Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit Informatik

Texploration: Automatische Analyse von grossen Textsammlungen

Autoren	Dominik Steiner Gëzim Zeneli
Hauptbetreuung	Mark Cieliebak
Nebenbetreuung	Pius von Däniken

Datum	07.06.2019

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

……

Unterschriften:

……...

……

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Texploration ist eine Webplattform für die Analyse von grossen Textsammlungen, welche im letzten Jahr im Rahmen einer Projekt- und Bachelorarbeit entwickelt wurde. Die Applikation automatisiert verschiedene Aufgaben im Bereich der Datenanalyse und Klassifikation.

Diese Arbeit verfolgte zwei Ziele. Zum einen sollten verschiedene Bereiche des Quellcodes verbessert und überarbeitet werden. Zum anderen sollten bestehende Algorithmen optimiert und neue hinzugefügt werden.

Der Fokus lag zu Beginn auf dem Engineering Teil. Es wurde ein Refactoring im Backend durchgeführt, welches zu einer starken Verbesserung der Codequalität führte. Ausserdem wurden durch den Einsatz eines Flask Server (Python Webserver) die Bereiche Node.js und Python stärker gekapselt. Im Anschluss wurde noch ein Konzept für den Einsatz von Microservices in der Texploration entwickelt.

Im zweiten Teil hingegen, wurde der Fokus hauptsächlich auf die Optimierung der bestehenden Algorithmen gelegt. Zusätzlich wurde die neue Komponente Language Detection eingeführt. Mithilfe der Language Detection kann der Benutzer sehen, ob sein Korpus mehrsprachig ist. Des Weiteren wurde die Komponente Topic Modeling überarbeitet und optimiert. Neu funktioniert das Topic Modeling nicht nur für englische Texte, sondern auch für andere Sprachen. Ausserdem wurde für Korpora, die grösser sind als 1 000 000 Dokumente, die Laufzeit optimiert, indem ein Random Sampling betrieben wird. Zusätzlich wurden noch verschiedene Varianten aufgezeigt, wie das Topic Modeling in einem nächsten Schritt verbessert werden könnte.

In der Zukunft sollte als nächstes das Microservices-Konzept umgesetzt werden. Dies würde vielversprechende Erweiterungsmöglichkeiten garantieren. Einerseits würde es die Architektur vereinfachen und somit auch den Aufwand für das Einarbeiten eines frischen Entwicklers. Andererseits können danach noch schneller und einfacher Module erweitert oder erstellt werden.

Abstract

Texploration is a web platform for the analysis of large text collections. It was developed last year as part of a project and bachelor thesis. The application automates various tasks in the field of data analysis and classification.

This work had two objectives. On the one hand, different areas of the source code were to be improved and revised. On the other hand, existing algorithms were to be optimized and new ones added.

At the beginning the focus was on the engineering part. Refactoring in the back end was carried out, which led to a strong improvement of the code quality. In addition, the two areas Node.js and Python were isolated more strongly by the employment of a Flask server (Python web server). Subsequently, a concept for the use of microservices in Texploration was developed.

In the second part, the focus was mainly on the optimization of existing algorithms. In addition, the new Language Detection component was introduced. With the help of Language Detection, the user can see whether his corpus is multilingual. Furthermore, component topic modeling has been revised and optimized. The topic modelling now works not only for English texts, but also for other languages. For corpora larger than 1 000 000 documents, the runtime has been optimized by random sampling. In addition, different variants were shown which suggests how the topic modeling could be improved.

In the future, the next step should be to implement the microservices concept. This would guarantee promising extension possibilities. On the one hand, it would simplify the architecture and thus the effort for the integration of a fresh developer. On the other hand, modules can be extended or created more quickly and easily.

Vorwort

Im zweiten Semester haben wir im Modul "Algorithmen und Datenstrukturen" einen Einblick erhalten, was alles dank Algorithmen erreicht werden kann. Seitdem sind wir beide vom maschinellen Lernen fasziniert und haben nun die Möglichkeit, unser Wissen im Rahmen dieser Bachelorarbeit zu vertiefen.

An dieser Stelle bedanken wir uns herzlich bei unseren Betreuern, Mark Cieliebak und Pius von Däniken, für Ihre Hilfe und Engagement bei unserem Projekt. Zudem bedanken wir uns auch bei Herrn Walter Eich für seine fachmännische Beratung im Bereich Microservices. Ausserdem möchten wir uns bei Patrick Nef und Kevin Schwarz für das Gegenlesen der Arbeit und bei Arbnor Zeneli für das Korrekturlesen des Abstracts bedanken.

Inhaltsverzeichnis

1 Einleitung

Oftmals müssen Forscher und Datenwissenschaftler mit unbekannten Korpora arbeiten. Für gewöhnlich analysieren sie diese Texte mit selber geschriebenen Tools. Manche wichtige Schritte können nicht erledigt werden, da die Analysen viel Arbeit benötigen[1].

1.1 Texploration

Die Texploration ist im letzten Jahr während einer Projekt- und Bachelorarbeit entstanden. Mit dieser Webplattform versuchte man unter anderem das genannte Problem zu lösen.

Das Frontend wurde hauptsächlich mit Vue.js entwickelt. Für das Backend wurde Node.js und eine SQLite Datenbank verwendet. Die Businesslogik der Analysen wurde in Python umgesetzt.

Es wird nun näher auf die Funktionen der Texploration eingegangen.

Bevor der Benutzer einen Korpus analysieren kann, muss er den Datensatz im CSV-Format hochladen. Hier hat er die Möglichkeit, die Datei automatisch von der Texploration parsen zu lassen oder das Format manuell anzugeben.

Anschliessend hat der Benutzer bei dieser Datei die Option, entweder eine Corpus Analyse oder die Classification zu starten.

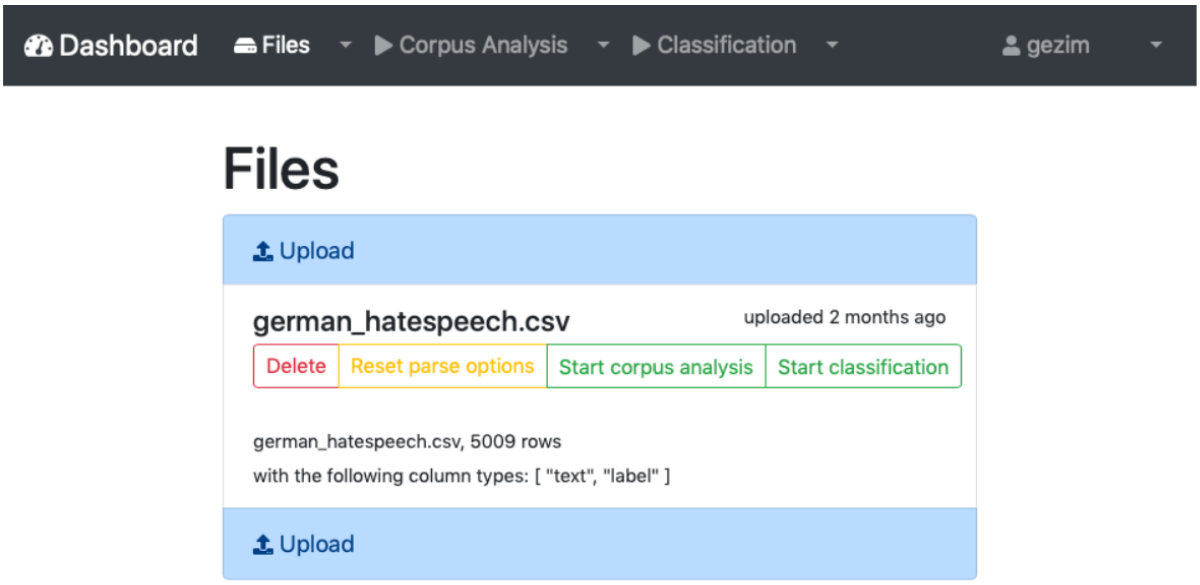
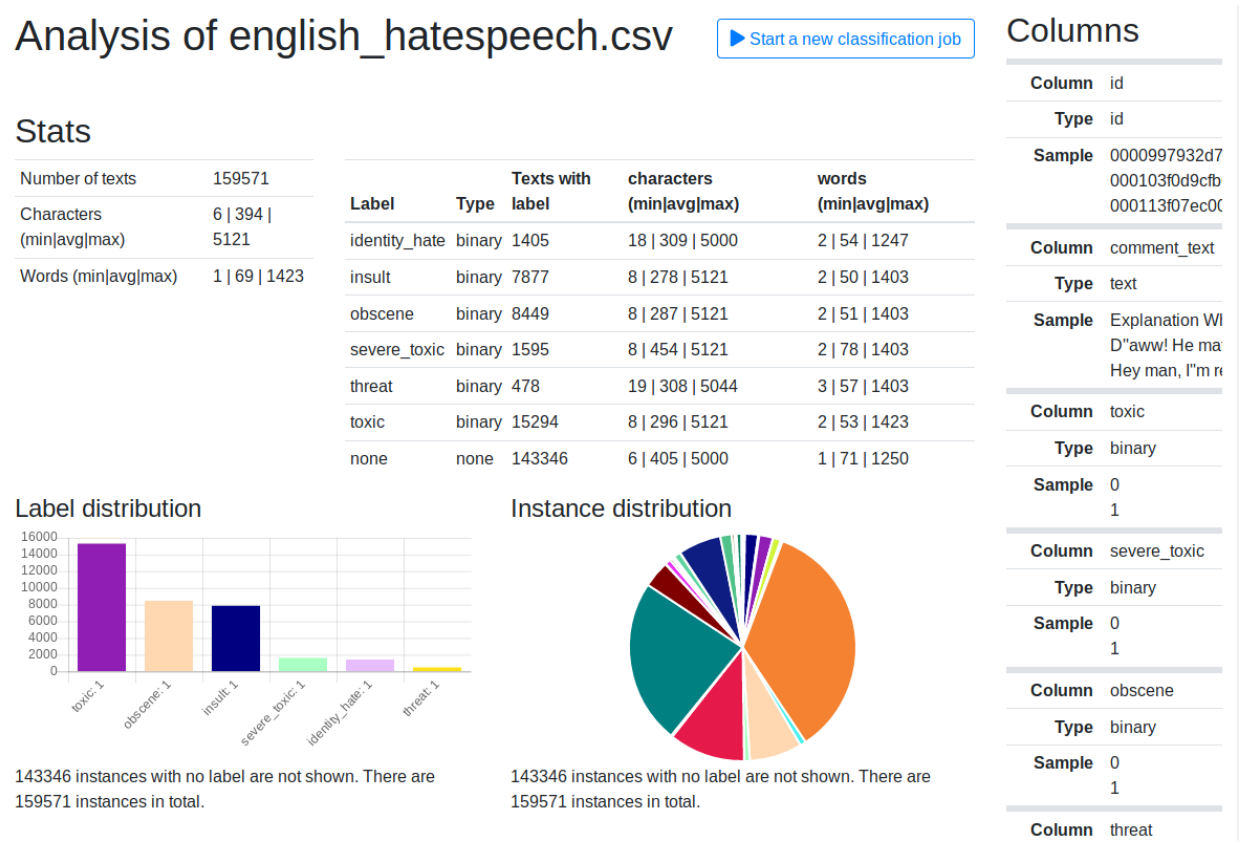


Abbildung 1 Auflistung der Optionen für eine hochgeladenen Datei

1.1.1 Corpus Analyse

Mithilfe der Corpus Analyse kann man sich einen Überblick über eine Textsammlung verschaffen.

Im ersten Teil der Analyse, die in der Abbildung 2 dargestellt ist, werden die allgemeinen Informationen zu den einzelnen Labels angezeigt. Unter anderem wird die Verteilung der Labels und verschiedene Statistiken zur Anzahl der Schriftzeichen sowie Wörter berechnet und dargestellt.



Analysis of english_hatespeech.csv

Start a new classification job

Stats

Number of texts	159571
Characters (min|avg|max)	6 | 394 | 5121
Words (min|avg|max)	1 | 69 | 1423

Label	Type	Texts with label	characters (min|avg|max)	words (min|avg|max)
identity_hate	binary	1405	18 | 309 | 5000	2 | 54 | 1247
insult	binary	7877	8 | 278 | 5121	2 | 50 | 1403
obscene	binary	8449	8 | 287 | 5121	2 | 51 | 1403
severe_toxic	binary	1595	8 | 454 | 5121	2 | 78 | 1403
threat	binary	478	19 | 308 | 5044	3 | 57 | 1403
toxic	binary	15294	8 | 296 | 5121	2 | 53 | 1423
none	none	143346	6 | 405 | 5000	1 | 71 | 1250

Label distribution

143346 instances with no label are not shown. There are 159571 instances in total.

Instance distribution

143346 instances with no label are not shown. There are 159571 instances in total.

Columns

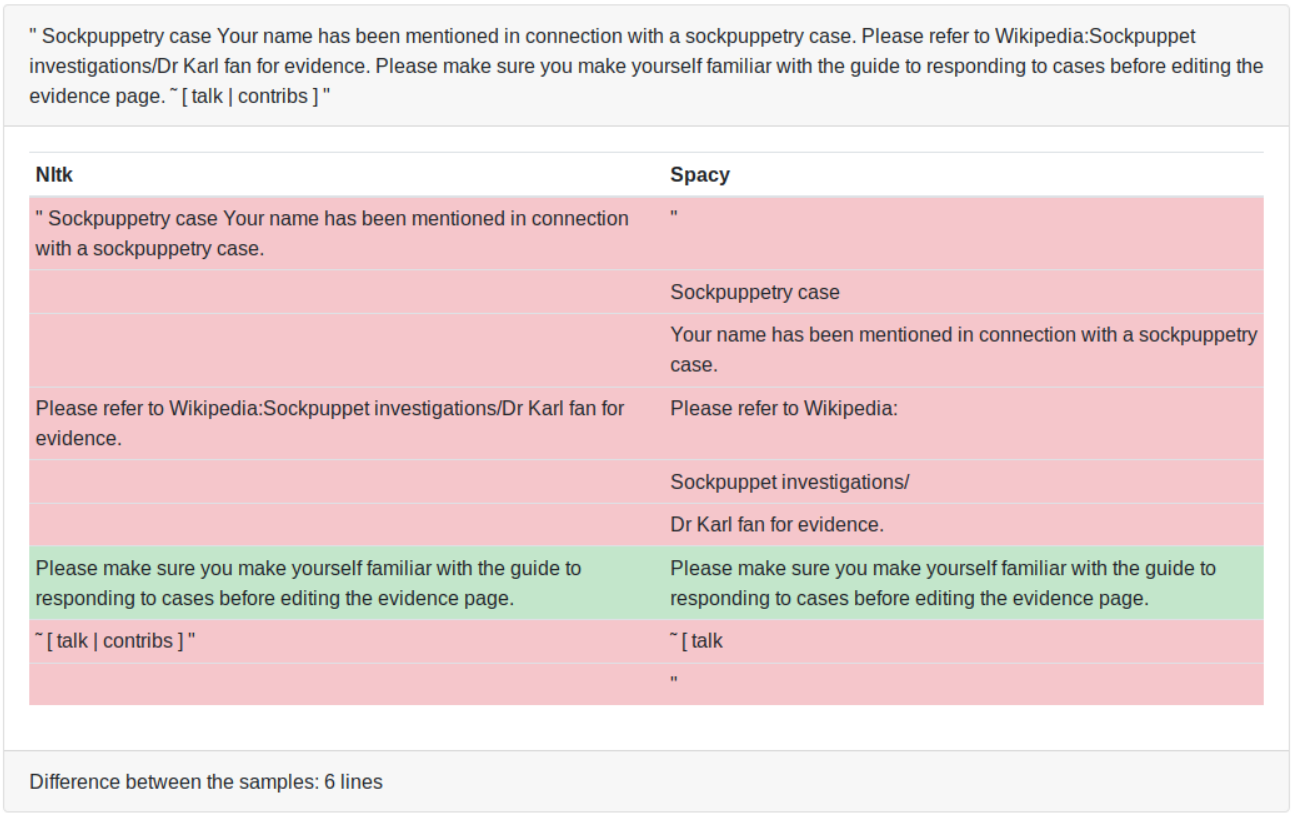
Column	id
Type	id
Sample	0000997932d7 000103f0d9cfb 000113f07ec0(
Column	comment_text
Type	text
Sample	Explanation Wh D"aww! He ma Hey man, I"m r
Column	toxic
Type	binary
Sample	0 1
Column	severe_toxic
Type	binary
Sample	0 1
Column	obscene
Type	binary
Sample	0 1
Column	threat

Abbildung 2 Ein Ausschnitt einer Corpus Analyse

Zusätzlich wird noch eine dynamische Word Cloud dargestellt. In der Abbildung 3 wird ein Beispiel einer Word Cloud dargestellt. Hier besteht die Möglichkeit, nur die Wörter pro Labels anzuzeigen.

Abbildung 3 Ein Beispiel einer Word Cloud der Corpus Analyse

Das Sentence Splitting beschäftigt sich mit der Aufteilung eines Textes in einzelne Sätze[2]. Weiter unten in der Analyse sieht der Benutzer einen Vergleich von zwei verschiedenen Modellen für das Splitting. Diese werden anhand mehrerer Beispiele mit Texten aus dem Korpus belegt.



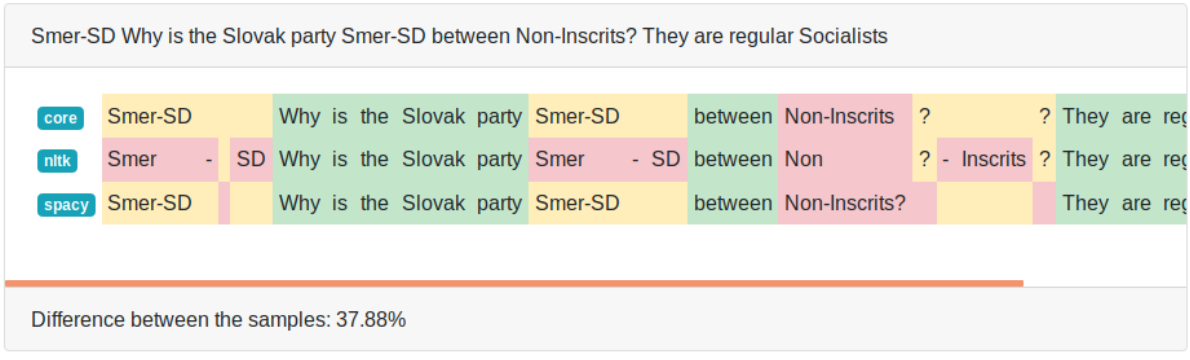
" Sockpuppetry case Your name has been mentioned in connection with a sockpuppetry case. Please refer to Wikipedia:Sockpuppet investigations/Dr Karl fan for evidence. Please make sure you make yourself familiar with the guide to responding to cases before editing the evidence page. ~ [talk | contribs] "

Nltk	Spacy
" Sockpuppetry case Your name has been mentioned in connection with a sockpuppetry case.	"
	Sockpuppetry case
	Your name has been mentioned in connection with a sockpuppetry case.
Please refer to Wikipedia:Sockpuppet investigations/Dr Karl fan for evidence.	Please refer to Wikipedia:
	Sockpuppet investigations/
	Dr Karl fan for evidence.
Please make sure you make yourself familiar with the guide to responding to cases before editing the evidence page.	Please make sure you make yourself familiar with the guide to responding to cases before editing the evidence page.
~ [talk | contribs] "	~ [talk
	"

Difference between the samples: 6 lines

Abbildung 4 Zeigt ein Beispiel für einen Sentence Splitting-Vergleich

Unter anderem existiert ein Vergleich von verschiedenen Tokenizer, ein Beispiel davon wird in der Abbildung 5 angezeigt. Ihre Aufgabe ist es einen Text in einzelne Wörter aufzuteilen[2].



Smer-SD Why is the Slovak party Smer-SD between Non-Inscrits? They are regular Socialists

core	Smer-SD			Why	is	the	Slovak	party	Smer-SD			between	Non-Inscrits		?			?	They	are	reg
nltk	Smer	-	SD	Why	is	the	Slovak	party	Smer	-	SD	between	Non		?	-	Inscrits	?	They	are	reg
spacy	Smer-SD			Why	is	the	Slovak	party	Smer-SD			between	Non-Inscrits?						They	are	reg

Difference between the samples: 37.88%

Abbildung 5 Ein Beispiel für einen Tokenizer-Vergleich

1.1.2 Classifier und Evaluation

Falls die hochgeladenen Daten mit Labels versehen wurden, können Classifier angewendet werden. Hier hat der Benutzer die Möglichkeit, verschiedene Classifier auf seinen Daten trainieren zu lassen. Ausserdem sieht der Benutzer die Resultate der Classifier und kann die einzelnen Classifier-Model herunterladen. Hierfür stehen fünf Classifier zur Auswahl: Naive Bayes, Support Vector Machine, Stochastic Gradient Descent, AdaBoost und Convolutional Neural Network.

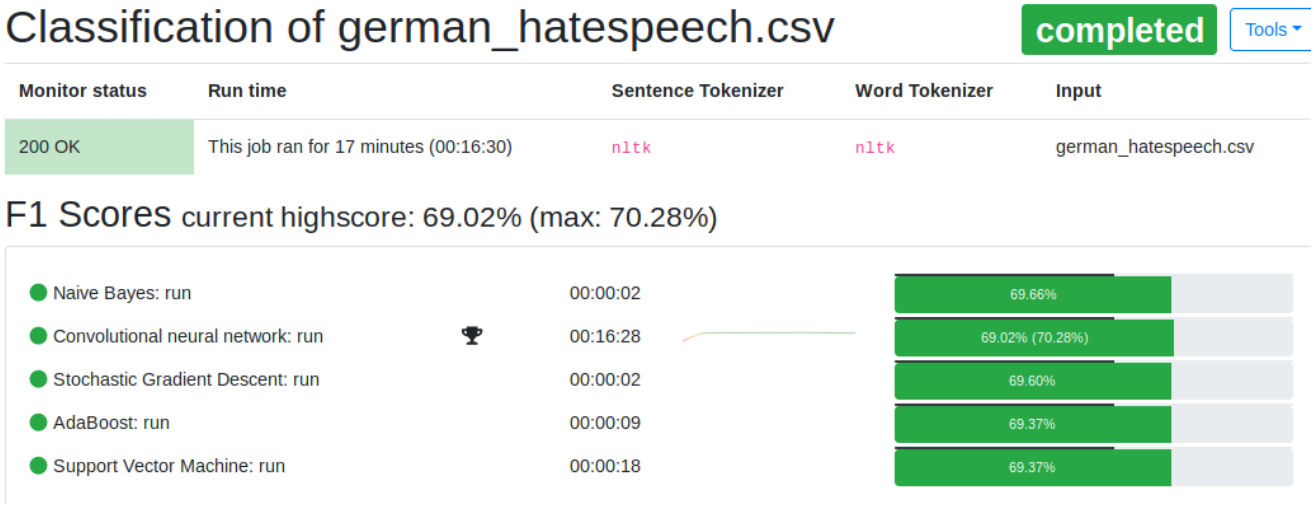


Abbildung 6 Eine Übersicht der erreichten Scores der Classifier

Mit den trainierten Models der Classifier besteht auch die Möglichkeit einen eigenen Text klassifizieren zu lassen. In Abbildung 7 wird anhand eins Beispiels aufgezeigt, wie es bei einem Korpus mit Labels aussehen könnte. Bei diesem Beispiel handelt es sich um eine Sentiment Analyse, hier versucht man die Haltung eines Textes zu analysieren, also ob es positiv oder negativ gemeint ist[2].

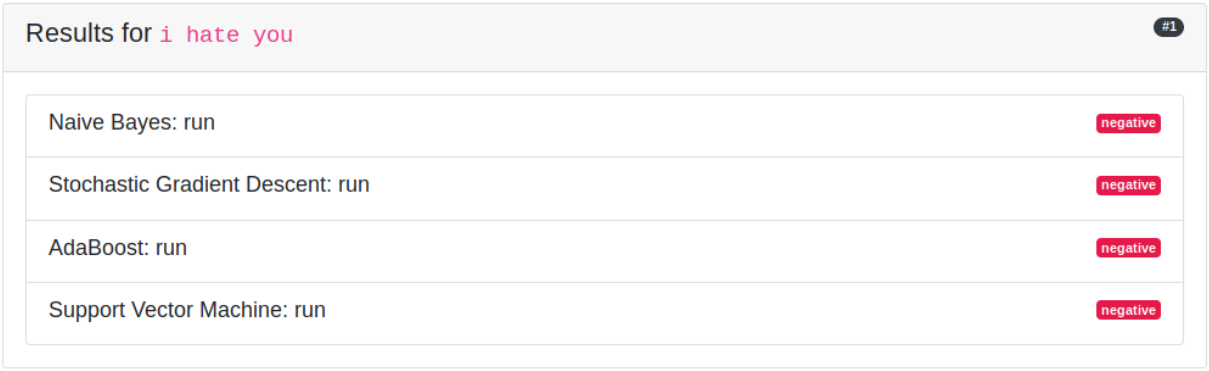


Abbildung 7 Ein Beispiel einer Evaluation mit dem Text als Input "i hate you".

1.2 Vergleich mit anderen Softwares

Im Folgenden wird analysiert, welche ähnliche Software zur Texploration existiert.

AntConc: Diese Software ist dazu gedacht Korpora zu analysieren. Das Tool ist zudem nicht erweiterbar und in sich geschlossen. Classifier oder sonstige Erweiterungen können somit nicht hinzugefügt werden. Zudem ist das User Interface nicht zeitgemäss aufgebaut. Folgende Funktionen werden gemäss der Webseite angeboten[3]:

- Concordance Tool: Zeigt Suchergebnisse im KWIC-Format (Keyword in Context) an.
- Concordance Plot Tool: Zeigt die Position eines Suchresultats in einem Diagramm an.
- Clusters/N-Grams: Visualisiert Cluster basierend auf Suchbedingungen.
- Word List: Zeigt alle Wörter sortiert nach ihrer Häufigkeit an.
- Keyword List: Die Funktion visualisiert, welche Wörter, im Vergleich zu einem Referenzkorpus, ungewöhnlich oft im Korpus vorkommen.

CorpusExplorer: Genau wie AntConc ist auch der CorpusExplorer nur für Corpus Analysen gedacht. Jedoch besitzt diese Software eine Schnittstelle für Erweiterungen durch .NET Sprachen. Somit könnte die Software um eine Classifier Funktion erweitert werden. Leider besitzt das Programm kein Benutzerverwaltung und es können nicht mehrere Personen gleichzeitig dran arbeiten. Folgende Funktionen werden gemäss der Webseite angeboten[4].

- Visualisierungen: Es werden 45 verschiedene Visualisierungen der Analyse angeboten, z.B: Frequenzanalyse, N-Gramme, Prasen, KWIC, DIFF, usw.

- Webcrawler: Funktion, um eigene Korpora zu erstellen.
- Annotationsansicht: Texte werden automatisch annotiert.
- Dokumentverteilung: Verteilung über Norm-Satz/Dokument
- N-Grams: Visualisierung N-Grams in Graphen
- Einflussfaktoren: Einfluss bestimmter Metadaten im Korpus
- Word Cloud

Google Cloud AutoML: Cloud AutoML ist eine Produktsuite für den Bereich maschinellem Lernen, die auch Entwicklern mit wenigen Vorkenntnissen erlaubt hochwertige Modelle zu trainieren. Die Software bietet vor allem Funktionen im Bereich Classification und Sentiment Analysis an. Abgesehen von den fehlenden Funktionen im Bereich Corpus Analysis, ist die Software auch kostenpflichtig[5].

Vereinzelt wurden auch Implementationen von Topic Modeling Algorithmen und Classifiern in Python-Scripts gefunden. Diese wurden hier jedoch nicht genauer betrachtet, da sie lediglich einen sehr kleinen Teil der Anforderungen abdecken.

Es existieren bereits Softwarelösungen, die diverse Aspekte von Texploration befriedigen. Eine Software, die alle Anforderungen vom Texploration deckt, konnte jedoch nicht gefunden werden.

1.3 Zielsetzung

Diese Arbeit verfolgt zwei Ziele, zum einen die Verbesserung des Quellcodes der Texploration und zum anderen die Erweiterung bestehender oder Implementation neuer Algorithmen. Im Genaueren werden im ersten Teil die Codequalität sowie die Architektur verbessert. Anschliessend sollten Erweiterungen im Bereich Language Detection und Topic Modeling durchgeführt werden. Die exakte Aufgabenstellung befindet sich im Anhang.

Das Kapitel 2 beschäftigt sich mit der Softwarearchitektur. Im Kapitel 2.1 wird das durchgeführte Refactoring thematisiert. Anschliessend wird ein Konzept für den Einsatz der Microservices in Texploration vorgestellt.

Das Kapitel 3 behandelt die Erweiterungen und Einführungen der Algorithmen in Texploration, wobei sich das Kapitel 3.1 mit dem Konzept und der Implementation der

Language Detection befasst. Im darauffolgenden Unterkapitel wird das Konzept und die Implementation des Topic Modeling behandelt.

Diese Arbeit richtet sich primär an Softwareentwickler und Personen mit solchen Kenntnissen.

2 Software Enginnering

Um langfristig die Software warten zu können, müssen schon von Beginn an wichtige Entscheidungen, in Bezug auf die Architektur, dem UI und anderen wichtigen Komponenten, getroffen werden.

2.1 Refactoring

Das Refactoring besteht aus zwei Teilen, zum einen aus der "Analyse des Codes" und zum anderen der "Ausführung des Refactorings".

2.1.1 Anfangszustand

Das Texploration besteht aus drei Bereichen: dem Frontend, der API und dem Backend. Während dem Refactoring liegt der Fokus hauptsächlich auf den letzteren zwei Bereichen. Aus diesem Grund wird in den folgenden Kapiteln nicht näher auf das Frontend eingegangen.

Die Softwarearchitektur sieht folgendermassen aus:

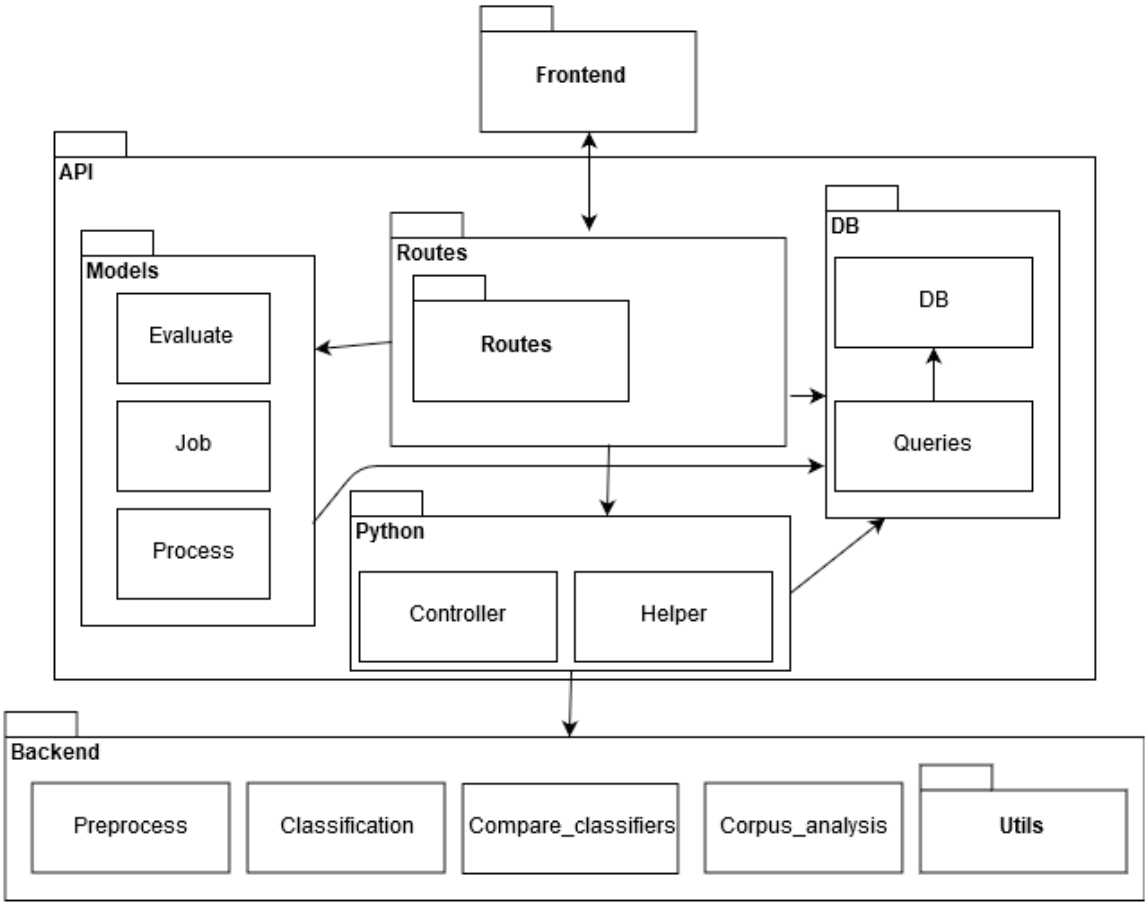


Abbildung 8 Architekturdiagramm im Anfangszustand

In der Abbildung ist zu beachten, dass die "Routes" auch die dazugehörige Business-Logik beinhalten.

Ein Teil der Datenbank-Queries befinden sich im gleichnamigen Modul "Queries". Der Rest ist über alle Komponenten des API-Bereiches verstreut.

Die Module im Backend-Bereich bestehen aus einzelnen Python-Scripts, die per Command-Line-Befehl von der Python Komponente des API-Bereiches aufgerufen werden.

Im Folgenden wird auf den aktuellen Stand der Codequalität eingegangen. Sowohl im API- als auch im Backend-Bereich existiert hierzu viel Optimierungsbedarf.

Eines der SOLID Prinzipien ist das "Single Responsibility Principle". Sie besagt, dass eine Klasse nicht mehr als einen Grund haben darf, um sie modifizieren zu wollen[6]. Im Projekt existieren jedoch Module oder Klassen, bei denen das nicht der Fall ist.

Gemäss Clean Code sollte jede Methode nur eine Aufgabe haben und sie sollte von keiner anderen Methode bereits ausgeführt werden[7]. Dieser Grundsatz wird oft gebrochen, was in der Texploration zu langen Funktionen sowie Codeduplikaten führt.

Ausserdem sollten die Variablen sowie Funktionsnamen beschreibend sein[7]. Dies ist oft nicht gegeben. Um den Ablauf mancher Module zu verstehen, reicht reines Lesen des Codes nicht mehr aus. Bei solchen Modulen ist Debugging nötig, um sie zu verstehen.

2.1.2 Gründe für Refactoring

Im Anfangszustand ist die Kopplung sowie die Kohäsion innerhalb des Backend- und API-Bereiches noch nicht auf dem gewünschten Niveau. Ausserdem ist die Codequalität in vielen Modulen optimierungsbedürftig. Dies erschwert die Erweiterung bestehender und die Einführung neuer Komponenten.

In der alten Version des Quellcodes würde ein Refactoring langzeitig einen grösseren Gewinn erbringen. Zusätzliche Features können anschliessend schneller und einfacher implementiert werden. Aus diesem Grund wurde entschieden ein Refactoring durchzuführen, bevor neue Features implementiert werden.

2.1.3 Zustand nach Refactoring

Nach dem Refactoring sieht die Softwarearchitektur folgendermassen aus.

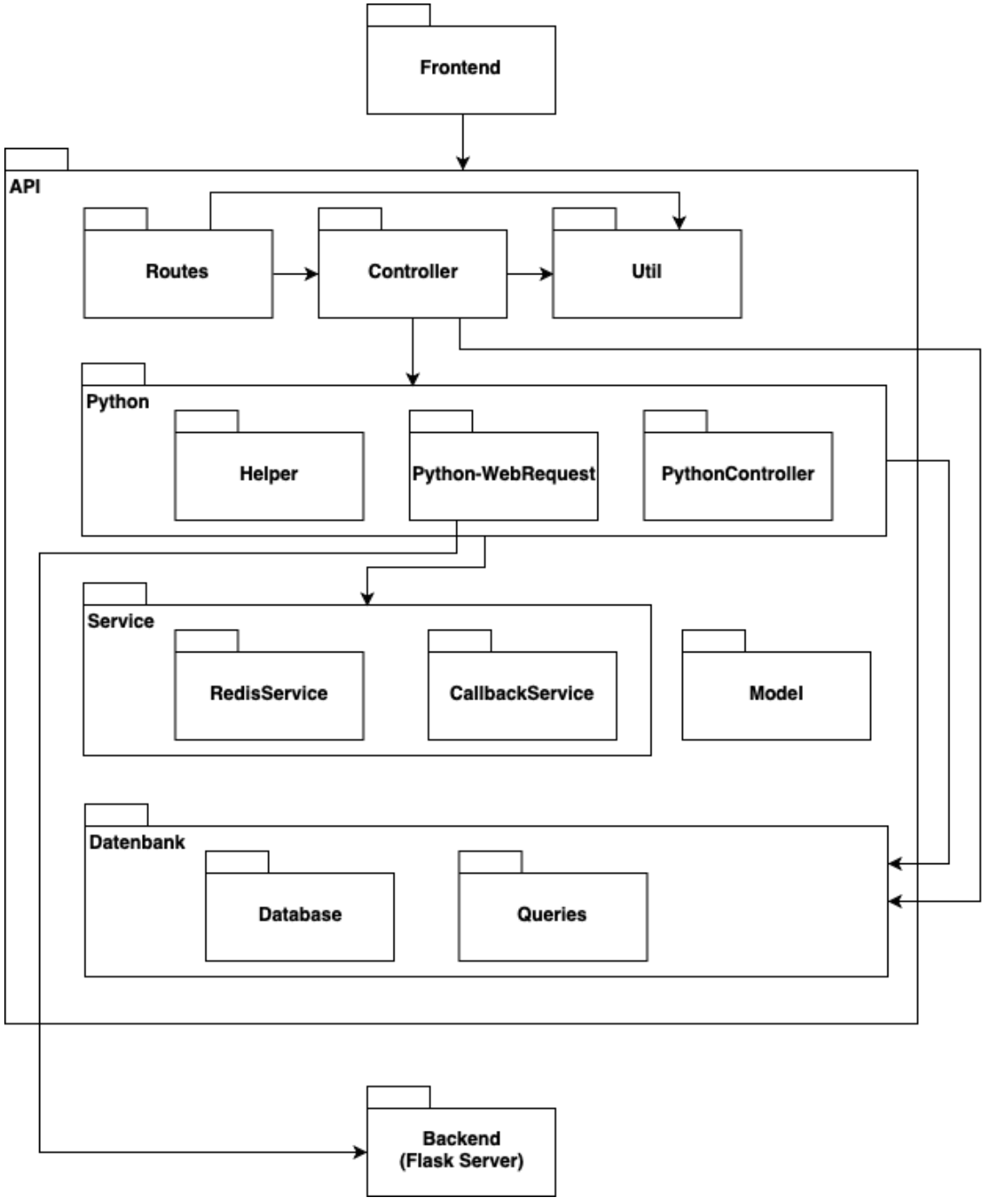


Abbildung 9 Softwarearchitektur der Texploration nach dem Refactoring

Wie aus der Abbildung 9 zu entnehmen ist, wurde die Business-Logik in die Controller ausgelagert. Auf diese Art kann die Logik besser wiederverwendet werden.

Um die Modularisierung weiterzutreiben, wurde die Logik des ursprünglichen Python-Controller in verschiedene Job Models verlagert.

Zwischen der API und dem Backend wurde eine Middleware in Form eines Python-Webservers (Flask) eingeführt. Alle Python-Jobs werden durch HTTP-Requests gestartet. Da der API-Teil nicht mehr für den Aufruf und Konfiguration der Python-Scripts zuständig ist, konnten so Teile entkoppelt werden. Ein weiterer Vorteil ist, dass die API und der Flask-Server theoretisch nicht auf demselben Server laufen müssen.

Es wurde auch darauf geachtet, dass die Codequalität verbessert wird. Diesbezüglich wurde die meiste Arbeit in den API-Bereich investiert. Im Genaueren wurde aus den Modulen mit zu vielen Verantwortlichkeiten, neue Module abgeleitet, damit die Wiederverwendbarkeit und Kohäsion besser werden. Funktionen, die zu lang sind oder zu viele Aufgaben haben, wurden in kleinere Funktionen aufgeteilt. Dabei wurden auch die Variablennamen verbessert. Zudem wurde versucht, Codeduplikate auf ein Minimum zu halten.

2.1.3.1 Backend

Das neue Backend unterscheidet sich erheblich von der alten Version. Es wurden Adapter eingeführt, damit Jobs über mehrere Prozesse (Multiprocessing) gleichzeitig angesteuert werden können. Das führt dazu, dass die Stabilität des Programms verbessert wird und Wartezeiten der HTTP-Request auf ein Minimum beschränkt werden können.

Jeder Job wird beim Start einem Adapter übergeben. Anschliessend wird der Adapter in einem eigenen Prozess gestartet. Auf diese Weise wird sichergestellt, dass der Flask-Server nach dem HTTP-Request nicht langzeitig blockiert wird. Beim Starten des Jobs wird eine Meldung an das Node.js übergeben, sodass die API beginnt periodisch den Status des Prozesses abzufragen und das UI (User Interface) aktualisiert. Wenn der Prozess mit den Berechnungen fertig ist, wird eine weitere Meldung an die API verschickt.

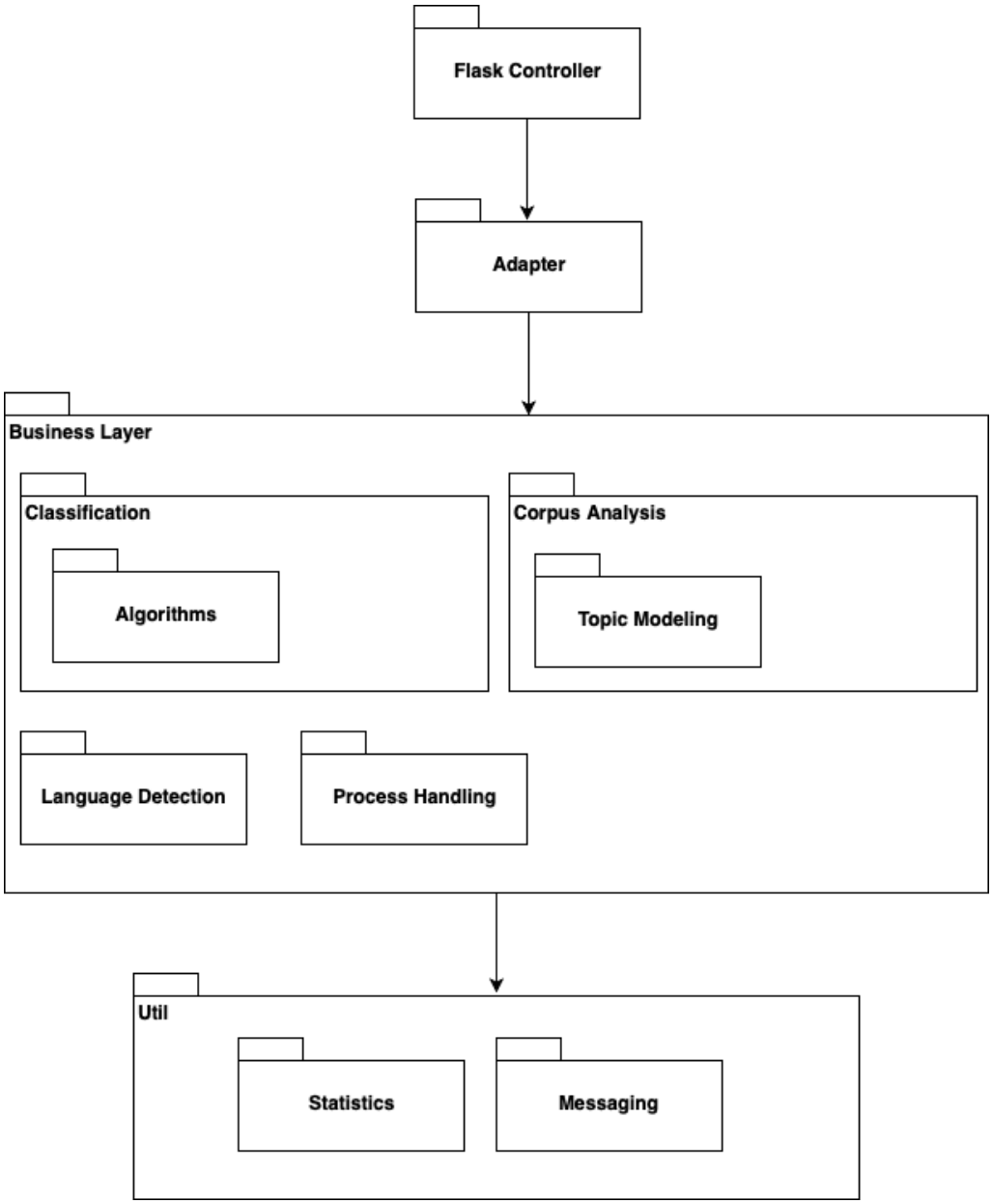


Abbildung 10 Architektur des neuen Python Backend

In der Texploration werden alle Anfragen an das Backend über einen Flask Controller gesteuert, dieser befindet sich im gleichnamigen Package zuoberst in der Abbildung 10.

Alle Berechnungen und Operationen des Business Layer werden über einen eigenen Adapter angesteuert. Nachdem der Adapter initialisiert und der Prozess gestartet wurde, wird das gewünschte Modul im Business Layer ausgeführt. Hier wurden

folgende vier Module identifiziert: Preprocessing, Classification, Corpus Analysis und das Process Handling.

Auf das Package "Process Handling" wird hier noch genauer eingegangen. Alle Prozesse, die während der Laufzeit erstellt werden, werden in einer Dictionary mit der UUID als Key abgespeichert. Wenn die Prozesse abgeschlossen sind oder ein Fehler aufgetreten ist, werden sie aus der Dictionary entfernt. Durch diese Lösung können alle Prozesse zentralisiert werden und das Handling der Prozesse gestaltet sich einfacher. Ausserdem vereinfacht diese Struktur die Einführung einer Job Queue.

Die Implementationen der Algorithmen befinden sich im Package "Algorithmen". Hier werden auch alle zukünftigen Algorithmen eingefügt, mit denen die Software später erweitert wird.

Das Modul Language Detection wurden in das Package Util verschoben, da es generisch konzipiert wurden und zukünftig auch von anderen Modulen eingesetzt werden könnte.

2.1.3.2 Kommunikation vom Backend zur API

Die API muss immer informiert werden, wenn ein Prozess in Python gestartet oder gestoppt wird. Dies erfolgte früher über das Node.js Modul "Child Process". Da man neuerdings im API-Bereich nicht mehr direkt auf die Prozessinformationen zugreifen kann, wurde diese Kommunikation auf eine Art Event Bus ausgelagert. Hierzu wurde ein Redis-Channel verwendet[8]. Wenn ein Prozess im Backend startet oder stoppt, wird eine Nachricht mit den benötigten Informationen über einen Redis-Kanal an die API verschickt.

2.1.4 Hinzufügen neuer Classifier

In den nachfolgenden zwei Unterkapitel wird thematisiert, welche Änderungen im Code durchgeführt werden müssen, um einen zusätzlichen Classifier-Algorithmus hinzuzufügen. Die Anleitung ist hauptsächlich für zukünftige Entwickler gedacht.

2.1.4.1 Erweiterungen im Flask Server

Im Pfad "flask/algorithms/classifier/" befinden sich die implementierten Classifier. Grundsätzlich steht es dem Entwickler frei, wie er die Algorithmen implementieren will, solange die folgenden Voraussetzungen erfüllt werden:

1. Der Klassenaufbau folgt der Vorlage in Abbildung 11. Es muss die Funktion train() implementiert werden.

```
1
2   class AlgorithmName:
3
4       def train(self, x_train, y_train, x_test):
5                   """
6                   Train algorithm
7
8                   Returns the y_predicted and the classifier
9                   """
10
11          return ''
```

Abbildung 11 Vorlage für einen Algorithmen-Adapter

2. In der Datei classification.py muss die Methode "train" um ein weiteres Codestück erweitert werden. In Abbildung 12 ist ein Beispiel für den AdaBoost Algorithmus zu sehen.

```
elif 'ADA' == config_data['classifier']:
    ada = aada.ADAB()
    y_predicted, classifier = ada.train(x_train, y_train, x_test)
```

Abbildung 12 Beispiel einer If-Abfrage für einen Algorithmus

2.1.4.2 Erweiterungen in der API

In der API müssen folgende zwei Änderungen durchgeführt werden.

1. Um das trainierte Model herunterladen zu können, muss in der Datei "api/src/api/download.js" ein weiterer if-Block gemäss dem Beispiel von Abbildung 13 hinzugefügt werden.

```
else if (req.params.name === 'Stochastic Gradient Descent')
  model = 'SGD';
```

Abbildung 13 Codeblock für den Export eines trainierten Models

2. Des Weiteren muss der Algorithmus in der Datenbank hinterlegt werden. Das entsprechende SQL Skript befindet sich im Pfad "api/migrations/005-algorithms.sql". Für den Algorithmus sollten die folgenden Informationen bereitgestellt werden:
   - eine ID
   - den Namen des Algorithmus

- eine kurze Beschreibung
- einen Link, der den Algorithmus genauer beschreibt
- ein Kürzel des Algorithmus.

```
INSERT INTO `algorithms`
    (`id`, `name`, `description`, `description_source`, `config_base`)
    VALUES
    (
        5,
        'Convolutional neural network',
        "In machine learning, a convolutional neural network (CNN, or ConvNet)",
        'https://en.wikipedia.org/wiki/Convolutional_neural_network',
        '{"classifier": "CNN"}'
    )
;
```

Abbildung 14 Beispiel eines SQL-Insert für den CNN-Algorithmen

Des Weiteren bedingt jeder Algorithmus einen weiteren Eintrag in der Tabelle "algorithm_configs".

```
INSERT INTO `algorithm_configs` (`algorithm_id`, `name`)
    VALUES (1, 'run');
```

Abbildung 15 Beispiel eines SQL-Insert für die Algorithmus-Konfiguration

2.1.5 Erweiterung Softwarearchitektur

Im Folgenden wird diskutiert, welche zusätzlichen Erweiterungen in der Zukunft noch ausgeführt werden könnten.

2.1.5.1 Einführung Job Queue

Prozesse, wie das Topic Modeling oder die Classification, können oftmals über eine längere Zeit die Ressourcen beanspruchen. Nichtsdestotrotz muss aber das System auch für andere Anfragen Ressourcen zur Verfügung stellen. Eine Lösung dafür wäre eine Job-Queue im Backend. Nachdem das Backend vom Frontend einen Job erhält, wird der Job in eine Queue eingefügt. Wenn das Backend wieder bereit ist, wird der eingefügte Job von der Queue herausgeholt. Da dann die Jobs nacheinander abgearbeitet werden, kann das System nicht überlastet werden.

Redis ist eine bekannte Software, welche eine solche Funktion anbietet. Sie wird bereits in der Texploration verwendet und wäre somit einsatzbereit. Des Weiteren ist

Redis programmiersprachenunabhängig und ermöglicht somit eine problemlose Kommunikation zwischen dem Node.js Server und Python im Backend.

2.1.5.2 Einführung von Adaptern

Damit der Umgang mit mehreren Prozessen und allfälligen Fehlern bei den Algorithmen besser abläuft, wurde in der überarbeiteten Version der Texploration das Adapter-Pattern eingeführt. Die Adapter können noch erweitert werden, indem sie nach Algorithmentypen unterteilt werden. Momentan ist jede Implementation eines Algorithmus für die Formatierung des Resultats verantwortlich. Es wäre vorteilhaft, wenn die Aufgabe von einem Adapter übernommen werden würde.

Wie in der Abbildung 16 zu entnehmen ist, werden hier zwischen verschiedenen Adapter unterschieden: Classifier, Evaluation der Classifier, Unsupervised Algorithmen und WordRanker. Jede neue Implementation eines Algorithmus würde eines der in Abbildung 16 abgebildeten Interfaces implementieren.

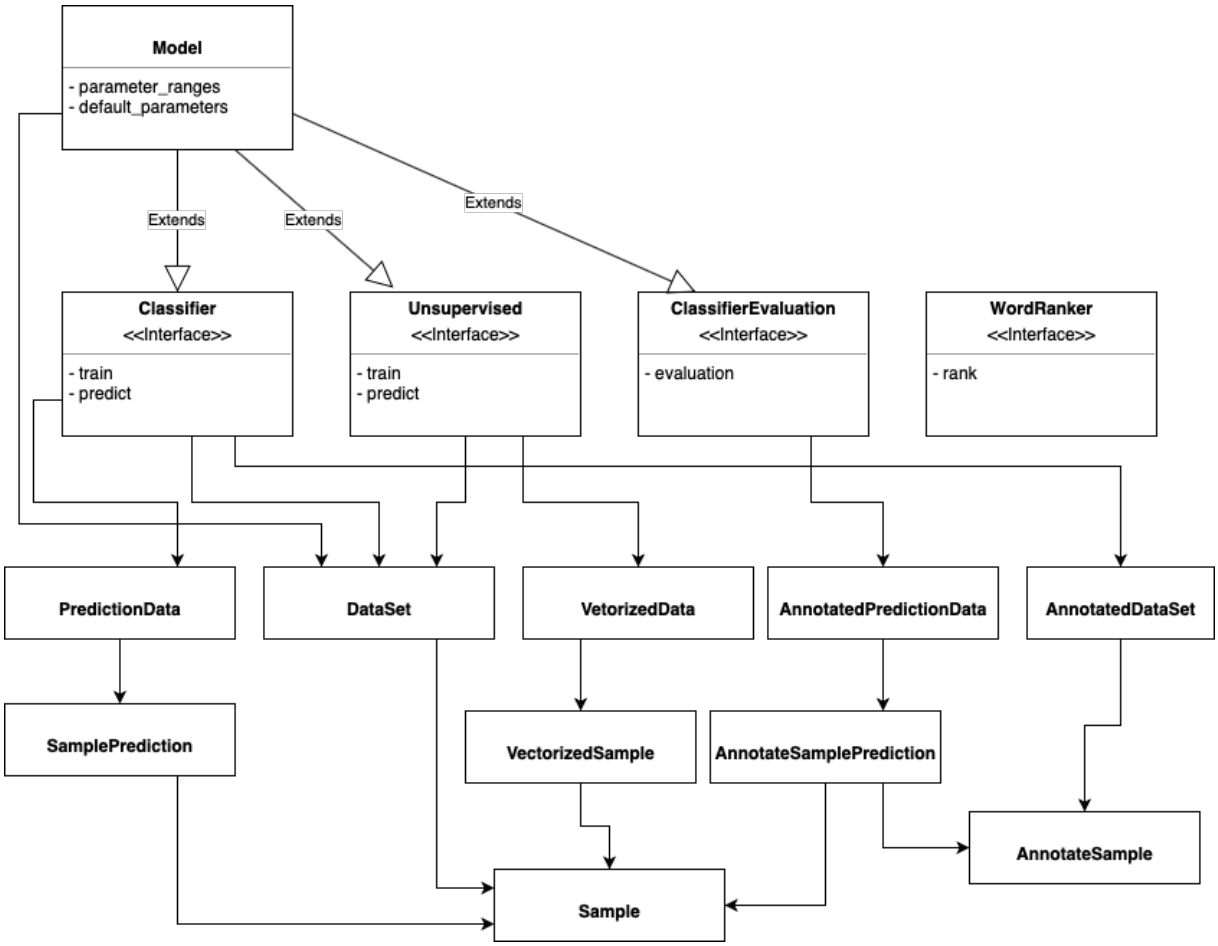


Abbildung 16 Minimal gehaltenes Klassendiagramm der Adapter-Struktur

Das Interface "Classifier" beinhaltet die zwei Methoden train() und predict(). Die Evaluation eines Classifiers würde in einem separaten Adapter implementiert werden. In Abbildung 16 wäre das der "ClassifierEvaluation" Adapter. Das Evaluation Interface würde nur die gleichnamige Methode "evaluation" beinhalten.

Das nächste Interface "Unsupervised" würde, wie der Classifier, nur die Methoden train() und predict() beinhalten.

Der untere Teil der Abbildung 16 besteht aus Software-Models, die in den Adaptern eingesetzt werden.

2.2 Aktuelle Datahandling

In den folgenden zwei Kapiteln wird analysiert, welche Daten der Texploration an welchen Speicherorten ablegt werden. Während dem Refactoring wurden hier keine Änderungen gemacht.

2.2.1 Datenbank

Ein Teil der Daten wird in der SQLite3[9] Datenbank gespeichert, welche als Datei im Pfad "api/db/database.sqlite" abgelegt ist. Darauf kann nur die API zugreifen. Das dazugehörige Datenbankschema befindet sich im Kapitel 6.2.

Folgend wird auf die relevanten Komponenten der Datenbank eingegangen.

In der Relation "algorithms" werden die Informationen zu den Classifier-Algorithmen gespeichert. Dazu gehören der Name, eine Beschreibung sowie die Konfiguration des Algorithmus.

Die Metadaten der hochgeladenen Dateien werden in der Relation "files" gespeichert. Im Genaueren wird für jede Datei eine eindeutige UUID generiert, zudem wird der Pfad zur Originaldatei sowie der Pfad zur geparsten Version der Datei gespeichert.

Eine Corpus Analyse entspricht einem Eintrag in der "Project" Relation. Jede Analyse besitzt eine UUID. Ausserdem werden zu jedem Projekt die Konfiguration sowie der Pfad zur Output-Datei gespeichert. Der Pfad zur Output-Datei befindet sich zweifach in der Tabelle, zum einen als absoluter Pfad in der Spalte "filepath_corpus_analysis" und zum anderen als relativer Pfad in der Spalte "uuid"

Beim Starten einer Classification wird jeweils ein Eintrag in der Relation "job" erstellt. Jeder Job erhält eine eindeutige UUID. Ausserdem werden die verwendeten Dateien, der F1-Score des Baseline Algorithmus und die Metainformationen (Start- und Endzeit) in dieser Relation gespeichert. Unter anderem steht hier auch ein Verweis auf den besten Classifier.

Zu jedem Classifier wird ein Eintrag in der Relation "processes" erstellt. Darüber hinaus stehen dort die genauen Informationen zu jedem ausgeführten Classifier, demzufolge wären das der F1-Scores, die Confusion Matrix, der Classification Report, die Laufzeit usw.

2.2.2 Dateisystem

Der restliche Teil der Daten wird direkt auf dem Dateisystem, im Pfad "api/storage/", abgelegt. Keines dieser Dateien besitzt eine Dateinamenerweiterung. Auf diese Daten greifen sowohl die API als auch das Python-Backend zu. Diese Daten werden auch zum Teil direkt an das Frontend geschickt.

Im Weiteren wird auf die Struktur des "storage" Ordner genauer eingegangen.

2.2.2.1 Ordner "classification"

In diesem Ordner werden alle Dateien der Classification abgespeichert. Im Hauptverzeichnis befindet sich pro Job ein Ordner mit dessen UUID als Namen.

Pro verwendeten Classifier werden die folgenden Dateien generiert:

Tabelle 1 Dateien, die pro Classifier generiert werden

Dateiname	**Beispiel Dateiname**	**Inhalt**
"processId"	614	Die JSON-Datei enthält den Classifier-Report, die Confusion Matrix und den F1-Score.
"processId_csv"	614_csv	Enthält die verwendeten Testdaten sowie das vom Classifier zugeteilte Label.

Ausserdem werden die JSON-Datei "base" und die verschiedenen "comparison" Dateien der "Classifier Comparison" in diesem Ordner gespeichert. Die base Datei enthält lediglich den F1-Score des Baseline-Algorithmus.

2.2.2.2 Ordner "corpus_analysis"

Im Ordner "corpus_analysis" werden die Ergebnisse zum gleichnamigen Modul gespeichert. Hier befindet sich pro Analyse eine Datei, die jeweils die UUID des Projektes als Dateinamen hat. Die JSON-Datei enthält alle Informationen, die im UI bei der Corpus Analysis angezeigt werden (zum Beispiel Tokens, Topic Modeling, Labels, etc.)

2.2.2.3 Ordner "evaluation"

Die Daten der Classifier-Evaluation werden im Ordner "evaluation" gespeichert. Jeder gestartete Prozess besitzt einen eigenen Ordner, mit der UUID des Jobs als Namen.

Für jeden Classifier werden die folgenden Dateien gespeichert:

Tabelle 2 Dateien, die für jeden Classifier erstellt werden

Dateiname	**Beispiel**	**Inhalt**
"ProcessId"	614	Die JSON-Datei enthält das Label, welches der Classifier dem Input zugewiesen hat.
"ProcessId.done"	614.done	Diese Datei hat keinen Inhalt und wird nach der Fertigstellung der Evaluation eines Classifiers erstellt. Nachdem die API die Evaluation gestartet hat, überprüft sie periodisch das Vorhandensein dieser Datei. Sollte sie existieren, weiss die API, dass der Classifier fertig ist.

2.2.2.4 Ordner "language_detection"

Das neu eingebaute Modul "Language Detection" speichert die Dateien in diesem Ordner. Die Ergebnisse dieser Analyse stehen jeweils in einzelnen Dateien, die den Namen des Projekt UUID tragen.

2.2.2.5 Ordner "models"

Die Models der Classifier werden als Pickels in diesem Ordner gespeichert. Auch sie besitzen die Job UUID als Namen. Ausserdem befinden sich hier die ZIP-Dateien der Models, die der Benutzer von der Webseite herunterladen kann.

2.2.2.6 Ordner "output"

Auf dem UI wird bei der Classification die Möglichkeit angeboten, den "Raw Output" der Classifier zu sehen. Im Ordner "output" liegen die Dateien, die diesen Output beinhalten. Der Output ist die Ausgabe der implementierten Libraries, welche normalerweise in der Konsole angezeigt werden würden. Der Name der Dateien wird aus der UUID sowie der ProcessId zusammengesetzt.

2.2.2.7 Ordner "preprocess"

Dieser Ordner wird beim Starten des Node.js-Servers erstellt, aber nie verwendet.

2.2.2.8 Ordner "uploads"

Die hochgeladenen Dateien werden in diesem Ordner gespeichert. Hier ist der Dateiname die UUID der Files-Relation.

2.2.2.9 Ordner "parsed"

Nachdem eine hochgeladene Datei geparst wurde, wird eine Kopie im Ordner "parsed" abgelegt. Der Name der Datei bleibt aber derselbe. Die Originaldatei bleibt unverändert im Ordner "uploads".

2.3 Konzept für die Microservices

Das Microservice Konzept ist eine junge Softwarearchitektur, die in den letzten Jahren vor allem bei Grosssystemen eingesetzt wurde.

2.3.1 Motivation zur Einführung von Microservices

Langfristig soll Texploration Wissenschaftlern zu Verfügung stehen, ihre Algorithmen zu testen und zu vergleichen. Dabei werden oftmals Algorithmen ausgeführt, die viel Rechenleistung beanspruchen. Microservices erlauben dem Entwickler jeden Service individuell, sowohl horizontal als auch vertikal, zu skalieren[10].

2.3.2 Funktionsweise von Microservices

Microservices sind eine Softwarearchitektur, welche stark von der serviceorientierten Architektur (SOA) beeinflusst wurde. Die Idee dahinter ist eine grosse monolithische Software in kleine Module aufzuteilen. Alle Module arbeiten selbstständig, bilden aber als Ganzes eine einheitliche Software. Zudem können sie auf dem gleichen oder jeweils einzeln auf unterschiedlichen physikalischen Servern ausgeführt werden. Die Kommunikation zwischen den einzelnen Modulen erfolgt meist über HTTP-Requests oder Messaging-Systemen. Der signifikante Unterschied von Microservices zur serviceorientierten Architektur ist, dass die Microservice-Architektur kein "Deployment-Monolyth" ist und jeder Service somit separat deployed werden kann. [11]

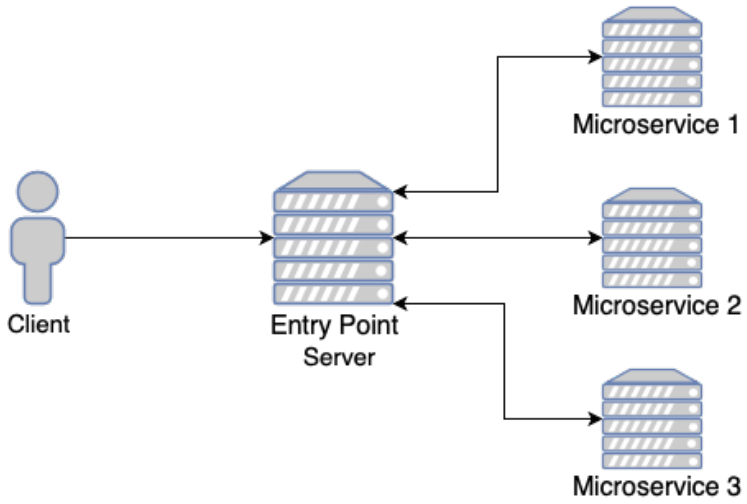


Abbildung 17 Einfacher Aufbau eines Microservice-Systems

Wie in Abbildung 17 aufgeführt, wird vom Client ein HTTP-Request an den Webserver verschickt. Dieser Webserver dient gleichzeitig auch als Entry Point für die Applikation. Anschliessend wird die Anfrage direkt an den zuständigen Microservice weitergeleitet. Dieser bearbeitet die Anfrage und retourniert dem Entry Point das Resultat. [11]

2.3.3 Patterns

Hier wird genauer darauf eingegangen, welche Patterns eingesetzt werden können, um den neuen Anforderungen dieser Technologie gerecht zu werden.

2.3.3.1 API-Gateway Pattern

Das API-Gateway fungiert als eine Art "Entry Point" zur API. Des Weiteren ist es die einzige Anlaufstelle für den Client und dient somit als Interface für die Software. Alle anderen Services sind für den Client nicht sichtbar. Ein grosser Nachteil eines API-Gateways ist, dass hier am ehesten ein Bottleneck entstehen kann, denn die ganze Client-Server-Kommunikation läuft über diesen Service. [11]

2.3.3.2 Service Discovery Patterns

Hierbei werden zwei verschiedene Arten von Pattern unterschieden. Zum einen existiert das "Client-side Discovery Pattern", bei dem jeder Service die Anfragen selbst durchführt, und zum anderen das "Server-side Discovery Pattern", welches über ein Forwarding-Konzept funktioniert. Bei beiden Patterns wird zudem noch ein weiterer Service, der "Discovery Server", hinzugeschaltet.

2.3.3.2.1 Discovery Server

Der Service Discovery Server führt eine "Service Registry". Die Registry beinhaltet die Kontaktangaben aller Services und funktioniert wie ein Telefonbuch. Sollte beispielsweise ein Service einen anderen kontaktieren wollen, würde er mithilfe des Discovery Servers an seine Kontaktinformationen gelangen. Der grosse Vorteil eines Discovery Servers ist die Zentralisierung der Kontaktinformationen. Das führt dazu, dass Änderungen oder neue Einträge nur hier angepasst werden müssen. Während der Laufzeit kann jedoch dieses Vorgehen auch zum Problem werden. Der Discovery Server läuft Gefahr zum Bottleneck zu werden, da jede Kommunikation von ihm verwaltet wird. [11]

2.3.3.2.2 Client-side Discovery Pattern

Das Client-side Pattern verfolgt die Idee, dass Services direkt untereinander kommunizieren. In der vereinfachten Abbildung 18 ist das Vorgehen genauer zu erkennen. Service1 kontaktiert zuerst den Discovery Server, welcher wiederum in seiner internen Datenbank (der Service Registry) die Informationen des Service2 sucht. Falls ein Eintrag in der Datenbank gefunden wurde, schickt der Server dem Service1 die Kontaktinformation zurück. Der Service1 führt anschliessend die eigentliche Abfrage aus und kontaktiert den Service2 über einen HTTP-Request. [11]

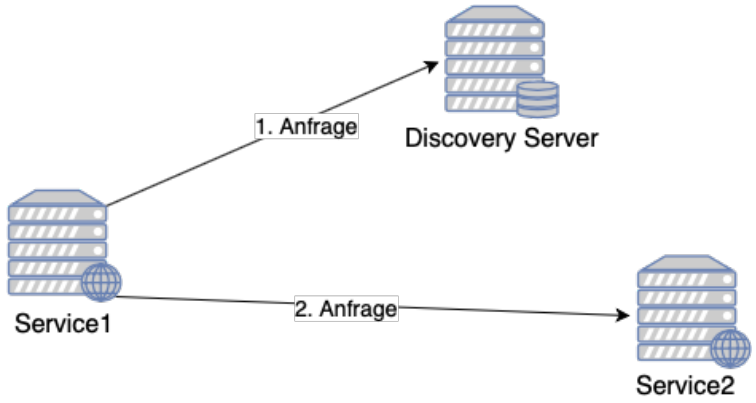


Abbildung 18 Konzeptioneller Aufbau des Client-side Discovery Patterns

Für das Entwicklerteam bietet dieses Vorgehen einen grossen Vorteil, denn es ist trivial zu verstehen und die Implementation ist unkompliziert. Für den Service1 wirkt sich das Pattern nachteilig aus, da es für zwei HTTP-Requests verantwortlich ist.

2.3.3.2.3 Server-side Discovery Pattern

Beim serverseitigen Pattern existiert ausserdem ein weiterer Service, der Load Balancer. Bei grösseren Systemen laufen Services oftmals redundant, damit hohe Auslastungen besser abgefangen werden können. Im Client-side Pattern hätte der anfragende Service selbst bestimmen müssen, welche der Instanzen er anfragen soll. Hier wird das vom Load Balance geregelt. [11]

Wie in Abbildung 19 aufgezeigt, kontaktiert der Service1 erst den Load Balancer, welcher in der Service Registry überprüft, ob der gesuchte Service existiert. Die Service Registry kann sich entweder im Discovery Server oder im Load Balancer befinden. Wenn der gesuchte Eintrag gefunden wurde, bestimmt der Load Balancer zu welcher Instanz die Anfrage weitergeleitet werden soll.

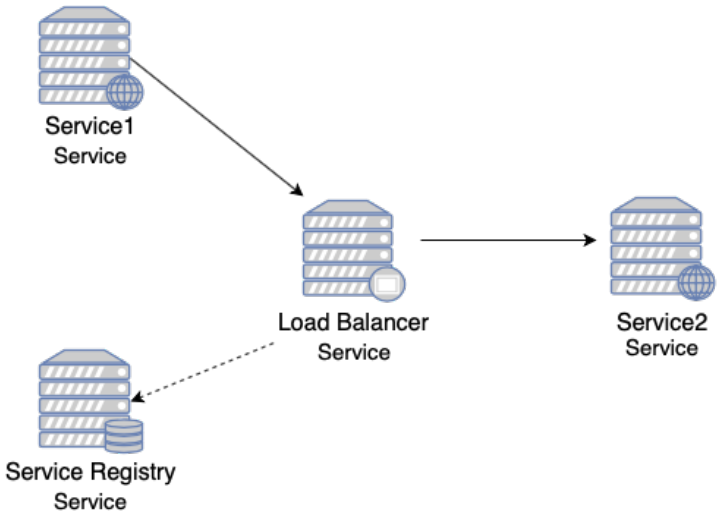


Abbildung 19 Konzeptioneller Aufbau des Server-side Discovery Patterns

Auch dieses Pattern bietet diverse Vor- und Nachteile. Die sicherlich grössten Vorteile sind die erhöhte Wartbarkeit und Skalierbarkeit bei der Einführung von redundanten Instanzen. Dies führt aber auch dazu, dass die Komplexität des Systems (sowie die des Service Registry) steigt. Der Load Balancer ist hier stark gefährdet ein Bottleneck zu werden. [11]

2.3.3.3 Data Storage Pattern

Das Basisprinzip bei Microservices ist, dass jeder Service selber seine eigenen Daten verwaltet. Zwei Services sollten sich nicht einen Datenspeicher teilen müssen.[12] Trotz dessen existieren viele Patterns für die Datenspeicherung, die sich vor allem im

Speicherort (Zentralisiert oder pro Service abspeichern) und im Speicherprinzip (Datenbank, Cluster, etc.) unterscheiden.

Folgend werden lediglich zwei Patterns angeschaut, die sich im Speicherort unterscheiden. Beide Patterns werden später im Microservice-Konzept (Kapitel 2.3.6.4) eingesetzt.

2.3.3.3.1 Shared Database Server

Beim Shared Database Server Pattern wird ein zusätzlicher Service ins System hinzugefügt. In diesem Service befindet sich eine Datenbank Software, z.B. eine NoSQL oder Spark Datenbank. Jeder Service kontaktiert eigenständig den Datenbank Server und fragt seine Daten ab. Der Einsatz von Spark oder NoSQL ermöglicht zudem eine einfache Handhabung von Objekten, die normalerweise nicht in eine relationale Datenbank passen würden. [11]

Abbildung 20 Konzeptioneller Aufbau des Shared Database Server Pattern

Beim Einsatz eines Service Discover Patterns würde zwar ein Load Balancer oder ein Discovery Server hinzugeschaltet werden, aber aus Gründen der Einfachheit wurden sie nicht in der Abbildung 20 dargestellt.

2.3.3.3.2 Database-per-Service Pattern

Dieses Pattern folgt dem Grundprinzip der Microservices, jeder Service ist für seine eigenen Daten verantwortlich. Meistens wird dafür pro Service eine eigene Datenbank aufgesetzt.

2.3.3.4 Multi/Single Service per Host Pattern

Services oder dessen Instanzen können bei Microservice-Systemen parallel auf einem physikalischen Host laufen. Dabei muss auf die Verteilung der Ressourcen geachtet werden, wenn ein Service alle Ressourcen beanspruchen würde, droht den anderen Services ein Starvation-Problem. [11]

2.3.3.5 UI Patterns

Generell unterscheidet man zwischen dem "Monolithic UI" und dem "Composite UI" Pattern.

2.3.3.5.1 Monolithic UI

Das einfachere der beiden Patterns ist das Monolithic UI. Dabei besteht das UI aus einem einzigen, eigenständigen Teil. Dieser greift auf die Microservices über den API-Gateway zu. Der wesentliche Nachteil bei diesem Pattern ist, dass die Seite monolithisch aufgebaut ist. Bei jeder Änderung im UI, muss die ganze Seite deployed werden.[13]

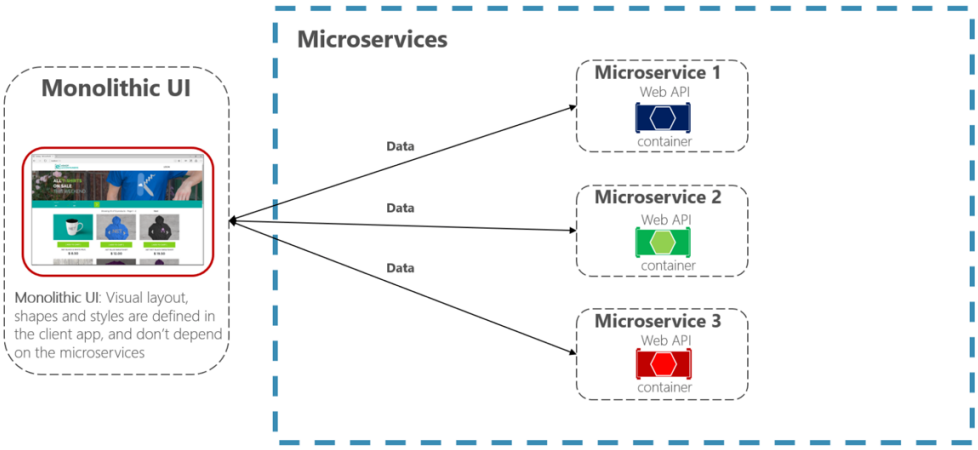


Abbildung 21 Monolithic UI Pattern[13]

2.3.3.5.2 Composite UI

Wie in Abbildung 22 dargestellt, funktioniert jede der UI Composition Microservices als eigenständiger API Gateway und ist nur für einen bestimmten Teil im UI verantwortlich[13].

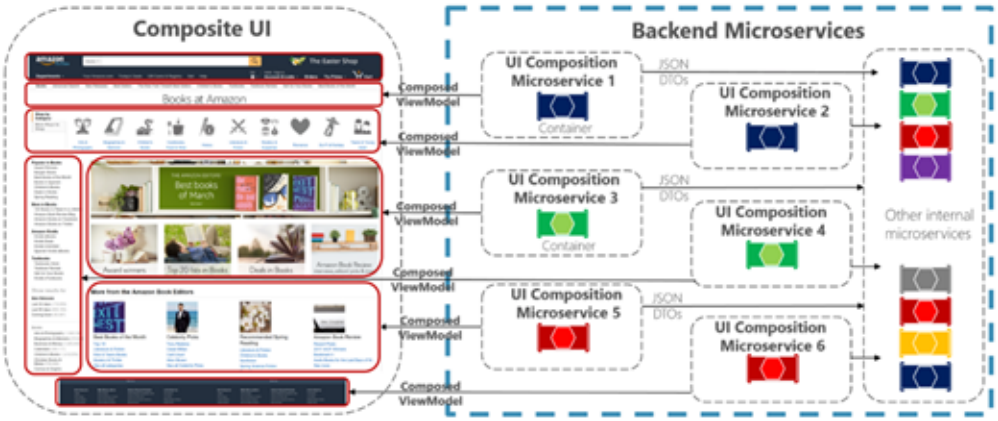


Abbildung 22 Composite UI Pattern[13]

2.3.4 Vorteile von Microservices

Hier wurden lediglich die drei wichtigsten Vorteile der Microservices gemäss Fowler aufgezählt.

- Unabhängigkeiten: Microservice können unabhängig von anderen Systemkomponenten entwickelt werden. Das führt dazu, dass die Skalierung von Entwicklungsprozessen ohne grossen Kommunikationsaufwand durchgeführt werden kann. [14]
- Deployment: Microservice-Systeme können gegen Ausfall robust gemacht werden. Die Architektur lässt zu, dass wenn ein Modul ausfallen würde, die anderen normal weiter funktionieren. [14]
- Technologien: Da die Kommunikation über HTTP-Request erfolgt, sind die einzelnen Microservices nicht an spezifische Programmiersprachen gebunden. [14]

2.3.5 Nachteile von Microservices

Im Folgenden wurden die vier Nachteile mit dem grössten Einfluss aufgeführt.

- Distribution: Bei der Entwicklung muss mehr Aufwand in die Robustheit investiert werden, da HTTP-Requests zum einen fehleranfälliger und zum anderen auch langsamer als interne Programm-Calls sind. [14]

- Konsistenz: Distributed Systems sind in Bezug auf die Konsistenz der Daten sehr schwierig zu warten. Im schlimmsten Fall müssen alle Services selbst auf die Konsistenz der Daten achten. [14]
- Komplexität: Es braucht ein erfahrenes Team, um eine grosse Anzahl an Services zu warten. Vor allem die Organisation des Deployment, das Logging und das Monitoring gestalten sich komplex. Nicht zu unterschätzen ist auch das Thema Lastenverteilung, sehr gefragte Services benötigen mehr Ressourcen als solche die nicht oft gebraucht werden. [14]
- Migration: Bei einer bereits bestehenden Software, ist der Aufwand oftmals enorm, um dieses in ein Microservice-System zu migrieren. [6]

2.3.6 Einsatz von Microservices in Texploration

Texploration ist bereits modular aufgebaut und bietet somit die besten Vorbedingungen um es als Microservice-System zu implementieren. Der Einsatz von Microservices bringt diverse Vorteile mit sich:

- Betriebssicherheit: Bei grossen Datensätzen dauern die Prozesse oftmals mehrere Stunden, wenn nicht gleich Tage. Bei einem Absturz eines Service wären die anderen Services davon nicht betroffen und Prozesse könnten normal weitergeführt bzw. zu Ende gebracht werden.
- Entwicklung: Texploration hat das Ziel eine Open-Source Software zu werden. Dafür wird ein einfacher und klar strukturierter Aufbau erwartet. Microservices sind von Natur aus eigenständige Programmteile mit einer sehr geringen Kopplung. Des Weiteren haben Microservices den grossen Vorteil, dass die Entwicklung einfach skaliert werden kann, da die Services voneinander unabhängig sind.

Wie in Abbildung 23 zu entnehmen ist, wird die Software in vier fachbezogene und in drei technische Services unterteilt.

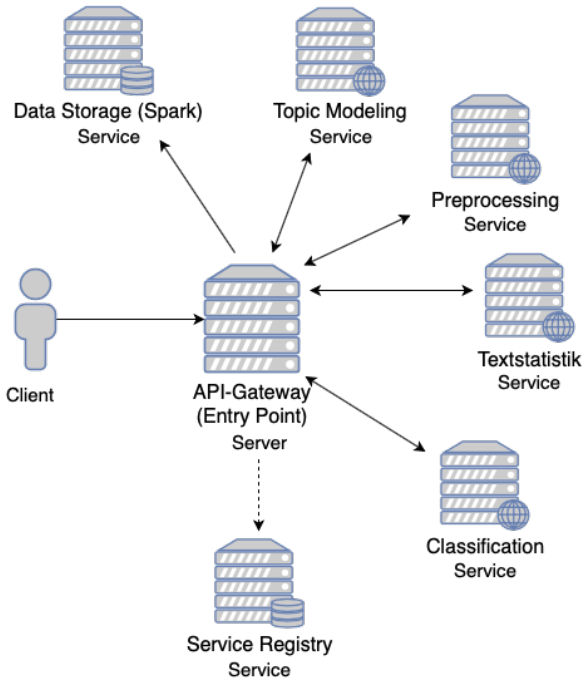


Abbildung 23 Konzept: Einführung von Microservices in Texploration

2.3.6.1 Identifikation der fachbezogenen Microservices

Um die Microservices genau zuschneiden zu können, wurde ein vereinfachtes Domain Model (s. Abbildung 24) erstellt und die zusammengehörigen Teilbäume zu einem Service zusammengefügt.

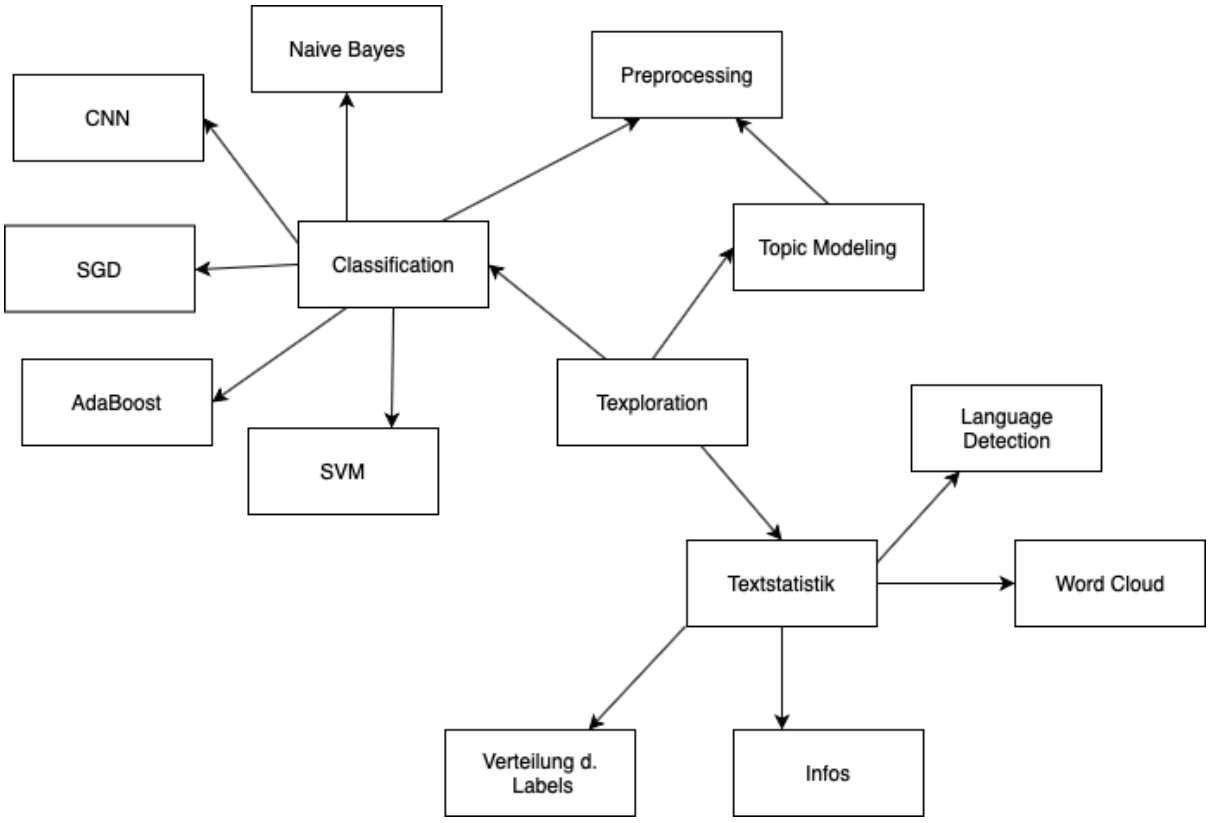


Abbildung 24 Vereinfachtes Domain Model von Texploration

Aus dem Diagramm lassen sich folgende vier, voneinander unabhängige, Microservices identifizieren.

Tabelle 3 Beschreibung der identifizierten Microservices

Servicename	**Beschreibung**
Preprocessing	Im Preprocessing wird der Text hauptsächlich normalisiert. Das beinhaltet vor allem die Funktionen, löschen von URL-Pfaden, das Entfernen von @-Zuweisungen und Tags. Aber auch der Tokenizer wird im Preprocessing ausgeführt. Zudem ist das Preprocessing der einzige Microservice, der die Daten wieder zurück in die Datenbank schreibt und nicht in einem lokalen Speicher behält. Das hat den Vorteil, dass die anderen Service die vorbereiteten Daten direkt bei der Datenbank abfragen können.

Topic Modeling	Der Service Topic Modeling ist nur für das Topic Modeling zuständig, jedoch nicht für dessen Preprocessing o.ä.
Textstatistik	In diesem Service werden, abgesehen vom Topic Modeling, alle Funktionen der Corpus Analyse durchgeführt. Das beinhaltet die generelle Statistik über dem Dataset, die Verteilung der Labels, die Word Cloud und die Language Detection
Classification	Der Service "Classification" ist für die Ausführung der implementierten Classifier zuständig. Des Weiteren wird hier auch die Evaluation ausgeführt.

2.3.6.2 Technische Services

Neben den fachbezogenen Services gibt es noch die technischen Services, welche für den Betrieb notwendig sind. Folgende Services konnten diesbezüglich identifiziert werden.

Tabelle 4 Beschreibung der einzusetzenden Services

Servicename	Beschreibung
Service Registry	In der Registry befinden sich, wie im Kapitel 2.3.3.2.1 beschrieben, alle Kontaktinformationen der Services.
Data Storage	Im Service "Data Storage" wird eine zentralisierte Datenbank aller Rohdaten und den geparsten Datensätzen geführt.
API-Gateway	Das Gateway wird alle Requests entgegennehmen und zu den entsprechenden Services weiterleiten.

In diesem Konzept wird der Load Balancer nicht berücksichtigt, da die Software klein ist und es von jedem Service zu Beginn lediglich eine Instanz geben wird. Des Weiteren würde ein Load Balancer sehr viel Komplexität mit sich bringen und man müsste schon vorab das Benutzerverhalten kennen. Nach Bedarf kann ein Load Balancer immer noch nachträglich eingeführt werden.

2.3.6.3 Risiken bei einem Umbau

Der grösste Kostenfaktor bei der Implementierung eines neuen Backends ist die Modularisierung der Python Klassen. In der aktuellen Version sind die meisten Module

stark gekoppelt, daher kann der Code nicht ohne grossen Aufwand in selbständige Services aufgeteilt werden.

2.3.6.4 Datenbank

Langfristig muss Texploration in der Lage sein, grosse Datenmengen zu verwalten. Aus diesem Grund ist es sinnvoll bereits von Anfang an mit Spark zu arbeiten. Spark ist ein Framework, welches in der Lage ist, grosse Datenmengen zu handeln[15]. Zudem bietet Spark eine unkomplizierte Art und Weise an, um Objekte zu speichern. Im Kontext der Texploration wären das die Input-Files und die trainierten Modelle, welche dann Versionsweise abgespeichert werden würden.

In der Texploration wäre ein Mix der beiden Patterns (s. Absatz 2.3.3.3) zu erwägen. Die Rohdaten, welche der Benutzer hochlädt, und die bearbeiteten Rohdaten nach dem Preprocessing würden zentralisiert im Data Storage Service abgelegt werden. Wohingegen die trainierten Modelle aus den Services Topic Modeling und Classification intern in den Services gespeichert werden würden. Mit diesem Prinzip kann man die Services weiter voneinander entkoppeln, da jeder Service nur für seine eigenen Models zuständig ist. Des Weiteren würde auch der Data Storage Service entlastet werden, da der Datenverkehr verkleinert werden würde.

2.3.6.5 User Interface

Das UI besteht im Texploration aus verschiedenen kleinen Teilen. Aus diesem Grund wurde im Konzept das Composite UI Pattern ausgewählt. Es konnten folgende Bereiche im UI identifiziert werden:

- Analyse einer Datei: Präsentiert eine Statistik des Inhalts sowie die Label Distribution und die Instance Distribution.
- Word Cloud: Grafische Ansicht der am meisten vorkommenden Wörter
- Tokenizer-Vergleich: Grafischer Vergleich unterschiedlicher Tokenizer
- Classifier: Anzeigen des F1-Scores und Resultat der Classifier

3 Algorithmen

Im Texploration wurden diverse Algorithmen für verschiedenste Aufgaben eingesetzt. Um dem Benutzer zukünftig einen möglichst reibungslosen Einsatz zu gewährleisten, wurde die Language Detection und das Topic Modeling analysiert und auf die Bedürfnisse von Texploration optimiert.

3.1 Language Detection

Die Language Detection hat das Ziel, die Sprache eines Textes zu erkennen. Eine solche Komponente ist für die Texploration sehr interessant, denn aktuell hat der Benutzer keine Möglichkeit zu erkennen, ob sein Korpus mehrsprachig ist.

3.1.1 Implementation im Backend

Im Folgenden wird betrachtet, wie die Language Detection in der Texploration im Backend implementiert wurde.

3.1.1.1 Sprachenerkennung

Für die Erkennung der Sprache wird das Framework "langid.py" verwendet. Mit Hilfe eines Naive Bayes-Classifier erkennt es 97 verschiedene Sprachen [16]. Die Trainingsdaten stammen aus den folgenden Textsammlungen: JRC-Acquis, ClueWeb 09, Wikipedia, Reuters RCV2 und Debian i18n. [17]

Langid.py untersteht der BSD Lizenz. Dadurch ist dieses Framework für den Einsatz in kommerziellen und Open-Source Projekten geeignet. [18]

3.1.1.2 Interface der Businesslogik

Im neuen Language Detection Modul werden zwei Funktionen angeboten. Zum einen eine Methode, die alle vorkommenden Sprachen auflistet, und zum anderen eine, die nur die meistverwendete Sprache zurückgibt.

Der Rückgabewert ist eine Liste, die jeweils pro Sprache den ISO-normierten String der Sprache (ISO 639-1) und einem Float für die Wahrscheinlichkeit der Sprache, auflistet.

Der Eingabewert für beide Funktionen ist eine Liste von Texten. Ein Beispiel für mögliche Ausgabewerte ist in der Abbildung 25 und Abbildung 26 zu sehen.

```
{
    "de": 0.9922140147734079
}
```


Abbildung 25 Beispiel für einen Ausgabewert für die Funktion, die die meistverwendete Sprache anzeigt.


```
{
    "af": 0.0003992812936713915,
    "da": 0.00019964064683569574,
    "de": 0.9922140147734079,
    "el": 0.0003992812936713915,
    "en": 0.001597125174685566,
    "et": 0.0003992812936713915,
    "eu": 0.00019964064683569574,
    "fi": 0.00019964064683569574,
    "fr": 0.00019964064683569574,
    "gl": 0.00019964064683569574,
    "it": 0.00019964064683569574,
    "la": 0.0003992812936713915,
    "lb": 0.001597125174685566,
    "lv": 0.00019964064683569574,
    "nl": 0.0007985625873427 83,
    "pl": 0.00019964064683569574,
    "pt": 0.00019964064683569574,
    "sv": 0.0003992812936713915
}
```


Abbildung 26 Beispiel für einen Ausgabewert für die Funktion, die alle Sprachen anzeigt

3.1.1.3 Struktur

Im Folgenden wird betrachtet, wie das Softwaremodul der Language Detection strukturiert ist.

Der LangIdAdapter ist ein Adapter mit dem Ziel, das Interface der verschiedenen Language Detection Libraries an ein normiertes Interface anzupassen. Jede Language Detection Library wird in einem eigenen Adapter implementiert. Würde man zum Beispiel die Library LangDetec[19] verwendet wollen, würde dafür der Adapter LangDetecAdapter erstellt werden.

Die Business-Logik des Moduls liegt in der LanguageDetection Klasse. Diese Klasse ist so gestaltet, dass sie keine Abhängigkeiten zur Struktur oder Datahandling des Texploration hat. Auf diese Weise kann gewährleistet werden, dass dieses Modul ohne Probleme in einen Microservice ausgelagert werden kann.

Die Aufgabe des LanguageDetectionWrapper ist es, die Eingabedaten sowie Rückgabewerte der LanguageDetection an die Struktur der Texploration anzupassen.

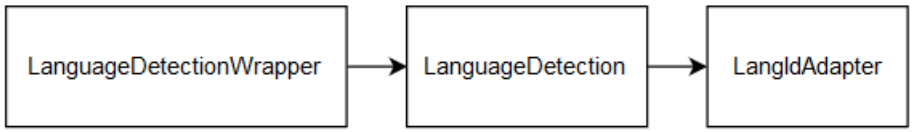


Abbildung 27 Ein Diagramm der involvierten Klassen bei der Language Detection

3.1.2 Konzept User Interface

Bei der Corpus Analyse sollte der Benutzer sehen können, ob sein Korpus mehrsprachig ist und vor allem auch, welche Sprachen seine Datensätze beinhalten. Um dies zu bewerkstelligen, wird dem Benutzer ein interaktives Kreisdiagram dargestellt, das den Anteil der vorkommenden Sprachen anzeigt.

Wenn eines der Kreisdiagrammsegmente vom Benutzer ausgewählt wird, erhält er nähere Informationen zur ausgewählten Sprache. Dort kann er sich die Verteilung der Label-Spalte oder Beispiele solcher Datensätze anzeigen lassen.

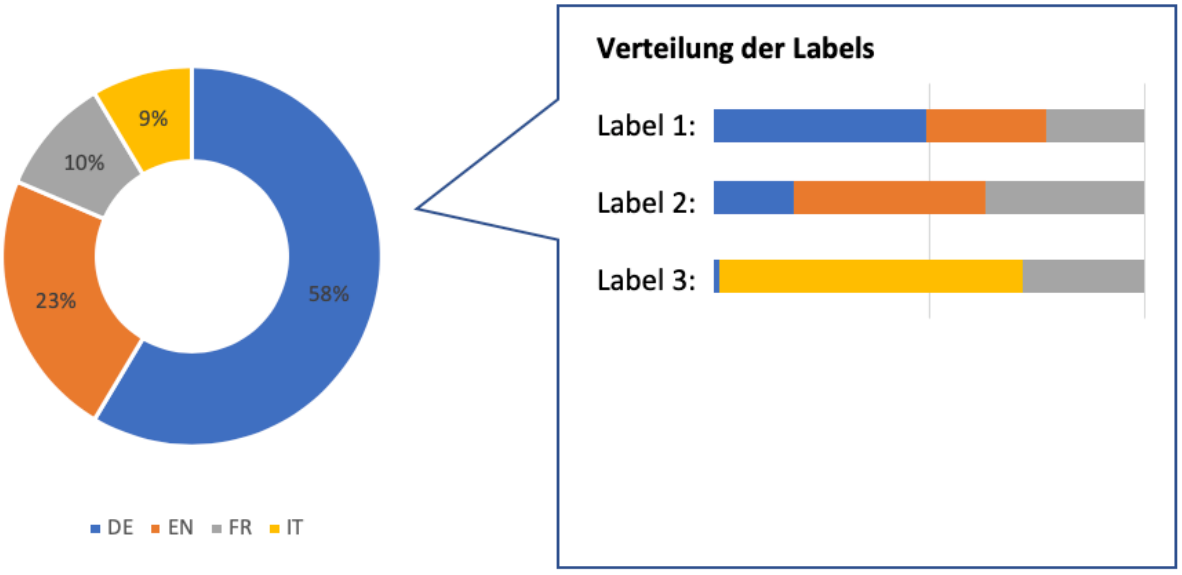


Abbildung 28 Konzept des UI für die Language Detection

3.2 Topic Modeling

Ein Topic Model gibt das Thema des Texts basierend auf einem statistischen Model an. In der Abbildung 29 behandelt das Topic 2 alle Themen rund um "Polizei und Verbrechen". Das Topic Modeling befindet sich in der Texploration in der "Corpus Analyse".

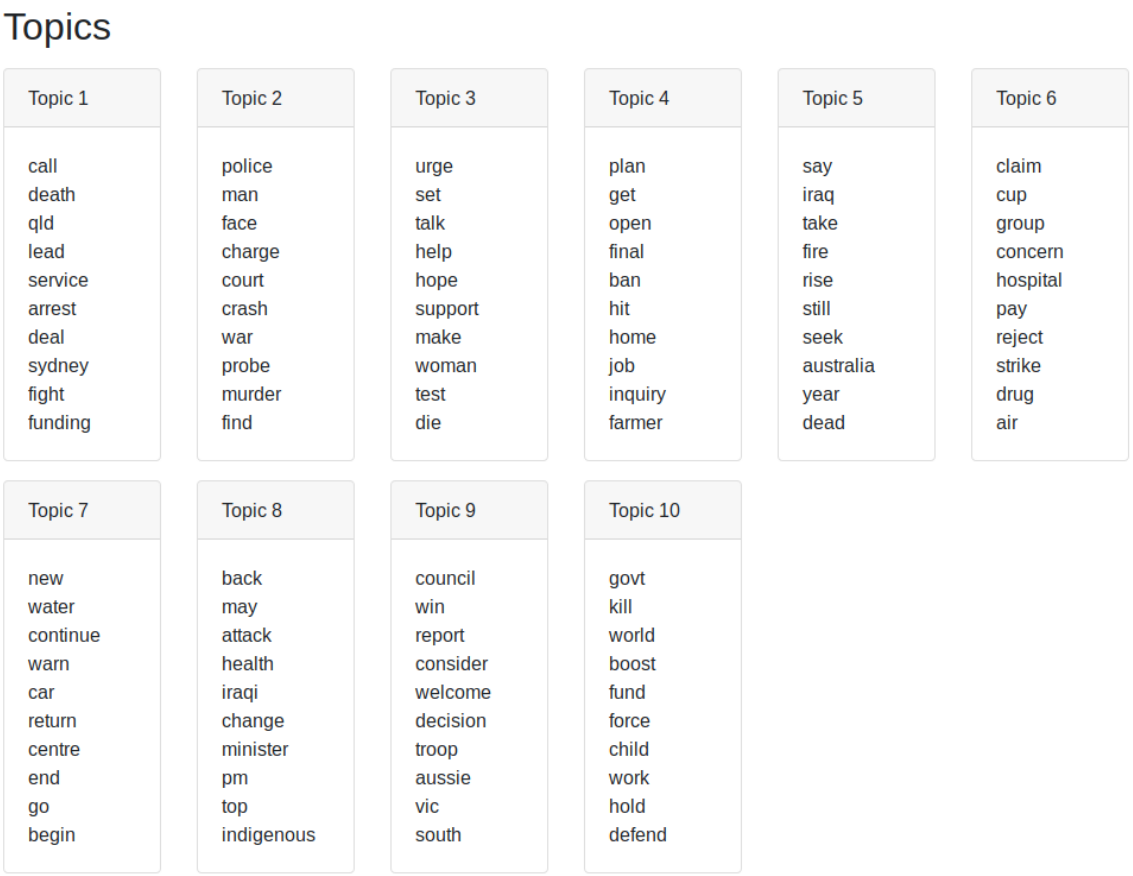


Abbildung 29 Ausschnitt des Topic Modeling

Das Topic Modeling hilft dem Benutzer einen Überblick über die behandelnden Themen im Text zu erhalten. Oftmals ist der Inhalt unbekannt und der Korpus viel zu gross, um manuell die Informationen durchzulesen[20].

3.2.1 Theorie

In den folgenden Kapiteln werden theoretisch relevante Aspekte für das Topic Modeling näher erläutert.

3.2.1.1 TF-IDF

TF-IDF heisst ausgeschrieben term frequency-inverse document frequency und ist ein statistisches Mass, um die Wichtigkeit einzelner Wörter in einem Korpus zu messen. Die Wichtigkeit wird mittels eins Gewichts angegeben. Je höher dieses Gewicht, desto wichtiger ist das Wort. [21]

Um das Gewicht W für das Wort x im Dokument y zu berechnen, sieht die Formel wie folgt aus: [22]

$$W_{x,y} = tf_{x,y} * \log\left(\frac{N}{df_x}\right)$$

Das tf berechnet, wie oft das Wort x im Dokument y vorkommt[22]. Dieser Wert kann auch normalisiert werden, indem es durch die totale Anzahl der Wörter geteilt wird[21].

Das N steht für die totale Anzahl der Dokumente und df_x für die Anzahl der Dokumente, die das Wort x beinhalten. Der rechte Faktor der Multiplikation wird unter anderem auch IDF (inverse document frequency) genannt[22].

3.2.1.2 Topic Modeling mit NMF (Non-Negative Matrix Factorization)

Die Non-Negative Matrix Factorization ist eine Gruppe der unsupervised Algorithmen. Es konnte schon in verschiedenen Bereichen erfolgreich angewendet werden, wie zum Beispiel in der Bioinformatik, Bildverarbeitung, Audioverarbeitung und der Textanalyse.[23]

Die Idee des NMF ist es, für eine nicht negative Matrix A, die zwei Matrizen W und H mit der Dimension k zu finden. Die Formel lautet wie folgt:

$$A = W * H$$

Das Resultat kann als "Dimension Reduction" oder für das Topic Modeling verwendet werden.[23]

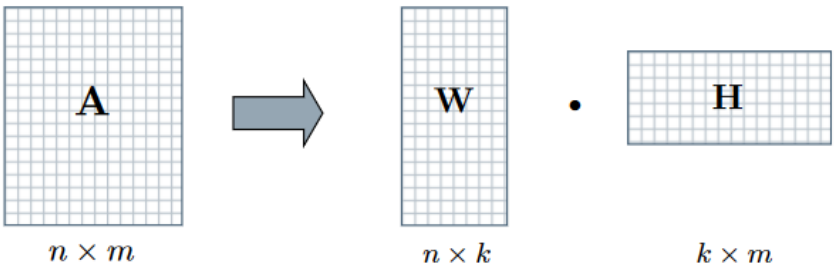


Abbildung 30 Input Matrix A und Output-Matrizen w und H[23]

Es existieren verschiedene Implementationen dafür. Die Implementation von scikit-learn verwendet Lineare Algebra[24].

Das NMF angewendet auf das Topic Modeling sieht folgendermassen aus. Die Input Matrix A ist eine term-document Matrix. Dort ist sichtbar, wie oft jedes einzelnes Wort pro Dokument auftaucht. Der Output wäre die term-topic Matrix W und topic document matrix H[25]. Die term-topic Matrix zeigt pro Wort an, welche Gewichtung es pro Topic besitzt. Bei der topic document matrix wird angezeigt, welche Topics in jedem Dokument vorkommen[23]. Mit diesen Matrizen besitzt man alle benötigten Informationen für das Topic Modeling.

3.2.1.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) ist ein mittlerweile etabliertes Verfahren, um generative Wahrscheinlichkeitsmodelle zu erstellen. Es spielt eine bedeutsame Rolle in der Bioinformatik, in den Sozial- und Politikwissenschaften und in vielen anderen Bereichen. [26]

3.2.1.3.1 Funktionsweise

Der ganze Algorithmus besteht im Allgemeinen aus vier Teilen, dem Preprocessing, dem Training, dem Scoring und der Evaluation[27].

Das Preprocessing ist dafür zuständig, dass die Daten für den Fortlauf des Algorithmus vorbereitet werden[27]. Nach dem Preprocessing kommt die eigentliche Ausführung des Algorithmus.

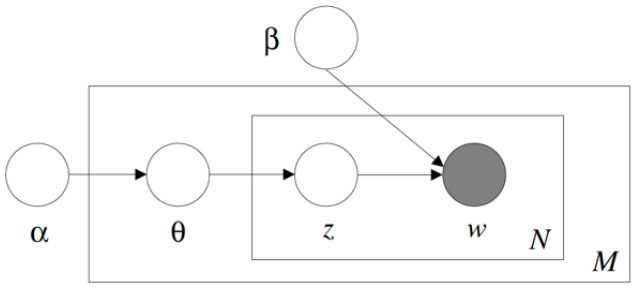


Abbildung 31 Grafische Darstellung des LDA Models[28]. Die Grafik zeigt auf, wie die Parameter aufeinander wirken.

Tabelle 5 Beschreibung der einzelnen Parameter von LDA

Symbol	**Beschreibung**
w	Wörter in einem Dokument i

z	Topic eines Wortes j in einem Dokument i
θ	Verteilung der Topics in einem Dokument i
N	Menge aller Wörter die vom TFIDF zurückgegeben wurden.
M	Menge aller Dokumente.
α	Parameter, der die Verteilung der Wörter per Topic definiert.
β	Parameter, der die Verteilung der Dokumente per Topic definiert.

Das Model kombiniert die drei Elemente "Wort", "Topic" und "Kontext". α und β definieren die Regeln wie ein Topic zusammengesetzt ist. Als Resultat erhält man eine Verteilung der Topics pro Dokument. Anschliessend können die Dokumente in ein Simplex über alle Topics eingefügt werden[27]. Ein Beispiel eines Simplex mit drei Topics ist in Abbildung 33 zu sehen. Der rote Punkt ist ein Dokument, welches neu eingefügt wurde. Die Grafik zeigt auf, welche Dokumente sich nahe dem roten Punkt befinden und somit ähnlich wie das rote Dokument sind.

LDA assumes the following generative process for each document $\mathbf{w}$ in a corpus D:

1. Choose $N \sim \text{Poisson}(\xi)$.
2. Choose $\theta \sim \text{Dir}(\alpha)$.
3. For each of the N words w_n:
   (a) Choose a topic $z_n \sim \text{Multinomial}(\theta)$.
   (b) Choose a word w_n from $p(w_n \mid z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

Abbildung 32 Textuelle Beschreibung des LDA Algorithmus[29]

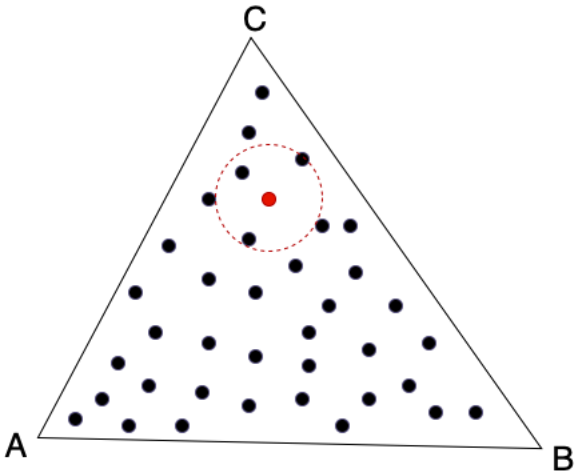


Abbildung 33 Grafische Darstellung eines Simplex in LDA mit den drei Topics A, B und C. Die Grafik wurde vom ursprünglichen Paper von Andrew Ng et al. abgeleitet und für diese Dokumentation angepasst[28].

3.2.1.3.2 Einflussfaktoren

Zu den Einflussfaktoren gehören die Anzahl von Dokumenten (D), die Länge der Dokumente (N), der Parameter für die Verteilung (α und β) und die definierte Anzahl Topics (K) [26].

3.2.1.3.3 Anzahl Dokumente

Den grössten Einfluss bei der Anwendung von LDA besitzt der Faktor "Anzahl Dokumente", denn es ist fast unmöglich Topics aus einer kleinen Anzahl von Dokumenten zu bestimmen. Sollte jedoch die Menge von Dokumenten bereits sehr gross sein, bringt eine noch grössere Menge von Dokumenten keinen signifikanten Qualitätsunterschied im Resultat mit sich. [26]

3.2.1.3.4 Länge der Dokumente

Schlechte Resultate können aber auch aus viel zu kurzen Dokumenten entstehen. Zu lange Dokumente haben wiederum einen schlechten Einfluss auf die Performance. In der Praxis werden solche Dokumente fraktioniert eingesetzt, wobei das Resultat dem eigentlichen Resultat nahekommt. [26]

3.2.1.3.5 Anzahl Topics

Ein weiterer Einflussfaktor ist die Anzahl Topics, denn je weniger Topics ausgewählt werden, desto grösser werden die Überschneidungen. Im Umkehrschluss bedeutet

das, je mehr Topics es gibt, desto feingranularer können die Topics erstellt werden. [26]

3.2.1.4 Berechnung der Topic Model Qualität

Gemäss Newman et al. ist es mithilfe des PMI-Scores möglich, die Qualität eines Topic Models zu messen. Der PMI-Score basiert auf dem pointwise mutual information[30]. Dieser Score verwendet die Assoziation zwischen allen Wortpaaren in einem Topic als Grundlage für die Berechnung. Auf diese Weise kann ermittelt werden, wie kohärent ein Topic ist[31].

Um die Qualität eines Topic Models zu berechnen, wird die folgende Formel verwendet:[32]

$$score = \frac{1}{K}\sum_k PMI - Score(k)\ ,$$

wobei K Topics gegeben sind. Die Funktion $PMI - Score$ ist wie folgt definiert:[31]

$$PMI - Score(k) = \frac{1}{T(T-1)/2}\sum_{1\le i<j\le T} PMI(w_i, w_j),$$

wobei T Wörter pro Topic gegeben sind und w die Wörter des jeweiligen Topics sind. Die PMI-Funktion ist folgenderweise definiert: [31]

$$PMI(w_i, w_j) = \log\frac{p(w_i, w_j)}{p(w_i)p(w_j)}$$

$p(w_i, w_j)$ steht für die Wahrscheinlichkeit des gleichzeitigen Auftretens der Wörter w_i und w_j. Die Funktionen $p(w_i)$ bzw. $p(w_j)$ geben an, wie wahrscheinlich die einzelnen Wörter sind. Diese Wahrscheinlichkeiten werden mithilfe einer externen Datenquelle berechnet.[31]

3.2.2 State of the Art

Es wurde eine Recherche betrieben, um den state of the art verschiedener Aspekte des Topic Modelings zu dokumentieren.

Laut Knispelis [33] wurde bei dem dänischen Medienunternehmen Issuu ein Topic Modeling Verfahren basierend auf dem LDA Algorithmus eingesetzt, um dem Benutzer eine möglichst grosse Vielfalt von themenähnlichen Artikeln anzubieten. Issuu bietet einen Onlinedienst an, um Magazine und Zeitungen zu publizieren.

Liu, Tang, Dong, Yao und Zhou [34] beschreiben, dass Topic Models immer öfter in der Bioinformatik eingesetzt werden, um biologische Informationen besser interpretieren zu können. Zwar besteht hierbei das Problem, dass noch keine spezifischen Algorithmen für die Bioinformatik existieren, jedoch sehen viele Wissenschaftler das Topic Modeling als nützliche Methode an, um versteckte Informationen und Beziehungen zwischen den Daten zu erkennen.

Huang, Hsieh und Wang [35] benutzen Topic Models für das Zusammenfassen von Meetings. Die Software arbeitet mit einer Bluemix Spracherkennung, sodass das ganze Meeting abgehört werden kann. Anschliessend wird der LDA- und TextTiling Algorithmus eingesetzt, um die Topics zu bestimmen. Anhand der TextTeaser Software wird der Text schlussendlich zusammengefasst.

Han, Kamber und Pei [36] haben entdeckt, dass man Topic Modeling neben anderen Verfahren auch für Text Data Mining einsetzten kann.

Stevens, Kegelmeyer, Andrzejewski und Buttler [37] haben anhand von Experiment herausgefunden, dass sowohl LDA, LSA, SVD und NMF unterschiedliche Stärken haben. Aus diesem Grund kann man keine generelle Aussage machen, welche der Algorithmen am besten ist. Man muss fallbedingt unterscheiden. Falls jedoch ein Benutzer mit dem Topic Modeling interagieren sollte, würde LDA am geeignetsten sein.

O'Callaghan, Greene, Carthy und Cunningham[38] haben mithilfe von mehreren Korpora beliebte Varianten von LDA und NMF verglichen. Es wurden dabei mehrere verschiedene Bewertungsmasse verwendet. Die Schlussfolgerung der Studie ist, dass LDA besser auf Korpora mit einer breiten Themenverteilung läuft. NMF ist hingegen besser auf Nischen- und nicht-Mainstream-Inhalten.

Cheng, Yan, Lan und Guo[39] haben den Algorithmus Biterm Topic Model (BTM) vorgestellt. Das BTM sollte das Problem lösen, dass LDA und PLSA auf kurzen Texten schlechte Ergebnisse liefern können. Eine Evaluation hat ergeben, dass BTM nicht nur auf kurzen Texten, sondern auch auf normalen Texten bessere Ergebnisse als das LDA liefert.

3.2.3 Aktuelle Situation

Vor der Ausführung des Topic Models wird das Preprocessing durchgeführt. Dort werden im Text unerwünschte Wörter oder Zeichen entfernt. Anschliessend entfernt man alle englischen Stopwords. Danach werden Bigrams geformt und die Lemmatization mithilfe eines englischen Spacy Models durchgeführt.

Nach dem Preprocessing wird TF-IDF auf den Text durchgeführt. Die daraus resultierende Matrix wird der Non-Negative Matrix Factorization (NMF) übergeben, um das Topic Model zu berechnen.

3.2.4 Problem

Die aktuelle Lösung erbringt bei kurzen Dokumenten ein gutes Resultat, jedoch ist sie nicht für lange Dokumente optimiert. Ausserdem gibt es bei grossen Datenmengen noch Optimierungsbedarf. Darüber hinaus funktioniert das Topic Modeling nur mit englischen Datensätzen, da momentan nur eine englische Stoppwort-Liste und ein englisches Model für das Tokenizing verwendet wird.

3.2.5 Konzept

Im Anschluss wird das erarbeitete Konzept für das Topic Modeling vorgestellt.

3.2.5.1 Grosse Datenmenge

Damit die Software auch bei sehr grossen Datenmengen eine gute Performance erreichen kann, wird ab einer Anzahl von 1 000 000 Dokumenten das Random Sampling Verfahren angewendet. Beim Random Sampling wird lediglich eine zufällig ausgewählte Menge von Dokumenten analysiert. Dies hat zur Folge, dass das Dataset um einiges kleiner wird und schneller analysiert werden kann. Der Qualitätsunterschied der beiden Resultate ist dabei nur sehr minim. [26]

3.2.5.2 Fallunterscheidung Dokumente

Um ein besseres Resultat erreichen zu können, werden bei unterschiedlichen Textarten die Topics auf verschiedene Weisen berechnet. Momentan wird zwischen den zwei Fällen "kurzer Text" und "langer Text" unterschieden.

Um die Art des Dokuments zu unterscheiden, wird über alle Texte iteriert und anschliessend der Median der Anzahl Zeichen berechnet. Sollte der Median nach dem Preprocessing grösser als 300 Zeichen sein, wird der ganze Dataset als "Langer Text" eingestuft, ansonsten als kurzer Text.

3.2.5.2.1 Kurzer Text

Als kurze Texte werden Dokumente eingestuft, die kürzer als 300 Zeichen sind. Für gewöhnlich gehören Tweets, SMS-Nachrichten oder auch Zitate dazu.

Tweets haben oftmals die Eigenheit, dass sie viele Hashtags, Personenzuweisungen (@-Symbole) und Links beinhalten. Da die Personenzuweisungen sowie die Links nicht zur Qualität des Resultats beitragen, werden sie beim Preprocessing entfernt. In der alten Version wurde bei einigen Korpora der Begriff "https" als eines den stärksten gewichteten Worten gewertet. Durch die obengenannte Modifikation ist dies kein Problem mehr.

Hierzu wird der im Kapitel 3.2.1.3 beschriebene Algorithmus LDA eingesetzt.

3.2.5.2.2 Langer Text

In die Kategorie "Langer Text" fallen alle Dokumente, die mehr als 300 Zeichen betragen. Auch hier wird für das Topic Modeling der LDA Algorithmus eingesetzt.

Lange Texte werden im Vergleich zu kurzen Texten mit anderem Alpha und Beta Parametern aufgerufen.

Texte, die länger als 5000 Zeichen sind, werden im Preprocessing fraktioniert, sodass daraus mehrere Texte entstehen, die maximal 5000 Zeichen lang sind. Dies erbringt einen erheblichen Performancegewinn, die Qualität des Resultates bleibt jedoch gleich. [26]

Auch wenn ein Korpus als "Langer Text" eingestuft wird, können einzelne Dokumente immer noch sehr kurz sein und somit zu einem schlechteren Resultat führen. Um dieses Problem zu umgehen, können mehrere kurze Dokumente mit dem gleichen Topic zusammengefasst werden. Das Resultat bei den zusammengefassten Dokumenten ist generell besser, als bei den normalen.[26]

3.2.5.3 Optimierung der LDA-Parameter

Im Anwendungsfall der Texploration sind bei LDA das α und β die einflussreichsten Parameter, die nicht vom Benutzer definiert werden. Das bedeutet, dass je nach Textart diese Parameter im Code optimiert werden können.

Da viele Kombinationen der Konfigurationen existieren, ist vermutlich eine automatische Ermittlung die bessere Herangehensweise, um eine optimale Konfiguration zu finden.

Eine solche Optimierung konnte von Panichella et al.[40] mit Hilfe eines genetischen Algorithmen durchgeführt werden.

Der genetische Algorithmus folgt der Idee der biologischen Evolution[41]. Dabei startet der Algorithmus mit einer zufälligen Population von Chromosomen, die jeweils eine mögliche Lösung für das gesuchte Problem darstellen[42]. Mithilfe der Fitness-Funktion kann berechnet werden, wie gut ein Chromosom die Lösung zum Problem ist. Diese Chromosomen können in verschiedenen Formaten gespeichert werden, zum Beispiel in Binärform. Diese Chromosomen werden iterationsweise, wobei eine Iteration auch Generation genannt wird, neu generiert.

Um ein neues Chromosom zu generieren, werden zwei zufällige Chromosomen der aktuellen Generation als Eltern ausgewählt. Die Chromosomen mit den besseren Werten der Fitness-Funktion haben dabei eine höhere Wahrscheinlichkeit gewählt zu werden. Sind die Eltern ausgewählt, wird mit der Crossover Wahrscheinlichkeit entschieden, ob das neue Chromosom eine Kreuzung der Eltern-Chromosomen ist oder eine direkte Kopie der Eltern. Anschliessend wird mittels der Mutation ein oder mehrere Teile des Chromosoms verändert. Bei der Binärform wäre es das Flippen eines Bits[43]. Ausserdem kann ein Elitismus von einem oder mehreren Chromosomen gehalten werden. Das bedeutet, dass jeweils die X besten Chromosomen automatisch in die nächste Generation übernommen werden[40].

Es werden Generationen so lange weiterberechnet, bis man mit der Lösung zufrieden ist.

3.2.5.4 Mini-Analyse

Je nach Datenmenge kann die Berechnung des Topic Modeling bis zu mehreren Tagen dauern. Oftmals hat der Benutzer nicht die Zeit und auch nicht das Bedürfnis, um eine vollumfängliche Analyse durchzuführen. Die Texploration sollte eine Funktion anbieten, um eine Mini-Analyse durchzuführen. Innert weniger Minuten bekommt der Benutzer bereits ein Resultat und erhält einen besseren Überblick über den Korpus.

Um das Resultat schnellstmöglich berechnen zu können, wird das Random Sampling Verfahren eingesetzt, jedoch befindet sich das Limit der Dokumente nicht bei 1 000 000, sondern bereits bei 10 000.

3.2.5.5 Verbesserung der Darstellung der Ergebnisse

Laut Song et al. existiert Optimierungsbedarf bei der Darstellung der Ergebnisse eines LDA-basierten Topic Models, da sie nicht immer ideal für das menschliche Verständnis strukturiert sind. Von diesen Autoren wurden zwei Aspekte erkannt, die bei Topic Models verbessert werden könnten: Reihenfolge der Topics und die Reihenfolge der Keywords innerhalb eines Topics[44].

3.2.5.5.1 Reihenfolge der Topics

In der Studie von Song et al. wurden verschiedene Methoden evaluiert, wie die Reihenfolge der Topics optimiert werden kann. Die besten Ergebnisse lieferte der Laplacian Score Algorithmus[44]. Dieser Algorithmus bewertet die Wichtigkeit der einzelnen Topics anhand des Laplacian Scores[45].

Die Zeitkomplexität dieser Methode beträgt ungefähr $O(M^2)$, wobei M die Anzahl Dokumente ist[44].

3.2.5.5.2 Reihenfolge der Keywords

Die Keywords eines Topics werden bereits in LDA nach ihrer Wahrscheinlichkeit (likelyhood) sortiert. Diese Reihenfolge kann noch verbessert werden, so dass die Topics für den Benutzer möglicherweise verständlicher werden[44].

In der Studie von Alokaili et al. [46] wurden die folgenden Methoden für die Gewichtung der Keywords analysiert:

- Originale LDA Ranking
- Normalisiertes LDA Ranking
- TF-IDF Ranking
- Inverse Document Frequency (IDF) Ranking

Die Auswertung in der Studie hat ergeben, dass das normalisierte LDA Ranking die schlechtesten Resultate liefert, welches der Studie von Song et al. wiederspricht. Da aber die Studie von Alokaili et al. über eine umfangreichere Evaluation verfügt, wird angenommen, dass sie auch die genaueren Ergebnisse erzielten. Die beste Methode dieser Studie ist das IDF-Ranking.

Die Formel für das IDF-Ranking lautet wie folgt:

$$score_{w,t} = \hat{\varphi}_{w,t} \log \frac{|D|}{|D_{w|}}$$

$\hat{\varphi}_{w,t}$ steht für die Wahrscheinlichkeit (likelyhood) des Wortes w im gegebenen Topic t. Dies wird mit der Inverse Document Frequency ($\log \frac{|D|}{|D_{w|}}$) des Wortes gewichtet. [46]

3.2.5.6 Smart LDA

Die Qualität des Resultats ist oftmals stark von der Anzahl ausgewählter Topics abhängig. Wenn der Benutzer jedoch lediglich zehn Topics angezeigt haben möchte, dann wird das zum Problem für das LDA. Bei einer kleinen Anzahl Topics nimmt die Qualität stark ab und es gibt innerhalb der Topics viele Überschneidungen[26]. Eine Möglichkeit dieses Problem zu umgehen bietet die selberentwickelte Funktion Smart LDA. Auch wenn der Benutzer nur zehn Topics auswählt, werden im Hintergrund 50 oder mehr Topics berechnet und die besten Zehn werden dem Benutzer angezeigt.

Im Grunde genommen, wird das Topic Modeling genau gleich ausgeführt wie sonst, der einzige Unterschied sind die Anzahl Topics. Nachdem die Topics bestimmt wurden, werden sie alle anhand eines intrinsischen Masses[47] bewertet. Anschliessend werden die besten Topics ausgewählt.

3.2.5.7 User Interface

Für das UI sollten diverse Anpassungen vorgenommen werden. Neu eingeführt werden Balkendiagramme. Darauf ist die Wahrscheinlichkeit (likelyhood) der Worte dargestellt, die aus dem jeweiligen Topic generiert werden. Wie aus der Abbildung 34 zu entnehmen ist, wird der Begriff «st» im ersten Topic am stärksten gewichtet. In den anderen Topics ist er hingegen weniger relevant. Des Weiteren ist im Diagramm eine interaktive Komponente eingebaut. Wenn ein Wort vom Benutzer ausgewählt wird, werden alle Vorkommnisse des Wortes innerhalb der anderen Topics farblich hervorgehoben. In der Abbildung 34 wird das Wort «matter» hervorgehoben, welches dadurch überall dunkelblau hinterlegt wurde.

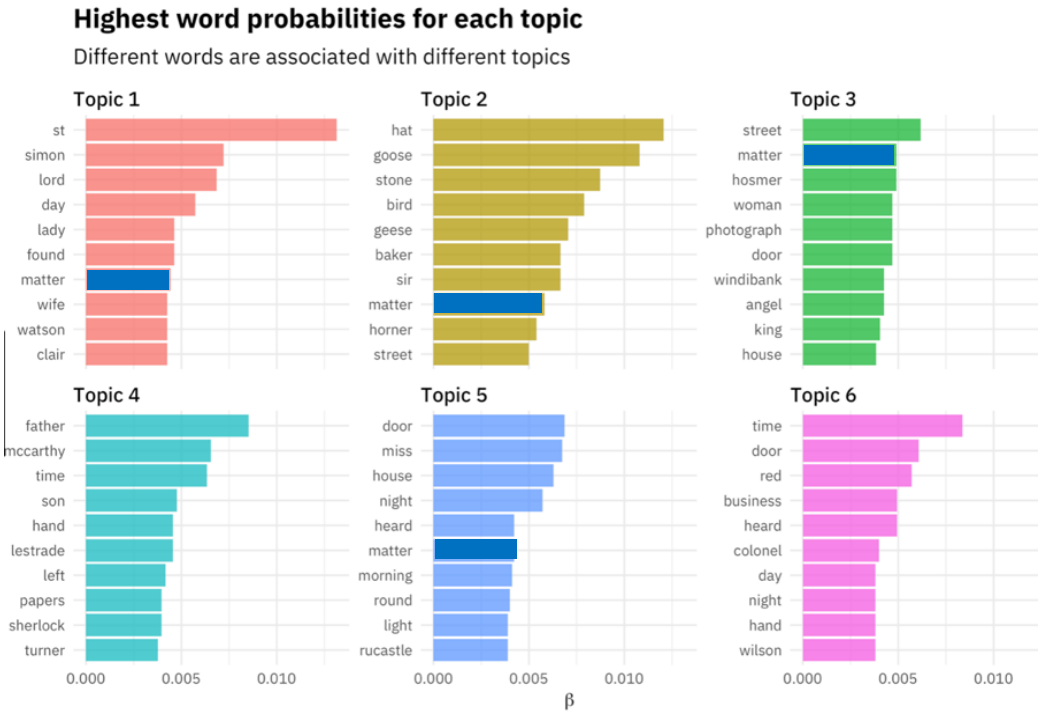


Abbildung 34 Geplantes UI für das Topic Modeling. Übernommen von Silge[48] und angepasst.

3.2.6 Implementation

Im Folgenden wird erwähnt, welche Aspekte des Konzeptes wie umgesetzt wurden.

Die vorgestellte Fallunterscheidung wurde mithilfe eines Experimentes (Kapitel 3.2.6.2) angepasst und implementiert. Unter anderem wurde ein Versuch gestartet, mithilfe des genetischen Algorithmus die LDA-Parameter zu optimieren.

Die Mini-Analyse wurde so wie im Konzept beschrieben umgesetzt.

Die Verbesserung der Topic-Reihenfolge, die im Kapitel 3.2.5.5.1 beschrieben wird, wurde nicht implementiert. Der Algorithmus arbeitet langsam und ist dementsprechend für den Anwendungsfall der Texploration unpassend. Aus dem gleichen Grund wurde auch die Smart LDA Funktion nicht implementiert.

Hingegen wurde die Reihenfolge der Keywords mit der Methode verbessert, die im Kapitel 3.2.5.5.1 beschrieben wird.

3.2.6.1 LDA Parameter

Für das LDA wurde die Library Scikit-learn[49] verwendet. Dort existiert die Möglichkeit, die Parameter α und β zu setzen. Wenn sie aber nicht definiert sind, werden sie folgendermassen gesetzt:

$$Wert = \frac{1}{K} ,$$

wobei K für die Anzahl der Topics steht.

3.2.6.1.1 Setup des Genetischer Algorithmus

Für die Parameteroptimierung wurde ein genetischer Algorithmus verwendet. Er wurde wie folgt konfiguriert.

Das Chromosom ist aus zwei Zahlen zusammengesetzt, die jeweils für den α- und β-Parameter des LDA stehen. Es wurde die Binärform verwendet. Die Population und der Elitismus wurde im Vergleich zur Studie von Panichella et al.[40], aus zeitlichen Gründen, auf 50 bzw. 1 reduziert. Die Crossover-Rate (0,7) und Mutationsrate (0,01) sind dieselben. Für die Fitness-Funktion diente der PMI-Score, der in Kapitel 3.2.1.4 erklärt wird.

Als Datensätze werden jeweils zwei für das Konzept repräsentative Korpora verwendet, ein deutscher Twitter Hatespeech-Datensatz und ein Datensatz von Reviews. Die Anzahl der Topics ist auf 50 gesetzt.

3.2.6.1.2 Resultat vom genetischen Algorithmus

Bei den kurzen Texten konnte eine Verbesserung von 7,5% (1,20 auf 1,29) erzielt werden. Bei langen Texten verbesserte sich das Topic Model um 12,8% (0,86 auf 0,97). In der folgenden Tabelle sind die besten Parameterwerte für die beiden Textsorten dargestellt:

Tabelle 6 Die vom generischen Algorithmus ermittelten Parameter für die beiden Textsorten

Textsorte	**α-Parameter**	**β-Parameter**
kurz	0,297	4,484
lang	0,0312	6,8828

Im Grossen und Ganzen ist der genetische Algorithmus eine gute Methode, um die Parameter automatisch zu optimieren. Die erreichten Ergebnisse sind ein guter Ansatz, jedoch müssen noch weitere Einstellungen getestet werden, um ein robustes Konzept für die Parameter aufstellen zu können.

Um das Potential des genetischen Algorithmus stärker auszuschöpfen, können folgenden Verbesserungen durchgeführt werden.

Im Topic Model ist das mehrfache Vorkommen eines Topics nicht erwünscht. Im PMI-Score wird dies aber nicht berücksichtigt. Dies kann zum Beispiel zur Inflation des Scores führen, wenn ein Topic mit einem guten Score oft im Topic Model mit denselben Wörtern wiederholt wird. Dies könnte behoben werden, indem der Score reduziert wird, wenn ein Topic mit ähnlichen oder gleichen Wörtern mehrmals erscheint.

Die verwendete Fitness-Funktion könnte verbessert werden, indem pro Chromosom mehrere Topic Models von verschiedenen Datensätzen mit ähnlichen Eigenschaften in die Berechnung des Fitness-Wertes miteinbezogen werden. Dies würde sicherstellen, dass die gewählten Parameter nicht nur für einen spezifischen Datenset passend sind, sondern auch für andere. Das würde aber dazu führen, dass die Berechnung einer Generation um ein Mehrfaches länger dauern würde.

3.2.6.2 Vergleich der Qualität von LDA und NMF

Um die Qualität zwischen LDA und NMF zu vergleichen, wurde ein Experiment durchgeführt.

3.2.6.2.1 Setup

Die beiden im Konzept definierten Fälle (s. Kapitel 3.2.5.2) werden mithilfe eines jeweils repräsentativen Korpus getestet. Es wird analysiert, wie der TF-IDF und NMF mit 10, 30 und 50 Topics im Vergleich zum LDA abschneidet.

Für den Fall "Kurzer Text" wird ein deutscher Twitter-Hatespeech Datensatz verwendet, der pro Dokument durchschnittlich 144 Zeichen beinhaltet. Für den anderen Fall wird ein englischer Review-Korpus verwendet, bei dem die durchschnittliche Dokumentenlänge 1227 Zeichen beträgt.

Für den PMI wurde als externe Datenquelle entweder der deutsche Wikipedia Korpus (1 563 795 Dokumenten und 299 389 636 Tokens) oder der englische (735 385 Dokumenten und 395 992 677 Tokens) verwendet. Diese Korpora haben ungefähr die gleiche Dateigrösse.

Das Preprocessing ist bei jeder der getesteten Methoden das gleiche. Das Topic Model wurde mit 10 Wörtern pro Topic generiert.

Für beide Datensätze werden die Parameter des LDA bei 50 Topics mithilfe des genetischen Algorithmus optimiert.

3.2.6.2.2 Vergleich

In der folgenden Tabelle sind die Resultate vom Vergleich der Methoden für den Twitter-Datensatz abgebildet.

Tabelle 7 Qualität der besten Topic Models der jeweiligen Algorithmen mit dem Twitter-Datensatz

Topics-Anzahl	NMF	LDA	LDA mit optimierten Parametern
10	1,03	1,11	-
30	1,16	1,19	-
50	1,16	1,2	1,29

Ausserdem sind diese Ergebnisse in der folgenden Abbildung dargestellt.

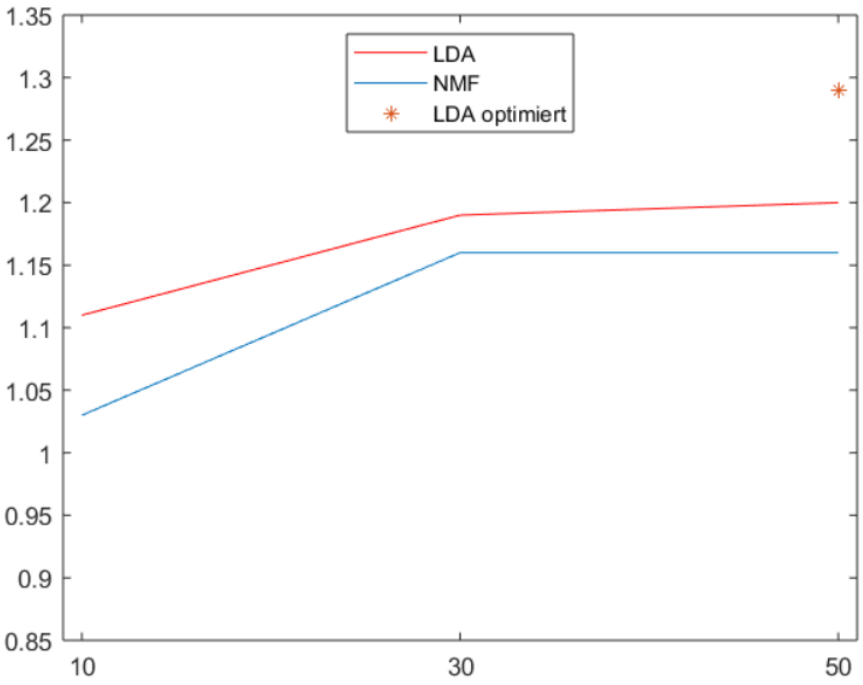


Abbildung 35 Darstellung der Resultate des Twitter-Korpus

Die Resultate für den Review-Korpus stehen in der nächsten Tabelle.

Tabelle 8 Qualität der besten Topic Models der jeweiligen Algorithmen mit dem Review-Datensatz

Topics-Anzahl	NMF	LDA	LDA mit optimierten Parametern
10	0,98	0,87	-
30	1,05	0,87	-
50	1,06	0,86	0,96

Diese Daten sind in der Abbildung 36 dargestellt.

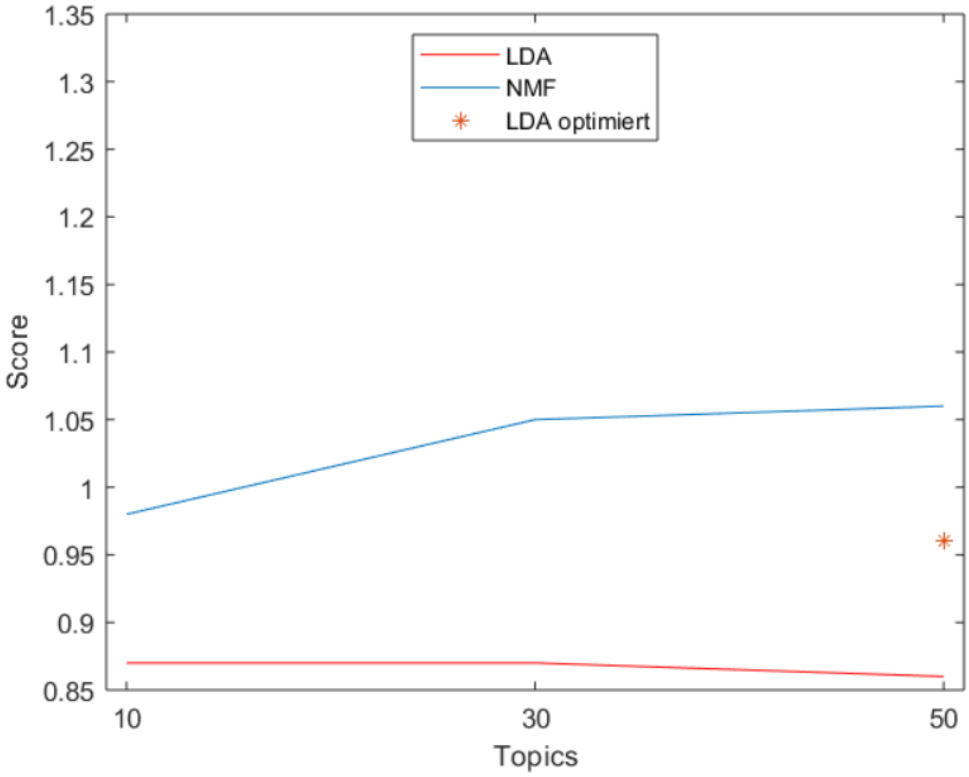


Abbildung 36 Darstellung der Resultate auf dem Review-Korpus

Zusammengefasst liefert der LDA bei kurzen Texten die besseren Ergebnisse. Bei langen Texten unterliegt er hingegen dem NMF. Mithilfe des genetischen Algorithmus konnte der PMI-Score des LDA um 0,09 – 0,10 gesteigert werden. Trotz dessen ist der NMF bei langen Texten besser.

Da bei langen Texten das LDA trotz der Parameteroptimierung schlechter als die bisherige Variante ausfällt, wird das Konzept für das Topic Modeling angepasst. Neu wird das LDA nur für kurze Texte angewendet. Für lange Texte wird NMF angewendet.

3.2.6.3 Laufzeitanalyse

Im Folgenden wird analysiert, wie lange die Laufzeit des implementierten LDA je nach Dokumentenanzahl ist.

3.2.6.3.1 Setup

Für das Experiment wurde eine leicht modifizierte Version der Texploration verwendet. Dort wurde jeweils die verwendete Zeit für das Preprocessing und für die Ausführung des LDA ausgegeben.

Die Software wurde in einer virtuellen Maschine mit einem Ubuntu Betriebssystem ausgeführt. Es wurden 10GB RAM und 4 CPU-Kerne eines i7-6700HQ Prozessor zugewiesen.

Als Datenquelle für das Topic Modeling diente der Leipzig Korpus[50], der drei Millionen deutsche Sätze beinhaltet. Die Dokumente von diesem Korpus sind vergleichsweise kurz. Mithilfe von Random Sampling wurden die Korpora mit der gewünschten Anzahl an Dokumenten generiert. Für das Experiment werden Dokumente mit eintausend, zehn Tausend, fünfzig Tausend, hundert Tausend, fünfhundert Tausend und einer Million Dokumente verwendet. Für jeden dieser Einstellungen werden jeweils 5 Durchläufe gemacht.

Das Preprocessing für das Topic Modeling wurde nicht verändert. Die Anzahl der Topics und der Wörter wurde standardmässig bei 10 belassen.

3.2.6.3.2 Resultat

Die Ergebnisse des Experimentes sind in der nachfolgenden Tabelle dargestellt.

Tabelle 9 Übersicht der LDA- und Preprocessing-Laufzeiten über die Anzahl Dokumente

Anzahl Dokumente	**Dateigrösse (MB)**	**Dauer Preprocessing [s]**	**Dauer LDA [s]**	**Dauer Total [s]**
1 000 000	110,0MB	2673 ± 72	688 ± 57	3361 ± 108

500 000	55,4MB	1313 ± 35	329 ± 10	1642 ± 33
100 000	11,0MB	263 ± 12	63 ± 3	326 ± 14
50 000	5,5MB	127 ± 1	34 ± 0	161 ± 1
10 000	1,1MB	28 ± 1	6 ± 0	34 ± 0
1 000	0,1MB	5 ± 1	0 ± 0	5 ± 1

Ausserdem sind diese Werte in der Abbildung 37 visualisiert.

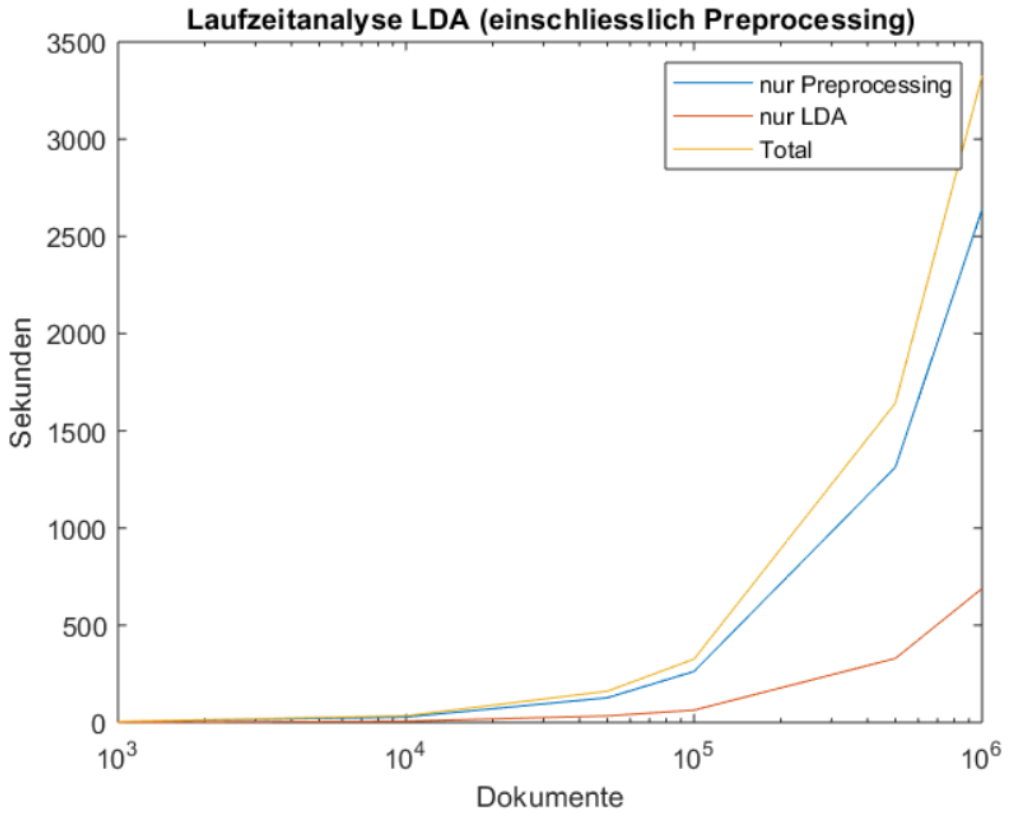


Abbildung 37 Darstellung der im Experiment erreichten Ergebnisse als Diagramm

Wie aus dem Diagramm ersichtlich ist, dauert das Preprocessing um ein Mehrfaches länger als das LDA selbst. Hier müsste man genauer analysieren, wieso das Preprocessing so lange dauert und wie man es optimieren könnte.

Ausserdem dauert das Topic Model mit LDA für eine Million kurze Dokumente etwa eine Stunde, welches sich noch im akzeptablen Rahmen befindet. Dies zeigt unter

anderem, dass die obere Grenze, ab dem das Random Sampling anfängt, gut angesetzt ist.

4 Diskussion und Ausblick

Dieses Kapitel reflektiert die erreichten Ziele und gibt einen Ausblick in die Erweiterungsmöglichkeiten von Texploration.

4.1 Erreichte Ziele

Texploration funktioniert jetzt stabiler als vorher und die bestehenden Funktionen konnten grossteils optimiert und erweitert werden. Neu hinzugekommen ist das Feature Language Detection. Zudem wurde das Topic Modeling mit dem LDA Algorithmus auf die Bedürfnisse von Texploration optimiert. Des Weiteren wurden spezifische Konzepte für die Software ausgearbeitet, damit zukünftige Entwickler einen einfachen Einstieg in das Programm erhalten. Aus diesen Gründen erachten wir, dass die Aufgabenstellung als erfüllt gilt.

4.1.1 Refactoring

Im Vergleich zur Ausgangslage wurden viele Änderungen in der Architektur als auch im Code vorgenommen. Die meisten Änderungen wurden in der API und im Backend durchgeführt, wobei neu die zwei Teile über einen Redis-Kanal kommunizieren können.

Des Weiteren wurden im Backend Adapter eingeführt, sodass die Algorithmen in verschiedenen Prozessen ausgeführt und spezifisch verwaltet werden können. Sie sind auch in der Lage Fehler zu behandeln.

Im Backend wurden zudem alle Python Scripts zu einer Software zusammengefasst. Zudem wurden Klassen eingeführt, so dass Entwickler besser verstehen können, wie die Software aufgebaut ist.

Eine weitere Optimierung im Code ist die Einführung eines Flask-Servers im Backend. Anfragen an das Backend werden durch HTTP-Requests erledigt. Ausserdem wurden, durch den Einsatz eines Flask Server (Python Webserver), die Bereiche Node.js und Python voneinander isoliert.

4.1.2 Microservice Konzept

Für den Fortgang des Projekts wurde ein Konzept entwickelt, welches den zukünftigen Entwicklern helfen soll, das Projekt umzustrukturieren.

4.1.3 Language Detection

Für dieses Feature wurde ein komplett unabhängiges Modul implementiert, welches zwei Funktionen erfüllt. Zum einen kann die Hauptsprache eines Korpus abgefragt werden und zum anderen kann die Verteilung der Sprachen pro Korpus abgefragt werden.

4.1.4 Topic Modeling

Im Topic Modeling wurde der LDA Algorithmus implementiert und für die Bedürfnisse von Texploration optimiert. Die Resultate sind bei kurzen Texten um einiges besser als beim vorherigen Algorithmus, bei längeren Texten hingegen ist LDA immer noch relativ schlecht.

Des Weiteren wurden für das Topic Modeling diverse Analysen durchgeführt, welche den LDA Algorithmus optimieren. Eines der durchgeführten Experimente ist das Optimieren der Parameter für den LDA anhand eines genetischen Algorithmus.

4.2 Erweiterungen

Texploration könnte zukünftig eine Open-Source Software sein, bei der viele Software Entwickler daran arbeiten können. Um dieses Ziel zu erreichen, müssen noch diverse Hürden in der Architektur und in der UI überwunden werden

4.2.1 Architektur

Der grösste Einfluss auf die Softwarearchitektur ist die Einführung von Microservices (s. Kapitel 2.3.6).

Zusammen mit der Einführung einer neuen Architektur kommen noch andere Elemente hinzu.

- Data Handling: Die Kommunikation zwischen den Programmteilen erfolgt nicht mehr über lokale Files, sondern über klar strukturierte HTTP-Requests oder Redis Kanäle. Zudem werden die Userdaten und die trainierten Models in Spark-Datenbanken abgelegt.
- User Interface: Das UI muss für eine reine Microservices Architektur angepasst und strukturiert werden.

4.2.2 Ausbau der Adapter

Wissenschaftler sollten in der Lage sein ihre Algorithmen schnell und einfach in das System implementieren zu können. Ein Adapter würde dabei als Schnittstelle zwischen Texploration und dem Algorithmus fungieren. Es soll lediglich definiert werden müssen, welche Art von Diagramm angezeigt werden soll, welche Werte zueinander finden müssen und wie der Algorithmus zum Frontend kommunizieren soll.

4.2.3 Mehr Classifier

Damit die Software bei Machine Learning Competitions besser zum Einsatz kommt, können noch mehr Classifier implementiert werden.

4.2.4 Preprocessing Konfigurieren

Das Preprocessing ist ein wichtiger Faktor für die Classifier. Daher ist es vorteilhaft, wenn der Benutzer das Preprocessing nach seinen Bedürfnissen anpassen und erweitern könnte. Wünschenswert wäre es, wenn es auch effizient wäre.

4.2.5 Projekte und Experimente

Ein weiterer Interessanter Ausbauschritt ist die Einführung von Projekten und Experimenten. Wenn ein neues Projekt erstellt wurde, können die dazugehörenden Personen dem Projekt zugewiesen werden. Ausserdem können noch vordefinierte Datensätze hinterlegt werden, welche spezifisch für das Projekt gedacht sind.

Pro Projekt könnten Experimente durchgeführt werden und in einer Historie nachverfolgt werden. Andererseits könnte man verschiedene Diagramme anzeigen lassen, zum Beispiel wie sich der F1-Score eines Algorithmus über die Zeit verbessert hat.

4.2.6 Support weiterer Dateitypen

Bis anhin werden nur CSV Dateien unterstützt. Eine Erleichterung für den Benutzer wäre, wenn Texploration unterschiedliche Dateitypen, wie JSON- oder XML-Dateien, unterstützen würde. Zudem ist es von Vorteil, wenn Texploration automatisch erkennen könnte, was der eigentliche Inhalt ist und was nur Metadaten.

4.2.7 Benutzerverwaltung

Um einen einfachen Betrieb zu bewerkstelligen, ist eine simple Benutzerverwaltung notwendig. Folgende Funktionen sollte eine Benutzerverwaltung anbieten:

- Erstellen, Bearbeiten und Löschen von Benutzern
- Erstellen und Zuweisen von Rollen
- Benutzerprofil anpassen
- Passwort ändern oder zurücksetzen
- Projekte für andere mit spezifischen Rechten freigeben

4.2.8 Referenzdatensätze zur Verfügung stellen

Bei neuentwickelten Algorithmen will man oftmals lediglich einen schnellen Vergleich zu anderen Algorithmen erhalten. Eine Lösung dafür wäre den neuen Algorithmus mit Referenzdatensätzen zu trainieren und es dann mit den Standard-Classifiern im Projekt zu vergleichen. Hierzu müssten auf der Webseite diese Referenzdaten für alle Benutzer direkt verfügbar sein.

4.2.9 Sicherheit

Nicht zuletzt ist auch der sicherheitstechnische Aspekt relevant. Wenn die Software einmal in Betrieb ist und reale Datensätze von kommerziellen Kunden eingesetzt werden, ist es wichtig diese durch entsprechende Vorkehrungen zu schützen. Nicht nur Kundendaten sind davon betroffen, auch Experimente, Analysen oder sonstige Element müssen vor Fremdeinwirkung geschützt werden.

4.3 Vision

Texploration hat ein grosses Potential eine Standardsoftware für Wissenschaftler im Bereich Künstliche Intelligenz und Datenwissenschaften zu werden. Der grosse Vorteil für die Benutzer ist, dass die ganze Testumgebung nicht mehr neu aufgebaut werden muss. Unter Umständen lassen sich so mehrere Tage Arbeit ersparen. Ein neuer Algorithmus wird in Texploration über einen Adapter eingebunden und ausgeführt. Anschliessend können die Resultate mit diversen state of the art Algorithmen verglichen werden.

Des Weiteren kann Texploration in Machine Learning Competitions eingesetzt werden, um einen allgemeinen Einblick in den Datensatz zu erhalten.

5 Verzeichnisse

5.1 Literaturverzeichnis

[1] L. Metzler *et al.*, “A Framework for Question-Driven Text Corpus Exploration.” Winterthur, 2019.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, Third Edit. 2018.

[3] A. Laurence, “AntConc,” 2019. [Online]. Available: https://www.laurenceanthony.net. [Accessed: 30-May-2019].

[4] J. O. Rüdiger, “CorpusExplorer Beschreibung.” [Online]. Available: https://notes.jan-oliver-ruediger.de/software/corpusexplorer-overview/. [Accessed: 30-May-2019].

[5] “Cloud AutoML.” [Online]. Available: https://cloud.google.com/automl/. [Accessed: 30-May-2019].

[6] R. C. Martin, *Clean Architecture*, 1. Auflage. Mitp Verlag, 2018.

[7] R. C. Martin, *Clean Code*. Prentice Hall, 2015.

[8] “Redis.” [Online]. Available: https://redis.io/. [Accessed: 13-Mar-2019].

[9] “SQLite Client for Node.js Apps,” 2019. [Online]. Available: https://www.npmjs.com/package/sqlite. [Accessed: 25-Apr-2019].

[10] L. Delgado, “Scalability within micro-services architecture,” 2016. .

[11] D. Taibi, V. Lenarduzzi, and C. Pahl, “Architectural Patterns for Microservices: A Systematic Mapping Study,” no. Closer, pp. 221–232, 2018.

[12] A. B. Mike Wasson, “Data considerations for microservices,” 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/microservices/design/data-considerations. [Accessed: 17-May-2019].

[13] M. Wenzel, N. Anil, L. Latham, and B. Wagner, “Creating composite UI based on microservices,” 2018. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/architect-microservice-container-applications/microservice-based-composite-ui-shape-layout.

[14] M. Fowler, “Microservice Trade-Offs.” [Online]. Available: https://www.martinfowler.com/articles/microservice-trade-offs.html#ops. [Accessed: 02-May-2019].

[15] The Apache Software Foundation, “Spark,” 2019. [Online]. Available: https://spark.apache.org. [Accessed: 07-May-2019].

[16] M. Lui and T. Baldwin, “langid.py: An off-the-shelf language identification tool,” *ACL Demo*, no. July, pp. 25–30, 2012.

[17] “langid.py,” 2017. [Online]. Available: https://github.com/saffsd/langid.py. [Accessed: 16-Apr-2019].

[18] The Linux Information Project, “BSD License Definition,” 2005. [Online]. Available: http://www.linfo.org/bsdlicense.html. [Accessed: 16-Apr-2019].

[19] M. Danilk, “langdetect 1.0.7,” 2016. [Online]. Available: https://pypi.org/project/langdetect. [Accessed: 07-May-2019].

[20] L. Renker, “Exploration von Textkorpora - Topic Models als Grundlage der Interaktion,” Fachhochschule Köln, 2015.

[21] “Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining.” [Online]. Available: http://www.tfidf.com/. [Accessed: 30-May-2019].

[22] M. Cieliebak, “L02: Unsupervised Learning 2,” 2019.

[23] D. Greene, “Matrix Factorization For Topic Models Insight Latent Space Workshop,” 2013.

[24] “Non-negative matrix factorization (NMF or NNMF),” 2019. [Online]. Available: https://scikit-learn.org/stable/modules/decomposition.html#nmf. [Accessed: 30-May-2019].

[25] E. Bolshakova, N. Loukachevitch, and M. Nokel, “Topic Models Can Improve Domain Term Extraction,” in *Advances in Information Retrieval*, 2013, pp. 684–687.

[26] Q. U. Edu, “Understanding the Limiting Factors of Topic Modeling via Posterior Contraction Analysis,” 2014.

[27] A. Knispelis, *LDA Topic Models*. YouTube, 2016.

[28] D. M. Blei, B. B. Edu, A. Y. Ng, A. S. Edu, M. I. Jordan, and J. B. Edu, “Latent Dirichlet Allocation,” 2003.

[29] “LDA Description.” [Online]. Available: https://www.zinkov.com/images/lda_story.png. [Accessed: 30-May-2019].

[30] D. Newman, S. Karimi, and L. Cavedon, “External evaluation of topic models,” *Proc. 14th Australas. Doc. Comput. ing Symp.*, pp. 1–8, 2011.

[31] D. Newman, E. Bonilla, and W. Buntine, “Improving topic coherence with regularized topic models,” *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2011.

[32] X. Cheng, X. Yan, Y. Lan, and J. Guo, “BTM: Topic Modeling over Short Texts,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 12, pp. 2928–2941, 2014.

[33] “issuu Homepage,” 2019. .

[34] L. Liu, L. Tang, W. Dong, S. Yao, and W. Zhou, “An overview of topic modeling and its current applications in bioinformatics,” 2016. [Online]. Available: https://springerplus.springeropen.com/articles/10.1186/s40064-016-3252-8. [Accessed: 02-Jun-2019].

[35] T.-C. Huang, C.-H. Hsieh, and H.-C. Wang, “Automatic meeting summarization and topic detection system,” 2018.

[36] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd Editio. Elsevier Inc., 2012.

[37] K. Stevens, P. Kegelmeyer, D. Andrzejewski, and D. Buttler, “Exploring Topic Coherence over many models and many topics,” Los Angeles, 2012.

[38] D. O’Callaghan, D. Greene, J. Carthy, and P. Cunningham, “An analysis of the coherence of descriptors in topic modeling,” *Expert Syst. Appl.*, vol. 42, no. 13, pp. 5645–5657, 2015.

[39] X. Yan, J. Guo, Y. Lan, and X. Cheng, “A biterm topic model for short texts,” pp. 1445–1456, 2016.

[40] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia, “How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms,” *Proc. - Int. Conf. Softw. Eng.*, pp. 522–531, 2013.

[41] D. (Colorado S. U. Whitley, “A Genetic Algorithm Tutorial by Darrell Whitley,” *Stat. Comput.*, no. 4, pp. 65–85, 1994.

[42] M. Obitko, “IV. Genetic Algorithm,” 1998. [Online]. Available: https://www.obitko.com/tutorials/genetic-algorithms/ga-basic-description.php. [Accessed: 25-May-2019].

[43] M. Obitko, “XI. Crossover and Mutation,” 1998. [Online]. Available: https://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php. [Accessed: 25-May-2019].

[44] Y. Song, S. Pan, S. Liu, M. X. Zhou, and W. Qian, “Topic and keyword re-ranking for LDA-based topic modeling,” p. 1757, 2009.

[45] R. Liu, N. Yang, X. Ding, and L. Ma, “An unsupervised feature selection algorithm: Laplacian score combined with distance-based entropy measure,” *3rd Int. Symp. Intell. Inf. Technol. Appl. IITA 2009*, vol. 3, pp. 65–68, 2009.

[46] A. Alokaili, N. Aletras, and M. Stevenson, “Re-Ranking Words to Improve Interpretability of Automatically Generated Topics.” 2019.

[47] Q. Pleplé, “Topic Coherence To Evaluate Topic Models,” 2013. [Online]. Available: http://qpleple.com/topic-coherence-to-evaluate-topic-models/. [Accessed: 01-Jun-2019].

[48] J. Silge, “Text Mining of Stack Overflow Questions,” 2017. [Online]. Available: https://stackoverflow.blog/2017/07/06/text-mining-stack-overflow-questions/. [Accessed: 26-May-2019].

[49] “Scikit-learn,” 2019. [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 20-May-2019].

[50] R. Tatman, “3 Million German Sentences,” 2017. [Online]. Available: https://www.kaggle.com/rtatman/3-million-german-sentences. [Accessed: 12-May-2019].

[51] “pickle — Python object serialization.” [Online]. Available: https://docs.python.org/3/library/pickle.html. [Accessed: 04-Jun-2019].

[52] K. P. Shung, “Accuracy, Precision, Recall or F1?,” 2018. [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. [Accessed: 04-Jun-2019].

[53] L. Metzler and N. Siddiqui, “Developing a Framework for Visual Corpus Exploration and Comparison of Machine and Deep Learning Algorithms,” 2018.

5.2 Glossar

Begriff	**Definition / Erklärung**
Korpus	Eine Sammlung von Text-Dokumenten[2]
Pickel	"The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream" [51]
Tokenizer	Aufteilung eines Textes in einzelne Worte (Tokens)[2]
F1-Score	Ist ein statistisches Mass, das die Performance eines Classifiers angibt.[52]
Bigram	Ist eine Sequenz von zwei Wörtern, zum Beispiel "ich gehe", "gehe zur" oder "zur Schule".[2]
Lemmatization	Bestimmung der Wortwurzel (sang, singte ➔ singen)[2]

5.3 Abbildungsverzeichnis

5.4 Tabellenverzeichnis

6 Anhang

6.1 Offizielle Aufgabenstellung

Ist ein Tweet positiv oder negativ? Hat jemand Selbstmord-Gedanken? Welche Note sollte eine elektronische Prüfung erhalten?

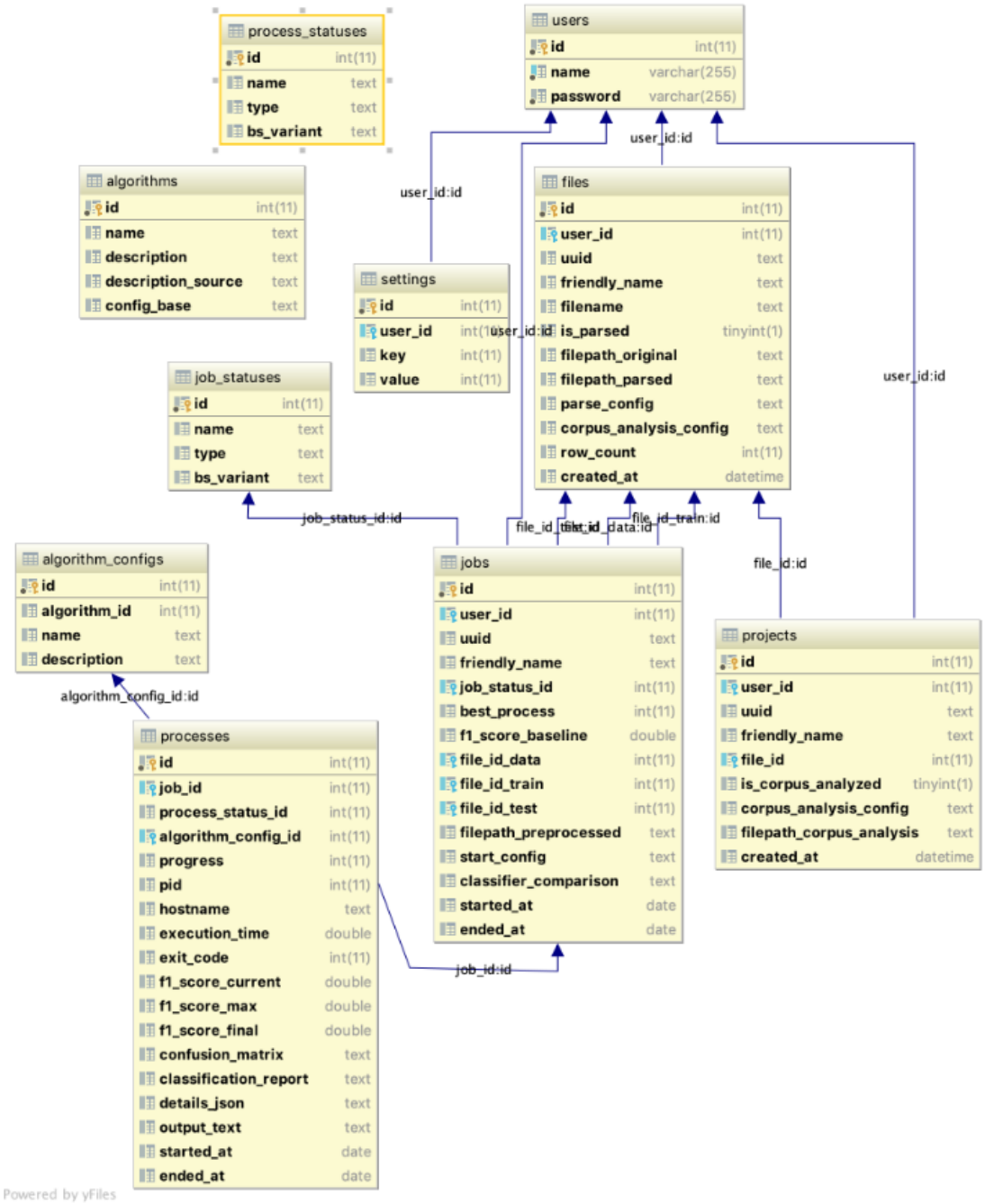
Diese Fragen, so unterschiedlich sie klingen, haben eines gemeinsam: Man muss einen Text in vorgegebene Kategorien einteilen, z.B. positiv/negativ oder Selbstmordgefährdet ja/nein. Dies wird heute typischerweise mit einem Machine-Learning-System gelöst, das für die jeweilige Aufgabenstellung konfiguriert und trainiert wird. Dabei werden z.B. Feature-basierte Systeme eingesetzt oder Neuronale Netze.

Wir haben am InIT ein "Corpus-Analytics-Tool" entwickelt, mit dem man einen Textcorpus (z.B. 10'000 Tweets) visuell analysieren kann. Das Tool kann z.B automatisch wichtige Themen erkennen, mehrere Machine-Learning-Algorithmen vergleichen etc. Damit kann man quasi auf Knopfdruck jedes beliebige Klassifikationsproblem lösen.

Das Framework besteht aus einem Backend mit verschiedenen Machine-Learning Verfahren und Optimierungsalgorithmen, und einer intuitiven GUI, mit der man die Daten analysieren und die Prozesse starten und steuern kann.

Aufgabenstellung In dieser Arbeit soll das bestehende Tool erweitert werden, z.B. um neue Machine-Learning-Algorithmen, Performance-Optimierung, bessere Visualisierung der Ergebnisse, etc. Die genauen Ziele werden gemeinsam festgelegt.

6.2 Datenbankschema



6.3 Installationseinleitung

Im Laufe dieser Arbeit hat sich in Bezug auf die Installation der Texploration nur sehr wenig geändert. Aus diesem Grund wird in den folgenden Unterkapiteln die originale Installationsanleitung der Texploration[53], die von Linus Metzler und Nadina Siddiqui erstellt wurde, mit kleinen Anpassungen wiederverwendet. Die Anpassungen sind jeweils mit eckigen Klammern markiert, wobei sich die Anpassung zwischen den

Klammern befindet. Da diese Anleitung ursprünglich in Englisch verfasst wurde, ist auch der folgende Text Englisch.

6.3.1 Login Information[53]

By default, the following combinations of username and password are present in the database:

Username	Password
linus	123
nadina	456

6.3.2 Installation Instructions[53]

There are two different types of installs covered by this chapter:

1. Installing texploration to run it ([Chapters 6.3.2.1] and [6.3.2.2]) using Docker.
2. Installing texploration for development ([Chapter 6.3.2.1]).

The code has been written using Visual Studio Code (API and frontend) and JetBrains PyCharm([flask]). These tools have proven to be very useful and assistive during development yet are not required for further development.

6.3.2.1 Development

This type, while recommended for development-only usage, can also be used to run texploration. Additionally, the reader is asked to read Chapters 4.2, 4.3, and 4.4, [which are located in [53]], as these provide knowledge about the tech stack used. To get started, [the Texploration repository has to be cloned.]

Setting up texploration for development is a very similar process to what Docker does when building the application using the Dockerfile (Chapter 4.5 [in [53]*). As such, the reader is asked to consult that file in case anything is unclear.*

For development, the following software has to be installed and added to the system's *PATH* environment variable: Node.js 9, Yarn, [Redis] and Python 3. Additionally, global packages for both Python and Node.js have to be installed:

- virtualenv package for Python: pip3 install virtualenv

- nodemon and cross-env for Node.js/Yarn: yarn global add nodemon cross-env

To satisfy the dependencies required by the api and frontend, executing yarn install within the root folder of the corresponding [folder] suffices.

To satisfy the dependencies required by [flask], pip can be used to install the requirements by executing pip3 install -r requirements.txt. Additionally, further files need to be downloaded for some of the algorithms:

- For nltk, execute python3 -c "import nltk; nltk.download('punkt'); nltk.download('stopwords')"
- For spacy, execute [the following two commands]:
    - python3 -m spacy download en
    - python3 -c "import spacy; spacy.cli.download('en_core_web_sm'); spacy.cli.download('de_core_news_sm');"
- For Stanford's CoreNLP, the archive located at https://nlp.stanford.edu/software/stanford-corenlp-full-2018-02-27.zip needs to be downloaded and unpacked at the root folder of [flask]. Additionally, http://nlp.stanford.edu/software/stanford-english-corenlp-2018-02-27-models.jar needs to downloaded to the root folder, too.

While [flask] does not need to be started, the API and the frontend both need to be and this can be achieved by running, again in the corresponding root folder, yarn dev. [Before the API and frontend is executed, Redis must be started.]

6.3.2.2 Docker from Sources

This type, while recommended for run-only usage, can also be used for development, yet no steps have been taken to facilitate usage of the image for development. Most notably, the source folders are copied rather than mounted i.e. the image has to be rebuilt after every code change for it to be reflected in the image. Additionally, the reader is asked to read Chapter 4.5 in [53] and have a [local copy of the repository].

In addition to the above setup, the reader should have Docker and Docker Compose installed on the host machine. The build process has only been tested on Ubuntu.

To build the image, the buid.sh file in the docker repository can be executed which will build the image and start.sh to start a container based on the image. The reader is asked to consult in [53] the Chapter 4.5, especially Table 8, for further details.

6.3.2.3 Docker from Archive

This type is recommended for run-only usage of texploration and assumes the reader is in possession of the accompanying app.tar file as well as the [docker folder]. Additionally, the reader is asked to read Chapter 4.5 in [53].

The tarball is a complete image of texploration as exported by save-image.sh and can be imported by executing docker load < app.tar. To start and stop the container, the same scripts as described in Chapter 9.3.2 (and Table 8), which is located in [53], can be used.