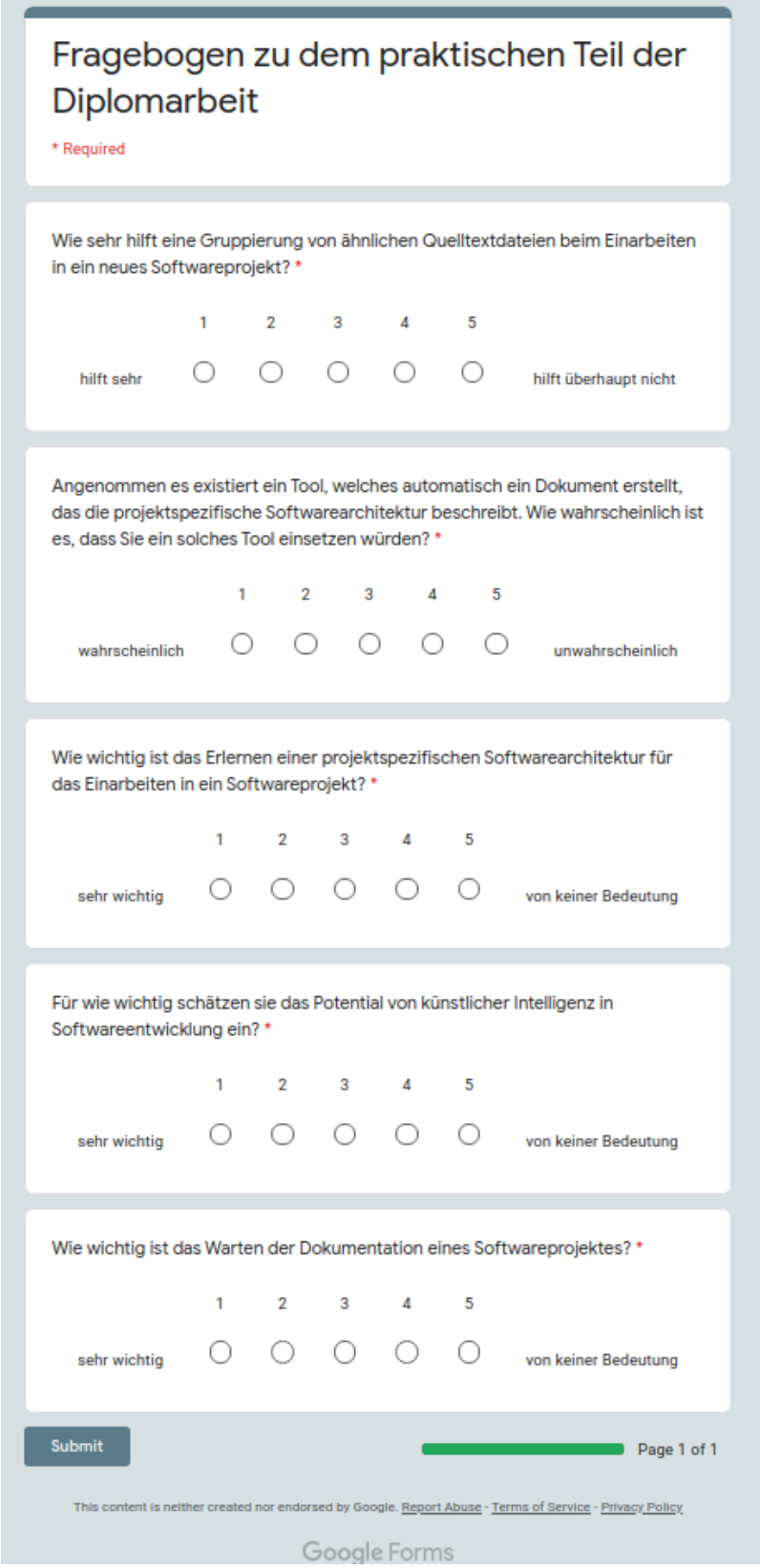


Abbildung 6.1: Screenshot der Umfrage zur Bedarfsanalyse.

6.1.2 Entwurf der Studie zur Evaluierung der Ergebnisse der Kommandozeilenapplikation

Kapitel 5.6.4 hat bereits die Anforderungen an das Kommandozeilenprogramm getestet und evaluiert. In diesem Kapitel wird versucht, mit einer Umfrage die Ergebnisse der Kommandozeilenapplikation zu bewerten und statistisch zu analysieren. Dabei liegt der Fokus dieses Kapitels auf dem Entwurf dieser Umfrage.

Mit dem Ergebnis dieser Umfrage soll folgende Forschungsfrage beantwortet werden:

Forschungsfrage über die Evaluierung der Ergebnisse des Programmes
Decken sich die Ergebnisse der Applikation, welche im Rahmen dieser Diplomarbeit entwickelt wurden, mit dem Clustering-Ergebnis von erfahrenen Personen im Bereich der Softwareentwicklung?

Dazu verwenden wir folgende Hypothese, die mit Hilfe der Antworten aus der Umfrage bestätigt bzw. widerlegt werden soll:

Hypothese über die Evaluierung der Ergebnisse des Programmes
Die projektspezifische Softwarearchitektur kann automatisiert von einem Softwareprojekt abgeleitet werden.

Da die Bewertung der Zuordnung von Quelltextdateien zu Clustern nicht trivial ist, muss eine andere Forschungsmethode als ein klassischer Fragebogen gefunden werden.

Der naheliegendste Ansatz wäre, Personen im Bereich der Softwareentwicklung dieselbe Aufgabe zu geben wie der Applikation und anschließend die Ergebnisse mit den Zuordnungen zu vergleichen. Die Analyse wäre sehr detailliert, allerdings ist das Problem an diesem Ansatz der hohe Aufwand der Befragten. Dadurch würde sich auch die Teilnehmerzahl sehr begrenzen, weswegen nach einer alternativen Forschungsmethode gesucht wurde, um die Ergebnisse des Kommandozeilenprogrammes zu bewerten.

Aus diesem Grund wurde eine Umfrage gewählt, die als Webapplikation auf dem Computer durchgeführt wird. Jeder Teilnehmer bekommt mindestens 20 zufällige Aufgaben aus unterschiedlichen, vorher definierten Softwareprojekten der Programmiersprachen JavaScript, Python und Java. Für jede Aufgabe bekommt der Befragte 6 verschiedene Quelltextdateien angezeigt. Der Befragte muss jede Quelltextdatei zu einem von 2 Clustern zuordnen, wobei 2 Cluster gefunden werden müssen. Diese Einschränkung musste eingebaut werden, da der Befragte nicht alle Quelltextdateien des gesamten Softwareprojektes kennt und daraus keinen Kontext bezüglich der Anzahl der Cluster erkennen kann.

Abbildung 6.2 zeigt einen Screenshot der Webapplikation, welche als Umfrage fungiert. Dabei sind 6 Quelltextdateien zu erkennen. Die Quelltextdateien werden mit einem bekannten Farbschema dargestellt und sind nicht bearbeitbar. Ober den Quelltextdateien befindet sich ein Kombinationslistenfeld, in dem der Befragte zwischen den Werten „Cluster 1“ und „Cluster 2“ auswählen kann. Nach den 6 Quelltextdateien befindet sich ein Button mit der Aufschrift „Submit“, mit welchem der Befragte die Möglichkeit hat, seine Zuordnung der Analyse bereitzustellen.

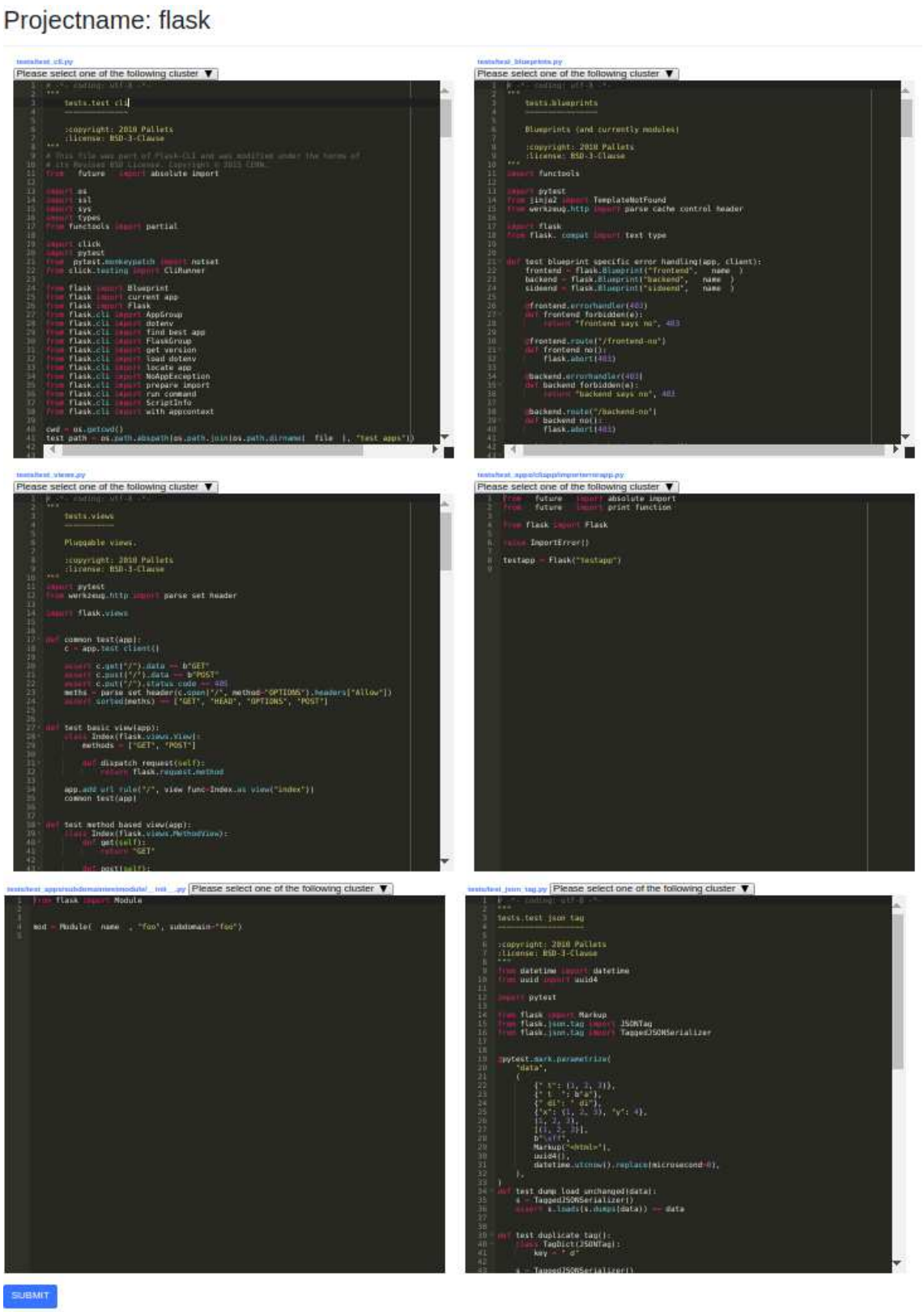


Abbildung 6.2: Screenshot der Umfrage zur Evaluierung des Ergebnisses der Applikation.

Nachdem eine Quelltextdatei einem Cluster zugeordnet wurde, bekommt sie eine farbliche Umrandung. Dabei wurde lila für „Cluster 1“ und grün für „Cluster 2“ gewählt. Abbildung 6.3 zeigt die farbliche Umrandung für 2 Quelltextdateien nach der Zuweisung zu Clustern.

Projectname: flask

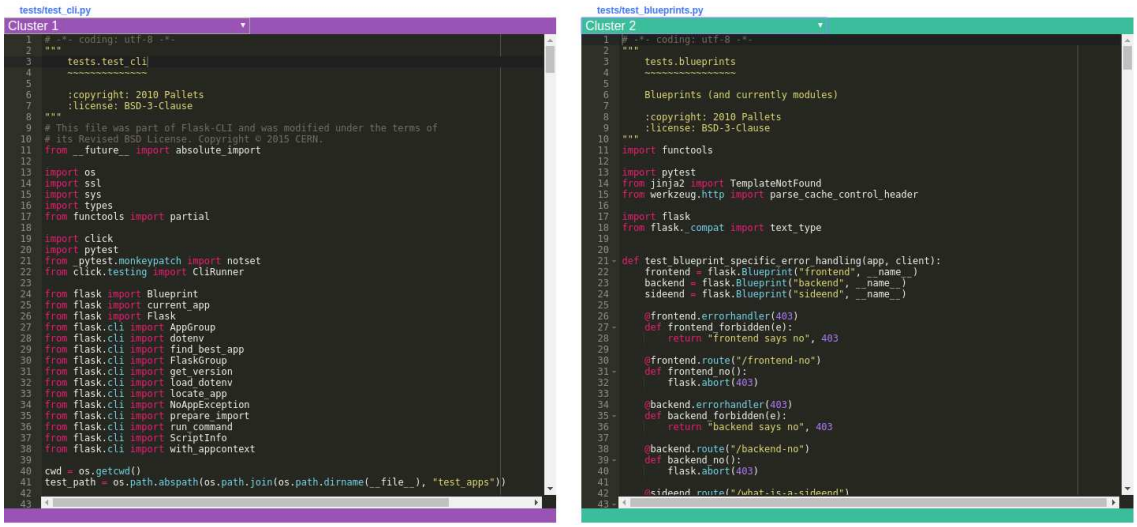


Abbildung 6.3: Screenshot der Umfrage zur Evaluierung des Ergebnisses der Applikation, mit bereits zugewiesenen und eingefärbten Clustern

Für die Bewertung der einzelnen Aufgaben wurden 2 verschiedene Bewertungssysteme verwendet.

- Bewertung nach Wahrscheinlichkeiten
- Bewertung der Cluster mittels normalisierter Transinformation

In beiden Bewertungssystemen werden Punkte vergeben, wenn der Befragte gleiche Zuordnungen zu Clustern, wie das Programm, welches im Rahmen dieser Diplomarbeit entwickelt wurde, vergibt. Ein Bewertungssystem verwendet die normalisierte Transinformation, die in Kapitel 2.5.3 erklärt wurde. Dabei wird die Normalisierte Transinformation der Zuweisungen von Quelltextdateien zu Clustern von den Befragten mit der Zuweisung von Quelltextdateien zu Clustern des Programmes berechnet. Der Wert ist eine Zahl zwischen 0 und 1, wobei die Zahl 1 repräsentiert, dass der Befragte alle Quelltextdateien denselben Clustern zugewiesen hat, wie das Programm. Je kleiner der Wert ist, umso mehr unterschiedliche Zuweisungen gibt es zwischen den Zuweisungen des Befragten und des Programmes.

Das zweite Bewertungssystem vergibt 4 unterschiedliche Anzahlen an Punkten, wobei die Punkte höher sind, je unwahrscheinlicher es ist, die Anzahl an Zuweisungen zufällig zu erraten. Dabei wird die Punktevergabe in der Tabelle 6.1 aufgelistet und erklärt. Die höchste Punkteanzahl von 41 wird erreicht, wenn der Befragte alle Zuordnungen von Quelltextdateien zu Clustern gleich wählt, wie die Kommandozeilenapplikation. Durch die Aufgabenstellung, dass definitiv 2 Cluster zugewiesen werden sollen, ist die kleinste Punkteanzahl 4. Der Grund dafür ist, dass durch die Aufgabenstellung und der Bewertung sicher 3 idente Zuweisungen existieren müssen. Die Zuweisungen des Programmes ist ein Array von 6 binären Zahlen. Zum Beispiel wäre die Zuweisung $Z_A = [0, 1, 1, 0, 0, 0]$ eine gültige Zuweisung der Applikation. Für die Bewertung der Zuweisung des Befragten Z_B, wird das Array Z_B negiert. Wäre also $Z_B = [1, 0, 0, 1, 1, 1]$, dann ist $-Z_B = [0, 1, 1, 0, 0, 0]$. Für die Bewertung werden beide Arrays Z_B und $-Z_B$ mit Z_A über eine Konjunktion verknüpft. Die Punkteanzahl P wäre der maximale Wert der Summe des Arrays aus den Konjunktionen. Dadurch ist es nicht möglich weniger als 3 idente Zuweisungen zu haben,

da die negierte Form der Zuweisungen mehr idente Zuweisungen besitzt und diese auch in die Bewertung einfliesst. Die Formel der Bewertung ist in 6.1 definiert.

$$P = \max(\sum_{x \in (Z_B \wedge Z_A)} x, \sum_{y \in (Z_B \wedge Z_A)} y) \tag{6.1}$$

Anzahl an identen Zuweisungen R	$P(R)$	**Punkte**
6	$\frac{1}{\frac{6!}{5!\cdot 1!}+\frac{6!}{4!\cdot 2!}+\frac{6!}{3!\cdot 3!}} = \frac{1}{41}$	41
5	$\frac{1}{\frac{5!}{5!}+\frac{5!}{4!\cdot 1!}+\frac{5!}{3!\cdot 2!}} = \frac{1}{16}$	16
4	$\frac{1}{\frac{4!}{4!}+\frac{4!}{3!\cdot 1!}+\frac{4!}{2!\cdot 2!}} = \frac{1}{9}$	9
3	$\frac{1}{\frac{3!}{3!}+\frac{3!}{2!\cdot 1!}} = \frac{1}{4}$	4

Tabelle 6.1: Tabelle zur Erklärung der Punktevergabe des Bewertungssystems nach Wahrscheinlichkeiten.

In der Tabelle 6.2 werden 4 Beispiele gezeigt, welche die Bewertung nach Wahrscheinlichkeiten veranschaulichen sollen.

Zuweisungen des Programmes $Z_P = (Q_1, Q_2, Q_3, Q_4, Q_5, Q_6)$	**Zuweisungen des Befragten** $Z_B = (Q_1, Q_2, Q_3, Q_4, Q_5, Q_6)$	**# Idente Zuweisungen**	**Punkte**
(1,2,2,2,2,2)	(2,1,1,1,1,1)	6	41
(1,2,2,1,2,2)	(2,2,1,2,1,1)	5	16
(1,1,2,1,2,2)	(1,2,1,2,1,1)	4	9
(1,2,2,2,2,2)	(1,2,1,2,1,1)	3	4

Tabelle 6.2: Tabelle zur Berechnung von Beispiel Zuweisungen nach dem Bewertungssystems nach Wahrscheinlichkeiten.

6.2 Ausführung der Studie zur Evaluierung der Zuordnung

In diesem Kapitel wird die Ausführung der zwei Studien beschrieben, welche in Kapitel 6.1 entwickelt wurden.

Dabei wurden 10 Personen ausgewählt, die bereits Erfahrungen mit Softwareentwicklung haben. Jeder der 10 Personen hat zuerst die Umfrage, welche als Webapplikation entwickelt wurde, beantwortet. Dafür musste jeder Befragte mindestens 20 mal 6 Quelltextdateien zu Clustern zuordnen. Danach wurde der Fragebogen zur Bedarfsanalyse per Email an die Befragten gesandt, sodass sie in Ruhe den Fragebogen zu einem anderen Zeitpunkt beantworten konnten.

Die ausgewählten Softwareprojekte, von denen die Quelltextdateien zur Zuordnung angezeigt werden, sind in der Tabelle 6.3 dargestellt. Dabei wurde versucht, bekannte und vergleichbare Softwa-

reprojekte zu wählen. So wurde zu jeder Programmiersprache ein Instagram-Klon in die Auswahl genommen. Ebenso wurde zu jeder Programmiersprache ein Framework für einen Web Server und eine Utility Bibliothek ausgewählt.

Projektname	**Programmiersprache**
Axios [7]	JavaScript
hospitalrun-frontend [36]	JavaScript
node-twitter [58]	JavaScript
React [70]	JavaScript
pinstagram [65]	JavaScript
jQuery [44]	JavaScript
flask [25]	Python
HTTPie [37]	Python
Instagram clone [39]	Python
okHttp [60]	Java
Jenkins [65]	Java
Spring Boot [83]	Java
Guava [32]	Java
my-moments [56]	Java

Tabelle 6.3: Auswahl der Softwareprojekte, die für die Umfrage zur Evaluierung der Ergebnisse der Kommandozeilenapplikation herangezogen wurden.

Zeitlich hatten die Befragten keine Vorgaben. Die durchschnittliche Dauer der Umfrage für die Webapplikation für alle Aufgaben hat 90 Minuten betragen. Somit benötigten die Befragten durchschnittlich 4,5 Minuten pro Aufgabe, um die 6 Quelltextdateien Clustern zuzuordnen.

6.3 Analyse der Ergebnisse der Studie zur Evaluierung der Zuordnung

Die Ergebnisse der zwei Umfragen, welche im Kapitel 6.2 beschrieben wurden, wurden bewertet und analysiert. Dazu werden zuerst die Ergebnisse der Umfrage für die Bedarfsanalyse in Kapitel 6.3.1 präsentiert. Anschließend widmet sich Kapitel 6.3.2 den Ergebnissen der Umfrage zur Evaluierung der Resultate der Applikation, welche im Rahmen dieser Diplomarbeit verfasst wurde.

6.3.1 Ergebnisse der Studie zur Bedarfsanalyse

In diesem Kapitel werden die Antworten zu dem Fragebogen der Bedarfsanalyse ausgewertet. Dabei wird dieselbe Reihenfolge für die Auswertung gewählt, wie schon bei dem Entwurf des Fragebogens in Kapitel 6.1.1.

Abbildung 6.4 zeigt dabei ein Diagramm, welches die Ergebnisse der ersten Fragen veranschaulicht. Durchschnittlich wurde diese Frage mit 1,6 Punkten beantwortet. Abbildung 6.5 zeigt eben-

falls ein Diagramm, welches die Ergebnisse der zweiten Frage veranschaulicht. Diese Frage wurde im Schnitt mit 2,4 Punkten beantwortet. Ein Diagramm mit den Ergebnissen der dritten Frage wird in Abbildung 6.6 angezeigt. Die durchschnittliche Punkteanzahl der Antworten auf diese Frage betrug 1,5 Punkte. Abbildung 6.7 zeigt die Ergebnisse der vierten Frage, die als Diagramm veranschaulicht wurde. Im Schnitt wurde diese Frage mit 1,6 Punkten beantwortet. Durchschnittlich wurde die fünfte Frage mit 1,7 Punkten beantwortet. Das Diagramm, welches die Ergebnisse zu dieser Frage veranschaulicht, wird in Abbildung 6.8 angezeigt.

Zusätzlich werden in der Darstellung 6.9 Boxplots zu den Antworten der Fragen angezeigt, um die Verteilung zu veranschaulichen.

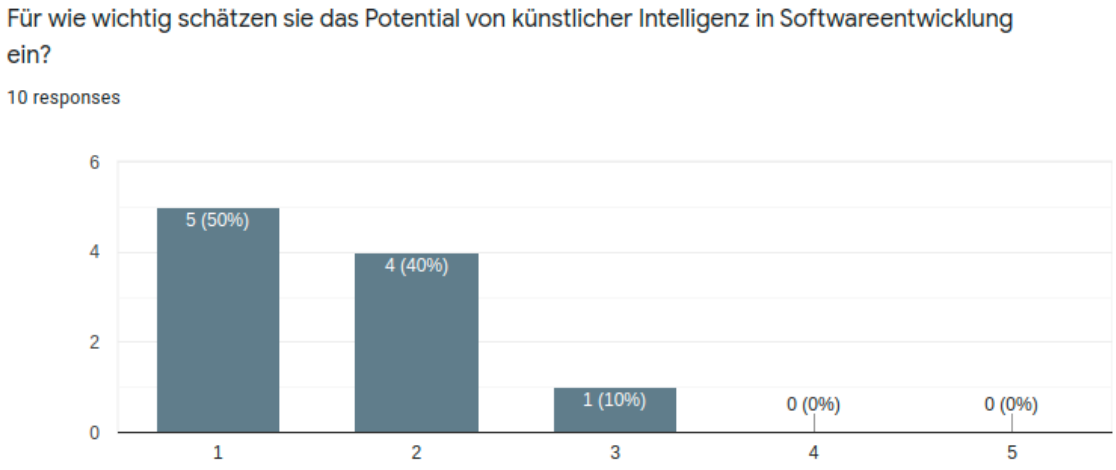


Abbildung 6.4: Ergebnisse der ersten Frage der Umfrage zur Bedarfsanalyse

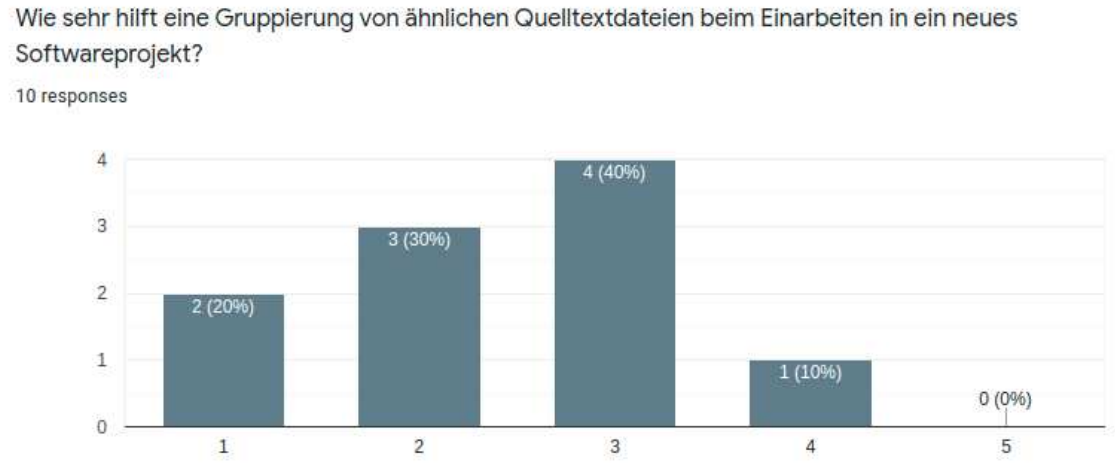


Abbildung 6.5: Ergebnisse der zweiten Frage der Umfrage zur Bedarfsanalyse

Abbildung 6.6: Ergebnisse der dritten Frage der Umfrage zur Bedarfsanalyse

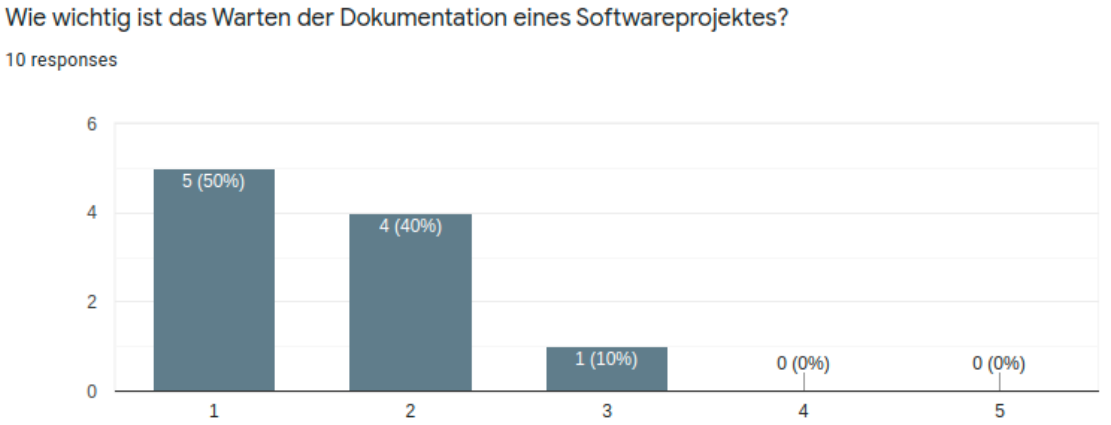


Abbildung 6.7: Ergebnisse der vierten Frage der Umfrage zur Bedarfsanalyse

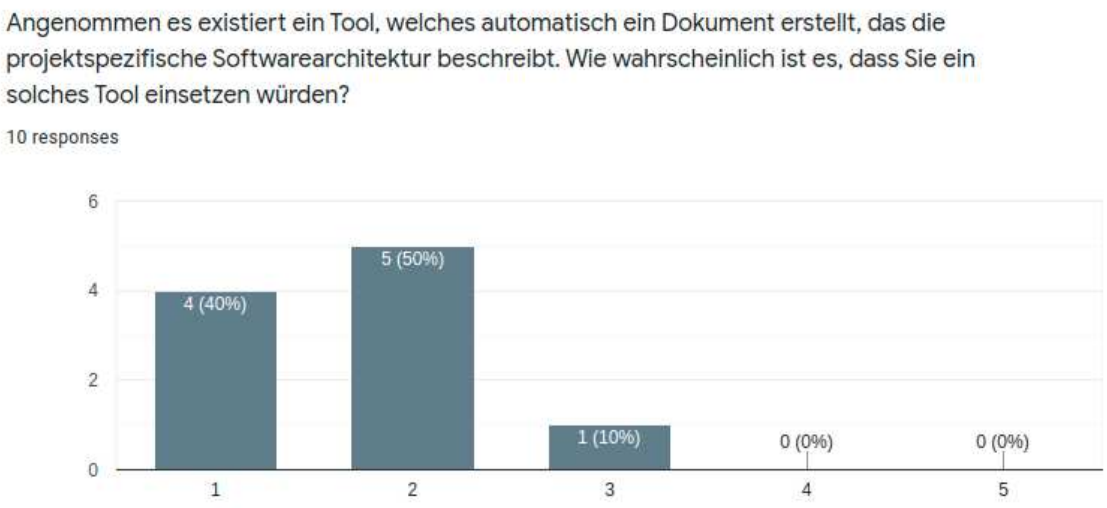
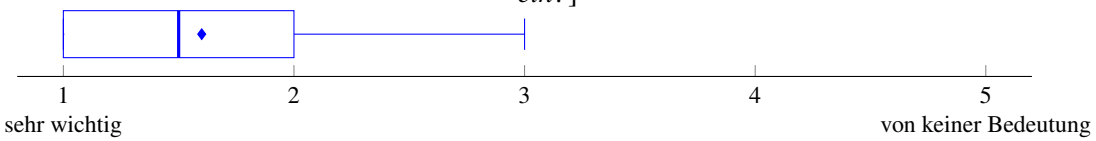
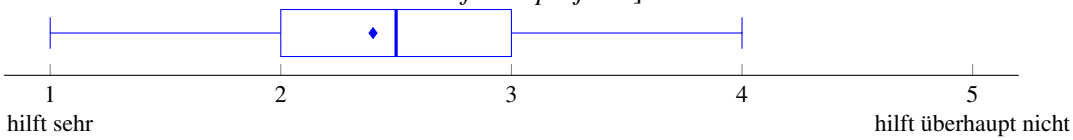


Abbildung 6.8: Ergebnisse der fünften Frage der Umfrage zur Bedarfsanalyse

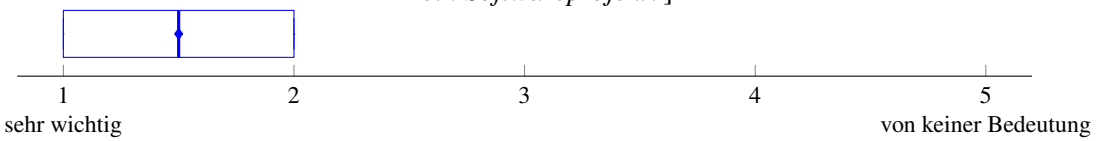
[*Für wie wichtig schätzen sie das Potential von künstlicher Intelligenz in Softwareentwicklung ein?*]



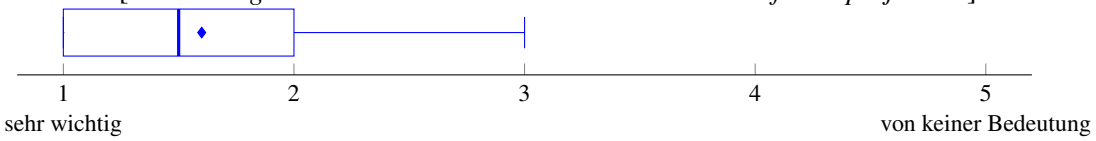
[*Wie sehr hilft eine Gruppierung von ähnlichen Quelltextdateien beim Einarbeiten in ein neues Softwareprojekt?*]



[*Wie wichtig ist das Erlernen einer projektspezifischen Softwarearchitektur für das Einarbeiten in ein Softwareprojekt?*]



[*Wie wichtig ist das Warten der Dokumentation eines Softwareprojektes?*]



[*Angenommen, es existiert ein Tool, welches automatisch ein Dokument erstellt, das die projektspezifische Softwarearchitektur beschreibt. Wie wahrscheinlich ist es, dass Sie ein solches Tool einsetzen würden?*]

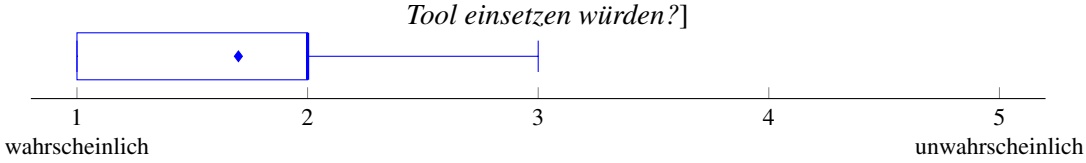


Abbildung 6.9: Boxplot zu den Antworten der Bedarfsanalyse.

In der Tabelle 6.4 werden alle unsere Forschungvsariablen, welche in Kapitel 6.1.1 definiert wurden, mit den Durchschnittswerten der Antworten der Umfrage bewertet.

Forschungsvariable V_i	**Bewertung**
V_1	1,6
V_2	2,4
V_3	1,5
V_4	1,6
V_5	1,7

Tabelle 6.4: Auswahl der Softwareprojekte, die für die Umfrage zur Evaluierung der Ergebnisse der Kommandozeilenapplikation herangezogen wurden.

Aus den Forschungsvariablen leiten wir folgende Aussage ab, die in derselben Reihenfolge wie die Forschungsvariablen präsentiert werden:

1. Personen im Bereich der Softwareentwicklung schätzen das Potential von KI in Softwareentwicklung sehr hoch ein.
2. Gruppierungen von ähnlichen Quelltextdateien eines Softwareprojektes würde den meisten Personen im Bereich der Softwareentwicklung in der Einarbeitungsphase des Projektes helfen.
3. Dokumentation über die projektspezifische Softwarearchitektur würde allen Befragten in der Einarbeitungsphase eines Softwareprojektes helfen.
4. Die Relevanz des Wartungsaufwandes von Dokumentation in Softwareprojekten wird sehr hoch eingeschätzt.
5. Es gibt Bedarf nach einem Programm, welches automatisiert Dokumentation über die projektspezifische Softwarearchitektur generiert.

Diese Ergebnisse liefern eine Evidenz, dass die Hypothese zutrifft, jedoch ist weitere Forschung in diesem Gebiet notwendig um die Hypothese zu bestätigen.

Somit geht diese Studie davon aus, dass Bedarf nach einer Applikation besteht, welche automatisiert Dokumentation über die projektspezifische Softwarearchitektur generiert.

6.3.2 Ergebnisse der Studie zur Evaluierung der Resultate der Applikation

In diesem Kapitel werden die Zuweisungen der Quelltextdateien zu Clustern der Befragten ausgewertet und mit den Zuweisungen der Applikation verglichen, welche im Rahmen dieser Diplomarbeit implementiert wurde.

Abbildung 6.11 zeigt die Anzahl der identen Zuweisungen von Quelltextdateien zu Clustern zwischen den Befragten und der Applikation, welche im Rahmen dieser Diplomarbeit verfasst wurde. In der Darstellung 6.10 wird ein Boxplot der Ergebnisse angezeigt.

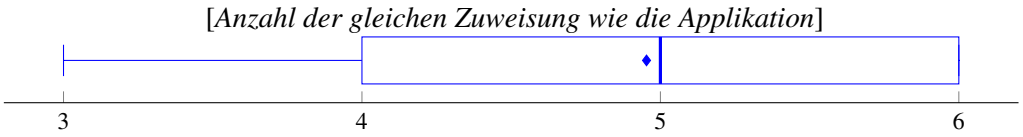


Abbildung 6.10: Anzahl der gleichen Zuweisung wie die Applikation.

In Abbildung 6.11 ist deutlich zu erkennen, dass die am häufigsten vorkommende Gruppe diejenige ist, in der alle 6 Quelltextdateien ident den Clustern zugewiesen wurden, die dieses Programm ebenfalls gewählt hat.

Insgesamt wurden 215 Aufgaben von 10 Befragten aufgezeichnet.

Nach der Bewertung der Wahrscheinlichkeit, welche in Kapitel 6.1.2 definiert ist, ergibt sich somit ein Durchschnittswert von 23.90 Punkten für alle Befragten. Im Schnitt gab es somit über 5 idente Zuweisungen zwischen den Befragten und der Applikation. Zum Vergleich wurde ein Algorithmus geschrieben, welcher die Zuweisung zufällig simuliert. Dieser Algorithmus lieferte für 1.000.000 Aufgaben einen durchschnittlichen Wert von 9,973095 und liegt somit im Durchschnitt knapp über 4 identen Zuweisungen.

Nach der Bewertung der normalisierten Transinformation der Cluster haben die Befragten ein durchschnittliches Ergebnis von 0,563 Punkten erreicht. Auch für dieses Berwertungssystem wurde eine Simulation mit 1.000.000 Aufgaben durchgeführt, welche die Zuordnungen zufällig treffen. Diese Simulation lieferte einen Durchschnittswert von 0,1987 Punkten.

Somit liegt nach beiden Berwertungssystemen die durchschnittliche Punktezahl der Befragten deutlich über der durchschnittlichen Punktezahl zufälliger Zuordnung. Daraus kann der Schluss gezogen werden, dass die Hypothese, welche in Kapitel 6.1.2 aufgestellt wurde, der Wahrheit entspricht. Um diese Hypothese jedoch zu bestätigen, sind weiter Forschungen notwendig.

Dementsprechend ist mit dieser Studie eine Evidenz geliefert worden, dass die Cluster der Applikation ähnliche Cluster abbilden wie erfahrene Personen im Bereich der Softwareentwicklung.

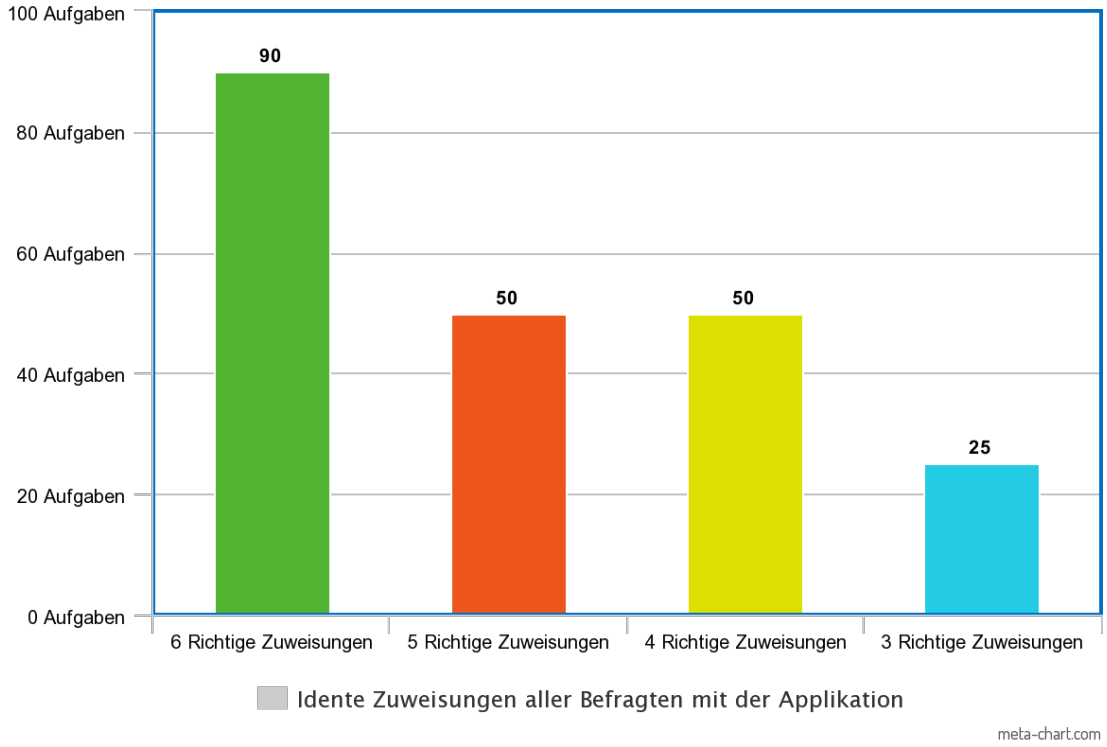


Abbildung 6.11: Ergebnisse der Umfrage nach identen Zuweisungen von Quelltextdateien zu Clustern der Befragten mit der Applikation

6.3.3 Forschungsvalidität

Durch den hohen Aufwand der Umfrage an die Befragten, wurde nur 10 Leute beauftragt diese Umfragen durchzuführen. Da dies ein kleiner Wert einer Stichprobengröße ist, kann die Hypothese nicht bestätigt werden, sondern nur Evidenz liefern, dass die Hypothesen der Wahrheit entsprechen.

Außerdem sind alle Befragten aus den professionellen Arbeitsumfeld des Autoren, da fachkundige erfahrene Personen für die Umfrage gesucht wurden.

Ein weitere Faktor, der die Ergebnisse verzerren könnte ist die Auswahlgröße der Softwareprojekte. Hier wäre eine größere Auswahlgröße notwendig um detailliertere Aussagen über die Ergebnisse schlussfolgern zu können.

6.3.4 Fallbeispiel eines Softwareprojektes

In diesem Kapitel werden anhand eines Beispielprojektes die Ergebnisse der Applikation beschrieben und in einen Kontext gestellt. Die Applikation, welche in dieser Diplomarbeit entwickelt wurde, erkennt zwar Ähnlichkeiten von Quelltextdateien, benennt diese jedoch nicht. Deswegen wird

in diesem Kapitel anhand eines Fallbeispiels beschrieben, wie die Resultate der Applikation zu deuten sind.

Als Beispiel wurde das JavaScript-Softwareprojekt pinstagram [65] gewählt, weil es überschaubar und damit leicht nachzuvollziehen ist.

In Abbildung 6.12 werden alle gefundenen Cluster und deren zugewiesene Quelltextdateien des Softwareprojektes angezeigt.

In dem Cluster mit dem Titel „Cluster #2“ sind alle Quelltextdateien, die Routen für das Softwareprojekt definieren, enthalten. Die Quelltextdateien, welche die grafische Oberfläche und deren Interaktion beinhalten, werden in „Cluster #1“ eingeteilt. Die Modelle sind in dem Cluster „Cluster #3 “ zusammengefasst. Der Cluster „Cluster #0“ listet alle Quelltextdateien auf, welche Controller und Middlewares dieses Softwareprojektes implementieren. Außerdem sind die restlichen Quelltextdateien auch diesen Cluster „Cluster #0“ zugeordnet. In Abbildung 6.13 sind die Ergebnisse in einem 3-dimensionalen Raum abgebildet. Daraus lässt sich ablesen, welche Cluster einander ähnlich sind. Je geringer die Distanz zwischen den Punkten ist, desto ähnlicher sind sich die Quelltextdateien. Demnach sind sich die Quelltextdateien aus „Cluster #0“ und „Cluster #3“ am ähnlichsten. Außerdem ist „Cluster #0“ der Cluster, in dem die Quelltextdateien am unterschiedlichsten sind, da dieser Cluster eine Mischung aus Controller, Middlewares und den restlichen Quelltextdateien ist.

Cluster #0
./handlers/errors.js
./handlers/passport.js
./handlers/mail.js
./controllers/notificationControllers.js
./controllers/authControllers.js
./controllers/userControllers.js
./controllers/commentControllers.js
./controllers/imageControllers.js
./webpack.config.js
./helpers.js
./server.js

Cluster #1
./public/javascript/main.js
./public/javascript/modules/modalRemove.js
./public/javascript/modules/handleLikes.js
./public/javascript/modules/handleFollow.js
./public/javascript/modules/handleNotifications.js
./public/javascript/modules/handleComments.js
./public/javascript/modules/uploadImage.js
./public/javascript/modules/modal.js
./public/javascript/modules/shortDom.js

Cluster #2
./routes/api.js
./routes/auth.js
./routes/routes.js

Cluster #3
./models/User.js
./models/Image.js
./models/Comment.js

Abbildung 6.12: Listenansicht des Ergebnisses des Fallbeispiels pinstagram [65].

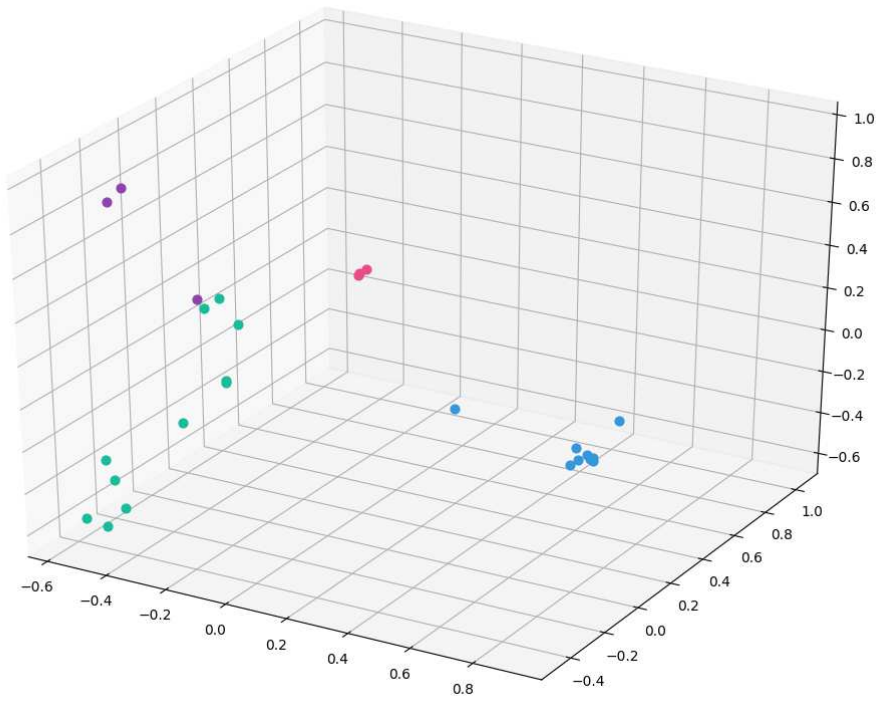


Abbildung 6.13: Ergebnisses des Fallbeispiels pinstagram [65] abgebildet in einem 3-dimensionalen Raum.

7 Zusammenfassung und Ausblick

Dieses Kapitel vermittelt einen Überblick und eine kurze Zusammenfassung aller Abschnitte dieser Diplomarbeit. In Kapitel 7.1 werden mögliche zukünftige Arbeiten beschrieben, die auf Erkenntnissen dieser Diplomarbeit aufbauen.

In der Einleitung wurde im Kapitel 1.1 die Problemstellung dieser Diplomarbeit beschrieben, dass die Einarbeitungsphase in ein neues Projekt für Personen im Bereich der Softwareentwicklung mit viel Aufwand verbunden ist, da Verständnis der projektbezogenen Softwarearchitektur für das neue Projekt notwendig ist.

Aus der Problemstellung sind Ziele dieser Arbeit definiert worden. Das Resultat dieser Diplomarbeit ist nach der Zielsetzung eine Kommandozeilenapplikation, welche Personen im Bereich der Softwareentwicklung helfen soll, Verständnis über die projektbezogene Softwarearchitektur zu erlangen.

In Abschnitt 1 sind alle relevanten Grundlagen erklärt worden, um dem Rest dieser Diplomarbeit folgen zu können.

Um den heutigen Stand der Technik in dem Forschungsgebiet „Machine Learning“ von Softwareprojekten aufzuzeigen, wurden alle relevanten wissenschaftlichen Dokumente in Kapitel 3 beschrieben.

Bevor in Abschnitt 5 die Implementierung anhand eines iterativen Entwicklungsmodels erklärt wurde, sind die Anforderungen in Kapitel 4 analysiert und beschrieben worden.

Danach wurde, in Kapitel 6, das Ergebnis des resultierenden Kommandozeilenprogrammes mit einer Umfrage evaluiert.

7.1 Ausblick

Dieser Abschnitt soll den Ausblick auf mögliche zukünftige Arbeiten beschreiben, die auf Erkenntnissen dieser Diplomarbeit aufbauen.

7.1.1 Unterstützung mehrerer Programmiersprachen

Das Kommandozeilenprogramm, welches das Resultat dieser Diplomarbeit ist, unterstützt die folgenden drei Programmiersprachen:

- Java
- Python
- JavaScript

Aus der Tabelle 1.1 in Kapitel 1, welche die 20 populärsten Programmiersprachen nach dem PYPL Ranking von Juni 2019 auflistet [66], ist ersichtlich, dass diese drei Programmiersprachen 57.5% der Popularität ausmachen.

Mit einer Erweiterung könnte das Programm die projektbezogene Softwarearchitektur von Projekten ableiten, welche in einer anderen Programmiersprache entwickelt wurden, als die drei oben angeführten.

7.1.2 Integration

Das Resultat dieser Diplomarbeit ist ein Kommandozeilenprogramm. Diese Applikation könnte von Drittanbietern aufgerufen werden, um die Ergebnisse, die dieses Programm liefert, visuell darzustellen.

So könnte ein Onlinedienst, der Quelltextdateien mittels eines Version Control System (VCS) auf Server bereitstellt, eine automatisierte Dokumentation über die projektspezifische Softwarearchitektur dieser Projekte anbieten. Der Aufruf des Programmes könnte nach „Merges“ oder „Commits“ automatisch gestartet werden.

7.1.3 Visualisierung

Die Quelltextdateien und die dazugehörigen Cluster werden im Rahmen dieser Diplomarbeit auf zwei Arten visualisiert:

- **Baumstruktur**
  In der Baumstruktur werden Quelltextdateien als Blätter und Verzeichnisse als innere Knoten dargestellt.
- **Listen**
  Es gibt n Listen, wobei jede Liste einen Cluster repräsentiert. In dieser Liste werden alle Quelltextdateien, die zu diesem Cluster zugewiesen wurden, aufgelistet.

Diese zwei Arten geben einen guten Überblick über die verschiedenen Cluster und die dazugehörigen Quelltextdateien, zeigen aber nicht an, wie ähnlich sich die verschiedenen Cluster sind.

Diese Visualisierungen könnten verbessert werden, sodass auch hier Graphen angezeigt werden, in dem Punkte Quelltextdateien und die Farbe der Punkte die verschiedenen Cluster symbolisieren. Der Abstand der Punkte zueinander würde damit die Ähnlichkeit der Punkte darstellen.

Es könnten mehrere Visualisierungen, die mit Interaktionen verbunden sind, eingebaut werden, sodass der Benutzer einen genaueren Einblick in die Unterschiede der verschiedenen Quelltextdateien und Cluster bekommt.

7.1.4 Features der Feature Extraction erweitern

Um das Kommandozeilenprogramm, dass Quelltextdateien Clustern zuordnet, entwickeln zu können, wurde ein Algorithmus entwickelt, um von n Quelltextdateien eine $n \times m$ Matrix abzuleiten. Dieser Algorithmus leitet Eigenschaften aus Quelltextdateien ab und wird daher als „Preprocessing und Feature engineering“ bezeichnet. In der Matrix, die das Resultat des „Feature engineering“ ist, ist n die Anzahl der Quelltextdateien in dem Softwareprojekt und m ist die Anzahl der gefundenen Features.

Weitere Forschung wäre notwendig, um die Anzahl der Features anzupassen und das Clusterverfahren zu optimieren. Ein Ansatz wäre, die Historie der Quelltextdateien in das „Feature engineering“ mit einzubeziehen. Diese Historie ist bereits im VCS erhalten.

7.1.5 Verbesserungsvorschläge in der Softwarearchitektur

Aufbauend auf dieser Arbeit, könnte ein Programm geschrieben werden, welches die Softwarearchitektur des ausgewählten Projektes mit Softwarearchitekturen ähnlicher Softwareprojekte vergleicht.

Somit könnte diese Applikation Vorschläge von Bibliotheken geben, welche in ähnlichen Softwareprojekten verwendet wurden.

Zusätzlich könnte das Programm Quelltextbeispiele vorschlagen um die Bibliothek in das Projekt zu integrieren und im Zuge von Quelltext-Analyse-Programme eingesetzt werden.

7.1.6 Unterstützung bei Quelltext Restrukturierung

Mit den Kenntnissen, die aus dieser Arbeit hervorgehen, wäre es möglich ein Programm zur Unterstützung von Restrukturierung zu entwickeln. Damit könnte das Programm zum Beispiel eine neue Verzeichnis-Struktur für bestehende Software-Komponenten vorschlagen.

Literatur

Wissenschaftliche Literatur

[1] Amir Ahmad und Lipika Dey. “A k-mean clustering algorithm for mixed numeric and categorical data”. In: *Data & Knowledge Engineering* 63.2 (2007), S. 503–527.

[2] Alfred V Aho, Ravi Sethi und Jeffrey D Ullman. *Compilers, principles, techniques*. Bd. 7. 8. 1986, S. 9.

[3] Akiko Aizawa. “An information-theoretic perspective of tf–idf measures”. In: *Information Processing & Management* 39.1 (2003), S. 45–65.

[5] Imad Eddine Araar und Hassina Seridi. “Software features extraction from object-oriented source code using an overlapping clustering approach”. In: *Informatica* 40.2 (2016).

[6] David Arthur und Sergei Vassilvitskii. “k-means++: The advantages of careful seeding”. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial und Applied Mathematics. 2007, S. 1027–1035.

[8] Sanghamitra Bandyopadhyay und Sriparna Saha. *Unsupervised classification: similarity measures, classical and metaheuristic approaches, and applications*. Springer Science & Business Media, 2012.

[9] Victor R Basil und Albert J Turner. “Iterative enhancement: A practical technique for software development”. In: *IEEE Transactions on Software Engineering* 4 (1975), S. 390–396.

[10] Christopher M Bishop. *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006.

[11] Kurt Bittner und Ian Spence. *Managing iterative software development projects*. Addison-Wesley Professional, 2006.

[12] Tony Buzan und Barry Buzan. *The mind map book*. Pearson Education, 2006.

[13] Tadeusz Caliński und Jerzy Harabasz. “A dendrite method for cluster analysis”. In: *Communications in Statistics-theory and Methods* 3.1 (1974), S. 1–27.

[14] Andrew Carnie. *Syntax: A generative introduction*. Bd. 16. John Wiley & Sons, 2012.

[15] Dan Cireşan, Ueli Meier und Jürgen Schmidhuber. “Multi-column deep neural networks for image classification”. In: *arXiv preprint arXiv:1202.2745* (2012).

[16] David L Davies und Donald W Bouldin. “A cluster separation measure”. In: *IEEE transactions on pattern analysis and machine intelligence* 2 (1979), S. 224–227.

[17] Scott Deerwester u. a. “Indexing by latent semantic analysis”. In: *Journal of the American society for information science* 41.6 (1990), S. 391–407.

[18] Norman R Draper und Harry Smith. *Applied regression analysis*. Bd. 326. John Wiley & Sons, 1998.

[19] Scott Emmons u. a. “Analysis of network clustering algorithms and cluster quality metrics at scale”. In: *PloS one* 11.7 (2016), e0159161.

[20] Wolfgang Ertel. *Grundkurs künstliche Intelligenz: eine praxisorientierte Einführung*. Springer-Verlag, 2016.

[22] Martin Ester u. a. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *Kdd*. Bd. 96. 34. 1996, S. 226–231.

[23] Jan Faigl und Geoffrey A Hollinger. “Autonomous data collection using a self-organizing map”. In: *IEEE transactions on neural networks and learning systems* 29.5 (2017), S. 1703–1715.

[24] Anna Rita Fasolino und Giuseppe Visaggio. “Improving software comprehension through an automated dependency tracer”. In: *Proceedings Seventh International Workshop on Program Comprehension*. IEEE. 1999, S. 58–65.

[26] Robert W Floyd. “Algorithm 97: shortest path”. In: *Communications of the ACM* 5.6 (1962), S. 345.

[27] Alexander J Gates und Yong-Yeol Ahn. “The impact of random models on clustering similarity”. In: *The Journal of Machine Learning Research* 18.1 (2017), S. 3049–3076.

[29] Robert L Glass. “Frequently forgotten fundamental facts about software engineering”. In: *IEEE software* 3 (2001), S. 112–110.

[33] Hongchen Guo, Junbang Ma und Zhiqiang Li. “Active Semi-supervised K-Means Clustering Based on Silhouette Coefficient”. In: *International Conference on Intelligent and Interactive Systems and Applications*. Springer. 2018, S. 202–209.

[34] Douglas M Hawkins. “The problem of overfitting”. In: *Journal of chemical information and computer sciences* 44.1 (2004), S. 1–12.

[35] Thomas Hofmann. “Probabilistic latent semantic indexing”. In: *ACM SIGIR Forum*. Bd. 51. 2. ACM. 2017, S. 211–218.

[38] Anna Huang. “Similarity measures for text document clustering”. In: *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*. Bd. 4. 2008, S. 9–56.

[43] Stephen C Johnson. “Hierarchical clustering schemes”. In: *Psychometrika* 32.3 (1967), S. 241–254.

[45] James M Keller, Michael R Gray und James A Givens. “A fuzzy k-nearest neighbor algorithm”. In: *IEEE transactions on systems, man, and cybernetics* 4 (1985), S. 580–585.

[46] David J Ketchen und Christopher L Shook. “The application of cluster analysis in strategic management research: an analysis and critique”. In: *Strategic management journal* 17.6 (1996), S. 441–458.

[47] Han Kyul Kim, Hyunjoong Kim und Sungzoon Cho. “Bag-of-concepts: Comprehending document representation through clustering words in distributed representation”. In: *Neurocomputing* 266 (2017), S. 336–352.

[48] David G Kleinbaum u. a. *Logistic regression*. Springer, 2002.

[49] Adrian Kuhn, Stéphane Ducasse und Tudor Gírba. “Semantic clustering: Identifying topics in source code”. In: *Information and Software Technology* 49.3 (2007), S. 230–243.

[50] Cheng-Yuan Liou u. a. “Autoencoder for words”. In: *Neurocomputing* 139 (2014), S. 84–96.

[51] P. Louridas und C. Ebert. “Machine Learning”. In: *IEEE Software* 33.5 (2016), S. 110–115.

[52] Nenad Medvidovic und Richard N Taylor. “Software architecture: foundations, theory, and practice”. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM. 2010, S. 471–472.

[53] Richard J Miara u. a. “Program indentation and comprehensibility”. In: *Communications of the ACM* 26.11 (1983), S. 861–867.

[54] Todd K Moon. “The expectation-maximization algorithm”. In: *IEEE Signal processing magazine* 13.6 (1996), S. 47–60.

[55] Kevin P Murphy u. a. “Naive bayes classifiers”. In: *University of British Columbia* 18 (2006), S. 60.

[57] Andreas Noack. “Energy Models for Graph Clustering.” In: *J. Graph Algorithms Appl.* 11.2 (2007), S. 453–480.

[61] Paolo Oliveri u. a. “The impact of signal pre-processing on the final interpretation of analytical outcomes–A tutorial”. In: *Analytica chimica acta* (2018).

[62] Pariwat Ongsulee. “Artificial intelligence, machine learning and deep learning”. In: *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*. IEEE. 2017, S. 1–6.

[63] Airel Pérez-Suárez u. a. “OClustR: A new graph-based algorithm for overlapping clustering”. In: *Neurocomputing* 121 (2013), S. 234–247.

[67] Dorian Pyle. *Data preparation for data mining*. morgan kaufmann, 1999.

[68] Anand Rajaraman und Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[69] Juan Ramos u. a. “Using tf-idf to determine word relevance in document queries”. In: *Proceedings of the first instructional conference on machine learning*. Bd. 242. Piscataway, NJ. 2003, S. 133–142.

[71] Sandeep Reddivari, Mahesh Kotapalli und Nan Niu. “SDVisu: A tool for clustering-based visual exploration of static dependencies”. In: *2017 Computing Conference*. IEEE. 2017, S. 1373–1374.

[73] Peter J Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of computational and applied mathematics* 20 (1987), S. 53–65.

[74] Arthur L Samuel. “Some studies in machine learning using the game of checkers. II—recent progress”. In: (1988), S. 366–400.

[75] Advait Sarkar. “The impact of syntax colouring on program comprehension.” In: *PPIG*. 2015, S. 8.

[76] Mirco Schindler, Oliver Fox und Andreas Rausch. “Clustering source code elements by semantic similarity using Wikipedia”. In: *2015 IEEE/ACM 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. IEEE. 2015, S. 13–18.

[77] Erich Schubert u. a. “DBSCAN revisited, revisited: why and how you should (still) use DBSCAN”. In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), S. 19.

[79] George AF Seber und Alan J Lee. *Linear regression analysis*. Bd. 329. John Wiley & Sons, 2012.

[80] Haipeng Shen und Jianhua Z Huang. “Analysis of call centre arrival data using singular value decomposition”. In: *Applied Stochastic Models in Business and Industry* 21.3 (2005), S. 251–263.

[81] Jonathon Shlens. “A tutorial on principal component analysis”. In: *arXiv preprint arXiv:1404.1100* (2014).

[84] Chintakindi Srinivas, Vangipuram Radhakrishna und CV Rao. “Clustering Software Project Components for Strategic Decisions and Building Reuse Libraries”. In: *Proceedings of the The International Conference on Engineering & MIS 2015*. ACM. 2015, S. 62.

[85] Chintakindi Srinivas und CV Rao. “A Feature Vector Based Approach for Software Component Clustering and Reuse Using K-means”. In: *Proceedings of the The International Conference on Engineering & MIS 2015*. ACM. 2015, S. 67.

[87] Douglas Steinley. “K-means clustering: a half-century synthesis”. In: *British Journal of Mathematical and Statistical Psychology* 59.1 (2006), S. 1–34.

[88] Richard S Sutton, Andrew G Barto u. a. *Introduction to reinforcement learning*. Bd. 2. 4. MIT press Cambridge, 1998.

[89] Kardi Teknomo. “K-means clustering tutorial”. In: *Medicine* 100.4 (2006), S. 3.

[93] Leo R Vijayasarathy und Charles W Butler. “Choice of software development methodologies: Do organizational, project, and team characteristics matter?” In: *IEEE software* 33.5 (2015), S. 86–94.

[94] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), S. 395–416.

[95] Michael E Wall, Andreas Rechtsteiner und Luis M Rocha. “Singular value decomposition and principal component analysis”. In: *A practical approach to microarray data analysis*. Springer, 2003, S. 91–109.

[96] Stephen Warshall. “A theorem on boolean matrices”. In: *Journal of the ACM*. Citeseer. 1962.

[97] Ingo Wegener. *Theoretische Informatik: Eine algorithmenorientierte Einführung*. Springer-Verlag, 2013.

[98] Junjie Wu. *Advances in K-means clustering: a data mining thinking*. Springer Science & Business Media, 2012.

[99] Peipei Xia, Li Zhang und Fanzhang Li. “Learning similarity with cosine similarity ensemble”. In: *Information Sciences* 307 (2015), S. 39–52.

[100] Shichao Zhang, Chengqi Zhang und Qiang Yang. “Data preparation for data mining”. In: *Applied artificial intelligence* 17.5-6 (2003), S. 375–381.

[101] Yin Zhang, Rong Jin und Zhi-Hua Zhou. “Understanding bag-of-words model: a statistical framework”. In: *International Journal of Machine Learning and Cybernetics* 1.1-4 (2010), S. 43–52.

Online-Referenzen

[4] *alexa*. 2019. URL: https://www.alexa.com/ (besucht am 18. 10. 2019).

[7] *Axios*. 2019. URL: https://github.com/axios/axios (besucht am 04. 12. 2019).

[21] *esprima*. 2019. URL: https://esprima.org/ (besucht am 14. 10. 2019).

[25] *flask*. 2019. URL: https://www.palletsprojects.com/p/flask/ (besucht am 04. 12. 2019).

[28] *GitHub*. 2019. URL: https://github.com (besucht am 30. 09. 2019).

[30] *Google*. 2019. URL: https://www.google.com (besucht am 30. 09. 2019).

[31] *Google Trends*. 2019. URL: https://www.google.com/trends (besucht am 03. 10. 2019).

[32] *Guava*. 2019. URL: https://github.com/google/guava (besucht am 04. 12. 2019).

[36] *HospitalRun*. 2019. URL: https://github.com/HospitalRun/hospitalrun-frontend (besucht am 04. 12. 2019).

[37] *HTTPie*. 2019. URL: https://httpie.org/ (besucht am 04. 12. 2019).

[39] *Instagram clone*. 2019. URL: https://github.com/benigls/instagram (besucht am 04. 12. 2019).

[40] *JDependencyFinder*. 2019. URL: http://depfind.sourceforge.net/ (besucht am 27. 09. 2019).

[41] *jedit*. 2019. URL: http://www.jedit.org (besucht am 25. 09. 2019).

[42] *Jenkins*. 2019. URL: https://jenkins.io/ (besucht am 02. 11. 2019).

[44] *jQuery*. 2019. URL: https://jquery.com/ (besucht am 04. 12. 2019).

[56] *my-moments*. 2019. URL: https://github.com/amrkhaledccd/my-moments (besucht am 04. 12. 2019).

[58] *node-twitter*. 2019. URL: https://github.com/vinitkumar/node-twitter (besucht am 04. 12. 2019).

[59] *Octoverse*. 2019. URL: https://octoverse.github.com (besucht am 18. 10. 2019).

[60] *okHttp*. 2019. URL: https://square.github.io/okhttp/ (besucht am 04. 12. 2019).

[64] *pinstagram*. 2019. URL: https://github.com/afuh/pinstagram (besucht am 19. 10. 2019).

[65] *pinstagram*. 2019. URL: https://github.com/afuh/pinstagram (besucht am 04. 12. 2019).

[66] *PopularitY of Programming Language*. 2019. URL: http://pypl.github.io/PYPL.html (besucht am 30. 09. 2019).

[70] *react*. 2019. URL: https://reactjs.org/ (besucht am 04. 12. 2019).

[72] *RedMonk Ranking*. 2019. URL: https://redmonk.com/sogrady/2019/07/18/language-rankings-6-19/ (besucht am 30. 09. 2019).

[78] *scikit-learn*. 2018. URL: http://scikit-learn.org (besucht am 10. 05. 2018).

[82] *similarWeb*. 2019. URL: https://www.similarweb.com/ (besucht am 18. 10. 2019).

[83] *Spring Boot*. 2019. URL: https://spring.io/projects/spring-boot (besucht am 04. 12. 2019).

[86] *StackOverflow*. 2019. URL: https://stackoverflow.com (besucht am 30. 09. 2019).

[90] *tensorflow*. 2019. URL: https://www.tensorflow.org/ (besucht am 19. 10. 2019).

[91] *The State of the Octoverse: machine learning*. 2019. URL: https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning (besucht am 18. 10. 2019).

[92] *Tiobe index*. 2019. URL: https://www.tiobe.com (besucht am 03. 11. 2019).