



**School of  
Engineering**

InIT Institute of Applied  
Information Technology

## **Bachelor Thesis Computer Science**

# Developing a Framework for Visual Corpus Exploration and Comparison of Machine and Deep Learning Algorithms

---

**Authors**

---

Linus Metzler  
Nadina Siddiqui

---

**Main supervisor**

---

Mark Cieliebak

---

**Sub supervisor**

---

Don Tuggener

---

**Date**

---

08.06.2018

---





## DECLARATION OF ORIGINALITY

### Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).



The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:

Winterthur, June 6, 2018 .....

Signature:

  
.....  
  
.....  
.....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all Bachelor thesis submitted.



## Abstract

Understanding a text classification corpus takes time and is integral to the quality of any further analysis or classification performed on it.

Building on previous work, which developed a framework with an intuitive user interface for text classification, the goal of this thesis is to extend the framework.

This project focuses on designing and developing two new modules. The first one analyzes a text corpus using a robust method to parse data files and perform corpus analyses, including conducting statistics on topics, text lengths and relevant words for the entire corpus and itemized by class. Additionally, multiple sentence splitters and word tokenizers are compared visually by highlighting the differences.

The second module adds a graphical comparison to the classifier framework built during the previous work. The classification results of the models are compared and filtered, for example, by complete agreement or cases where all models have predicted the wrong label. From each filter, a few samples are presented in the user interface featuring an easy-to-read color coding for the predictions of the algorithms.

The main challenges were to prepare and present the output from various algorithms and the corpus analysis in an appealing and intuitive fashion, and to ensure the user flow feels natural. Feedback from a user test indicates these challenges were well overcome.

There are many features that could be added, such as automatically finding the best parameters for the classifiers. Moreover, the framework could be further expanded by adding new modules, for instance, the capability of inspecting and analyzing individual texts and searching for terms within the corpus.



## Zusammenfassung

Einen Textkorpus zu verstehen ist überaus zeitaufwändig und ist ein wesentlicher Qualitätsbestandteil jeglicher darauf aufbauender Analysen oder Klassifizierungen.

In einer vorhergehenden Arbeit wurde ein System entwickelt, das eine intuitive Benutzeroberfläche zur Textklassifizierung anbietet. Darauf aufbauend werden weitere Module hinzugefügt und das bestehende erweitert.

Im Rahmen des Projektes werden zwei neue Module geplant und entwickelt. Das erste analysiert einen Textkorpus mittels einer robusten Methode, Dateien einzulesen und die Analyse durchzuführen, welche Statistiken über Themen, Textlängen und relevante Wörter für sowohl den ganzen Korpus wie auch aufgeschlüsselt nach Klassen, liefert. Weiter werden verschiedene Sentence Splitters und Word Tokenizers durch das visuelle Hervorheben der Unterschiede verglichen.

Das zweite Modul fügt einen grafischen Vergleich zum Klassifizierungsframework aus der vorherigen Arbeit hinzu. Die Klassifizierungsergebnisse der Modelle werden verglichen und gefiltert, zum Beispiel, nach kompletter Übereinstimmung oder nach Fällen, bei denen alle Modelle die falsche Klasse vorhergesagt haben. Von jedem Filter werden einige Beispiele in der Benutzeroberfläche präsentiert, wobei die Vorhersagen der Algorithmen farblich kodiert dargestellt werden.

Die primären Herausforderungen waren das Aufbereiten und Präsentieren der Ausgabe von verschiedenen Algorithmen und der Korpusanalyse. Dabei wurde darauf geachtet, dass der Benutzerfluss natürlich und die Darstellung intuitiv ist. Rückmeldungen aus einem Benutzertest bestätigen, dass diese Herausforderungen gemeistert wurden.

Es gibt viele Funktionen, die hinzugefügt werden können, wie zum Beispiel, dass die besten Parameter für die Klassifizierungsalgorithmen automatisch gefunden werden. Ferner könnte das Framework um weitere Module erweitert werden, wie beispielsweise die Möglichkeit, einzelne Texte zu untersuchen und analysieren sowie nach Begriffen im Korpus zu suchen.





## Preface

After having developed a framework to simplify machine learning algorithms for text classification through an intuitive user interface, we were able to build on that framework and extend it as part of our bachelor's thesis. This lays the foundation for the next iteration of the framework and facilitates tasks revolving around text classification. We are delighted knowing the next iteration has already been planned.

It was a privilege being able to work on the same project for a year with the support of the same supervisors since it allowed us to focus on one product and develop it to a point where it is functional and useful, as well as benefitting from knowing each other's skills and talents.

Much to our delight, we were asked to submit a paper about the framework we developed to the *Fachkonferenz Technik, Architektur und Life Sciences (FTAL)* conference taking place in Lugano, Switzerland on October 18 – 19, 2018.

We would like to thank our supervisors, Mark Cieliebak and Don Tuggener, for meeting with us every week to discuss what we achieved and what was still laying ahead of us which proved to be an invaluable feedback loop. Additionally, we would like to thank Dirk von Grünigen, Jan Deriu, Pius von Däniken, and Fernando Benites for taking their time to participate in our user test and Michael Longthorn for proofreading the abstract. Special thanks go to Monika Metzler for proofreading this thesis.

We would also like to thank the Queen of England<sup>1</sup> for keeping the mood in the team light.

---

<sup>1</sup> In form of Telegram stickers [56].



## Contents

Abstract.....	1
Zusammenfassung.....	3
Preface.....	5
1 Introduction .....	11
2 Modus Operandi .....	12
3 Use Cases .....	13
3.1 Use Case 1: File Parsing.....	13
3.2 Use Case 2: Analyses.....	13
3.3 Use Case 3: Column Overview .....	13
3.4 Use Case 4: Figures on Word and Character Lengths.....	13
3.5 Use Case 5: Label Distribution.....	14
3.6 Use Case 6: Label Instances .....	14
3.7 Use Case 7: Relevant Terms.....	14
3.8 Use Case 8: Topics .....	14
3.9 Use Case 9: Tokenizer Comparison .....	14
3.10 Use Case 10: Classifier Comparison .....	14
4 Application.....	17
4.1 Walkthrough.....	17
4.1.1 Login .....	17
4.1.2 Settings .....	18
4.1.3 Dashboard.....	18
4.1.4 Files .....	21
4.1.5 Corpus Analysis .....	25
4.1.6 Classification.....	39
4.2 Backend.....	53
4.2.1 Overview .....	53
4.2.1.2 <i>Python</i> .....	53
4.2.2 Corpus Analysis .....	55
4.2.3 Classification.....	60
4.3 API.....	65
4.3.1 Overview .....	65
4.3.2 Files .....	69
4.3.3 Corpus Analysis .....	70

4.3.4	Classification.....	70
4.4	Frontend.....	71
4.4.1	Overview.....	71
4.4.2	Files.....	72
4.4.3	Corpus Analysis.....	73
4.4.4	Classification.....	73
4.5	Docker.....	75
5	User Test.....	77
5.1	Setup.....	77
5.2	Instructions.....	77
5.3	Observations.....	78
5.3.1	Step 1: Upload and Parse.....	78
5.3.2	Step 2: Delete File (Hate Speech).....	78
5.3.3	Step 3: Start Corpus Analysis (Tweets) & Step 4: Load Corpus Analysis (Newsgroups) & Step 7 (optional): Load Corpus Analysis (Hate Speech).....	79
5.3.4	Step 5: Start & Monitor Classification (Tweets).....	80
5.3.5	Step 6: Evaluate Classification (Tweets).....	80
5.3.6	General Feedback.....	80
5.3.7	General Observations.....	81
6	Related Work.....	83
6.1	Corpus Analysis.....	83
6.1.1	Kaggle.....	83
6.1.2	Lucene.....	83
6.1.3	Elasticsearch.....	83
6.2	Classifier Comparison.....	84
6.3	Conclusion.....	84
7	Conclusion.....	85
7.1	Achieved Results.....	85
7.1.1	Backend.....	85
7.1.2	API.....	85
7.1.3	Frontend.....	86
7.1.4	Docker.....	86
7.2	Future Work.....	87
7.2.1	Repository: Backend.....	87
7.2.2	Repository: Frontend.....	87

7.2.3	Repository: API.....	88
7.2.4	Module: Neural Net Builder.....	88
7.2.5	Module: Sideload Classifier.....	88
7.2.6	Module: Single Text.....	89
7.2.7	Module: Per-User Settings.....	89
7.2.8	Module: User Management.....	89
7.2.9	Module: Range Filters.....	89
7.2.10	Module: Auto-Updating Data.....	90
8	References.....	91
8.1	Bibliography.....	91
8.2	Figures.....	96
8.3	Tables.....	98
8.4	Code Snippets.....	98
9	Appendix.....	99
9.1	Official description of the task.....	99
9.2	Login Information.....	100
9.3	Installation Instructions.....	100
9.3.1	Development.....	100
9.3.2	Docker from Sources.....	101
9.3.3	Docker from Archive.....	101
9.4	Meeting Notes.....	102
9.4.1	2018-02-06.....	102
9.4.2	2018-02-22.....	103
9.4.3	2018-02-26.....	104
9.4.4	2018-03-04.....	105
9.4.5	2018-03-12.....	106
9.4.6	2018-03-19.....	106
9.4.7	2018-03-26.....	106
9.4.8	2018-04-04.....	107
9.4.9	2018-04-09.....	108
9.4.10	2018-04-16.....	108
9.4.11	2018-04-23.....	109
9.4.12	2018-04-30.....	110
9.4.13	2018-05-07.....	112

9.4.14	2018-05-14.....	112
9.5	Timetable.....	114
9.6	User Test Notes (raw).....	115
9.6.1	Test 1 - Dirk.....	115
9.6.2	Test 2 - Jan.....	117
9.6.3	Test 3 - Pius.....	119
9.6.4	Test 4 - Fernando.....	121
9.7	License Information.....	124

## 1 Introduction

Analyzing a text corpus and performing classification on it has become increasingly popular in recent years, mainly in combination with machine and deep learning [1], [2, pp. 513-520], [3]. Existing solutions (see Chapter 6) are often based on a command-line interface which do not allow the output to be displayed graphically and require the user to be comfortable interacting with command-line tools and being able to create visualizations, either mentally or using additional tooling. Furthermore, comparing algorithms on the same text corpus is time-intensive as the output format has to be unified first and distinctive samples have to be found.

The goal of this thesis is to extend a classification framework [4] by adding new functionality in the form of two modules: corpus analysis and classifier comparison. The first one analyzes a text corpus using a robust method to parse data files and perform corpus analyses, including conducting statistics on topics, text lengths and relevant words for the entire corpus and itemized by class. The second module adds a graphical comparison to the classifier framework built during the previous work. The classification results of the models are compared and filtered, for example, by complete agreement or cases where all models have predicted the wrong label. Please refer to Chapter 9.1 for the official task description.

As the framework, called *texploration*, described in this thesis is aimed at researchers and other professionals, the reader is expected to be familiar with the terms used. Nevertheless, a short overview is provided as follows: A *corpus* consists of many *texts* which are stored in a *comma-separated value file (CSV)*. A corpus has multiple *topics* which can be inferred by appropriate algorithms. A *tokenizer* splits a string into *tokens*, and depending how it is configured, a token is either a sentence or word whereas a word is either a single word or a string of characters which are treated like a word such as emojis.

After briefly touching on the modus operandi (Chapter 2) and the application's use cases (Chapter 3), a walkthrough (Chapter 4.1) of *texploration* will follow in order to become familiar with the application. The details of the individual repositories making up *texploration* are explained in the subsequent chapters (Chapters 4.2, 4.3, 4.4, 4.5). In order to verify the claim of *texploration*'s user-friendliness, a user test was performed which is analyzed in (Chapter 5). Once *texploration* has been discussed in detail, it is put into context by looking at related works (Chapter 6) before concluding the thesis with the achieved results (Chapter 7.1) and an outlook at future work (Chapter 7.2).

## 2 Modus Operandi

The work on exploration started on February 6<sup>th</sup>, 2018 with the first meeting about the project between the team and the supervisors. Most of the code was written between then and the end of April when the coding team declared a “feature freeze”. The next month was used to fix some errors and write the thesis.

The workload was split by repository between the two people in the team. One person was responsible for the backend and the other for the remaining three repositories. Like this, the members could work mostly independent of each other. The team held an hour-long meeting most weeks followed by another hour-long meeting with the supervisors thus effectively working in one-week sprints. Even though a timetable was created at the beginning, the next steps were usually decided during those meetings. Those steps are listed in the meeting notes (Chapter 9.4) under “ToDo”.

Due to a two weeks illness of half the coding team, the project started later than planned which caused some tighter scheduling than initially planned but the lost time could eventually be made up over the next two months.

Apart from the meetings the team was in regular contact via Telegram<sup>2</sup>, sending back and forth over 3000 messages over the course of this project. In most cases, this worked very well, but on some rare occasions, a Skype call was necessary.

Since the API and the frontend are dependent on the backend, the latter was usually ahead. As soon as a new module was complete, the other person was informed about it by text and would, in turn, feed back error reports or request structural changes on the output. Once the module was implemented in the GUI, the person responsible for the backend would test it and note possible improvements.

---

<sup>2</sup> Telegram is a cloud-based instant-messaging platform. [54]



## 3 Use Cases

To ensure practical applicability of texploration, a number of use cases were devised and serve as the foundation for many of the available features.

### 3.1 Use Case 1: File Parsing

*As a user, I want to import a CSV file in any format I want.*

Due to the lack of standardization of the CSV format and the variety of choices in terms of delimiters and escape and quote characters [5], texploration allows the user to upload any CSV file which is then parsed using a best-effort strategy (Chapter 4.3.2) by the “Smart CSV” feature (Figure 13).<sup>3</sup> By means of the “Smart CSV” feature the burden of the file format is removed from the user and placed on the application allowing the user not having to concern themselves with low-level details.

### 3.2 Use Case 2: Analyses

*As a user, I want to choose which analyses I would like to run.*

When starting a corpus analysis, the general assumption by texploration is the user wants to run every analyzer. However, there are instances where only certain analyzers are needed or sensible. To satisfy both needs, there are a set of advanced options (Figure 19) when starting an analysis. These are hidden by default based on the aforementioned assumption yet can easily be revealed to toggle certain analyzers.

### 3.3 Use Case 3: Column Overview

*As a user, I want to see an overview of all columns.*

Once the user is using texploration, there should be no need to open the raw input file. To account for the instances where a user wishes to recall the contents of a certain column, a sample of the data is shown during corpus analysis (Figure 25). This data sample is based on the top rows of the input file, yet it attempts to showcase the different values in the file. An example is the case of the *binary* columns where instead of the first few rows, the two different values in that column are shown.

### 3.4 Use Case 4: Figures on Word and Character Lengths

*As a user, I want to see per-label analysis of word and character lengths.*

One of the key differences between different types of corpora is the minimum / average / maximum number of words / characters per text.<sup>4</sup> These key figures are displayed at the top of the corpus analysis (Chapter 4.1.5.2.1) to enable the user to see these figures immediately upon loading the corpus analysis while being able to focus on more in-depth and complex analyses further down the page.

---

<sup>3</sup> One corpus frequently used during testing uses a mixed choice of quote and escape characters within the same file which is thus not feasible to automatically parse “correctly” in the sense a human would expect.

<sup>4</sup> Example: the min/avg/max words for the Tweets corpus are 1/20/35 whereas the same figures for the 20 Newsgroups corpus are 15/314/13540.

These figures come in two forms: one table displays the figure corpus-wide while the other table displays the figures per-label.

### 3.5 Use Case 5: Label Distribution

*As a user, I want to know the distribution of labels in a corpus.*

To give the user a way to quickly see which labels are present and how frequent these labels are, a chart (Figure 29) displays the distribution of labels graphically (in addition the Stats table (Figure 29)). This chart takes into account the column mapping defined for the corpus and as a consequence e.g. only the number of texts, for which a binary label is true, is shown.

### 3.6 Use Case 6: Label Instances

*As a user, I want to see the different combination of labels.*

Depending on the type of corpus one text can have more than one label. The instance distribution chart (Figure 29) displays the label instances for texts which have at least one label, easily allowing the user to see which instances appear often in a corpus. An instance is defined as the set of labels a given text does have (Figure 30, Figure 31, Figure 32).

### 3.7 Use Case 7: Relevant Terms

*As a user, I want to see per-label analysis of relevant terms.*

Apart from the figures on words and characters corpus-wide and per-label (Chapter 3.4), the most relevant terms are displayed in the form of a word cloud (Chapter 4.1.5.2.2) allowing the user to quickly see which terms are relevant either corpus-wide or for any label or combinations of multiple labels.

### 3.8 Use Case 8: Topics

*As a user, I want to know what topics are present in a corpus.*

Topic modeling is a useful tool to see what kind of texts are present in a corpus and to see how they belong together [6]. By showing the user the results of user-configurable (Figure 19) topic modeling, texploration enables the user to see the topics present in a corpus.

### 3.9 Use Case 9: Tokenizer Comparison

*As a user, I want to know the differences between tokenizers on word- and sentence-level.*

When tokenizing words and sentences, different algorithms yield different results (Chapters 4.2.2.3.3 and 4.2.2.3.4) and different usages may call for a different tokenizer. To facilitate the, at times nuanced, comparison of different tokenizing algorithms, texploration displays samples where the tokenizers differ most using color-coding to facilitate seeing the differences (Chapters 4.1.5.2.5 and 4.1.5.2.4).

### 3.10 Use Case 10: Classifier Comparison

*As a user, I want to compare different classifiers.*

In addition to comparing tokenizers (Chapter 3.9), classifiers can be compared, too. To accomplish this in an informative and visually pleasing way, texploration displays several

sets (Chapter 4.1.6.2.2) highlighting the similarities and differences between individual tokenizers on a per-text basis and allows the user to download the complete comparison for further evaluations. The desire for this use case was strongly supported by the feedback from the user test (see Chapter 5.3).



## 4 Application

Texploration consists of three main parts (and code repositories), the backend (Chapter 4.2) the API (Chapter 4.3), and the frontend (Chapter 4.4) as well as a separate repository for the Docker image (Chapter 4.5). These four repositories are described in this chapter after presenting a walkthrough (Chapter 4.1) of texploration to get familiar with the application and its concepts and features.

### 4.1 Walkthrough

*Note: In a few instances, a different corpus is used to highlight a sub-section of a page than was used for the complete page. The intention is to show a corpus for which the chapter in question is most relevant and provides information. See Chapter 7.1.1 for details on why some corpora exhibit better results for certain chapters than other corpora do for the same chapters.*

*Note: one of the corpora used for analysis in this chapter contains hate speech and as a result some screenshots contain profanity.*

Visualizations help conveying data (and the resulting information), and to give the reader a visualization of subsequent chapters, a walk through texploration is presented in this chapter, following the order of how the application is typically used.

*Note: In this chapter, a screenshot will be followed by the accompanying text in which case the corresponding figure will not be referenced. In any other case and in case of possible ambiguity, the corresponding figure will be referenced.*

#### 4.1.1 Login

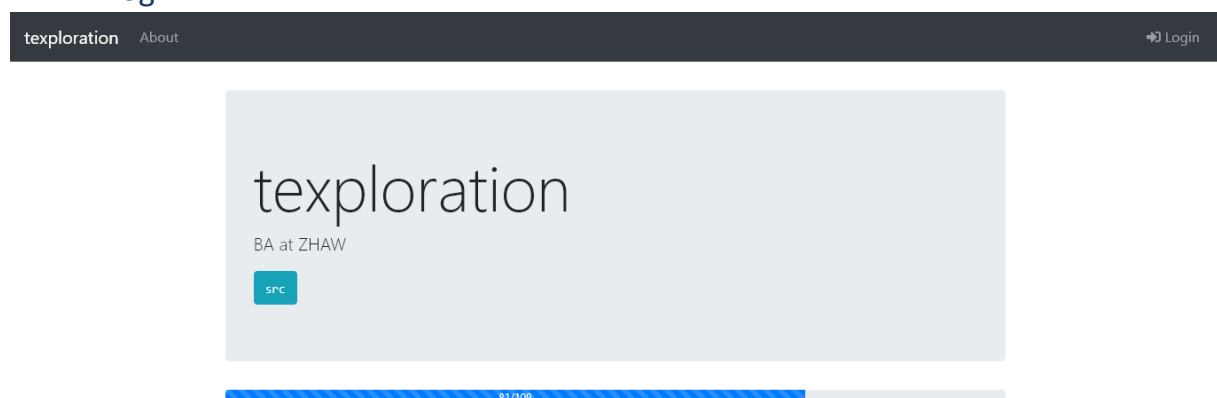
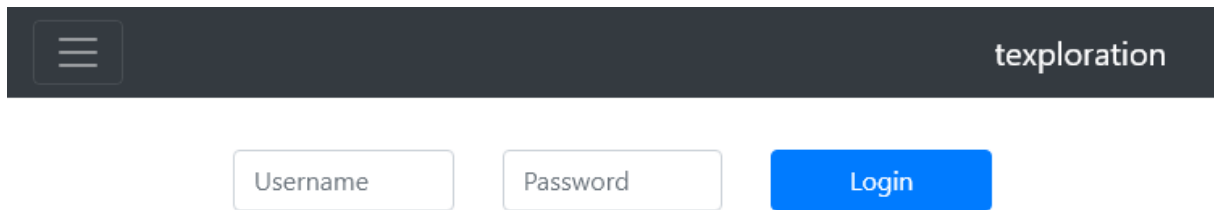


Figure 1: Home screen

Since the whole application requires a user to be logged in, the home screen is very basic and serves merely as a placeholder<sup>5</sup>.

---

<sup>5</sup> This also applies to the omitted “About” page.



The screenshot shows a dark header bar with a hamburger menu icon on the left and the text 'texploration' on the right. Below the header, there is a login form with three elements: a text input field labeled 'Username', another text input field labeled 'Password', and a blue button labeled 'Login'.

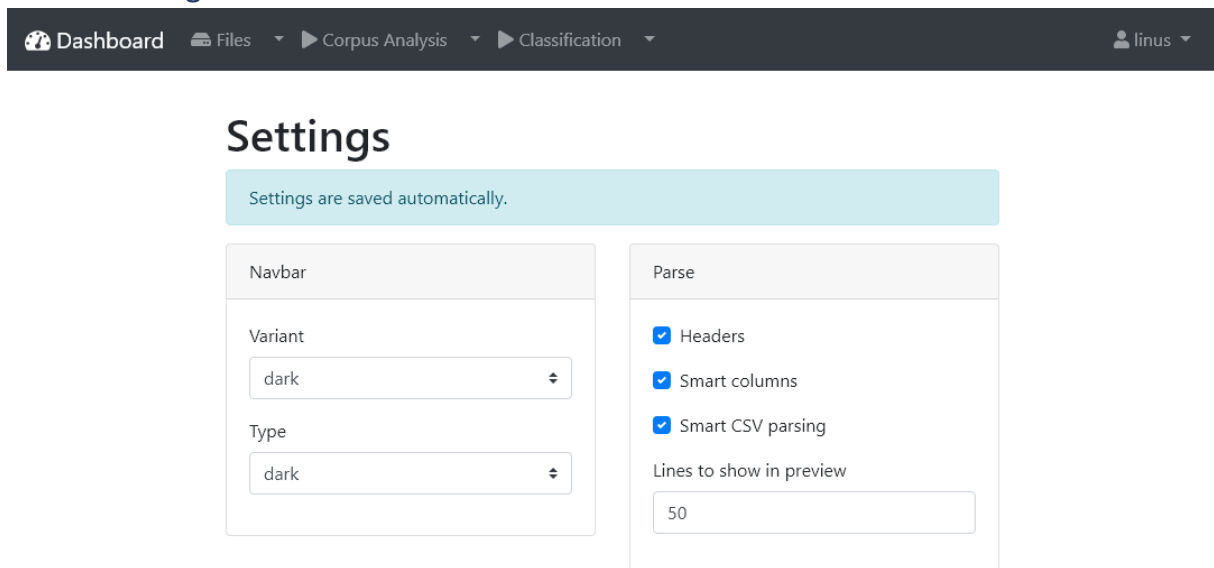
Figure 2: Login screen

The user will then click on “Login” in the top right and will then be greeted by a simple username-and-password login form.

Assuming the user has been authenticated successfully, a redirect will happen to the dashboard.

In case the login form was triggered by an authentication failure of the user, the user will be redirected to the originally intended location.

#### 4.1.2 Settings

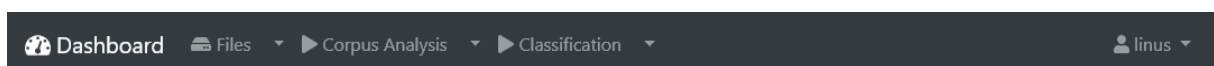


The screenshot shows the 'Settings' page. At the top, there is a dark navigation bar with 'Dashboard', 'Files', 'Corpus Analysis', and 'Classification' as menu items, and a user profile 'linus'. Below the navigation bar, the title 'Settings' is displayed. A light blue banner indicates 'Settings are saved automatically.' The settings are organized into two columns: 'Navbar' and 'Parse'. The 'Navbar' section has two dropdown menus for 'Variant' and 'Type', both set to 'dark'. The 'Parse' section has three checked checkboxes: 'Headers', 'Smart columns', and 'Smart CSV parsing'. Below these is a text input field for 'Lines to show in preview' with the value '50'.

Figure 3: Settings screen

Before looking at the main part of the application, a quick look will be taken at the settings. These allow a user to override defaults set by the application as well as to customize the application by setting a different color for the top-bar navigation.

#### 4.1.3 Dashboard



The screenshot shows the top navigation bar of the dashboard. It features a dark background with a hamburger menu icon on the left, followed by the text 'Dashboard'. To the right are menu items: 'Files', 'Corpus Analysis', and 'Classification', each with a dropdown arrow. On the far right is a user profile icon and the name 'linus' with a dropdown arrow.

Figure 4: Top-bar navigation for Dashboard

# Dashboard

Hello, linus

Upload	
<p><b>TV Show</b> <span style="float: right;">uploaded 3 hours ago</span></p> <div style="display: flex; gap: 5px;"> <span style="border: 1px solid red; padding: 2px 5px; color: red;">Delete</span> <span style="border: 1px solid blue; padding: 2px 5px; color: blue;">Parse</span> </div>	
<p><b>20 News Groups</b> <span style="float: right;">uploaded 3 days ago</span></p> <div style="display: flex; gap: 5px;"> <span style="border: 1px solid red; padding: 2px 5px; color: red;">Delete</span> <span style="border: 1px solid orange; padding: 2px 5px; color: orange;">Reset parse options</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start corpus analysis</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start classification</span> </div> <p style="font-size: 12px; margin-top: 5px;">20newsgroups.csv, 11314 rows with the following column types: [ "text", "label" ]</p>	
<p><b>Tweets without labels</b> <span style="float: right;">uploaded 4 days ago</span></p> <div style="display: flex; gap: 5px;"> <span style="border: 1px solid red; padding: 2px 5px; color: red;">Delete</span> <span style="border: 1px solid orange; padding: 2px 5px; color: orange;">Reset parse options</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start corpus analysis</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start classification</span> </div> <p style="font-size: 12px; margin-top: 5px;">tweets_test_no_label.csv, 3609 rows with the following column types: [ "id", "lang", "text" ]</p>	
<p><b>Tweets</b> <span style="float: right;">uploaded 5 days ago</span></p> <div style="display: flex; gap: 5px;"> <span style="border: 1px solid red; padding: 2px 5px; color: red;">Delete</span> <span style="border: 1px solid orange; padding: 2px 5px; color: orange;">Reset parse options</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start corpus analysis</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start classification</span> </div> <p style="font-size: 12px; margin-top: 5px;">en_full_headers.tsv, 18044 rows with the following column types: [ "id", "lang", "label", "text" ]</p>	
<p><b>Hate Speech</b> <span style="float: right;">uploaded 5 days ago</span></p> <div style="display: flex; gap: 5px;"> <span style="border: 1px solid red; padding: 2px 5px; color: red;">Delete</span> <span style="border: 1px solid orange; padding: 2px 5px; color: orange;">Reset parse options</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start corpus analysis</span> <span style="border: 1px solid green; padding: 2px 5px; color: green;">Start classification</span> </div> <p style="font-size: 12px; margin-top: 5px;">hate.csv, 159571 rows with the following column types: [ "id", "text", "ignore", "binary", "binary", "binary", "binary", "binary" ]</p>	
Upload	

Figure 5: Dashboard screen

The dashboard is the user’s central hub. It lists all files a user has uploaded. Additionally, there are two buttons<sup>6</sup> to upload a new file.

*Note: since the dashboard links to other parts’ functionalities, these are described in the corresponding chapters.*

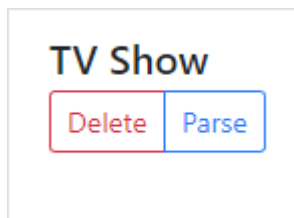


Figure 6: A file which has not yet been parsed

If a file has only been uploaded but not yet parsed, the file can either be deleted or parsed.

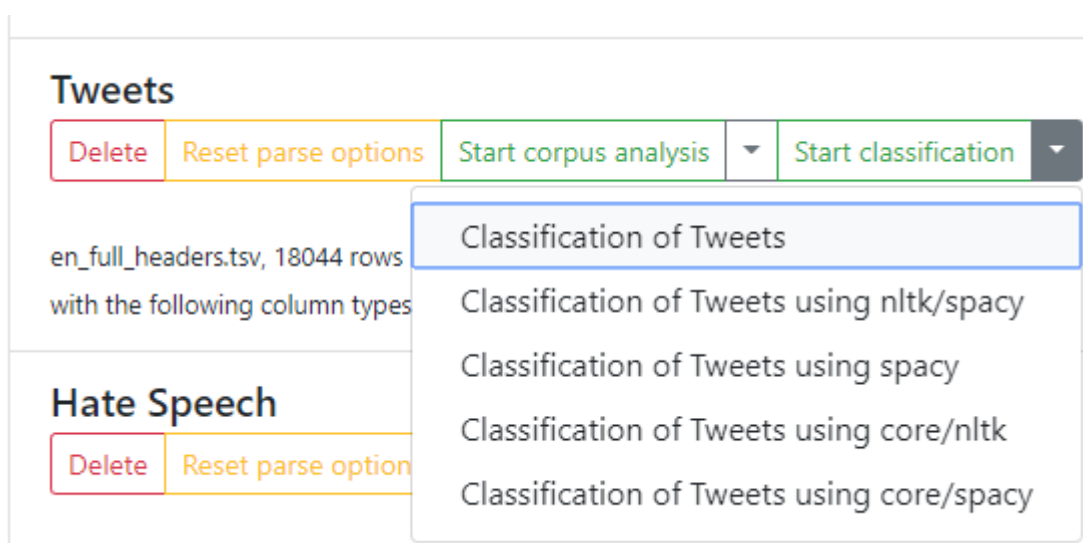


Figure 7: Classification dropdown menu for a file

If the user has already used a given file for a corpus analysis or classification, links are shown in the dropdown menu next to the corresponding button to quickly navigate to the analysis or classification in question. Additionally, the file can be deleted (which will also delete corpus analyses and classification tasks associated with that file) and the parse options can be reset, resetting it to the state it was in when it was uploaded.

In the example below, the “Tweets” file’s corpus has already been analyzed (indicated by the dropdown menu next to the green “Start corpus analysis” button) and several classification tasks have already been run which are listed in the open dropdown menu next to “Start classification”.

<sup>6</sup> Except for when there are no files. In that case only one button is displayed.



## 4.1.4 Files

### 4.1.4.1 List

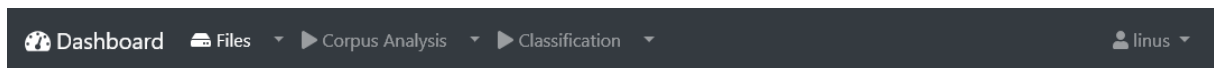


Figure 8: Top-bar navigation for Files

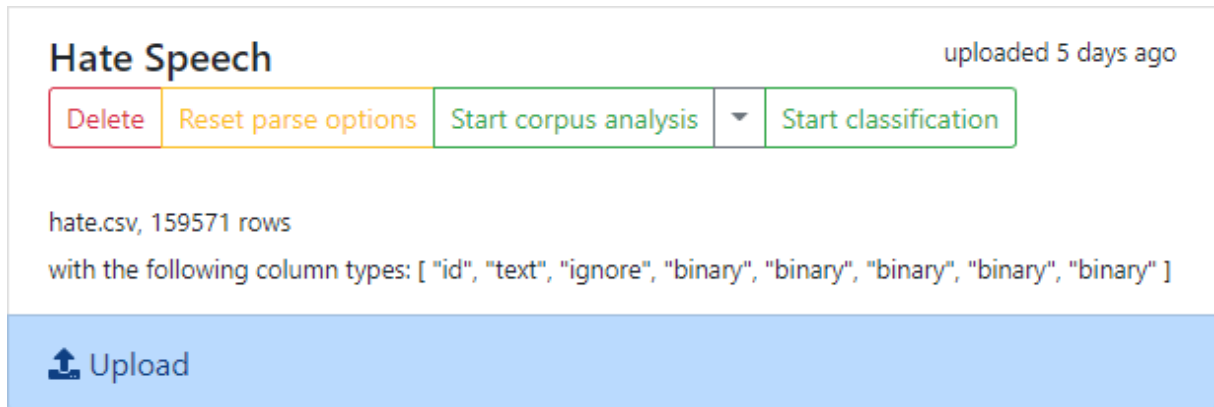


Figure 9: A single file in the file list and the upload button

When using the top-bar navigation to navigate to “Files”, the user is presented with the same screen as on the dashboard save for the grey rectangle above the file list.

### 4.1.4.2 Upload

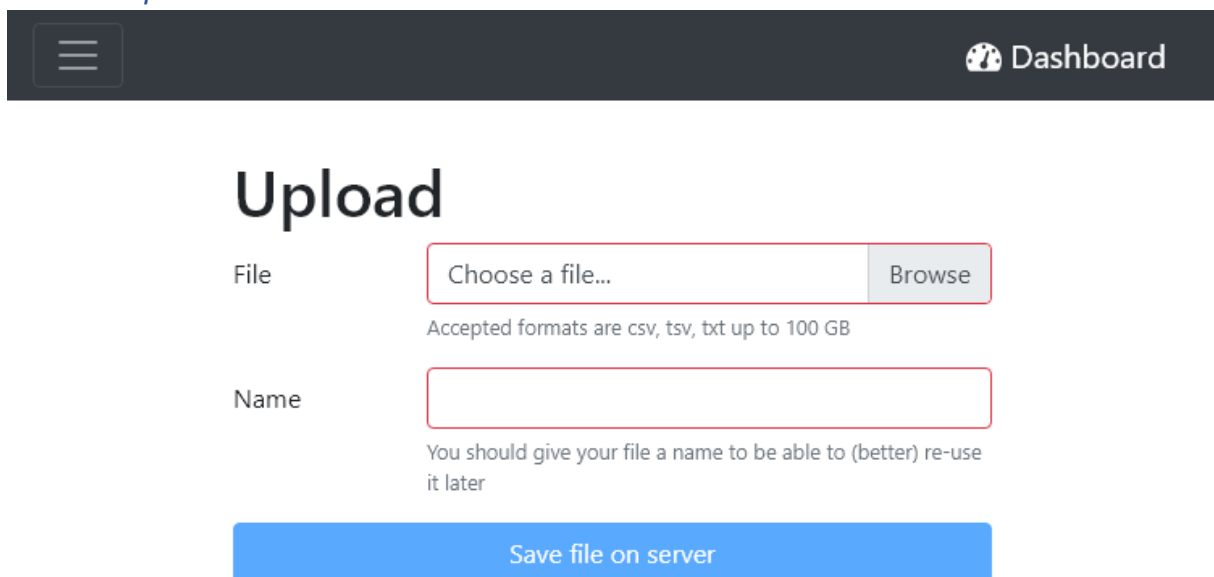


Figure 10: Upload screen

When the user clicks on the “Upload” button, a simple form is loaded consisting of a file upload input and a field to name the file. The latter is populated automatically using the filename of the uploaded file.

The UX pattern behind the automatic population of the name is used through the application; whenever the user *has* to name something, the name of the underlying resource is used to populate the input field which means the user *can* name something, yet a sensible default is provided.

Name

You should give your file a name to be able to (better) re-use it later

[Save file on server](#)

Figure 11: File name and button with passing validation state

Once both inputs are filled in, the red borders turn green and the button changes from the disabled state to enabled. This UI pattern can be found in all forms of the application; every input's border indicates its validation state and the button's disabled state indicates whether the form can be submitted.



Figure 12: Persistent toast notification while a file is being uploaded

Since the file can be quite large in size, a persistent notification will be displayed to let the user know the file is uploading. This notification, also called toast, shows up in the top right and is again a recurring UI pattern, yet in two different forms:

1. Toasts which automatically expire within a few seconds. This variant is used for e.g. success messages. These notifications have a progress bar at the top to indicate their expiry.
2. Toasts which are persistent ("sticky") until programmatically removed (or the user clicks on it to dismiss it). This variant is used for long-running tasks e.g. upload a file.

#### 4.1.4.3 Parse

## Parse TV Show

Headers	CSV Parsing
<input checked="" type="checkbox"/> Header row <input checked="" type="checkbox"/> Smart headers	<input checked="" type="checkbox"/> Smart CSV
<a href="#">Guess CSV format</a>	
<a href="#">Reset options</a>	

Figure 13: Parse options for a file

After the file has been uploaded, the user can set the parse options. CSVs are not standardized and delimiters, and quote and escape characters can vary widely. As to not bother the user with these nuances, there is an option called “Smart CSV” to let the application figure out the format.

Additionally, the user can indicate whether the file has headers and if the column mapping should be performed automatically. The column mapping defines which column contains what kind of content which is crucial for subsequent tasks.<sup>7</sup>

---

<sup>7</sup> See Table 4 for more details.

# Parse TV Show

## Columns

At least one column has to be of type **text**

### Column type

	<input type="text" value="ID"/>	Please select the type for this column
Season	<input type="text" value="Numeric label"/>	Please select the type for this column
EpisodeNo	<input type="text" value="Numeric label"/>	Please select the type for this column
Title	<input type="text" value="ID"/>	Please select the type for this column
AirDate	<input type="text" value="Date label"/>	Please select the type for this column
Writers	<input type="text" value=""/>	Please select the type for this column
Director	<input type="text" value="String label"/>	Please select the type for this column
SEID	<input type="text" value="ID"/>	Please select the type for this column

### Preview of column

<input type="text" value="0"/>
<input type="text" value="1.0"/>
<input type="text" value="1.0"/>
<input type="text" value="Good News, Bad News"/>
<input type="text" value="July 5, 1989"/>
<input type="text" value="Larry David, Jerry Seinfeld"/>
<input type="text" value="Art Wolff"/>
<input type="text" value="S01E01"/>

## Parse Results

### Data

The first true rows are shown.

	↑↓ Season	↑↓ Episode No	↑↓ Title	↑↓ Air Date	↑↓ Writers	↑↓ Director	↑↓ SEID
0	1.0	1.0	Good News, Bad News	July 5, 1989	Larry David, Jerry Seinfeld	Art Wolff	S01E01

Figure 14: Parse results and column mapping for a file

The API then performs an analysis of the CSV file, attempts to find the optimal set of CSV options and makes an educated guess about the column mapping. The user is then presented with the results and can modify the mapping. This screen contains the following elements, top-down from left to right:

1. (optional) A button to quickly set non-mapped columns as string labels.
2. (optional) A button to quickly ignore non-mapped columns.

3. (optional) An alert if no column has been set to be the column containing text which is required for corpus analysis and text classification.
4. A three-column list of all columns found in the input file: the left-most column contains the column name, the middle column offers a dropdown to choose the column's type, and the last column offers a preview of that column.
5. A big blue button to save the CSV and the chosen column mapping.
6. A small red button to start-over with parsing the CSV and analyzing the columns for the mapping.
7. (optional and not show) A list of errors encountered when parsing the CSV file. Whether there are errors is also shown in a toast notification.
8. A table containing the first 50 rows<sup>8</sup> of the CSV as it was parsed.

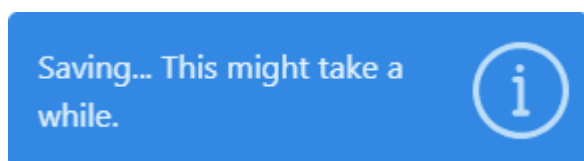


Figure 15: Persistent notification while a file is being saved



Figure 16: Notification after the file has been saved successfully

Should the user decide to be satisfied with how the file was parsed and the column mapping was performed, they will click the aforementioned “Save” button. This prompts the API to parse the complete CSV<sup>9</sup> and write it to disk using the format options. Since this conversion depends on the file size, this process might require a certain amount of time<sup>10</sup> and a persistent toast notification is displayed to inform the user the process is still ongoing. This toast will then be replaced by a green notification indicating success and the user is redirected to the list of files.

#### 4.1.5 Corpus Analysis

To gain insights and a deeper understanding of a corpus, the user can analyze the corpus using the Corpus Analysis feature. This consists of starting (Chapter 4.1.5.1), and optionally configuring, the analysis followed by examining (Chapter 4.1.5.2) the analysis.

<sup>8</sup> This number can be configured through the global settings (Chapter 4.1.2).

<sup>9</sup> In the previous steps, only the first 2'000 lines were parsed for performance reasons. See Chapter 4.3.2.

<sup>10</sup> Example for a 1.28 GB CSV file with 1 ID column, 1 text column, and 6 binary columns: just under a minute.

### 4.1.5.1 Start

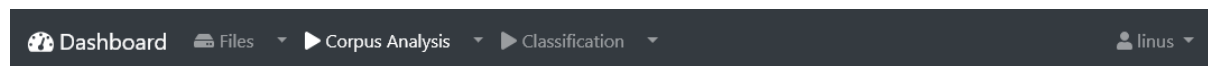


Figure 17: Top-bar navigation for Corpus Analysis

# Start

Choose a file

You can also [upload a file](#)

Name

You should give your corpus a name to be able to (better) re-use it later

Advanced options

Would you like to set advanced options for the analysis?

[Start](#)

Figure 18: Start screen

There are two ways to start a new corpus analysis:

1. By clicking on the green “Start corpus analysis” button in the list of files. In this case the two inputs at the top will be pre-populated with the corresponding file.
2. By clicking on “Corpus Analysis” in the top-bar navigation. In this case the file has to be chosen using the dropdown.

By default, the advanced options are hidden and starting a corpus analysis consists of merely choosing a file<sup>11</sup> and clicking the “Start” button.

---

<sup>11</sup> The name is, again, auto-populated using the file name.

## Start

Choose a file

You can also [upload a file](#)

Name

You should give your corpus a name to be able to (better) re-use it later

Advanced options

Would you like to set advanced options for the analysis?

**Analyzers**

Choose analyzers to run

- general stats
- topic modeling
- sentence tokenizer
- word tokenizer

**Parameters for topic modeling**

Number of topics

Number of words per topic

Number of features

enter 0 for no features

Max. size of n-grams

[Start](#)

Figure 19: Start screen with advanced options active

Using the advanced options, the user can set various parameters for topic modeling (Chapter 4.2.2.3.2) and (de-)activate certain analyzers.

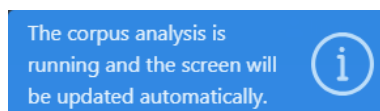


Figure 20: Persistent notification while the analysis is running

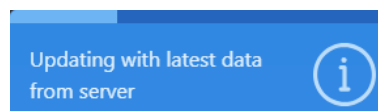


Figure 21: Notification when new data has been loaded

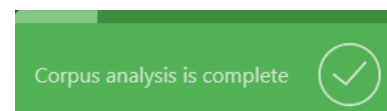


Figure 22: Notification when the analysis has been completed

The corpus analysis can take up to a few minutes yet intermediary output is provided which is then automatically loaded by the frontend. In order to make this transparent to the user and not to confuse them as to why only a partial output is displayed, a set of toast notifications are used:

- While the analysis is running, a sticky toast is displayed to indicate the analysis is still running and the screen will be updated periodically.
- Whenever new data is available, an expiring toast is shown to let the user know new data is available and was loaded.
- Once the analysis is complete, another expiring toast will be shown informing the user of the analysis' new state.

### 4.1.5.2 View

Note: since screenshots in this chapter are rather tall (up to ten times their width), no screenshot of the complete page is shown.

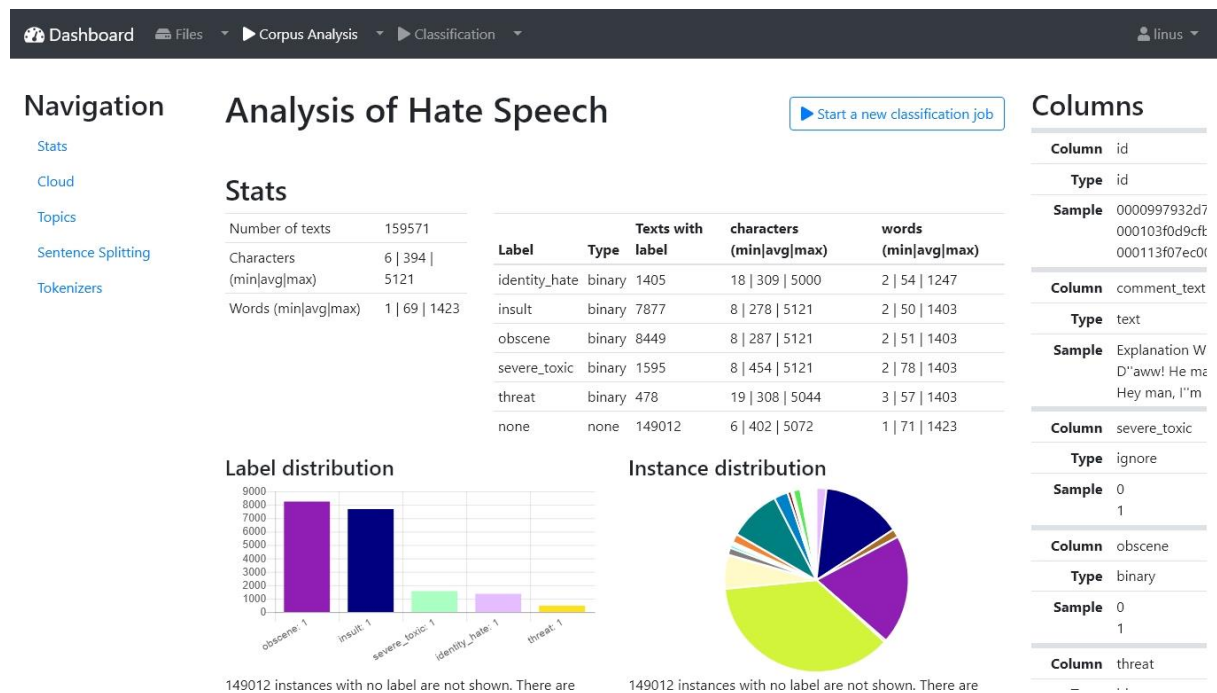


Figure 23: Top part of the corpus analysis screen for the hate speech corpus

## Navigation

- Stats
- Cloud
- Topics
- Sentence Splitting
- Tokenizers

Figure 24: Left sidebar navigation



## Columns

<b>Column</b>	id
<b>Type</b>	id
<b>Sample</b>	0000997932d7 000103f0d9cft 000113f07ec00
<b>Column</b>	comment_text
<b>Type</b>	text
<b>Sample</b>	Explanation W D''aww! He ma Hey man, I''m
<b>Column</b>	severe_toxic
<b>Type</b>	ignore
<b>Sample</b>	0 1
<b>Column</b>	obscene
<b>Type</b>	binary
<b>Sample</b>	0 1

Figure 25: Right sidebar containing a data sample of the hate speech corpus

The corpus analysis uses a three-column layout (see Figure 23):

- The left sidebar (Figure 24) contains an in-page navigation enabling the user to quickly navigate to a section within the analysis. This sidebar is always visible when scrolling.
- The main content area in the middle which will be explained in detail.
- The right sidebar (Figure 25) displaying the contents of the underlying CSV in a summarized fashion. For every column, its type (from the column mapping) and a data sample are shown.

## Analysis of Tweets

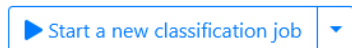


Figure 26: Corpus analysis header with classification start button and a (closed) dropdown menu

Next to the main content's heading, which corresponds to the analysis' name entered when it was started, is a button to quickly start a classification task using the same file. Should a classification have already been run, a dropdown menu is offered to load the classification task (Figure 26).

# Analysis of 20 News Groups

[▶ Start a new classification job](#)

For every column, only the top 5 values are shown. Show all?

Figure 27: Corpus analysis header with the (unchecked) checkbox to show all values

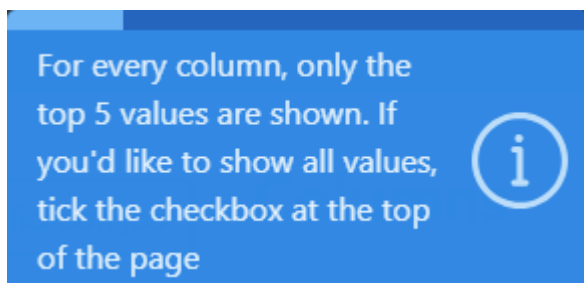


Figure 28: Toast notification indicating the number of values per label column has been capped

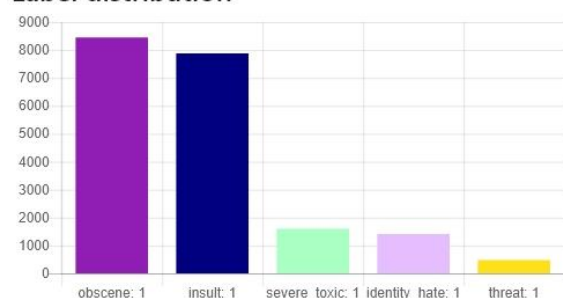
For some corpora<sup>12</sup>, there might be a greater-than-anticipated number of label values in a column. To prevent the user’s browser from freezing or even crashing, only the five most frequent values for a label column are shown by default. Should that be the case for an analysis, a notification will alert the user to this fact (Figure 28) and a checkbox is provided to show all values (Figure 27).

## 4.1.5.2.1 Stats

### Stats

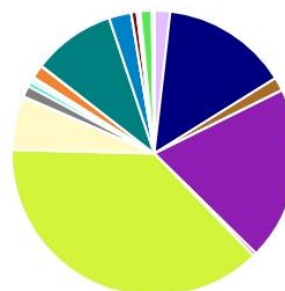
Number of texts	159571	<b>Label</b>	<b>Type</b>	<b>Texts with label</b>	<b>characters (min avg max)</b>	<b>words (min avg max)</b>
Characters (min avg max)	6   394   5121	identity_hate	binary	1405	18   309   5000	2   54   1247
Words (min avg max)	1   69   1423	insult	binary	7877	8   278   5121	2   50   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
		none	none	149012	6   402   5072	1   71   1423

#### Label distribution



149012 instances with no label are not shown. There are 159571 instances in total.

#### Instance distribution



149012 instances with no label are not shown. There are 159571 instances in total.

Figure 29: The Stats section for the hate speech corpus

The first section of the corpus analysis displays general statistics on the corpus. The two tables in the first row give information about the number of texts in general (left table)

<sup>12</sup> Or in the case of a mis-configuration of the column mapping.

or per label (right table), the minimum, average, and maximum number of words and characters per text as well as (right table only) the label type.

The second row of this section shows the label distribution, i.e. the number of texts with a given label, graphically as well as a graphical representation of the instances in the corpus. An instance is the set of labels for a given text. See the following figures (Figure 30, Figure 31, Figure 32) for examples of what constitutes an instance in the case of a corpus of hate speech with binary labels.

Labels, in every part of the application, are always colored using the same algorithm (Chapter 4.4.1.4) to ensure the same label (e.g. “obscene” or “positive”) is always colored the same way which allows the user to recognize a label graphically without having to read the text.

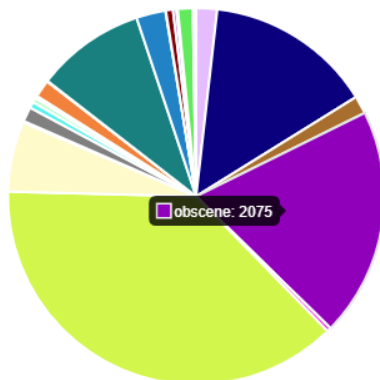


Figure 30: 2075 texts have the “obscene” label and no other label

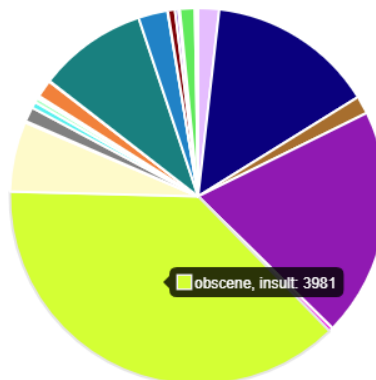


Figure 31: 3981 texts have both, “obscene” and “insult” as labels which does not include the 2075 texts which only have the “obscene” label set

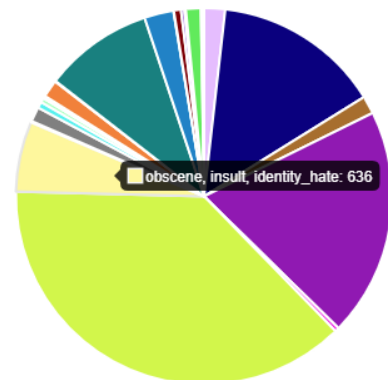


Figure 32: 636 texts have the three labels “obscene”, “insult”, and “identity\_hate” which, again, does not include either the 2075 “obscene” or the 3981 “obscene, insult” instances.



## Topics

<p>Topic 1</p> <p>write article people say think make know good time thing</p>	<p>Topic 2</p> <p>use problem work mac run monitor machine apple color computer</p>	<p>Topic 3</p> <p>team player year play win good season hockey nhl fan</p>	<p>Topic 4</p> <p>window run program application win version font screen problem driver</p>	<p>Topic 5</p> <p>nntp_post host edu university article distribution_ world write reply com usa</p>	<p>Topic 6</p> <p>key chip encryption clipper clipper_chip bit government security algorithm encrypt</p>
<p>Topic 7</p> <p>christian church religion bible christianity belief believe christ truth faith</p>	<p>Topic 8</p> <p>drive scsi disk ide controller format problem tape cable mac</p>	<p>Topic 9</p> <p>sale sell price offer include new interested condition manual ask</p>	<p>Topic 10</p> <p>god atheist believe sin faith man hell exist say love</p>	<p>Topic 11</p> <p>car engine buy model price dealer mile speed look good</p>	<p>Topic 12</p> <p>card driver video color graphic mode bit support monitor board</p>
<p>Topic 13</p> <p>file format program directory image convert gif disk help ftp</p>	<p>Topic 14</p> <p>gun firearm weapon law crime criminal handgun kill people police</p>	<p>Topic 15</p> <p>bike motorcycle ride dod dog advice buy mile road turn</p>	<p>Topic 16</p> <p>armenian turk serdar_argic turkish armenia turkey greek people russian muslim</p>	<p>Topic 17</p> <p>game win score baseball playoff play run hockey tv fan</p>	<p>Topic 18</p> <p>space nasa launch moon sci science orbit earth satellite mission</p>
<p>Topic 19</p> <p>thank mail address advance email information know info post help</p>	<p>Topic 20</p> <p>israel israeli arab jew palestinian jewish state kill peace attack</p>				

Figure 34: Complete card deck of all topics for the 20 News Groups corpus



Figure 35: Excerpt of the topics for the 20 News Groups corpus

Each card represents one topic, identified by a topic number in the grey header, followed by a plain list of the words in that topic.

Topic number	Topic words	Number of texts	Number of texts per label
1	write, article, people, say, think, make, know, good, time, thing	4596	
2	use, problem, work, mac, run, monitor, machine, apple, color, computer	2876	
3	team, player, year, play, win, good, season, hockey, nhl, fan	1023	
4	window, run, program, application, win, version, font, screen, problem, driver	343	

Figure 36: A part of the topic table showing the topic words, number of texts, and the number of texts per label for a topic

While the cards provide an overview, a user may want to explore the modeling in a more detailed fashion which can be done using the table following the deck. Apart from displaying the topic number and words, it also displays the number of texts in a given topic and a mini-chart of the number of texts per label contributing to that topic.

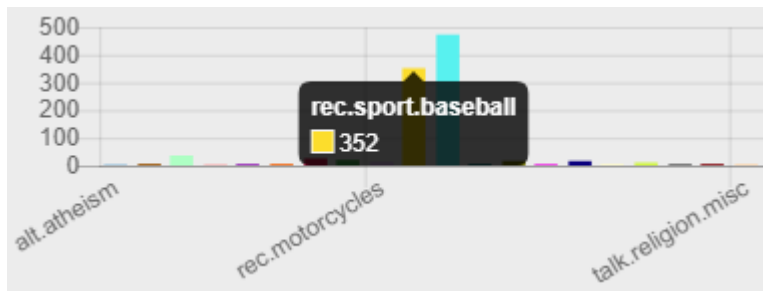


Figure 37: Detailed view of the mini-chart for topic 3

As an example, there is an interesting comparison between topic 1 and topic 3: topic 1 appears to be a very general topic, containing texts from every label which is self-explanatory when looking at the words in that topic. Topic 3, however, contains words which hint at sports discussions which is supported by the mini-chart (Figure 37) where two bars dominate the chart (*rec.sport.baseball* and *rec.sport.hockey*). Since there is not a lot of space, most labels are missing from the chart yet when hovering over a bar, the label and the count is displayed in a tooltip (Figure 37).

### 4.1.5.2.4 Sentence Splitting

## Sentence Splitting

Show  Highlight

all differences  non-agreement  full agreement



Figure 38: An excerpt of the sentence splitting comparison for hate speech

The goal of the section on sentence splitting is to highlight the differences between different algorithms. To accomplish that, sentence splitting is performed on the corpus and the ten texts with the biggest difference in lines i.e. sentences are displayed in tabular form, where each column represents one algorithm. The background color indicates whether the algorithms agree (green) or not (red). Furthermore, the header contains the corresponding text and the footer indicates the number of lines the algorithms differ in.



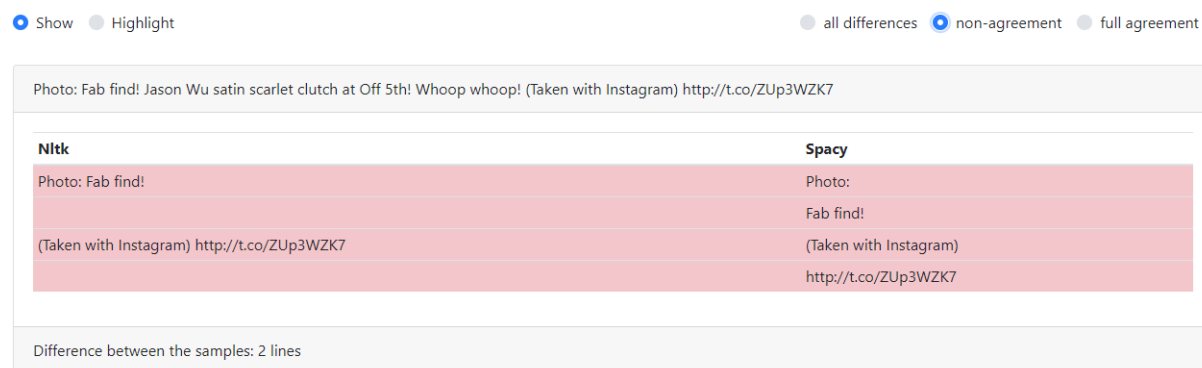


Figure 39: A single sentence splitting comparison displaying only non-agreement

To allow the user to quickly identify (dis-)agreement, a number of radio buttons are offered at the top of the section. By using these options, the user can e.g. only *show* lines for which the algorithms have a *non-agreement* (Figure 39, compare to the first sample in Figure 38: An excerpt of the sentence splitting comparison for hate speech; see also Chapter 4.1.5.2.5).

#### 4.1.5.2.5 Tokenizers

### Tokenizers

Tokenizers	Overlap
nltk ↔ core	96.06%
nltk ↔ spacy	94.48%
spacy ↔ core	94.23%

### Tokenizer Overlap

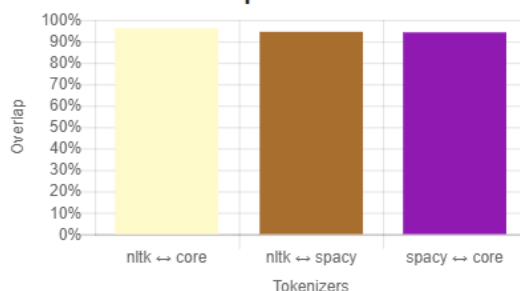


Figure 40: Table and chart comparing the overlap between tokenizers

The Tokenizer section is similar to the previous Sentence Splitting section yet instead of visualizing the differences between sentences, the user can see the differences between tokenizers i.e. algorithms that split a sentence into tokens which can be a single word, a contraction of two words, or special characters.

The samples for this section are again the sentences where the tokenizers differ the most. To get an idea on how similar the tokenizers are before looking at the samples, the user is presented with a table and a graph showing the overlap between each pair of tokenizers.

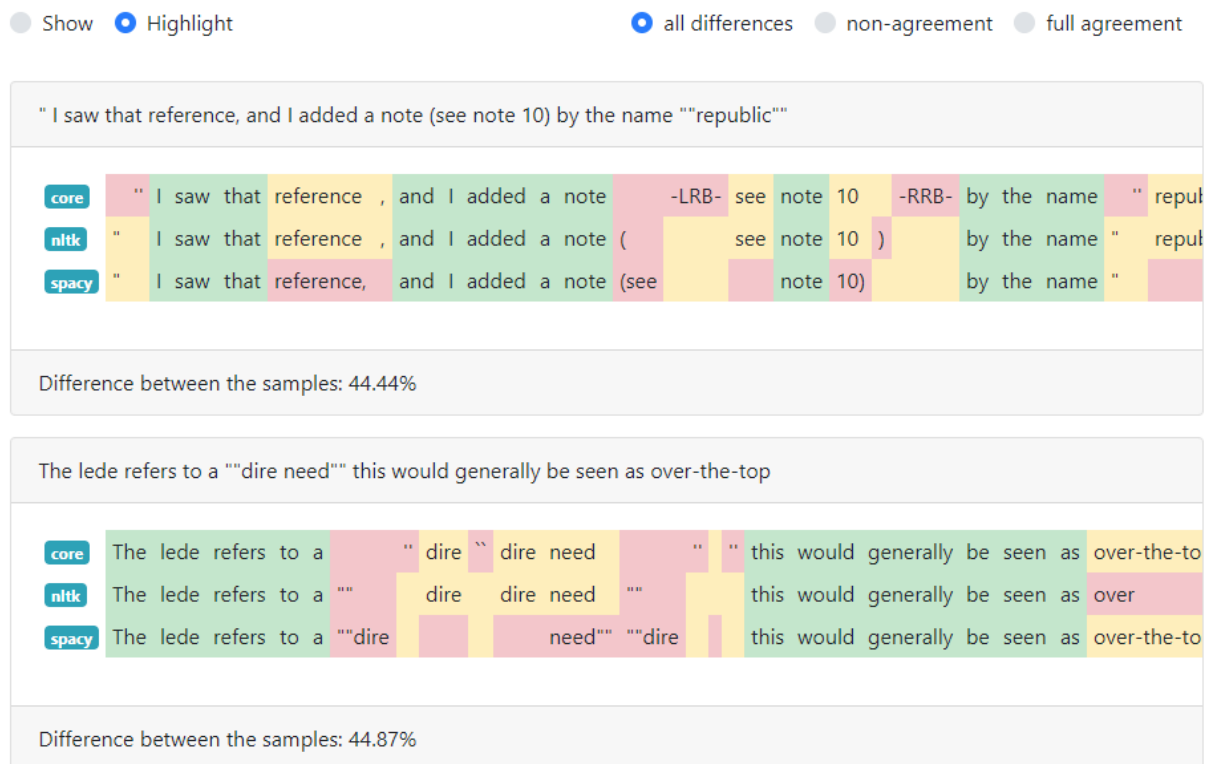


Figure 41: An example of the tokenizer comparison on the hate speech corpus

Contrary to the previous section, algorithms are displayed row-wise to account for the smaller size of tokens as compared to sentences. Additionally, since three instead of two algorithms are used, there is a third color, yellow, which is used for partial agreement when only two tokenizers agree.

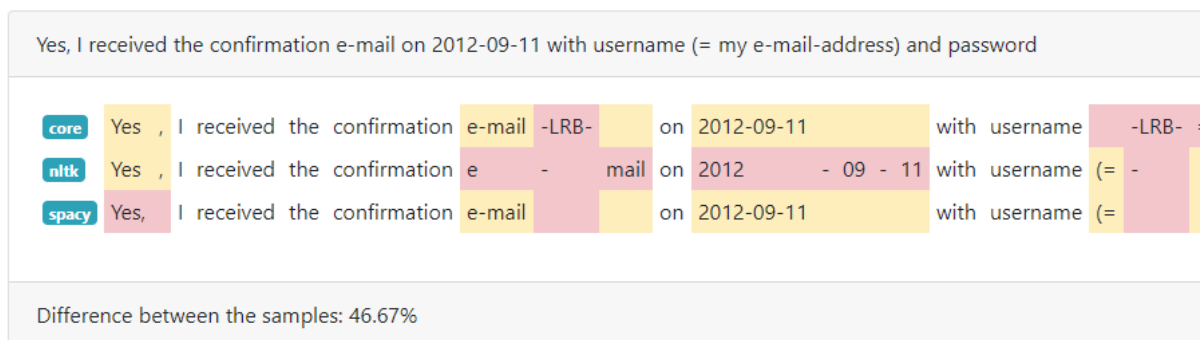


Figure 42: Tokenizer comparison only highlighting differences between the tokenizers

For the purpose of the radio buttons controlling which differences are highlighted (or shown), yellow (partial agreement) is treated as red (non-agreement) (see Figure 42).

This tokenizer comparison also benefits the user to spot peculiarities of a single tokenizer such as Stanford’s Core NLP to convert special characters into tokens (see Figure 42 where the left round bracket “(” has been substituted as “-LRB-”) or how quotation marks are treated (and sometimes standardized) as well as other textual features. For a more complete analysis, see Chapter 4.2.2.3.4.

### 4.1.5.3 Load

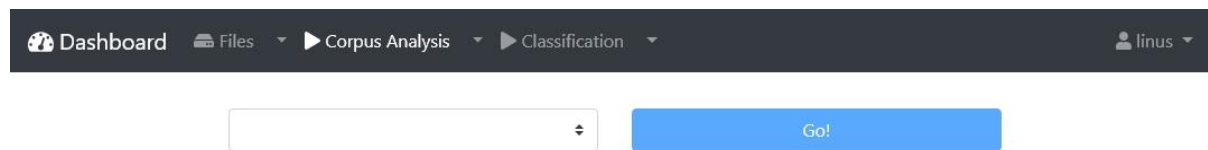


Figure 43: Load screen

To re-examine a previous corpus analysis, the user has two options to load an analysis:

1. Navigate to the list of files, and then click on the dropdown menu right of the “Start corpus analysis” button.
2. In the top-bar navigation the user can click on the dropdown menu right of “Corpus Analysis” and then click on “Corpus”.

In the latter case, the user can then use the dropdown menu to select the desired analysis and confirm the choice by clicking on “Go!”.

### 4.1.6 Classification

*Note: The work in this chapter build on previous work which has been document in [4]. While concepts are briefly re-explained in this document, the reader is kindly asked to refer to [4] for further details.*



Figure 44: The top-bar navigation for classification

Apart from analyzing a corpus, the user can also employ machine and deep learning algorithms by starting (Chapter 4.1.6.1) and running (Chapter 4.1.6.2) a classification task, as well as evaluating (Chapter 4.1.6.3) the models generated in this task.

### 4.1.6.1 Start

Please refer to [4, pp. 17-21] for further details.

## Start a Classification

The screenshot shows a web interface for starting a classification job, divided into three main sections:

- Data set:** Contains a heading "Data set", a label "Choose your type of file input", two buttons "Automatic split" (highlighted in blue) and "Separate train and test files", and a "Choose a file" label next to an empty dropdown menu.
- Tokenizers:** Contains a heading "Tokenizers", a "Word tokenizer" label next to a dropdown menu showing "nltk", and a "Sentence tokenizer" label next to another dropdown menu showing "nltk".
- Algorithms:** Contains a heading "Algorithms", a toggle switch for "All" (selected) and "None", a prompt "Please choose the algorithms you'd like to run on your data.", and a card for the "Naive Bayes" algorithm. The card includes a description: "In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features." and a "run" checkbox which is checked. A "[Source]" link is also present.

Figure 45: The upper part of the screen to start a new classification job

A classification task can be started in two different ways:

1. By clicking on “Classification” in the top-bar navigation (Figure 44).
2. By clicking the “Start a new classification job” button within a corpus analysis (Figure 26) or the “Start classification” button in the list of files (Figure 5).

The difference between the ways is reflected in the state of the first dropdown menu of the screen (Figure 46) where the file has to be chosen or is already populated with the chosen file, respectively.

### Data set

Choose your type of file input

Automatic split
Separate train and test files

Training file  Select train file first.

**Please note:** The column mapping of the two file has to be identical.  
Files with non-matching mappings will appear as disabled options.  
You can also [upload a file](#)

Figure 46: Choosing separate train and test files for a job

In case the user chooses a single file (Figure 45), a stratified train-test split will be performed automatically (Chapter 4.2.3.1.1). Should the user desire to supply a custom train-test split, there is an option to supply a training and test file by switching from “Automatic split” to “Separate train and test files”. Since the training and test file need to have an identical column mapping, the training file has to be selected first and only test files will be available as options whose column mapping corresponds to the column mapping of the selected training file.

### Tokenizers

Word  
tokenizer

Sentence  
tokenizer

Figure 47: Choosing word and sentence tokenizers when starting a job

Since the corpus analysis allowed for the comparison of different word and sentence tokenizers, the user can now choose which tokenizers to use for the classification task. The default is to use *nlTK* for performance reasons with *spacy* and *CoreNLP* (word tokenizer only) available as options.

## Algorithms

All
None

Please choose the algorithms you'd like to run on your data.

**Naive Bayes**

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

[\[Source\]](#)

run

**Support Vector Machine**

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting).

[\[Source\]](#)

run

**Stochastic Gradient Descent**

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

[\[Source\]](#)

run

**AdaBoost**

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

[\[Source\]](#)

run

**Convolutional neural network**

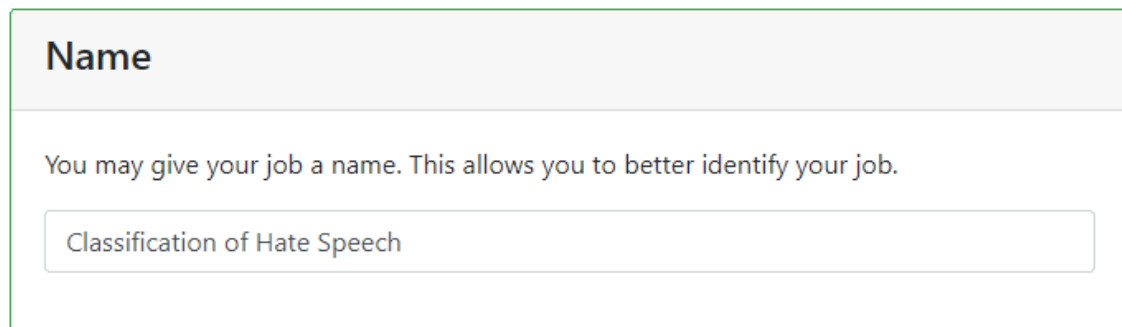
In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

[\[Source\]](#)

run

Figure 48: The list of available algorithms for classification

Once the input file(s) tokenizers have been set, the user can then proceed to choose the classifiers to use<sup>13</sup>. By default, all classifiers are checked i.e. will be used for the job.



The screenshot shows a web interface for creating a job. At the top, there is a header with the word "Name" in bold. Below the header, there is a text prompt: "You may give your job a name. This allows you to better identify your job." Underneath the prompt is a text input field containing the text "Classification of Hate Speech". Below the input field is a large blue button with the text "Start job on server" in white.

**Start job on server**

Figure 49: Name input and start button for a job

Before starting the job by clicking on the blue “Start job on server” button, the user may modify the automatically generated job name (“Classification of Hate Speech”).

#### 4.1.6.2 Monitor

As soon as the job has been started, the user will be shown the Monitor, consisting of three main sections which are top-down:

1. The header displaying job meta data and the input parameters as well as an overview of the processes in that job. (Chapter 4.1.6.2.1)
2. The classifier comparison showing information on which texts the classifiers produced agreeing or disagreeing results. (Chapter 4.1.6.2.2)
3. The processes’ detail boxes displaying the details of every process, including its status, confusion matrix, classification report, and epoch graph. (Chapter 4.1.6.2.3)

---

<sup>13</sup> Each classifier is referred to as a *process* in the context of a classification *job*.

### 4.1.6.2.1 Header

Please refer to [4, pp. 21-27] for further details.



Figure 50: The overview at the top of the job monitor

In the first part of the header, the user can see whether the job is still running or if it is complete, how long the execution of the job as a whole did take, and which input parameters were chosen for the tokenizers and which file(s) were used.

By using the “Tools” menu, processes, which are still running, can be terminated, the job can be evaluated (Chapter 4.1.6.3), and, if available, a corpus analysis based on the same file can be loaded.

The overview of the processes displays the status of each process, its F1 score, the execution time, and, if applicable, a mini-chart of its epoch graph.

### 4.1.6.2.2 Classifier Comparison

#### Classifier Comparison

[Download comparison](#)

Set	Negative	Neutral	Positive	Total	Set	Total
All incorrect	320	160	272	752	Complete agreement	1940
All correct	20	686	665	1371	Disagreement	1669
Rest	210	663	613	1486	<b>Total</b>	<b>3609</b>
<b>Total</b>	<b>550</b>	<b>1509</b>	<b>1550</b>	<b>3609</b>		

Figure 51: Classifier comparison overview tables for the tweets corpus

At the top of the classifier comparison, the user will find a button to download the complete comparison as well as an overview of the comparison.



The comparison consists of different sets<sup>14</sup> in the following order:

1. All incorrect (classified incorrectly by all classifiers)
2. Disagreement (classifiers disagree on labels)
3. All correct (classified correctly by all classifiers)
4. Complete agreement (all classifiers agree on labels)
5. Rest

The two overview tables display data on which set contains how many texts with a given label as well as the total number of texts in a set.

### All incorrect

Classified incorrectly by all classifiers

Correct label	Naive Bayes: run	Convolutional neural network: run	AdaBoost: run	Support Vector Machine: run	Stochastic Gradient Descent: run	Text
negative	!	!	!	!	!	The Philippines just passed a law worse than SOPA, which
negative	!	!	!	!	!	We have finally discovered a day of the week worse than M
negative	!	!	!	!	!	The conveyor belt of "goods" bought at Monsanto Superm
negative	!	!	!	!	!	Want to fight Obama's terrible #IranDeal? Help us send a
neutral	!	!	!	!	!	Blake was upset because I spoiled Harry Potter for her,if yo
neutral	!	!	!	!	!	@manneredguy @BallHogba @j85royals Like Juventus is b
neutral	!	!	!	!	!	I'm tired but I'm not... I may watch the end of Zombieland
neutral	!	!	!	!	!	Amazon Prime Day: Worst Sale Ever or Huge Success? - It v
positive	!	!	!	!	!	Thursday Blackheath By Election. Vote UKIP. You know it m
positive	!	!	!	!	!	Mark Martin winning the The No Bull Million during the Cc
positive	!	!	!	!	!	@Periplus_Store on the 6th books: "Harry Potter and The H
positive	!	!	!	!	!	http://t.co/E1la9Ydik5 If you're thinking about voting for D

All classifiers are wrong for **752 out of 3609** texts (20.84%).  
 negative: 320 / 550 (58.18%) | neutral: 160 / 1509 (10.60%) | positive: 272 / 1550 (17.55%)

Figure 52: The set of texts all incorrectly classified by every process

Every set has the same structure with an explanatory header, the result of the comparison in tabular form as its main section, a download button for the corresponding set, and the footer containing the same information as was present in the overview table.

<sup>14</sup> The classifier comparison is also part of the Evaluate feature (Chapter 4.1.6.3) where files may not have labels and as such not all comparisons can be performed.

Disagreement						
Classifiers disagree on labels						
Correct label	Naive Bayes: run	Convolutional neural network: run	AdaBoost: run	Support Vector Machine: run	Stochastic Gradient Descent: run	Text
negative	!	✓	!	✓	!	Obama is putting US on par with every other 3rd world co
negative	!	!	✓	!	!	I knew Amazon Prime Day was destined to fail when I saw
neutral	✓	!	!	!	!	You say it's not a right. SCOTUS says it is. Constitution says

Figure 53: An excerpt of the “Disagreement” set

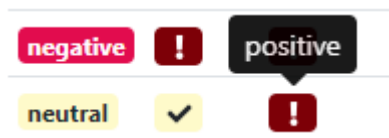


Figure 54: Tooltip appearing when hovering over the icon

The main comparison table presents the correct label for a given text in the first column, colorized according to the label, and the text in the last column. The middle columns each correspond to a process and each cell contains two pieces of information:

1. An icon which is either a checkmark or an exclamation sign indicating whether this column’s process predicted the label correctly or incorrectly, respectively.
2. The icon’s background color represents the color of the label the process in question predicted. By not displaying the label in text form, space is saved and there is less information to parse for the user, yet when hovering over the icon, the text form is displayed as a tooltip (Figure 54).

The different sets allow for quick capturing of which texts are challenging for the algorithm to predict correctly, or are ambiguous enough for the algorithms to predict a variety of different labels. Additionally, it is also beneficial to the user knowing which texts are always predicted correctly.

These comparison sets, together with the comparison in Chapters 4.1.5.2.4 and 4.1.5.2.5, are vital to the user’s deeper understanding of the similarities, differences, and nuances of the algorithms. By making use of color-coded information (red/yellow/green or label-based color), the user can spot outliers visually without having to read the text and is able to re-use the knowledge and pattern matching skills gained from one corpus analysis project or classification job for the next project or job.

**All correct**  
Classified correctly by all classifiers

---

**Correct**

label	Text
negative	@TechCrunch the phone will be too tall and bulky and annoying to tolerate use. @Microsoft may lose it like @BlackBer
negative	@MickeyW1776 @AmericannaClub He won't he does not want to alienate Trump conservative voters. May be looking

Figure 55: The "All correct" set lacks the column for every process.

In the case of the "All correct" set, the per-process columns are not displayed as they exclusively contain redundant information (Figure 55).

#### 4.1.6.2.3 Processes

Please refer to [4, pp. 21-27] for further details.

**completed** PID: 18240
F1 score: **55.15%** (58.70%)

---

### Convolutional neural network

run

#### Confusion Matrix

actual/predicted	negative	neutral	positive	all
negative	165	228	157	550
neutral	137	861	511	1509
positive	82	346	1122	1550
all	384	1435	1790	3609

#### Classification Report

	precision	recall	f1-score	support
negative	0.43	0.30	0.35	550
neutral	0.60	0.57	0.58	1509
positive	0.63	0.72	0.67	1550
avg/total	0.59	0.60	0.59	3609

#### F1 Score over Epoch Graph

Epoch

**Toggle raw output**

Ended 4 days ago on XPS8500 | Execution took 13 minutes

Figure 56: The process detail box of a CNN

For every process in a job, there is a detail box where the user can find more details about the execution and output of a process. This information is displayed as soon as it is available allowing the user, if supported by the underlying algorithm, to monitor the execution of the process while it is still running.

The user is presumably most interested in the process' output which, if applicable for this process<sup>15</sup>, is displayed in two tables and a chart. The tables contain the confusion matrix and the classification report. The chart displays the F1 score per epoch.

---

<sup>15</sup> Not every algorithm e.g. uses epochs (or does not make the intermediary results available) and as such the epoch graph is not shown in these cases.

### 4.1.6.3 Evaluate

Please refer to [4, pp. 27-31] for further details.

## Evaluate Classification of Tweets / Tweets

[Tools ▾](#)

Here you can evaluate models generated by a job. An evaluation consists of:

1. A set of models you'd like to use for the evaluation
2. Input data

### Models

Against which models would you like to run your input? All models No models Best model only

<input checked="" type="checkbox"/>	Naive Bayes: run	F1 score: <span style="background-color: #28a745; color: white; padding: 2px;">59.69%</span>
<input checked="" type="checkbox"/>	Convolutional neural network: run	F1 score: <span style="background-color: #28a745; color: white; padding: 2px;">69.45%</span>
<input checked="" type="checkbox"/>	AdaBoost: run	F1 score: <span style="background-color: #28a745; color: white; padding: 2px;">59.12%</span>
<input checked="" type="checkbox"/>	Support Vector Machine: run	F1 score: <span style="background-color: #28a745; color: white; padding: 2px;">64.84%</span>
<input checked="" type="checkbox"/>	Stochastic Gradient Descent: run	F1 score: <span style="background-color: #28a745; color: white; padding: 2px;">64.26%</span>

### Input

What would you like to feed as input to the computed models? You can provide a single string or a file.

Choose your type of input

A single string
Another file
The test file you provided for this job

Enter a string

Some required fields are empty.

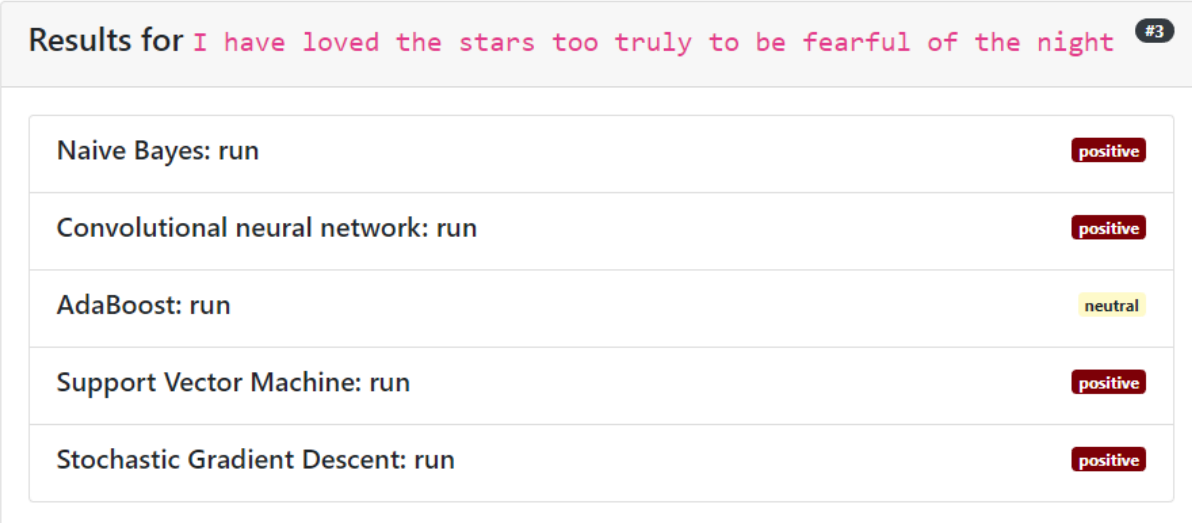
Figure 57: The Evaluate screen for a job with separate train and test files

In the process of performing a classification job, models are generated by the processes which can be used for string and file evaluation<sup>16</sup>.

<sup>16</sup> Some algorithms, such as the CNN, output intermediate models (e.g. after an epoch) which can be used for evaluation, too. For the purpose of the walkthrough, all processes have completed, and no process is

The evaluation screen consists of a simple header with a “Tools” menu, allowing the user to switch back to the monitor and load a corpus analysis, a list of available models, and the various input forms which are:

- A single string: this allows the user to enter a text which will then be evaluated by the models. This can be useful for e.g. checking how nuances are captured by the models.
- Another file: the user can choose a file stored on the server which will then be evaluated by the models.
- The test file you provided for this job (*only if one was provided*): if a test file was provided when starting the job (Chapter 4.1.6.1), this serves as a shortcut to feed the file to the models for evaluation.



The screenshot shows a results panel for a specific string evaluation. The header displays the string "I have loved the stars too truly to be fearful of the night" in pink, followed by a "#3" badge. Below the header is a table with five rows, each representing a different model and its evaluation result.

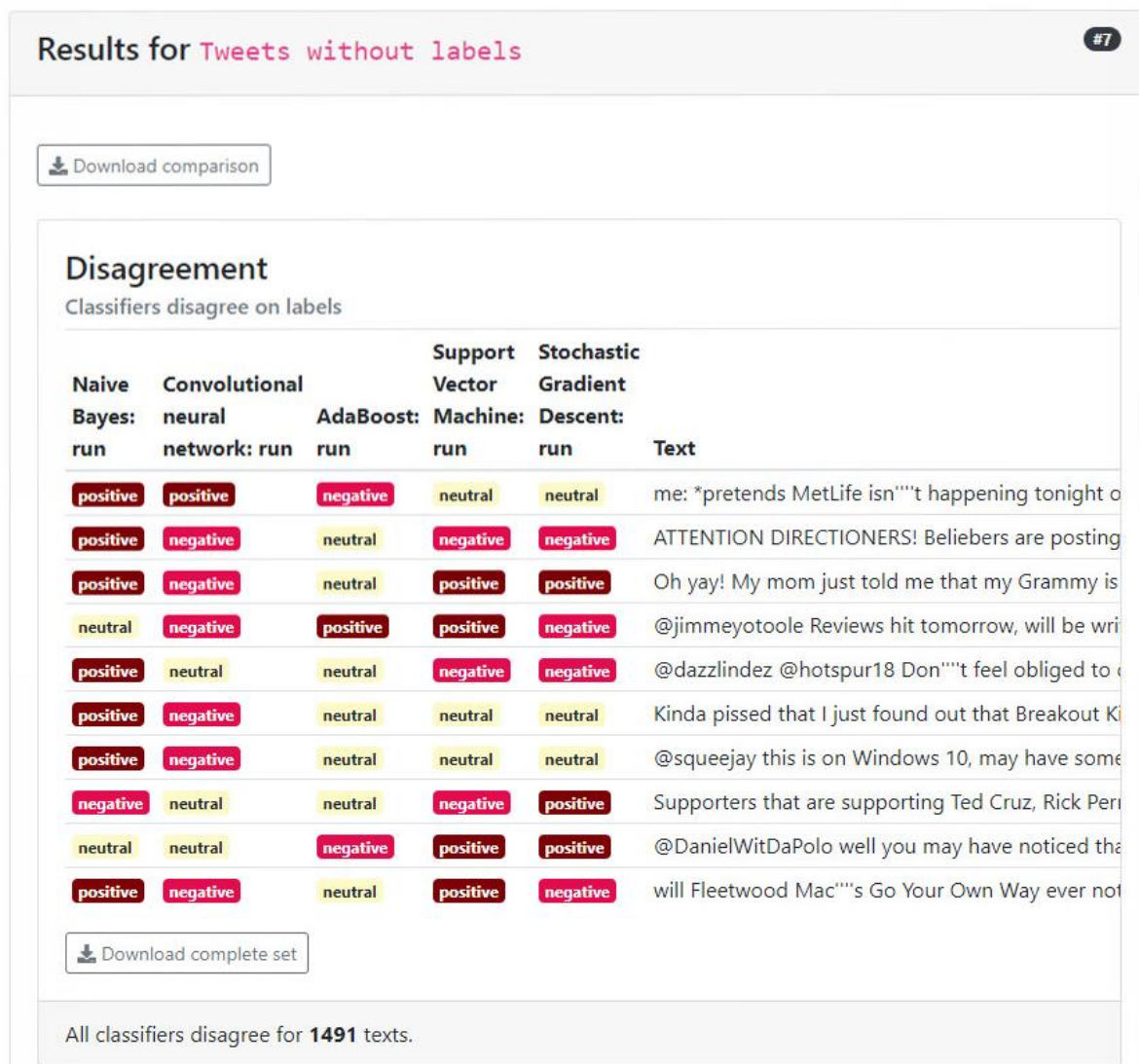
Model	Result
Naive Bayes: run	positive
Convolutional neural network: run	positive
AdaBoost: run	neutral
Support Vector Machine: run	positive
Stochastic Gradient Descent: run	positive

Figure 58: Results for a single string evaluation on models generated by a job on the tweets corpus

---

still running. Except for the different status color (blue instead of green) of the F1 score in the list of models, there is no difference for the user whether the model used for evaluation is intermediary or final.

The results of a single string evaluation are displayed in a simple list form, with the name of the process on the left, and the predicted label, appropriately colored, on the right.



**Results for Tweets without labels** #7

[Download comparison](#)

### Disagreement

Classifiers disagree on labels

Naive Bayes: run	Convolutional neural network: run	AdaBoost: run	Support Vector Machine: run	Stochastic Gradient Descent: run	Text
positive	positive	negative	neutral	neutral	me: *pretends MetLife isn""t happening tonight o
positive	negative	neutral	negative	negative	ATTENTION DIRECTIONERS! Beliebers are posting
positive	negative	neutral	positive	positive	Oh yay! My mom just told me that my Grammy is
neutral	negative	positive	positive	negative	@jimmeyotoole Reviews hit tomorrow, will be wri
positive	neutral	neutral	negative	negative	@dazzlindez @hotspur18 Don""t feel obliged to c
positive	negative	neutral	neutral	neutral	Kinda pissed that I just found out that Breakout Ki
positive	negative	neutral	neutral	neutral	@squeejay this is on Windows 10, may have some
negative	neutral	neutral	negative	positive	Supporters that are supporting Ted Cruz, Rick Perri
neutral	neutral	negative	positive	positive	@DanielWitDaPolo well you may have noticed the
positive	negative	neutral	positive	negative	will Fleetwood Mac""s Go Your Own Way ever not

[Download complete set](#)

All classifiers disagree for **1491** texts.

Figure 59: The header of a result card for file evaluation and the "Disagreement" set

When a file was chosen for evaluation, the result card consists of three sections, with a focus on the comparison of the underlying models:

1. The header displaying information about which file was used and offering a download button of the complete comparison of the algorithms on this file. (Figure 59)
2. Multiple comparison sets <sup>17</sup> containing the same information as described in Chapter 4.1.6.2.2. Please refer to the referenced chapter for more details<sup>18</sup>. (Figure 59)
3. A list of the models used on this evaluation with a download button to download the result. (Figure 60)

<sup>17</sup> While the information is the same, not all sets may be displayed as described in <sup>14</sup>.

<sup>18</sup> This is an instance of component re-use as described in Chapter 4.4.1.1.

Naive Bayes: run	<a href="#">Download</a>
Convolutional neural network: run	<a href="#">Download</a>
AdaBoost: run	<a href="#">Download</a>
Support Vector Machine: run	<a href="#">Download</a>
Stochastic Gradient Descent: run	<a href="#">Download</a>

Figure 60: Download buttons for each model to download the annotated file

The download button allows the user to download the chosen file which has been supplemented with a *predict* column<sup>19</sup> containing the label predicted by the model.

---

<sup>19</sup> This column is not called *predict* in the downloaded file, instead an abbreviation of the model name is used, e.g. *NB* for *Naive Bayes*.



## 4.2 Backend

The backend repository holds all Python files of the project and is the place where all the data crunching and classification is done. This chapter first gives an overview of the files before looking at the different parts in detail.

### 4.2.1 Overview

#### 4.2.1.1 Structure Overview

Of the ten Python files, four are called directly by the API. These are found in the root of the repository and are listed with their functions in Table 1 while the others are in the *utils* folder and described in Table 2.

File	Function	Imports <sup>20</sup>
<b><i>corpus_analysis.py</i></b>	Starts the different parts of the analysis as threads and manages the lock for the output file.	<i>io_utils.py</i> <i>topic_modeling.py</i> <i>stats.py</i> <i>sentence_tokenizer.py</i> <i>word_tokenizer.py</i>
<b><i>preprocess.py</i></b>	Readies the data for classification by applying the selected tokenizers and vectorizes the text.	<i>io_utils.py</i> <i>sentence_tokenizer.py</i> <i>word_tokenizer.py</i>
<b><i>classification.py</i></b>	Trains or evaluates the data, based on the input.	<i>cnn.py</i> <i>io_utils.py</i> <i>stats.py</i>
<b><i>compare_classifiers.py</i></b>	Compares the output of the different classifiers.	<i>io_utils.py</i>

Table 1: Callable Files

File	Function	Imports <sup>20</sup>
<b><i>io_utils.py</i></b>	Contains all functions that load or save files. Additionally, it parses the command line parameters and parses the data files.	
<b><i>topic_modeling.py</i></b>	Generates the specified number of topics, with the top words and the corresponding label distribution.	<i>io_utils.py</i> <i>stats.py</i>
<b><i>stats.py</i></b>	Generates statistics for label, instance and word distribution.	<i>word_tokenizer.py</i>
<b><i>sentence_tokenizer.py</i></b>	Compares sentence tokenizer and synchronizes matching tokens.	
<b><i>word_tokenizer.py</i></b>	Compares word tokenizer and synchronizes matching tokens.	
<b><i>cnn.py</i></b>	Contains all custom functions for the CNN.	<i>io_utils.py</i>

Table 2: Utility Files

#### 4.2.1.2 Python Calls

Each of the callable Python files requires a number of parameters. Since all files use the same parse function the whole application only relies on 5 switch characters *-c -o -d -e -r*.

<sup>20</sup> Does not list imports of library files

All parameters are paths and saved in the environment string of the operating system. This is done so that no additional arguments need to be passed along to other methods. Once saved, the path can be accessed from everywhere by calling `os.environ['path_variable']`.

Short	Long	File Type	Used for	Function
<b>-c</b>	<code>--config=</code>	JSON	corpus_analysis.py preprocess.py classification.py	Location of the configuration file
<b>-o</b>	<code>--output=</code>	JSON	All	Location of the output file(s)
<b>-d</b>	<code>--data=</code>	CSV	All	Location of the data file(s)
<b>-e</b>	<code>--test=</code>	CSV	preprocess.py	Location of the train file
<b>-r</b>	<code>--train=</code>	CSV	preprocess.py	Location of the test file

Table 3: Call Parameters

The specific requirements for the calls as well as the needed JSON configuration data are described at the beginning of the corresponding chapter.

#### 4.2.1.3 Data Files

Since the data file gets converted right after it is uploaded to the server (see Chapter 4.3.2) the backend expects it in only one format. The columns are separated with semicolons and if required single quotes are used around a data entry, furthermore, the encoding is expected to be UTF-8<sup>21</sup>. Should a line still have the wrong number of columns it is dropped from the data.

#### 4.2.1.4 Structure of the Data Frame

The CSV file is loaded into a pandas<sup>22</sup> data frame and if necessary some restructuring is done.

- Columns flagged as *ignore* are dropped
- If the file has a header, the names are kept, otherwise, one is added using the names as for the column type flags, e.g. *date*
- Unique names for column headers are forced by adding e.g. *\_1* to identical fields

The order of the columns is kept the same as in the original file for the corpus analysis, but some columns are added as helpers. In the classification process, the order is changed to keep label columns next to each other. Changes to the data frame will be noted in the corresponding chapters.

<sup>21</sup> Unicode Transformation Format 8-bit is a variable-width encoding that can represent every character in the Unicode character set.

<sup>22</sup> pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

<b>Name</b>	<b>Flag</b>	<b>Description</b>	<b>Example</b>
<b>String label</b>	<i>label</i>	Any string	<i>positive</i>
<b>Date label</b>	<i>date</i>	Date with or without time, multiple formats accepted	<i>July 5, 1989</i>
<b>Numeric label</b>	<i>numeric</i>	Any number	<i>235</i>
<b>Binary label</b>	<i>binary</i>	Only two unique labels per column, in case of one being either <i>0</i> or an empty field, it is not treated as a label for the corpus analysis.	<i>1</i>
<b>Text</b>	<i>text</i>	Text to be evaluated.	<i>Good morning</i>
<b>ID</b>	<i>id</i>	ID of the entry	<i>2345678876 5</i>
<b>Language identifier</b>	<i>lang</i>	Specifies what language the text entry is.	<i>en</i>
<b>(ignore)</b>	<i>ignore</i>	Column will be dropped.	

Table 4: Column types

#### 4.2.1.5 Structure of the labels

There are four different label types the user can select from for a column. The types and examples are listed in Table 4. This structure was chosen with the idea to create ranges or dates and numbers to consolidate the labels and to filter out-“off” values for binary columns. At the moment, only the latter has been implemented.

All labels are stored in a nested Python *dictionary*. The four label types make up the keys for the outer layer where the labels are grouped by their types. The label names are used as keys for the inner layer which stores information that is specific to a label gathered during the corpus analysis.

#### 4.2.2 Corpus Analysis

All information that is displayed in the “Corpus” tab (Chapters 4.1.5 and 4.4.3) is generated with the call of `corpus_analysis.py`. As shown in Table 5 the analysis is comprised of four parts that the user can choose to analyze.

Part	Returns
<b>Topic modeling</b>	<ul style="list-style-type: none"> <li>- Requested number of topics</li> <li>- Requested number of topics per label</li> <li>- Distribution of topics in the corpus</li> <li>- Distribution of topics in the labels</li> </ul>
<b>General statistics</b>	<ul style="list-style-type: none"> <li>- Column preview: up to three unique values per column</li> <li>- Distribution of labels</li> <li>- Distribution of instances<sup>23</sup></li> <li>- Top 100 Words in total and per label</li> <li>- Min/average/max character and word length of texts in total and per label</li> </ul>
<b>Word tokenizer</b>	<ul style="list-style-type: none"> <li>- A sample of 10 sentences split by three different tokenizers</li> <li>- Percentage of how much the tokenizers overlap</li> </ul>
<b>Sentence splitting</b>	<ul style="list-style-type: none"> <li>- A sample of 10 texts split by two different tokenizers</li> </ul>

Table 5: Parts of Corpus Analysis

#### 4.2.2.1 Call

The Python call requires a configuration, a data- and an output file. For example:

```
python corpus_analysis.py -c config.json -o out.json -d data.csv
```

#### 4.2.2.2 Configuration JSON

Code Snippet 1 is an example of a corpus analysis configuration json. The *columnMapping* lists the types of columns of the data in order. *header* shows the number of header rows of the data file. Because the data frame uses *None* for no header and *0* states that line 0 of the document contains the names of the columns, the backend expects *-1* to signal that the file does not contain a header. *analyse* holds the switches of what is to be analyzed in the corpus. The last field is *topic\_modeling* which holds settings for the number of topics and words per topic to be returned as well as the number of features and the maximum size of the n-grams. If the number of features is set to *0* the number of features is not limited.

<sup>23</sup> In case of multiple label columns, an instance represents the combination of all labels in a row.

```

1  {
2    "columnMapping": [
3      "ID",
4      "lang",
5      "binary",
6      "text"
7    ],
8    "header": 0,
9    "analyse": {
10     "topic_modeling": 1,
11     "general_stats": 0,
12     "sentence_tokenizer": 1,
13     "word_tokenizer": 0
14   },
15   "topic_modeling": {
16     "n_topics": 10,
17     "n_words_per_topic": 10,
18     "n_features": 0,
19     "n_grams": 1
20   },
21 }

```

*Code Snippet 1: Corpus analysis configuration JSON*

#### 4.2.2.3 Functionality

The *corpus\_analysis.py* parses the call arguments and loads the data and the configuration via the *io\_utils.py* file. Next, the label columns get validated: *binary* labels must only contain 2 unique values, both zero and empty fields are treated as off. The Python standard library *dateutil* tries to parse columns with the label set to *date*, if successful the dates are reformatted into a standard format, YYYY.MM.DD and if the time is available, hh:mm:ss. For the *numeric* label again a standard library function, *math.isnan()* is used to validate the column. If any field in a column fails the validation, the whole column is dropped for the following analysis. A warning is displayed showing which columns have been removed. If a user still wishes to analyze the column, the label should be set to *string label* since it is not validated.

At this point, all parts are ready to start the analysis. All selected parts for the analysis get started in separate processes. Each one returns a *dictionary* containing the gathered data. *corpus\_analysis.py* then saves the information in a json using a thread-safe function.

##### 4.2.2.3.1 General Statistic

Most of the statistics are gained by filtering the data frame or using its *apply()* function, which applies a method to, depending on the selection, each row or field in a column.

###### 4.2.2.3.1.1 Column Preview

The column preview should allow the user to get a brief look at the contents of each column. For this purpose, unique values are extracted from the data frame and up to three values are returned. Because of this the previews for the different columns do not necessarily contain data from the same rows.

#### 4.2.2.3.1.2 Distribution of Labels

The number of times a label appears in a column. Identical labels in different columns are treated separately. If the label type is binary and the value is either 0 or an empty field, it is not counted. Upon completion, all rows that have not been used are added up and returned in a field called *none*.

#### 4.2.2.3.1.3 Distribution of Instances

An instance is a combination of all labels in a row. If there is only one label column the output will be identical to the distribution of the labels as seen in 4.2.2.3.1.2. In preparation, an additional column is added to the data frame that holds the values of each row as a single string with the labels being separated by ` (Unicode 035b). The separation symbol was selected because it is highly unlikely to appear in a label. From this new column, all unique instances are extracted. The data is filtered for each of these to get the number of uses. If the labels are dates or numbers, the label name is added to the column header so it is clear what the value specifies e.g. *airdate\_2017-12-02*. Binary labels that are “off” are removed from the combined string. All rows without a label are counted in the field *none*.

#### 4.2.2.3.1.4 Character and word count

Character and word count are only evaluated in the column flagged as *text*. Both statistics are generated for the whole corpus and for each label. The shortest, the longest and the average length is returned. The character count is simply the length of the field. For the words, the text is split using the Spacy [7] tokenizer to get the number of words.

#### 4.2.2.3.1.5 Top 100 Words

Using the *TfidfVectorizer* from Sklearn the 100 most relevant words are extracted from the complete corpus and per label. In the app, the output is displayed as a word cloud, for this the total weight of each word is needed to scale them. Because the total weight is calculated as the sum of the weights in all the text entries, the total weights are divided by the number of texts that were analyzed to balance the scaling for corpora with an uneven label distribution.

#### 4.2.2.3.2 Topic Modeling

The biggest challenge while creating the topic modeling part, was to get the result in a reasonable timeframe. For this reason, gensim<sup>24</sup>, unfortunately, was not an option. From Sklearn Non-Negative Matrix Factorization (NMF) and Latent Dirichlet Allocation (LDA) were tested in combination with a tfidf vectorizer and a count vectorizer. The preliminary results were not satisfactory, though the best and the fastest combination was the NMF with the tfidf vectorizer. Tweaking the parameters for both the factorizer and the vectorizer did not have a notable effect. To get the result as seen in Figure 34 the corpus had to be preprocessed. The topics for the 20 News Groups<sup>25</sup> dataset is now very good, but the preprocess has hampered the performance considerably. A set of 18000 tweets still does not produce any useful topics which may be because the individual texts are too short or

---

<sup>24</sup> Gensim is a free topic modelling tool for Python. [radimrehurek.com/gensim/](http://radimrehurek.com/gensim/)

<sup>25</sup> 20 News Groups is widely used data set for machine learning development. It contain news articles from 20 different topics.

the corpus size is too small. For example<sup>26</sup>, the topic *game, game tomorrow, throne, game throne, season, watch game, game saturday, game friday, season game, game Sunday* is a mixture of the TV series Game of Thrones which airs on a Sunday and sports, both featuring the word *game* prominently.

#### 4.2.2.3.3 Sentence Tokenizer

The purpose of the sentence tokenizer part is to compare different splitters. At the current stage two tokenizers have been applied, the Spacy and the NLTK [8] sentence tokenizer. But the code was designed in a way that implementing and comparing additional tools should be easy.

Because only ten samples are needed for the comparison, only a portion of the dataset needs to be analyzed. While looping through the corpus, up to ten samples with the most difference in the number of tokens are stored. By this, example texts are saved that are treated very differently by the tokenizers and therefore offer the user some interesting cases. This difference is checked before the splitters are compared, and if it is lower than the current lowest difference in the top 10 the analysis is skipped. The process terminates early when the top 10 list has not changed for 500 texts.

If the tokens are not equal because one is longer than the other, the next instance is sought where two tokens start the same. For this, the first ten characters of the token following the longer sentence is compared to the tokens after the shorter one until a match is found. Only 10 characters are analyzed to avoid missing the overlap if the tokenizer that previously had the longer token next produces the shorter one.

As seen in Figure 61 the NLTK tokenizer mostly just splits at punctuations, while Spacy also splits apart long sentences into its subsets.

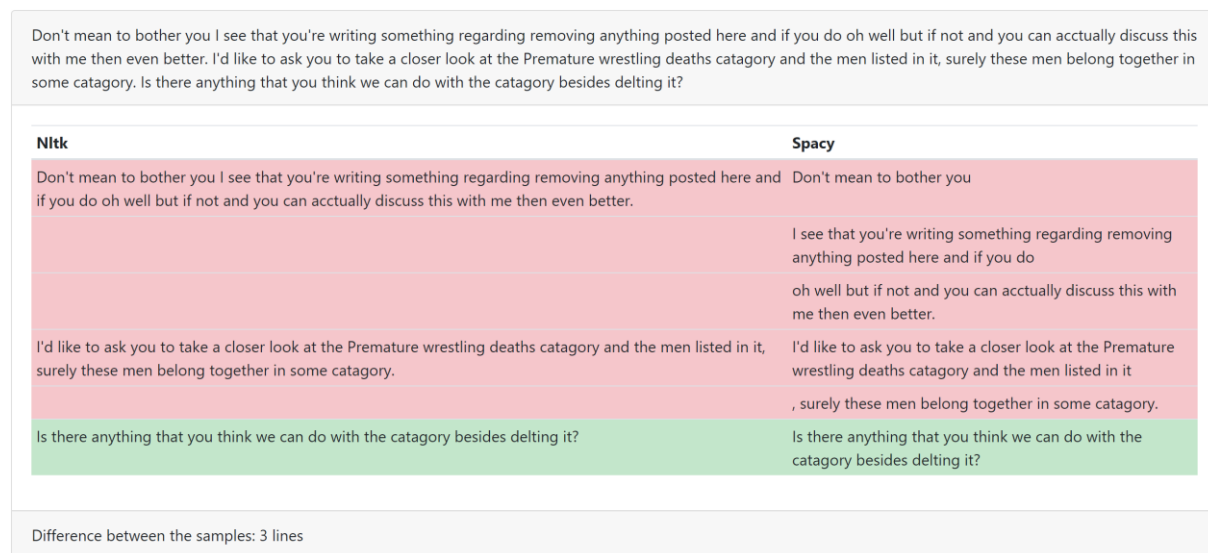


Figure 61: Sentence Tokenizer

<sup>26</sup> 20 topics with a maximum n-grams size of 2



#### 4.2.2.3.4 Word Tokenizer

The basic concept behind the word tokenizer is the same as the sentence tokenizer described in Chapter 4.2.2.3.3. Again, the tokenizers from NLTK and Spacy are used and additionally, the one from Stanford CoreNLP [9]. The comparison was implemented so that adding more tokenizer is kept simple.

Aligning the tokens proved a lot more difficult than for the sentence splitter for multiple reasons. The problems will reference to the example in Figure 62.

- One problem arises when a word is repeated and separated by a comma. Both NLTK and CoreNLP split the comma from the word, while Spacy keeps it together. So, when it searches for the next identical token after “for” it will mismatch. Instead of finding the second “tomorrow” it matches the first one of NLTK and CoreNLP to the second one in Spacy. Because only ten examples are needed these cases are skipped when they are detected.
- The Tokenizers try to standardize symbols, as seen with the apostrophe and the bracket. This creates tokens that can not be matched to the other tokenizers.
- After “be” there is no more token where all three classifiers agree. Instead of always checking if the end of a token list has been reached, a stop token is added to the end of the list.
- Aligning tokens that start the same is difficult because some tokens are only one or two characters long. This was solved with the class *Token* that tracks the previous and the following tokens. If those are equal for all tokenizers it is safe to assume that the token in the middle references the same word in the original sentence.

	I can't wait for tomorrow, tomorrow will be great! :)														
<b>core</b>	I	ca	n't		wait	for	tomorrow	,	tomorrow	will	be	great	!	:	-RRB-
<b>nltk</b>	I	can		'	t	wait	for	tomorrow	,	tomorrow	will	be	great	!	:)
<b>spacy</b>	I	can't			wait	for	tomorrow,	tomorrow	will	be	great!			:)	

Figure 62: Word tokenizers<sup>27</sup>

### 4.2.3 Classification

In the backend, the classification part of *texploration* consists of three elements: The data preprocessing, the classification and the comparison of the classifiers. Each part is a callable Python file, but they are only meant to be used together. As opposed to the previous work done in [4], the data preprocessing was split from the classification to reduce the overhead. Now the preprocessing is done once, and the classifiers can load the already processed data. The comparer can be started once at least two classifiers have successfully terminated, independent of the call being for training or an evaluation. Even though the corpus analysis (Chapter 4.2.2) can deal with multilabel and multiclass data, the classification expects a single label column.

<sup>27</sup> This example is manually generated and not actual output from *texploration*.



### 4.2.3.1 Calls

#### 4.2.3.1.1 Pre-process

There are three ways to call *preprocess.py*. Two are used when training the models.

```
python preprocess.py --config=config.json --output= output_directory --data=data.csv
```

```
python preprocess.py --config=config.json --output= output_directory --train=train.csv --
test=test.csv
```

The first one is used when the train and test data is in a single file. In this case, a stratified train-test-split is done on the corpus. The second option allows for selecting a separate test file. The output directory points to the location where the classification output data will be saved. Inside *texploration* the path is *api/storage/classification/job-UUID*. Additionally, a folder for the classification models and pre-process output is created at *api/storage/models/job-UUID*. When evaluating models, the first call is reused, in this instance, the data file is the file to be evaluated and the output directory points to the aforementioned *models* directory. The third call option is used when a single string is to be evaluated. In this case, the call is the same as the first one but without the data parameter since the string is passed inside the configuration JSON.

#### 4.2.3.1.2 Classification

*classification.py* is not dependent on training, evaluation or the single string option so there is only one way to call it.

```
python classification.py --config=config.json --output=output_file --data=models_directory
```

The data path needs to point to the models folder of the pre-process, when calling it from the API this is *api/storage/models/job-UUID*. Depending on the settings in the configuration file the output consists of one or two files: A JSON named as defined in the call and / or a CSV which has a “\_csv” suffix added to the *output\_file* name.

#### 4.2.3.1.3 Compare Classifiers

```
python compare_classifiers.py --output=output_file --data=classification_directory
```

The comparison is done on the prediction of the test data. So, the *data* parameter is the same as the output of the pre-process call, in this case *api/storage/classification/job-UUID*. The output file is a json with the comparison data that is shown in *texploration*, additionally up to four CSVs are generated using the output file’s name and a suffix, as detailed in Table 6.

### 4.2.3.2 Configuration JSONs

#### 4.2.3.2.1 Pre-process

The configuration JSON for *preprocess.py* contains six fields. Of those *columnMapping* and *header* function the same as the ones for the corpus analysis described in Chapter 4.2.2.2. *sentence\_tokenizer* and *word\_tokenizer* hold which tokenizers the user has selected. If *evaluate* is set to 0 the classifiers are trained, if it is 1 the models are evaluated. If the *single\_entry* field is not empty and *evaluation* is on, a text in *single\_entry* is evaluated. For an example see Code Snippet 2.

```

1  {
2      "columnMapping": [
3          "ID",
4          "lang",
5          "text"
6      ],
7      "header": 0,
8      "sentence_tokenizer": "nltk",
9      "word_tokenizer": "nltk",
10     "single_entry": "",
11     "evaluate": 0
12 }

```

Code Snippet 2: Preprocess configuration JSON

#### 4.2.3.2.2 Classification

Only three fields are required for the configuration JSON for *classification.py* as seen in the example Code Snippet 3. *columnMapping* lists the column names of the data in order and *evaluate* defines if the models will be trained or evaluated. The *classifier* field holds the abbreviation of the classifier to be used.

```

1  {
2      "classifier": "NB",
3      "evaluate": 1,
4      "columnMapping": [
5          "ID",
6          "language",
7          "text"
8      ]
9  }

```

Code Snippet 3: Classification configuration JSON

#### 4.2.3.3 Pre-process

The pre-process mainly differentiates between a training and a test file, because in case of a test file the tfidf vectorizer that was fitted on the training data is used to transform the test data. To standardize the process a single string entry is saved as a pandas data frame and treated the same as a test file. If *preprocess.py* is call in training mode, the *--data* input file is processed as a train file, otherwise as a test file.

Before the data is vectorized the selected tokenizers are applied to the data. The same tokenizers can be chosen that are compared in the corpus analysis. For the sentence splitting texplorator offers NLTK and Spacy, and the same applies for word tokenization with the addition of Stanford's CoreNLP.

Pickle<sup>28</sup> is used to save the output in the models folder: *api/storage/models/job-UUID*. The vectorized texts are dumped as "train\_x" or "test\_x" respectively. The raw data is

---

<sup>28</sup> Pickle is a Python library that allows objects to be saved to the hard drive and loaded back into Python at a later time

saved in the same way but using the suffix “\_data”. Additionally, the fitted tfidf vectorizer is also dumped.

#### 4.2.3.4 Classification

The *classification.py* file is an improved version of the previous work done in [4]. It consolidates all classifiers and the baseline as well as training and evaluation in one place, which before was distributed over four files and multiple call options.

At first, it is distinguished between training and evaluation. In case of training, a check is done if there are one or two data files and the files are loaded. If only one is found, the data is stratified and 20% is split off for testing. Then the selected classifier is trained and, in the end, evaluated with the test set. Three files are generated, the first two are saved in the output folder, while the third one is saved in the models folder, the same as the data is loaded from. One is a JSON, containing data about the performance of the classifier on the test data, as already described in [4]. Additionally to the previous work, a classification report is saved, as seen in Figure 56 containing per-class information on precision, recall, f1-scores and the number of texts per class evaluated. The second file is a CSV containing the original data plus a column with the predicted labels. For easier comparison, the column with the correct labels is placed at the end of the data, so it is next to the predicted values. The last file is the trained model that is named with the abbreviation of the classifier used.

If data is being evaluated or predicted, the data and the selected classifier is loaded for the model folder and the labels of the texts are predicted. If the data contains labels the same JSON and CSV are created as in a training process. If there are no labels but multiple texts, only the CSV is generated, since the performance of the classifier can not be evaluated. If the process was called with a single entry, the output only consists of a JSON containing the predicted class.

##### 4.2.3.4.1 Classifiers

Two of the classifiers, the Naïve Bayes, and the Support Vector Machine, were migrated from the [4]. The other three are a standard Adaptive Boost Classifier from Sklearn with a limit of 100 estimators, a Stochastic Gradient Descent Classifier, also from Sklearn, and a convolutional neural network short CNN built using Keras [10]

The current version of *texploration* only has a basic CNN with hard-coded layers and parameters, because the focus was on making it comparable with the other classifiers and standardize the process input and output to the one used by all the Sklearn models that are implemented. To be comparable the metrics of the CNN needed to be changed from the standard, accuracy, to f1-score. For this *f1* class in *cnn.py* was copied from the *deepmlsa* [11] deep learning tool. Using the Keras Callback class the validation f1-score after each epoch is saved to JSON. If the model has surpassed its previous best score the model is saved after the epoch. Early stopping has also been implemented to stop the training if the model has not improved in a defined number of epochs. With this, all the building blocks are there to create a more flexible version of the CNN with definable parameters and layers.

#### 4.2.3.5 Classifier Comparison

The comparison of the classifier has two main features. One is to generate an overview for the texploration GUI, as seen in Figure 52 and the other one is to create CSV files for more in-depth comparison. The process differentiates between data containing the correct labels and data without them. Table 6 lists the different analyses and in what case they are done. For each of these subsets of the data, samples are saved in a JSON and CSV that has all the columns of the original data file, with, if it exists, the label column at the end, followed by the predictions of the classifiers.

<b>Name</b>	<b>Description</b>	<b>Used when</b>
<b>Agree</b>	Texts for which all classifiers have predicted the same label	Samples: No correct label Statistics: Both
<b>Disagree- ment</b>	All cases where at least one classifier has predicted a different label than the rest	Both
<b>All correct</b>	All classifiers agree and have predicted the correct label	Correct Label
<b>All incor- rect</b>	None of the classifiers have predicted the correct label	Correct Label
<b>Rest<sup>29</sup></b>	All texts that neither fall into the All correct nor the All incorrect category	Correct label

Table 6: Comparison Breakdown

In order to select interesting and diverse examples, some of each label are selected, so that the total of samples is between ten and twenty. Furthermore, the texts chosen for the Disagreement samples have been classified the most variedly.

As seen in Figure 51, texploration also displays the number of texts per subset, both in total and itemized by label. The Rest category was created for this purpose, so the total of all subsets always equals the number of texts predicted.

<sup>29</sup> Only used for the statistics, no samples are displayed, and no CSV output

## 4.3 API

*Note: The work in this chapter builds on previous work which has been document in [4]. While concepts are briefly re-explained in this document, the reader is kindly asked to refer to [4] for further details.*

The API repository is the interface between the frontend (Chapter 4.4) and the backend (Chapter 4.2). It starts Python processes, stores the application's state, and serves data for the frontend.

### 4.3.1 Overview

#### 4.3.1.1 Tech Stack

The API is written in JavaScript, using Node.js [12]<sup>30</sup> and is thus platform-independent [13]. Yarn was chosen as the package manager for its speed, built-in security and its ability to lock down all dependencies (and their dependencies et cetera) to a fixed version allowing for reproducible installs [14].

Several aspects of the API can be configured through *config.js* such as whether to log HTTP requests, perform the storage cleanup (Chapter 4.3.1.3), and which port the API is exposed on. Other configuration settings can be changed, too, yet they are less likely to be changed.

The main packages used are *express* [15] for the web server, *sqlite* [16] for database access, *lodash* [17] to parse data structures, and *winston* [18] for logging. Additionally, *eslint* [19] is used to ensure a consistent look and feel of the code and to enforce best practices [19].

To combat the excessive use of callbacks, which can be found in many Node.js projects, *async/await* are used wherever possible save for a few instances of *Promises* and callbacks [20].

#### 4.3.1.2 Database

All data, except for files (Chapter 4.3.1.3), is persisted in an SQLite database residing in the repository's root directory. The connection is handled by the *sqlite* package as defined in *src/db/db.js* and is used by important aforementioned file or the adjacent *src/db/queries.js*.

Upon startup of the API server, there are two scenarios for the database: either it exists, or it does not yet exist. In the former case, the existing database contents will be used to perform a cleanup of the storage whereas in the latter case, the database will be created and the migrations in the *migrations* folder will be executed.

To facilitate deleting a file through the frontend and to ensure consistency, foreign keys are used throughout the migrations and are enforced by SQLite [21].

---

<sup>30</sup> At the time of development, Node.js 8 Carbon was the latest long-term support (LTS) version. However, towards the end of development, a new LTS version, Node.js 10, was released and the application and its dependencies are compatible with both versions.

The schema of the database is reflected in several parts of the application and the same names are used throughout the API and frontend. There are seven main models – users, settings, files, projects, jobs, processes, and algorithms.

Since the application was designed for multi-user support, there is a *user* model which stores the username, a bcrypted [22, pp. 81-92] hash of the user’s password and, most importantly, an ID. This ID is used to associate other models with a user.

A *setting* is a simple key-value pair associated with a user ID. To provide a default for a setting, an entry is added where the user ID is NULL. These settings are exclusively used by the frontend.

A *file* serves as a reference to a file a user has uploaded and can be used for corpus analysis and classification. The database stores a universally unique identifier (UUID v1) [23], a name, and meta data (JSON) on how the file was parsed (assuming it has been parsed). The UUID is used for routing within the frontend and calling the corresponding API endpoints. This also applies to UUIDs of other models.

“An *algorithm* refers to a directory in the classification repository, consists of a name and description, and defines the interpreter (e.g. Python). Such an example is the “Support Vector Machine” algorithm. Furthermore, an algorithm, for it to be usable, has one or more *algorithm configs* which specify the command line arguments passed to an executable or script for various different calls (train, evaluate string, evaluate file). Each *config* has a name and optionally a description.” [4, p. 12]

“A *process* is an instance of an *algorithm config* and as such corresponds to an OS-level process. A *process* stores structured (JSON) and unstructured (plain text) output of a running algorithm as well as extracted and computed data (e.g. maximum F1 score, current F1 score etc.). Additionally, meta information such as PID<sup>31</sup>, hostname, status, and timestamps are stored.” [4, p. 12]

“A *job* is started by a user in the GUI. It consists of one or more *processes*, which were created based on the *algorithm configs* the user selected and stores the ID(s) of the input file(s). Based on the training file, a baseline is computed and stored as well as a user-specified job name. This data is augmented by meta information such as job status and timestamps in addition to a UUID.” [4, p. 12]

A *project* is very similar to a job yet where jobs are used for classification, projects are used for corpus analysis. A project consists of a file reference, configuration of the analysis (JSON) as well as further meta data such as the project name and when it was created. The result of the analysis is also stored in this model.

#### 4.3.1.3 Storage

The storage component of the API is where files uploaded by the user, as well as output and models of the backend are stored. There are a number of subdirectories in the *storage* folder (Table 7), which are named after the purpose they serve. Most files are stored without a file extension.

---

<sup>31</sup> The process ID given to a running algorithm by the OS.

<b>Folder<sup>32</sup> classification</b>	<b>Purpose</b>
<b>corpus_analysis</b>	This folder stores the output (JSON and CSV) from processes with a folder with the job's UUID as well as the results of the classifier comparison for a given job.
<b>evaluation</b>	Storage location for the output file of the corpus analysis bearing the project's UUID as filename.
<b>models</b>	While an evaluation is not persisted in the database, it is persisted in the file system. There is a folder for every evaluation (UUID) containing the corresponding output files.
<b>output</b>	The (intermediary and final) models of a classification job and its processes are stored in this folder within a subfolder with the job's UUID as name. These models are used for evaluation. For more information, see Chapter 4.2.3.1
<b>parsed</b>	The raw combined <i>stdout</i> and <i>stderr</i> of processes (from classifications) are stored in this folder using a combination of the job's UUID and process' ID as a filename.
<b>preprocess uploads</b>	After a file uploaded by a user has been parsed i.e. its columns mapped and its CSV format detected, the file will be parsed with these CSV options and saved in this folder using a standardized CSV format. There are exclusively files in this folder and their filenames (UUIDs) correspond to an entry in the <i>files</i> table. See Chapters 4.2.3.1.1 and 4.2.3.3 Stores the original file as uploaded by the user. Upon upload, the file is temporarily stored in the system's temp folder and is then moved to this folder. There are exclusively files in this folder and their filenames (UUIDs) correspond to an entry in the <i>files</i> table.

Table 7: List of the storage folders

As is evident from the above table, the first level of hierarchy within a storage folder consists exclusively of UUIDs (for both files and folders). This allows for a comparatively simple cleanup which is performed automatically upon startup. All the UUIDs in the database

<sup>32</sup> The folder names used in the table align with the settings in *config.js* in the *directories* object. The *backend* folder is omitted in this list as it serves as a link to the backend (Chapter 4.2) which is not part of the *api* repository but its own repository.



are combined and every file or folder on the first level of hierarchy, which cannot be found in the combined UUIDs, is deleted from the file system.

#### 4.3.1.4 API Server

The API server located in *src/api/server.js* is based on the Express framework [15] and makes use of a number of other packages acting as middlewares for CORS [24], multipart forms / uploads, compression, logging, cookies, as well as authentication.

The majority of the routes is defined in separate files with the same directory with names corresponding to the database model they serve. All but one route use either the GET or POST HTTP verb except for a single DELETE verb used to delete a file.

Authentication is handled in *src/api/auth-middleware.js* and uses JSON Web Tokens (JWT) [25] which is sent either as a HTTP X-Access-Token header<sup>33</sup> or as a HTTP Cookie.<sup>34</sup> The middleware checks whether the request (and thus the user) is authenticated with a valid token and decrypts the token's payload and makes it available to the routes in *req.jwt*. Should the authentication fail, a 401 HTTP status code is returned, and the request is aborted.

#### 4.3.1.5 Models

The three files located in *src/models*, *evaluate.js*, *job.js*, and *process.js* are the core controllers for the respective parts of classification tasks. They interface with both the Python controller (Chapter 4.3.1.6) and the database and are invoked by the API.

#### 4.3.1.6 Python

The backend i.e. Python processes are exclusively spawned [26] by *src/python/controller.js* with the help of *src/python/helpers.js* which provides helper functions.

While development was done on Windows machines, the application can also be run from within an Ubuntu-based Docker container (Chapter 4.5) and, as is common with Python, packages can either be installed system-wide or in a local *virtualenv*-folder [27]. To account for all these options, the helper function will attempt to find the correct Python executable.

Since Node.js' *child\_process* module requires arguments to be in array form, an additional helper is used to make this conversion easier.<sup>35</sup>

The controller file consists of several exported functions<sup>36</sup> which then call the internal *run* function. This function marshals the arguments to Python, sets up pipes for *stdout* and *stderr* (both for logging and storage) and creates a temporary file for the *--config* parameter used by the backend CLI (see Table 3).

---

<sup>33</sup> Preferred method when using a command line tool like *curl* or *httpie*.

<sup>34</sup> Even though not used at the moment, it is also possible to send the token in the query string (GET) or request body (POST).

<sup>35</sup> It is important to point out, the combination of Node.js and Python does not like the short form of a parameter (e.g. *-c*) and instead requires the long form (e.g. *--config*).

<sup>36</sup> Exported functions in a Node.js environment can be used by other files i.e. modules.



### 4.3.2 Files

While CSV files are ubiquitously used as a file format for exchanging data between different systems, there is no commonly-agreed-upon standard format and definition of column delimiters, quotes, and escape characters [5]. To tackle this challenge, every data file the application uses has to be first uploaded and parsed. After the user has uploaded a file and activated the option for “Smart CSV”, the cartesian product of all the possible options for delimiters, quotes, and escape characters is taken and the first 2’048 lines of the CSV file are parsed using these options. Should a user desire so, it is possible to manually set these options through the frontend. Due to the efficiency of the underlying Papa Parse library, the time spent on this exhaustive parameter search is negligible [28].

Once all the different combinations have been evaluated, the options with the least number of errors, preferably 0, is returned together with a preview of 250 rows. The user can then either confirm or adjust the parse options and upon confirming the correct set of CSV options, the complete file is parsed by the API using these options and written to disk with a fixed set of CSV options<sup>37</sup>.

By using a file buffer (as opposed to reading the whole file into memory) for reading (from the *uploads* storage) and writing (to the *parsed* storage), a file of any reasonable size can be parsed without a memory leak, assuming sufficient patience from the user<sup>38</sup>.

In addition to the CSV parse options, every file needs to have a column mapping which maps each column in the file to a certain column type. These types are<sup>39</sup>:

- ID
- String label
- Text
- Language identifier
- Date label
- Numeric label
- Binary label
- (ignore)

By default, after the algorithm has found the correct set of CSV parse options, an attempt will be made to guess the column mapping assuming the “Smart Headers” option was set. In a first step, the column headers, if present, are used and, if not yet successful, the first 100 rows of the data are used to infer a column type. This approach relies on the number of characters within a column, the number of unique values, as well as attempting to parse date strings. If the user is not satisfied with the result, the mapping can be adjusted.

---

<sup>37</sup> The newline character is `\n`, the delimiter is the semicolon, and quotes are only used if necessary and if so, single quotes are used.

<sup>38</sup> During testing, only files with sizes of less than one gigabyte have been used.

<sup>39</sup> See Table 4 for a detailed explanation of these column types and their effects.

### 4.3.3 Corpus Analysis

As briefly touched on in previous (sub-)chapters, corpus analyses are captured as projects within the API<sup>40</sup>. Consequently, the relevant API file is *src/api/projects.js* which defines straightforward routes to create a project, retrieve metadata and data for a project, and list all projects. Additionally, there is an endpoint to get all classification jobs using the same file. To allow for a non-naïve auto-refresh of the frontend when an analysis is still running, there is a */check* endpoint which returns a hash of the output file, allowing the frontend to compare the retrieved hash with the current hash of the file and, if the hashes do not match, load new data.

### 4.3.4 Classification

Classification and the closely linked Evaluate feature are, when looking at the number of files and code, the largest part of the API. This is in part due to more calls to the Python backend (six calls compared to a single call for corpus analysis) as well as building on and integrating previous work which had slightly different approaches.<sup>41</sup>

Classification and evaluation happens in two steps with an optional third step:

1. Preprocessing: to reduce execution time, all classifiers in the next step re-use the shared result from the preprocessing step (Chapter 4.2.3.1.1). This step happens once per classification job as it is, although re-usable for every subsequent process, dependent on user configuration such as tokenizers, file input type etc.
2. Running the classifier(s). Immediately following the successful preprocess step, the classifier(s), as selected by the user, are started and their output is captured in various forms.
3. Classifier comparison (optional, only if more than one classifier): once a process exits successfully, the classifier comparison is started<sup>42</sup> and its JSON output is available through the API and its output CSV files<sup>43</sup> are available for download.

---

<sup>40</sup> And also in the frontend.

<sup>41</sup> As the attentive reader may know, the previous work referred to here was done by the same developers half a year earlier. The API (which was previously called backend) developer amassed a considerable amount of knowledge while coding on both the previous and this repository and this learning curve resulted in different paradigms.

<sup>42</sup> This only applies to classification. When evaluating, the comparison is only started after all processes have exited. This is a trade-off between resource usage (classification runs the comparison multiple times) and wait time (while classification processes can differ vastly in execution time, evaluation processes take a similar amount of time).

<sup>43</sup> Only applies to classification and file evaluation.

## 4.4 Frontend

*Note: The work in this chapter builds on previous work which has been documented in [4]. While concepts are briefly re-explained in this document, the reader is kindly asked to refer to [4] for further details.*

*Note: By definition, the frontend is centered on visuals. The reader is kindly asked to consult the walkthrough chapter (Chapter 4.1) for screenshots.*

The frontend is the primary source of interaction for any end-user of texploration and allows a user to upload and parse files, start and explore a corpus analysis, and start, monitor, and evaluate classification jobs.

The frontend is web-based and runs in any modern browser. The frontend interacts with the API to load all data and is responsible for transforming and displaying that data. As this happens in the browser, the frontend can be served as static HTML, CSS, and JS files while the API and the backend can be scaled vertically and horizontally to account for usage.

### 4.4.1 Overview

#### 4.4.1.1 Tech Stack

The frontend is built on Twitter Bootstrap v4 [29] using the Vue.js framework [30]. Combining feature-rich Bootstrap with data-driven Vue enables the code to be declarative and ensuring the current state depends on the underlying data and not vice versa. Vue files are called components which are self-contained and can be re-used in other components.

The underlying data is exclusively loaded via AJAX/XHR [31] from the API and most requests return JSON<sup>44</sup>.

Just like for the API code, ESLint [32] was used for the frontend to ensure consistent use of syntactic features, spacing, and preventing bad practices by using both a general ruleset and one specific to Vue.js.

#### 4.4.1.2 Authentication

As discussed in the API's description (Chapter 4.3), users need to authenticate to use texploration with a username and password. In addition to every request to the API containing authentication, the frontend's router checks for valid authentication, too.

Since the API and the frontend are exposed on different ports, simply setting a cookie does not work. Instead, a cookie is set by the frontend authentication utilities in `_Auth/utills.js_` to store the token and is then appended to every request to the server as a `X-Access-Token` HTTP header. This header is then checked by the authentication middleware (Chapter 4.3.1.4) in the API and if unsuccessful, a 401 status code is returned. Otherwise the request will be processed further by the API.

---

<sup>44</sup> Except for the raw output of classification processes and requests that are header-only that rely either on the HTTP status code or on HTTP headers.

Upon receiving a 401 response, the frontend will clear the cookie containing the token and redirect the user to re-authenticate. Once the user has been authenticated, the originally intended route is loaded.

The lifetime of the cookie corresponds to the lifetime of the token which is set at 24 hours and will be renewed 3 hours prior to expiry (if possible).

#### 4.4.1.3 Settings

The frontend has a small number of global settings which are persisted via the API (Chapter 4.3.1.2). The implementation is based on Vuex [33], a global i.e. across components storage for Vue. Apart from cosmetic options to change the navigation bar, the settings allow defaults for CSV parsing to be set. Immediately upon changing a setting, the value is persisted, and all components are informed and use the new value. The Vuex store is defined in *main.js*.

Where sensible, there is an (implicit) option to temporarily override the global setting locally such as whether the file to be uploaded has headers.

#### 4.4.1.4 Label Coloring

To ensure consistent colorization of the same label string, e.g. “positive” across the whole application including corpus analysis, classification, and evaluation, a function, *colorArrayRandom*, exists in *main.js* taking care of this. The function has the following properties:

- It uses a list of twenty colors which are as different from each other as possible.
- For the same input, the same output is returned.
- A color cannot be re-used i.e. it can appear once at maximum.
- Once the list of twenty colors is exhausted, another color is returned satisfying the above two properties.

Consistent colorization is very helpful to the user to quickly identify a value without having to read it (see also Chapter 5.3) and is thus very beneficial to *texploration*’s usability.

#### 4.4.2 Files

The components for managing files can be found in the *Files* folder and contain the necessary code to list, upload, and parse files.

The *List* component displays every file a user has uploaded together with buttons to start a (or, if applicable, load an existing) corpus analysis or classification, to reset the parse options, and to delete the file (and its associated projects and jobs<sup>45</sup>). Additionally, the column mapping is displayed together with the original file name and the number of rows in the file.

After using the *Upload* component to upload a new file, the *Parse* component (and its sub-component, *ParseResults*) will be shown to first set the options for CSV parsing such as smart headers and smart CSV (see Chapter 4.3.2 for a detailed explanation). Once the API has returned a result containing the best (i.e. the one resulting in the smallest number of errors) set of CSV parse options, this will be saved within the component. In the same response, the API also returns (if activated) the guessed column mapping which can the

---

<sup>45</sup> As described in Chapter 4.2, a corpus analysis is also called a project and a classification a job.

be adjusted or set by the user. Once the user is satisfied, the save button will trigger the API to save the column mapping and parse the complete CSV.

#### 4.4.3 Corpus Analysis

A corpus analysis is started using the *Start* component in the *Project* folder while all components to display an analysis are located in the *Corpus* folder. This split was performed to avoid the need for refactoring when extending projects.

The *Start* component lets the user set a number of advanced options, primarily to configure topic modeling, should they choose to do so, and is otherwise very simplistic.

Once the corpus analysis has been started, the *Project* component takes over the lead and instantiates a number of sub-components located within the same folder. As defined by the router<sup>46</sup>, this component is supplemented by a left and a right sidebar containing an in-page navigation and a preview of the file, respectively. When the analysis is still running, the component periodically checks for a newer version using a dedicated API endpoint which returns the current MD5 hash of the analysis output which the frontend compares to the current hash it stores.

Immediately upon receiving the response from API, the content is transformed using *lodash*. This deals with the various types of labels the user can set when uploading a file; if one column contains the label as string (which is the case for e.g. sentiment analysis), the values in that column are then flattened and their label type is added back into the object. This transformation happens only once and is then passed on to the subcomponents.

Aforementioned subcomponents are fairly uninvolved; they perform their own, much smaller transformations to get the data in the shape and form they need to display it as intended. The *Tokenizer* (and *Sentence*) components are of particular note however. To allow for a visual comparison of tokenizers (and sentences), the results of the different algorithms are displayed in a table and differences and similarities are colorized and can be configured using radio buttons to only show or highlight similarities or differences.

#### 4.4.4 Classification

These components were recycled from the previous work mentioned at the beginning of the chapter. Minor changes were made to accommodate for the new concept of files.

One new feature, however, was added which seems to resonate well with users (see Chapter 5.3): classifier comparison, which can be found in the *ClassifierComparison* component within the *Monitor* folder<sup>47</sup>.

The comparison (in its full form<sup>48</sup>) displays two small tables as an overview to get insights into which labels were predicted unanimously and which ones were more challenging to predict correctly.

---

<sup>46</sup> And not by the *Project* component in the *Corpus* folder.

<sup>47</sup> It is used by both the job monitor and job evaluation features

<sup>48</sup> This depends on whether the file the comparison was run on contains labels.

Following the tables, a number of so-called sets are displayed which contain samples and the comparison results. A set can be a collection of samples which were all predicted incorrectly, a large number of different labels were predicted, or where all classifiers agree on the correct label. In order to facilitate the comparison, the labels are colorized<sup>49</sup> and whether a classifier predicted a label correctly is presented by an icon on a background colored after the label it predicted.

---

<sup>49</sup> In the same way they are colorized in the corpus analysis.

## 4.5 Docker

While setting up texploration is fairly straightforward (see Chapter 9.3), starting a Docker container is less involved. To build the image, the script expects Docker and Docker Compose to be installed on the host system and the repositories *api*, *backend*, *docker*, and *frontend* to be in the same parent folder. There are a number of trivial convenience scripts provided as described in table (Table 8).

Script	Purpose
<b>build.sh</b>	Executing this script will build the texploration Docker image, assuming a folder structure as described above. The <i>Dockerfile</i> and <i>.dockerignore</i> will be copied (and removed afterwards) one level up to circumvent Docker security restrictions of referencing parent directories.
<b>start.sh</b>	Starts the container in detached mode. Works only if the image already exists.
<b>logs.sh</b>	Since the image will be instantiated as a detached container, this file can be used to tail/follow the log output.
<b>stop.sh</b>	Stops the container. Contrary to <i>stop_and_clear.sh</i> state (i.e. volumes) is preserved.
<b>stop_and_clear.sh</b>	Stops the container and deletes the attached volumes which store the API's database and storage.
<b>save-image.sh</b>	Saves the generated image.

Table 8: List of Docker scripts

By using Docker Compose, apart from exposing ports, volumes can be specified [34]. Volumes in Docker are one of the ways to persist data [35] across containers i.e. instantiations of the same image [36]. By default, data is ephemeral and is lost as soon as the container exits<sup>50</sup> [35]. There are a mere two directories that need to be persisted: the folder containing the SQLite database (Chapter 4.3.1.2) and the API's storage (Chapter 4.3.1.3). Volumes are created automatically by the Docker engine and mounted at the correct location [34].

While the API and the frontend communicate exclusively via HTTP and could thus live in different containers, the API and the backend communicate via spawned processes and file system access requiring these two repositories to be in the same container. Consequently, the Docker image created by the *Dockerfile* is a monolithic container based on the general-purpose current Ubuntu LTS<sup>51</sup> release 18.04 [37].

A Dockerfile is cached top-down i.e. if a command's output or state changes, subsequent commands' caches will be invalidated [38]. To minimize build time the Dockerfile is de-

<sup>50</sup> No matter what kind of exit: clean, forced, crash etc.

<sup>51</sup> Long-term support

signed to first install global dependencies, then local dependencies ordered by how infrequently these are likely to be changed, and lastly the actual code is copied into the container. Additionally, the Dockerfile is a good reference point on how to setup a development environment.

To be able to start two processes in parallel upon booting the container, *pm2* is used which relies on *process.json* for configuration and its health and monitoring endpoint is exposed to the host [39].

While Docker was not used during development save for testing, it is the easiest and simplest way to start and use exploration without having to install the other repositories' dependencies locally.



## 5 User Test

One of texploration's goals is to be user-friendly and intuitive to use. In order to validate that aspiration and to gain valuable insights from users likely to be using texploration, a user test was conducted towards the end of development. While only a small subset of the feedback went into this iteration of texploration due to the timing of the user test, the feedback gathered in the user test should prove to be a valuable resource for future work. See Chapter 9.6 for the raw test notes.

### 5.1 Setup

The test was performed with four researchers of ZHAW's InIT. While two testers<sup>52</sup> had no knowledge of texploration, two testers<sup>53</sup> were already part of the user test of the application texploration builds on [4, pp. 38-40].

Each test lasted between 30 and 45 minutes during which the tester was sitting in front of a laptop which was running texploration and connected to a second screen which allowed the observer to take notes. The observer's primary goal was to capture as many details as possible from the tester interacting with the application. Furthermore, the observer spoke as little as possible in order not to influence the user as little as possible.

On the laptop, there were two windows: one running Google Chrome displaying the frontend, the other one being Windows Explorer showing the files the tester might need<sup>54</sup>.

Additionally, some corpora had already been uploaded and parsed for use in some of the steps of the user test.

After welcoming the tester, the tester was asked to read the instructions (Chapter 5.2) and if they had any questions before starting the test. Following the test, any remaining questions the tester had were, if possible, clarified.

### 5.2 Instructions

The instructions were as follows:

*Our application offers corpus analysis and classification of text corpora (CSVs). We have provided you with two sets of data. One contains tweets for sentiment analysis and the other one hate speech.*

*During this test, we'd like you to do the following:*

- 0. Please always think out loud and tell us why and what you are doing*
- 1. Upload and parse the hate speech file*
  - a. Give it a name containing your first name*
  - b. You'll only need the column toxic<sup>55</sup>*

---

<sup>52</sup> Dirk and Pius (these names correspond to the headings in the raw notes)

<sup>53</sup> Jan and Fernando (these names correspond to the headings in the raw notes)

<sup>54</sup> The "Upload file..." dialog of Google Chrome was set to the same folder and as such the Windows Explorer window was purely for reference should the tester desire to inspect the files outside of texploration.

<sup>55</sup> During the test "ID and comment\_text" was added for clarification.

2. *Delete the file again*
3. *Start a corpus analysis for tweets*
4. *Load the corpus analysis for the news groups*
5. *Start a classification on tweets*
  - a. *leave the tokenizers nltk*
  - b. *explore the classifier comparison*
  - c. *kill the algorithms which are taking too long*
6. *Evaluate the classification*
  1. *with string input*
  2. *with a file (already uploaded)*
7. *(optional) Load the corpus analysis for hate speech*

### 5.3 Observations

*Note: this chapter attempts to highlight certain key findings and common themes among the feedback gathered, yet the reader is kindly asked to consult Chapter 9.6 for the complete test notes.*

Two testers' first reaction was to compliment on the graphical user interface and, upon encountering these, on the use of toast notifications (e.g. Figure 12).

#### 5.3.1 Step 1: Upload and Parse

While no tester had any trouble uploading and parsing the hate speech corpus (step 2), two testers were confused about the *Parse* screen (Chapter 4.1.4.3). This might be due to the nomenclature used for this screen and missing verbiage explaining the process.

Upon encountering the column mapping (Figure 14), there was some minor confusion about what had happened yet this was quickly resolved when the dropdown menu with the column types (Table 4) was opened and the testers looked at the data preview<sup>56</sup>.

Eventually, after explaining the feature at the end of the test, all testers were content about the "Smart columns" and "Smart CSV" (Figure 13) options during parsing, yet they were equally confused and curious at first.

#### 5.3.2 Step 2: Delete File (Hate Speech)

Save for the question of the reason behind deleting the file they had just uploaded<sup>57</sup>, there was no feedback on the delete button.

---

<sup>56</sup> It should be noted, not all testers were familiar with all the corpora used during the test.

<sup>57</sup> The hate speech corpus was used for uploading as it contained many columns. The reason behind deleting the file, except for testing that functionality, was for time reasons; an analyzed hate speech corpus was already prepared and analyzing it takes about ten minutes which was too long for the purpose of this test.

### 5.3.3 Step 3: Start Corpus Analysis (Tweets) & Step 4: Load Corpus Analysis (Newsgroups) & Step 7 (optional): Load Corpus Analysis (Hate Speech)

*Note: While not intended, the testers analyzed both the Tweets and the Newsgroups analysis. Henceforth, this chapter combines the results from steps 3 and 4. Additionally, also the results from step 7 using the Hate Speech corpus<sup>58</sup> are included in this chapter since the findings concern the same screen and most of the feedback was not corpus-specific.*

Some testers asked about setting advanced options (Figure 19) when starting the analysis, and after confirming they did not have to, but could, started the analysis. Since the analysis does not immediately after starting have any results, one tester navigated away despite the sticky toast (Figure 20).

Once results were being loaded, the testers started exploring the analysis.

Concerning the overview table (Figure 29), one tester suggested displaying the median word/character count in addition to the current minimum, average, and maximum figures as well as a histogram of the underlying data.

The word cloud (Figure 33) raised a number of questions, mostly revolving around how the terms were selected and what exactly it does as well as whether the feature's label selection was being applied globally.

One tester suggested making the topic cards (Figure 34) interactive instead of being a plain deck of cards.

While the goal of the topic's mini-chart (Figure 37) is to be small, testers asked for more details to be included such as a comparison of the number of texts with a given label contributing to a certain topic to the number of texts with the same label on the whole corpus. Overall, however, testers appreciated the mini-charts.

Sentence splitting (Chapter 4.1.5.2.4) was, at that time, merely called "Sentences" which resulted in a non-understanding of the feature for most testers. However, after seeing the similar "Tokenizers", these questions were resolved. Since the default state for the radio buttons (Figure 41) is to "highlight all differences", changing "highlight" to "show" yields in no visible changes yet in the testers thinking the feature was not working.

In the Tokenizer (Chapter 4.1.5.2.5) section, the names of the algorithms were abbreviated. The abbreviation of "Stanford Core NLP" as "core" lead some testers to believe "core" was texploration's default i.e. core algorithm.

Several testers suggested substituting e.g. URLs, dates and, in the case of the Tweets corpus, usernames and hashtags for the purpose of comparing tokenizers.

One tester noticed how the labels were consistently colored (Chapter 4.4.1.4).

As the corpus analysis page is lengthy, one tester suggested using collapsible sections i.e. accordions in addition to the left sidebar (Figure 24) providing an in-page navigation.

---

<sup>58</sup> Due to a bug, the instances and labels were not correctly computed for this corpus for the purpose of this test. Specifically, instead of taking the truthy values for the binary labels, the inverse was taken.

A general theme among the feedback was the desire for more details of how the data being presented was generated and chosen as well as explanatory texts for each section. Interestingly, the right sidebar displaying a data sample was rarely discovered or used.

At the time of the test, the “top 5” (Figure 27, Figure 28) checkbox was displayed much less prominently causing confusion, especially with the 20 Newsgroups corpus only showing 5 labels.

Furthermore, a number of testers struggled loading a corpus analysis (see also Chapter 5.3.7).

#### 5.3.4 Step 5: Start & Monitor Classification (Tweets)

*The interested reader is kindly asked to also consult [4, pp. 38-40] for further findings.*

Almost all testers inquired about the “automatic split” (Figure 46) and whether they could influence the split in any way (except for providing separate training and test files).

Additionally, some testers asked for more details about the individual algorithms concerning their implementation and parameters (see also Chapter 5.3.6).

The classifier comparison (Chapter 4.1.6.2.2) feature was met with much enthusiasm by the testers. Yet two testers had difficulties interpreting the overview tables at the top of the classifier comparison. One of the two testers, while liking the comparison, would have preferred to see the classification report and confusion matrix before the classifier comparison. Additionally, while downloading the comparison sets is possible, one tester would like to paginate through the complete set without having to download it.

A fair number of comments were made asking about the precise implementation of the algorithms and whether it was possible to integrate one’s own algorithms into exploration. The same applies to the metric used (the F1 score) which some testers would prefer to be something else or a more fine-grained metric (e.g. per-label scores).

#### 5.3.5 Step 6: Evaluate Classification (Tweets)

There are multiple possible ways to navigate from the job monitor to the evaluation screen and, while most testers used the “Tools” (Figure 50) menu, one tester used the top-bar nav (Figure 44).

After noticing models were being saved, the testers performed several rounds of single string evaluation (Figure 57) using all models and various, and also amusing, strings, some of which were crafted in a way to test the model for nuances.

Two testers stated they were delighted to see the classifier comparison also being available during evaluation, and one tester downloaded and inspected the file.

#### 5.3.6 General Feedback

One common theme was the desire for more details on the algorithms and the processing used by the backend in all parts of the application. Additionally, the testers also expressed the need to set and configure hyperparameters as well as more different forms of metrics in the classification feature.

Overall, while the testers were critical and provided a lot of valuable feedback, they said multiple times they really liked the tool and complimented on the work being done on texploration.

### 5.3.7 General Observations

The application makes use of split dropdown buttons in several locations such as the top-bar navigation (Figure 4) or the file list (Figure 5, Figure 7). These dropdown menus were almost exclusively discovered by accident or not at all even though they are meant to provide contextual shortcuts to other parts of the application.

Another recurring theme among the feedback was about one of texploration's goals: to be a tool providing an overview of a corpus and to understand it. A number of the feedback items received during the test would like to see more fine-grained control over the parameters of the underlying algorithms. Finding the balance between a sleek and simple interface and offering the user more options is a challenge that will have to be explored in further iterations of the application.



## 6 Related Work

This chapter takes a look at other projects that have dealt with similar tasks on corpus analysis and classifier comparison.

### 6.1 Corpus Analysis

The statistical analysis and the topic modeling is nothing new when looking at the individual parts but is not available in combination yet<sup>59</sup>. Numerous papers can be found on the comparison of tokenizers, but they only inform on how the tokenizers work and provide advice when best to use which one. A tool that offers a visual comparison of the user data could not be located<sup>59</sup>.

#### 6.1.1 Kaggle

Kaggle<sup>60</sup>, the self-declared “Home of Data Science & Machine Learning”, provides very similar statistics on the columns to texploration. Each column is attributed to one of four types: *String*, *Numeric*, *DateTime* or *Boolean*. Coincidentally, these are equal to the label types used in texploration as listed in Table 4. Furthermore, Kaggle displays different metrics on the columns depending on the data type, an example of which can be seen in Figure 63. For *String* it shows how any unique entries were found and lists the top 5 (Figure 27, Figure 28) by occurrence together with the respective percentages. A bar chart counting fields with the same value and the mean of the column are illustrated for both *DateTime* and *Numeric*. For *Boolean* it simply lists how many percent are true or false. These metrics are a more advanced form of the column preview in texploration (Figure 25) and are worth considering when proceeding the development of texploration.

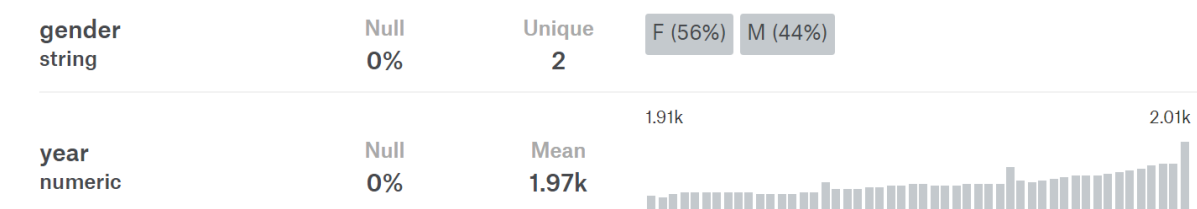


Figure 63 - Kaggle column metric [40]

#### 6.1.2 Lucene

Lucene is a Java-based, open source information retrieval tool, which has been around for over 20 years and is currently well established and widely used by, for example, Netflix, Twitter, and Instagram. At its core lies the Lucene inverted index that stores the data. Queries are inverted the same way to find and return related entries for a posed question. [41] Texploration uses a similar technique to establish the most relevant terms for the word cloud.

#### 6.1.3 Elasticsearch

First released in 2010, Elasticsearch has made a name for itself as a search engine and is used by Facebook, Mozilla, and GitHub to name a few. It is based on Lucene and offers advantages like a more intuitive API, inter-operation with other coding languages and

<sup>59</sup> To the best of our knowledge

<sup>60</sup> A platform for machine learning information, data, and competitions. [57]

clustering. [42] The latter allows for near real-time search results even on large datasets. [43] Implementing Elasticsearch within texploration in the future would prove beneficial, especially if the single text exploration module described in Chapter 7.2.6 will be developed, since it provides fast full text search with filtering options.

## 6.2 Classifier Comparison

There are numerous papers that compare classifiers and offer advice on which classifier is suitable on what kind of data, but until now no project exists<sup>59</sup> that provides a direct comparison of the classifiers on a data corpus chosen by the user like texploration does.

However, Sklearn also offers a visual comparison for classifiers by displaying the data as a two-dimensional scatter plot and underlaying it with the decision boundaries of the algorithm. While no direct comparison is done, it offers a quick overview of which algorithm is practical to use on a data set. As seen in Figure 64<sup>61</sup> a linear SVM is not practical for the second data set, but it performs very well on the third. It should be considered to be added to texploration as an additional visual aid.

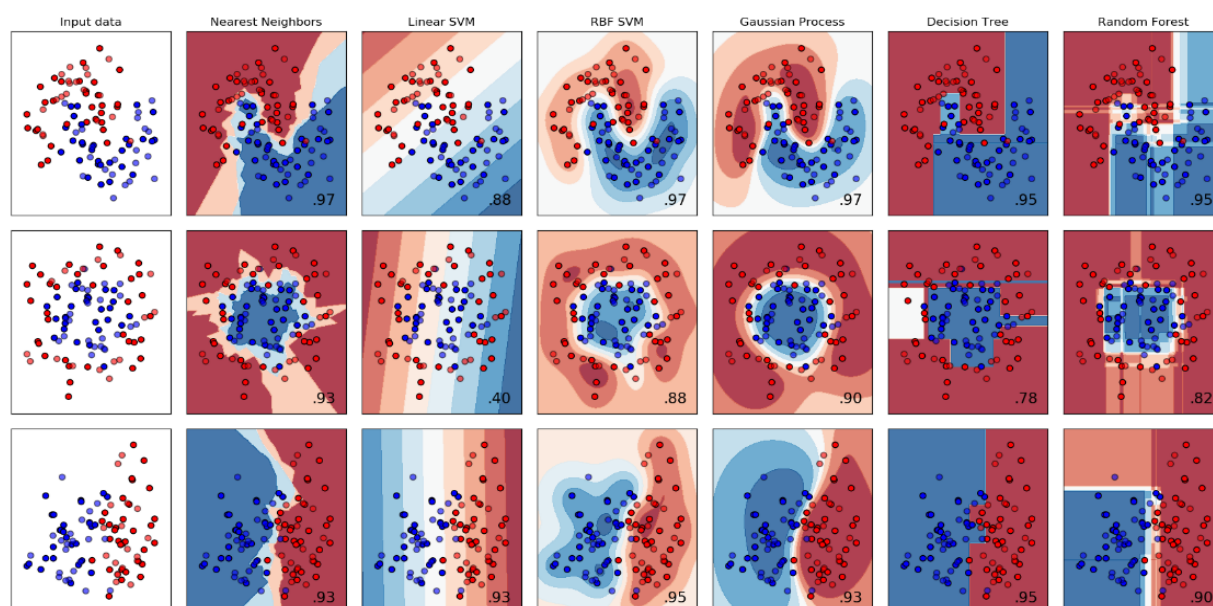


Figure 64 - Sklearn classifier comparison [44]

## 6.3 Conclusion

While all the comparisons implemented in texploration do not seem to have been done this way before, these works provide services and ideas to further develop this project and will be suggested in Chapter 7.2.

<sup>61</sup> Each row presents a binary data set that was used to train and test different classifiers. The numbers in the plots represent the accuracy of the model.



## 7 Conclusion

This chapter reflects on the results achieved (Chapter 7.1) and takes an outlook at future work (Chapter 7.2). As in previous chapters, the two parts are structured according to the repositories.

### 7.1 Achieved Results

#### 7.1.1 Backend

Even though *texploration* is built on previous work [4] nearly the whole code in the backend is new. The existing code for the classification was not deemed useful, since there were some problems with the time performance. The new and improved version now separates the preprocessing for the classification so that the former only has to be performed once for a classification job using multiple classifiers.

Deciding what information should be provided by the classifier comparison was a difficult process, since there are numerous statistics to be generated. Therefore, the output was revised multiple times during development to meet the requirements of a machine learning developer and as the user tests have proven, this task was completed successfully. Some of the other ideas for the comparison are listed in Chapter 7.2.1.

Building the corpus analysis was a constant balancing of the time performance versus the quality of the information generated. Especially when it comes to topic modeling, the current version is quite fast, but the results on some datasets have not been satisfactory.

Summarized, the backend, while there is still a great potential for improvement and expansion, runs stably and was built to be easily extended in the form of additional classifiers, such as new analyses on the corpus or tokenizer(s).

#### 7.1.2 API

Comparing this iteration of the API to the previous iteration<sup>62</sup> [4, pp. 12-14], a number of improvements were made, both in terms of features (see also Chapter 4.3) and robustness.

Besides adding support for the additional features driven by the backend, the API now supports JSON Web Token (JWT) for authentication of requests (Chapter 4.3.1.4) made against the API as well as a robust method to parse files (Chapter 4.3.2).

Every invocation of the Python backend (Chapter 4.3.1.6) is now made through a single method which provides logging and debugging information helpful for both API and backend development.

The folder structure used by the API clearly separates the source / code files from user-provided content and backend output (Table 7).

By using migrations for the SQLite database (Chapter 4.3.1.2) not only changes to schema are easier to retrace<sup>63</sup> but also friction among team members during development was reduced. Previously, when pulling the latest version of the schema, the database file had

---

<sup>62</sup> It was called “backend” in that iteration.

<sup>63</sup> E.g. by using *git blame*

to be deleted<sup>64</sup> whereas now the API automatically checks whether migrations are to be performed upon boot and proceeds to execute the migrations without any user interaction necessary.

The API is able to parse large<sup>65</sup> files while barely using additional memory. This is possible by using file buffers in the corresponding parts of the applications instead of reading the complete file into memory.<sup>66</sup>

### 7.1.3 Frontend

While most of the new components (Chapter 4.4.1.1) were driven by the work done in the backend (Chapter 7.1.1) and in fact do constitute the majority of the work being done, they have been described extensively in the walkthrough (Chapter 4.1). There are a number of other improvements however, as well as functionalities in components that can be re-used, directly or conceptually.

One key functionality is the color-coding of differences between the tokenizers which processes the backend's output data and formats in a way suited for the *Table* component [45] of the *BootstrapVue* framework.

The frontend now supports application-wide per-user settings (Chapter 4.4.1.3) which are stored using Vuex, "a first-party state management library for Vue.js" [33].<sup>67</sup>

In addition to adding authentication to the API (Chapter 4.3.1.4), the frontend has gained authentication support as well (Chapter 4.4.1.2).

Another new and appreciated (Chapter 5.3) feature is the consistent colorization (Chapter 4.4.1.4) of labels ensuring the same label is consistently colored across the frontend and across requests and users.

### 7.1.4 Docker

By using volumes and Docker Compose (Chapter 4.5), the Docker image now persists data whereas previously data was lost once the container stopped (or crashed) i.e. was ephemeral. Furthermore, by providing a number of convenience scripts (Table 8), using the Docker image, despite its added features, remains simple.

---

<sup>64</sup> The developer either had to notice the update manually unless informed by the committer or would see inexplicable errors happening when the API was running due to an incorrect database schema.

<sup>65</sup> See also Footnote 10

<sup>66</sup> During development of this feature, the API would crash some time after reading a file, no matter whether small (few MBs) or large (1 GB). After some time, this turned out to be a simple memory leak due to an unclosed buffer.

<sup>67</sup> Vuex is also used for the *TheSearch.vue* component found in the *Snippets* folder. While this component is currently not in use (due to feature re-prioritization), it is fully functional and to re-enable it, remove the *v-if="false"* from its instantiation in *TheNavigation.vue* and invoke it as it can be found in the *Corpus/Cloud.vue* component.

## 7.2 Future Work

In addition the repository structure, some aspects that are not part of a single repository, are listed as modules.

### 7.2.1 Repository: Backend

During the work on texploration, some problems were found but not yet addressed, as well as numerous ideas, for which there was not enough time during the project:

- **Tokenizer time performance:** Two of the implemented tokenizers, namely the Spacy [7] sentence tokenizer and the CoreNLP [9] word tokenizer take a very long time to complete.
- **Topic Modeling quality:** On some datasets the implemented topic modeling does not generate useful topics. This could potentially be solved by either building a better model and offering the user the choice between speed and quality, or by automatically determining the number of topics in the corpus by searching for the number of topics with the highest coherence of all topics.
- **Auto tune:** Automatically tune the parameters of the classifiers.
- **Parameter option:** Allow the user to choose the parameters for a classifier.
- **Add classifiers:** Implement more classifiers.
- **Expand classifier comparison:**
  - o Show two-dimensional plots of the classifiers as described in Chapter 6.2
  - o Analyze how many texts were classified correct over all classifiers.
  - o List data entries that were only predicted correct by one classifier to show how similar they predict.
- **Clustering:** Since the backend is computation-intensive in terms of both CPU and RAM usage, it may be desirable to have multiple instances of it distributed across multiple hosts, possibly using a cloud offering, in order to guarantee performance when running multiple corpus analyses or classifications.
- **Downsampling:** Offer the user an option to run a quick corpus analysis on only a percentage of the data.

### 7.2.2 Repository: Frontend

Development of the frontend is primarily pushed by new functionality implemented in the backend. There are, however, areas of interest, some of which were made evident during the user test (Chapter 5.3), which would benefit from attention in the next iteration of texploration:

- **Split navigation:** Both the top-bar navigation (e.g. Figure 4) and buttons within the main section of the content (e.g. Figure 7) use split buttons i.e. where the larger, clickable area represents one, primary action, and a dropdown menu next to it, represented by a downward-pointing triangle, offers additional, secondary actions. These dropdowns, while providing functionality, were rarely used or discovered during the user test.
- **Explanatory texts:** Despite the testers in the user test being more familiar with the research field of text classification, a number of features were not self-explanatory and only became clear after studying the feature for a prolonged time.

As also suggested by some testers, the corpus analysis would benefit from short, explanatory texts for each section.

- **Corpus analysis filters / per-label corpus analysis:** Currently, only the word cloud can be filtered by label(s) (Chapter 4.1.5.2.2). During the user test, the wish to see the corpus analysis for a given set of labels was expressed i.e. to be able to apply the word cloud filters to the complete analysis.
- **Renaming:** While a resource such as a file, project, or job can be named when it is being created, this name cannot be changed later by the user. In some instances, this may be a desirable feature.

### 7.2.3 Repository: API

Since the API itself is merely an interface between the frontend and the backend, there is little work to do besides adding new features. However, this shall not imply there is no room for improvement. Key areas of future work include:

- **SQL statements:** Due to integrating the previous work late in the process, there are two different forms of SQL statements present in the API: the previous work performs SQL statements using methods exposed by *src/db/queries.js* whereas in the new iteration, SQL statements are written as code in the corresponding places.
- **Data management:** There are several different ways of how JSON data provided by the backend is currently accessed: in some instances, the corresponding file on disk is read whereas in other instances the data is present in the database. While this is handled transparently by the implementation and of no concern to API consumers, it is inconsistent.
- **Routes:** Whereas the previous iteration favored calling methods in the body of a route, the current iteration favors putting the corresponding code within the body of the route as these functions are generally only invoked by one route.
- **Logging:** The current iteration provides extensive and useful logging for debugging. It is advisable, however, to improve both the logging capabilities for a production environment and for debugging as this proved to be a key point of information exchange between the frontend/API and backend developers.

### 7.2.4 Module: Neural Net Builder

Offer the user the option to define their own neural net by offering a selection of Keras [10] layers and parameters they can choose from. It would require a validator to check if the choice is a functional model.

### 7.2.5 Module: Sideload Classifier

During the user test the wish to test their own classifiers against the ones implemented in texploration was voiced multiple times. This could either be done by uploading the predictions on a test file and including this data in the comparison or loading a classification model which can be run just like the native algorithms. For the latter option, the input and output formats would need to be adjusted to fit the texploration standard.

### 7.2.6 Module: Single Text

Currently, it is not possible, except for the chosen samples, to view an individual text within a corpus and see statistics on it and how it fits into the corpus. One part of this new functionality would be, among many others, to display the labels applying to a selected text as well as comparing text length and word count to the corpus average values. Another aspect to this functionality could be a search form<sup>68</sup> to search for texts by content and label(s).

Implementing this feature would require work in all three repositories as well as the JSON format used to pass information between the frontend and backend through the API as there is currently no mapping between any form of output and the individual texts in the corpus this output stems from.

### 7.2.7 Module: Per-User Settings

The current iteration of *texploration* supports per-user settings (Chapter 4.4.1.3) allowing the user to tweak and customize some aspects of the application to their liking. The application would benefit, however, if more aspects were configurable on a per-user level such as custom column mapping hints<sup>69</sup>, the sort order for the list of files / projects / jobs or configuring the number of samples displayed in a certain section.

### 7.2.8 Module: User Management

Due to feature prioritization, the user management, while fully functional, is kept to a bare minimum<sup>70</sup>. Any changes such as adding a user or changing the password for a user have to be performed by running an SQL query against the database which requires knowledge about the database structure and the password hashing algorithm (Chapter 4.4.1.2). A non-exhaustive list of functionalities to add includes:

- Allowing users to change their password when logged in.
- Allowing users to reset their password in case they cannot recall it.
- Allowing new users / admins to sign up / create accounts.

### 7.2.9 Module: Range Filters

For the column types “numeric” and “date”, currently only the five most frequent values are displayed (Figure 28). Instead of capping these values, a user interface widget to filter these columns by range would be more sensible and might be more beneficial in terms of information content to the user. These widgets should take the spread of the data (e.g. whether it spans days or years for the “date” label) into account by adjusting the user interface elements accordingly.

Additionally, one might provide pre-defined filter range buckets based on the spread of the data, both equidistant (e.g. the same amount of days per range) as well as equal-count (i.e. where every bucket contains the same number of elements).

---

<sup>68</sup> To implement this feature, the hidden search functionality in the frontend (Footnote 67) could be re-stored.

<sup>69</sup> Currently, the column mapping guessing (Chapter 4.3.2) use a fixed algorithm which may or may not be suitable for some corpora. By letting the user add custom hints, the algorithm could be extended and customized to the user’s need. An example could be a “rating” column which could be *text*, *label*, or *number*.

<sup>70</sup> As of this iteration, there are two predefined users. See Chapter 9.2 for their credentials.

### 7.2.10 Module: Auto-Updating Data

The application uses auto-updating data in all instances where data is not ready immediately upon requesting it, such as when the user starts i.e. requests a corpus analysis and the results may not be ready for several minutes.

This is currently implemented by the browser polling the server periodically and unconditionally. While no performance issues have been reported so far due to this feature, issues are not unlikely to arise, both on the client side i.e. the frontend in the browser as well as on the server side i.e. the API:<sup>71</sup>

- Polling is done at maximum every 1,000 milliseconds<sup>72</sup> and, even though polling for changes is halted once the project or job has completed, this results in many requests being sent to the server.
- The corpus analysis is read from disk which leads to many I/O requests creating a possible bottleneck.
- When loading newer data, the complete component is re-rendered<sup>73</sup> which uses CPU and RAM on the user's computer. While this is partly mitigated using a hash to conditionally load new data (Chapter 4.3.3)<sup>74</sup>, this does not reduce the number of requests (in fact, this number increases<sup>75</sup>).

This could be solved by instead of having the client poll the server for updates, the server pushing updates to the client. This could be achieved using WebSockets. “[They] are an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.” [46]

During development of texploration, this functionality was explored and deemed to be feasible, yet out-of-scope within this iteration.

---

<sup>71</sup> This list does not aim to be complete.

<sup>72</sup> This depends on the component.

<sup>73</sup> While Vue only re-renders the sections that depend on data that has been changed [58], due to the nature of re-fetching all data, the complete component is re-rendered in most instances.

<sup>74</sup> This hash is in turn I/O-bound since it needs to hash the analysis file on the server. This could be mitigated by not comparing the file contents and instead comparing the file modification timestamp.

<sup>75</sup> When the hashes are different, another request is made to load the data.

## 8 References

### 8.1 Bibliography

- [1] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys (CSUR)*, vol. 34, no. 1, pp. 1-47, 2002.
- [2] X. Glorot, A. Bordes and Y. Bengio, "Domain adaptation for large-scale sentiment classification: a deep learning approach," in *ICML'11 Proceedings of the 28th International Conference on International Conference on Machine Learning*, Bellevue, Washington, USA, 2011.
- [3] A. Kachkach, "Problem-solving with ML: automatic document classification," Google Inc., 10 01 2018. [Online]. Available: <https://cloud.google.com/blog/big-data/2018/01/problem-solving-with-ml-automatic-document-classification>. [Accessed 03 06 2018].
- [4] N. Siddiqui and L. Metzler, "A Framework to Optimize Machine Learning Algorithms for Text Classification through an Intuitive User Interface," ZHAW InIT, Winterthur, 2017.
- [5] Wikipedia contributors, "Comma-separated values," Wikimedia Foundation, Inc., 29 04 2018. [Online]. Available: [https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values). [Accessed 12 05 2018].
- [6] J. W. Mohr and P. Bogdanov, "Introduction—Topic models: What they are and why they matter," *Poetics*, vol. 41, no. 6, pp. 545-569, 2013.
- [7] Spacy, "Industrial-Strength Natural Language Processing," [Online]. Available: <https://spacy.io/>. [Accessed 02 June 2018].
- [8] NLTK, "Natural Language Toolkit," [Online]. Available: <https://www.nltk.org/>. [Accessed 02 June 2018].
- [9] C. D. M. S. J. B. J. F. S. J. B. a. D. M. Manning, "The Stanford CoreNLP Natural Language Processing Toolkit," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 55-60, 2014.
- [10] Keras, "Keras," [Online]. Available: <https://keras.io/>. [Accessed 01 12 2017].
- [11] A. L. V. D. L. A. S. S. M. M. C. T. H. M. J. Jan Deriu, "Leveraging Large Amounts of Weakly Supervised Data for Multi-Language Sentiment Classification," in *WW 2017 - International World Wide Web Conference*, Perth, Australia, 2017.
- [12] Node.js Foundation, "Node.js," [Online]. Available: <https://nodejs.org/en/>. [Accessed 20 05 2018].



- [13] Node.js, "Downloads," [Online]. Available: <https://nodejs.org/en/download/>. [Accessed 12 05 2018].
- [14] Yarn, "Yarn," [Online]. Available: <https://yarnpkg.com/en/>. [Accessed 12 05 2018].
- [15] expressjs.com contributors, "Express - Node.js web application framework," [Online]. Available: <https://expressjs.com/>. [Accessed 01 12 2017].
- [16] Kriasoft, "kriasoft/node-sqlite," [Online]. Available: <https://github.com/kriasoft/node-sqlite>. [Accessed 20 05 2018].
- [17] @veksenn and @zthall, "Lodash," [Online]. Available: <https://lodash.com/>. [Accessed 20 05 2018].
- [18] winstonjs, "winstonjs/winston," [Online]. Available: <https://github.com/winstonjs/winston>. [Accessed 20 05 2018].
- [19] JS Foundation and contributors, "ESLint - PLuggable JavaScript linter," [Online]. Available: <https://eslint.org/>. [Accessed 12 05 2018].
- [20] M. Ogden, "Callback Hell," [Online]. Available: <http://callbackhell.com/>. [Accessed 12 05 2018].
- [21] SQLite, "SQLite Foreign Key Support," [Online]. Available: <https://www.sqlite.org/foreignkeys.html>. [Accessed 12 05 2018].
- [22] N. Provos, D. Mazières and T. J. Sutton, "Proceedings of 1999 USENIX Annual Technical Conference," in *A Future-Adaptable Password Scheme*, 1999.
- [23] P. Leach, M. Mealing and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," 07 2005. [Online]. Available: <https://tools.ietf.org/html/rfc4122>. [Accessed 12 05 2018].
- [24] MDN web docs, "Cross-Origin Resource Sharing (CORS)," Mozilla, 01 05 2018. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>. [Accessed 12 05 2018].
- [25] M. Jones, J. Bradley and N. Sakimura, "JSON Web Token (JWT)," 05 2015. [Online]. Available: <https://tools.ietf.org/html/rfc7519>. [Accessed 12 05 2018].
- [26] Node.js, "child\_process.spawn," [Online]. Available: [https://nodejs.org/api/child\\_process.html#child\\_process\\_child\\_process\\_spawn\\_command\\_args\\_options](https://nodejs.org/api/child_process.html#child_process_child_process_spawn_command_args_options). [Accessed 12 05 2018].
- [27] Python Software Foundation, "Installing Packages," 04 05 2018. [Online]. Available: <https://packaging.python.org/tutorials/installing-packages/>. [Accessed 12 05 2018].



- [28] M. Holt, "Papa Parse," [Online]. Available: <https://www.papaparse.com/>. [Accessed 12 05 2018].
- [29] @mdo and @fat, "Bootstrap · The most popular HTML, CSS, and JS library in the world.," [Online]. Available: <http://getbootstrap.com/>. [Accessed 12 05 2018].
- [30] E. You, "Vue.js," [Online]. Available: <https://vuejs.org/>. [Accessed 12 05 2018].
- [31] MDN web docs, "XMLHttpRequest," Mozilla, 12 04 2018. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>. [Accessed 12 05 2017].
- [32] JS Foundation and contributors, "ESLint - PLuggable JavaScript linter," [Online]. Available: <https://eslint.org/>. [Accessed 12 05 2018].
- [33] vuejs, "Vuex," [Online]. Available: <https://vuex.vuejs.org/en/>. [Accessed 21 05 2018].
- [34] Docker Inc., "Compose file version 3 reference," [Online]. Available: <https://docs.docker.com/compose/compose-file/>. [Accessed 12 05 2018].
- [35] Docker Inc., "Use volumes," [Online]. Available: <https://docs.docker.com/storage/volumes/>. [Accessed 12 05 2018].
- [36] Docker Inc., "Docker overview," [Online]. Available: <https://docs.docker.com/engine/docker-overview/>. [Accessed 12 05 2018].
- [37] Canonical Ltd., "Ubuntu release end of life," [Online]. Available: <https://www.ubuntu.com/info/release-end-of-life>. [Accessed 12 05 2018].
- [38] Docker Inc., "Best practices for writing Dockerfiles," [Online]. Available: [https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/). [Accessed 12 05 2018].
- [39] Keymetrics, "PM2 - Advanced Node.js Process Manager," [Online]. Available: <http://pm2.keymetrics.io/>. [Accessed 12 05 2018].
- [40] P. T. Mooney, "USA Name Data," Kaggle, 21 April 2018. [Online]. Available: <https://www.kaggle.com/datagov/usa-names/data>. [Accessed 1 June 2018].
- [41] A. Bialecki, R. Muir and G. Ingersoll, "Apache Lucene 4," in *SIGIA*, Portland, 2012.
- [42] P. Gupta and S. Nair, "Survey Paper on Elastic Search," *International Journal of Science and Research*, vol. 5, no. 1, pp. 333-336, 2016.
- [43] D. Kalyani and D. Dr. Mehta, "Paper on Searching and Indexing Using Elasticsearch," *International Journal Of Engeneering And Computer Science*, vol. 6, no. 6, pp. 21824-21829, 2017.

- [44] Sklearn, "Classifier comparison," [Online]. Available: [http://scikit-learn.org/stable/auto\\_examples/classification/plot\\_classifier\\_comparison.html](http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html). [Accessed 01 June 2018].
- [45] BootstrapVue, "Table," [Online]. Available: <https://bootstrap-vue.js.org/docs/components/table>. [Accessed 21 05 2018].
- [46] MDN web docs, "WebSockets," Mozilla, 29 03 2018. [Online]. Available: [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API). [Accessed 22 05 2018].
- [47] Microsoft, "Visual Studio Code," [Online]. Available: <https://code.visualstudio.com/>. [Accessed 30 05 2018].
- [48] JetBrains, "PyCharm," [Online]. Available: <https://www.jetbrains.com/pycharm/>. [Accessed 30 05 2018].
- [49] texploration, "texploration," [Online]. Available: <https://github.engineering.zhaw.ch/texploration>. [Accessed 30 05 2018].
- [50] Python Software Foundation, "Welcome to Python.org," [Online]. Available: <https://www.python.org/>. [Accessed 02 06 2018].
- [51] Docker Inc., "About Docker CE," [Online]. Available: <https://docs.docker.com/install/>. [Accessed 02 06 2018].
- [52] Docker Inc., "Install Docker Compose," [Online]. Available: <https://docs.docker.com/compose/install/>. [Accessed 02 06 2018].
- [53] Sklearn, "Sklearn," [Online]. Available: <http://scikit-learn.org>. [Accessed 1 June 2018].
- [54] Telegram, "Telegram Messenger," [Online]. Available: <https://telegram.org/>. [Accessed 02 06 2018].
- [55] A Telegram user, "Telegram: Add Sticker Set," Telegram, [Online]. Available: <https://t.me/addstickers/QueenElizabethII>. [Accessed 20 05 2018].
- [56] Telegram, "The Queen," [Online]. Available: <https://t.me/addstickers/QueenElizabethII>. [Accessed 01 06 2018].
- [57] Kaggle Inc., "Kaggle: Your Home for Data Science," [Online]. Available: <https://www.kaggle.com/>. [Accessed 03 06 2018].
- [58] Vue.js, "Reactivity in Depth," [Online]. Available: <https://vuejs.org/v2/guide/reactivity.html>. [Accessed 22 05 2018].



## 8.2 Figures

Figure 1: Home screen .....	17
Figure 2: Login screen .....	18
Figure 3: Settings screen .....	18
Figure 4: Top-bar navigation for Dashboard.....	18
Figure 5: Dashboard screen .....	19
Figure 6: A file which has not yet been parsed.....	20
Figure 7: Classification dropdown menu for a file .....	20
Figure 8: Top-bar navigation for Files .....	21
Figure 9: A single file in the file list and the upload button .....	21
Figure 10: Upload screen .....	21
Figure 11: File name and button with passing validation state.....	22
Figure 12: Persistent toast notification while a file is being uploaded.....	22
Figure 13: Parse options for a file.....	22
Figure 14: Parse results and column mapping for a file .....	24
Figure 15: Persistent notification while a file is being saved .....	25
Figure 16: Notification after the file has been saved successfully .....	25
Figure 17: Top-bar navigation for Corpus Analysis .....	26
Figure 18: Start screen.....	26
Figure 19: Start screen with advanced options active.....	27
Figure 20: Persistent notification while the analysis is running.....	27
Figure 21: Notification when new data has been loaded .....	27
Figure 22: Notification when the analysis has been completed .....	27
Figure 23: Top part of the corpus analysis screen for the hate speech corpus .....	28
Figure 24: Left sidebar navigation .....	28
Figure 25: Right sidebar containing a data sample of the hate speech corpus.....	29
Figure 26: Corpus analysis header with classification start button and a (closed) dropdown menu.....	29
Figure 27: Corpus analysis header with the (unchecked) checkbox to show all values ..	30
Figure 28: Toast notification indicating the number of values per label column has been capped .....	30
Figure 29: The Stats section for the hate speech corpus .....	30
Figure 30: 2075 texts have the “obscene” label and no other label .....	31
Figure 31: 3981 texts have both, “obscene” and “insult” as labels which does not include the 2075 texts which only have the “obscene” label set.....	31
Figure 32: 636 texts have the three labels “obscene”, “insult”, and “identity_hate” which, again, does not include either the 2075 “obscene” or the 3981 “obscene, insult” instances. ....	31
Figure 33: Word cloud for the hate speech corpus displaying the terms which do not have any labels.....	32
Figure 34: Complete card deck of all topics for the 20 News Groups corpus.....	34
Figure 35: Excerpt of the topics for the 20 News Groups corpus.....	34
Figure 36: A part of the topic table showing the topic words, number of texts, and the number of texts per label for a topic .....	34
Figure 37: Detailed view of the mini-chart for topic 3 .....	35

Figure 38: An excerpt of the sentence splitting comparison for hate speech .....	36
Figure 39: A single sentence splitting comparison displaying only non-agreement.....	37
Figure 40: Table and chart comparing the overlap between tokenizers .....	37
Figure 41: An example of the tokenizer comparison on the hate speech corpus .....	38
Figure 42: Tokenizer comparison only highlighting differences between the tokenizers .....	38
Figure 43: Load screen.....	39
Figure 44: The top-bar navigation for classification .....	39
Figure 45: The upper part of the screen to start a new classification job.....	40
Figure 46: Choosing separate train and test files for a job .....	41
Figure 47: Choosing word and sentence tokenizers when starting a job .....	41
Figure 48: The list of available algorithms for classification .....	42
Figure 49: Name input and start button for a job.....	43
Figure 50: The overview at the top of the job monitor .....	44
Figure 51: Classifier comparison overview tables for the tweets corpus.....	44
Figure 52: The set of texts all incorrectly classified by every process .....	45
Figure 53: An excerpt of the “Disagreement” set.....	46
Figure 54: Tooltip appearing when hovering over the icon.....	46
Figure 55: The “All correct” set lacks the column for every process. ....	47
Figure 56: The process detail box of a CNN.....	47
Figure 57: The Evaluate screen for a job with separate train and test files.....	49
Figure 58: Results for a single string evaluation on models generated by a job on the tweets corpus .....	50
Figure 59: The header of a result card for file evaluation and the “Disagreement” set....	51
Figure 60: Download buttons for each model to download the annotated file .....	52
Figure 61: Sentence Tokenizer .....	59
Figure 62: Word tokenizers .....	60
Figure 63 - Kaggle column metric [40] .....	83
Figure 64 - Sklearn classifier comparison [44] .....	84

### 8.3 Tables

Table 1: Callable Files .....	53
Table 2: Utility Files .....	53
Table 3: Call Parameters .....	54
Table 4: Column types.....	55
Table 5: Parts of Corpus Analysis .....	56
Table 6: Comparison Breakdown .....	64
Table 7: List of the storage folders .....	67
Table 8: List of Docker scripts.....	75
Table 9: Default username/password combinations for texploration .....	100
Table 10: Timetable for the Bachelor's thesis .....	114

### 8.4 Code Snippets

Code Snippet 1: Corpus analysis configuration JSON .....	57
Code Snippet 2: Preprocess configuration JSON.....	62
Code Snippet 3: Classification configuration JSON .....	62

## 9 Appendix

### 9.1 Official description of the task

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

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

Die manuelle Auswahl des geeigneten Systems für eine konkrete Klassifikationsaufgabe ist sehr viel Aufwand. Deswegen möchten wir gern ein Framework entwickeln, das verschiedene Basis-Systeme für Textklassifikation kennt und für einen neuen Task automatisch ein möglichst gutes Klassifikationssystem aufbaut. Damit könnte man quasi auf Knopfdruck jedes beliebige Klassifikationsproblem lösen. Das Framework soll jedes Basis-System für den Task trainieren und optimieren, und anschliessend das beste System daraus entwickeln.

Das Ziel dieser Arbeit ist der Ausbau eines Frameworks, das zurzeit in einer PA entwickelt wird.

Konkrete Aufgaben sind:

- Einarbeiten ins Thema und ins bestehende Framework
- Integration von verschiedenen Basis-Systemen. Dabei kann auf bestehende Systeme zurückgegriffen werden, die am InIT entwickelt wurden
- Anwendung des Frameworks auf Klassifikationstasks und Vergleich mit dem State of the Art in der Wissenschaft. Je nach Qualität der Resultate können die Ergebnisse als Open-Source-Tool oder sogar in einem wissenschaftlichen Paper publiziert werden.

## 9.2 Login Information

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

Username	Password
linus	123
nadina	456

Table 9: Default username/password combinations for texploration

## 9.3 Installation Instructions

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

1. Installing texploration to run it (Chapters 9.3.1 and 9.3.2) using Docker.
2. Installing texploration for development (Chapter 9.3.1)<sup>76</sup>.

The code has been written using Visual Studio Code [47] (API and frontend) and JetBrains PyCharm [48] (backend). These tools have proven to be very useful and assistive during development yet are not required for further development.

### 9.3.1 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 as these provide knowledge about the tech stack used. To get started, the *api*, *backend*, and *frontend* repositories [49] have to be cloned and should each have the same root folder.

*Setting up texploration for development is a very similar process to what Docker does when building the application using the Dockerfile (Chapter 4.5). 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<sup>77</sup>: Node.js 9<sup>78</sup> [13], Yarn [14], and Python 3 [50]. 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 repository suffices.

To satisfy the dependencies required by the *backend* repository, *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 `python3 -m spacy download en`

<sup>76</sup> While this method can also be used to run texploration, it is more time-consuming than the other, Docker-based install type.

<sup>77</sup> This usually happens automatically. A restart after installing these programs is recommended for the changes to the environment variable to be picked up by every other program and shell/terminal.

<sup>78</sup> See Footnote 30 for background on why Node.js 9 and not 8 or 10.



- 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 the repository. Additionally, <http://nlp.stanford.edu/software/stanford-english-corenlp-2018-02-27-models.jar> needs to be downloaded to the root folder, too.

While the backend 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*.

### 9.3.2 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 and have local copies of the *api*, *backend*, *frontend*, and *docker* repositories [49] within the same root folder.

In addition to the above setup, the reader should have Docker [51] and Docker Compose [52] installed on the host machine. The build process has only been tested on Ubuntu<sup>79</sup>.

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 Chapter 4.5, especially Table 8, for further details.

### 9.3.3 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<sup>80</sup> as well as the *docker* repository. Additionally, the reader is asked to read Chapter 4.5.

The tarball is a complete image of texploration as exported by *save-image.sh* (Table 8) 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) can be used.

---

<sup>79</sup> Both, bare-metal and virtualized.

<sup>80</sup> This file is distributed on a USB thumb drive.

## 9.4 Meeting Notes

### 9.4.1 2018-02-06

#### 9.4.1.1 PA

- gutes Englisch
- guter technischer Überblick
- mehr Software-Engineering als ML
- ML (zu) wenig thematisiert
- USer-TEst gut
- es hat funktioniert!
- zu wenig scientific report, zu fest blog post
- BA: mehr scientific
- Schriftgröße grösser
- related work zu kurz
- Forschungsstrang von AutoML; read papers
- DB Schema vergessen
- DB regenerating process
- spawn / child process Problem
- "start new job" unklar, was ausgewählt wird -> algo configs are confusing; remove text descriptions; describe algo configs
- deep-mlsa ist sentiment analysis und nicht text classification im Allgemeinen -> highlight the difference / it being non-generic
- Pokal grösser!!!
- BA: wissenschaftliche Fragenstelle mehr im Vordergrund
- eher lang und ausführlich
- bei cross reference: "Chapter" not "chapter"
- walk-through zuerst --> an den Leser denken
- Data upload: just one input auto-split train and test 80/20..
- Farbgebung running/killed
- "Monitor" <select> "Select" not "Enter"
- Monitor screen: size *and* number of sentences/rows etc.
- current/max highscore: when completed, show max (not interested in "current")
- "inter-annotator agreement" for Evaluate
- Don hat ein Paper für "facts & figures"
- Table 5 merged two tables -> anti-UX
- Table 5: bold/emphasize maximum value
- Table 5: weird results
- unusual to evaluate training set
- entweder nach jeder Epoche auf Test-Set evaluieren oder separates validation set (70% test, 10% validate, 20% test); cross-validation is a bad idea -> **BA**
- fix random seed für splits

#### 9.4.1.2 BA

- "no sensible use of deep learning in practice so far" - Mark via conference in Spain
- "Text Analytics Workbench"
- auto-sklearn

- Input: CSV mit Text und Labels oder ohne Labels
- verstehen, was in den Texten ist
- einzelnen Text (Satz, Tweet) und gesamte Collection kann betrachtet werden
- Einzeltext: statistische Verteilungen, Textlänge, Wordcloud, vgl. zu Durchschnitt, Sprache, Topic, evtl. sentiment analysis
- Collection: filtern (nach Label, Topic, Länge etc.) -> text exploration, Suche, häufigste Wörter
- Topic modeling
- preprocessing: tokenizing, sentence splitting, encodings etc.; find the "correct" tokenizer -> verschiedene anwenden und die Outputs vergleichen (auch visuell) und was sind die Unterschiede; same for sentence splitting
- ganzer Text oder einzelne Sätze haben Labels: Überblick bekommen, Statistik, Verteilung, Verteilung über Textlänge/tfidf, majority class -> remove that and look again
- Auch bei Plaintext: HTML, XML -> headings, captions, strong/em --> small language (front matter / yaml; markdown -> use for display); "AI Markup Language"
- job ads can be used
- Ergebnisse -> Exploration der Resultate..! Welche Sätze gehen gut, welche nicht? Zeigen wieso etc.
- wie werden die Algos konfiguriert? choose word embeddings?
- entity extraction etc.

## 9.4.2 2018-02-22

### 9.4.2.1 Meeting

- sketches by hand of the **MVP**
  - GUI (-> features)
- Mark's vision
  - statistics
- "single text" get an idea about a single document (sentence, website, book chapter)
  - text length, most frequent words **etc.**
  - 2) "corpus" get an idea about a text collection
    - distribution among texts, labels, text lengths **etc.**
    - filter by keyword/label/text length/...
    - from collection (2) to single document (1) for exploring
  - preprocessing
    - lowercasing, URL replacement, number replacement <-- regex
- choose a tokenizer and see (visual) result
  - 2) compare tokenizers (highlight matches/differences) and with facts&figures (graphs)
  - 3) sentence splitting
  - preview sentence splitting + tokenizers
  - save selection as config
  - algorithms
    - based on preprocessing config, run algos
    - output a classifier

- use default for statistics and also allow saved config from preprocessing to be used in statistics again
- text labelling
  - consider a text may have n labels, even per word (POS)
  - string/int/date/datetime/hierarchical/geo
- when uploading, allow to choose/assign columns (which one is text, which ones are labels)
- standard input: csv with ID (maybe not present) and label(s) + text
  - also: JSON
  - also: files + list with names and labels
- filter by labels, possibly time range
- find a good framework!!
- user management
  - user + password
  - ohne ZHAW Anbindung o.ä.
- store jobs etc. on server
- job (processs) management API?
- prod: runs on server, not locally
- spawn/exec -> display PID in GUI
- internal representation of data structure (JSON, Markdown etc.) and message passing format
  - store in ElasticSearch?

#### 9.4.2.2 *ToDo*

- **see above**
- check out ElasticSearch + Kibana
- Lucene
- find Vue framework for data-heavy apps
- talk to Christopherus Nal TE 425 or Melanie Imhof

#### 9.4.3 2018-02-26

##### 9.4.3.1 *Questions*

- replacement regex: on/off
- filters greedy oder everything-has-to-match
- display global/local inline/separate
- <https://github.engineering.zhaw.ch/texplorer>

##### 9.4.3.2 *Meeting*

- regex-replacement is part of pre-processing, toggleable
  - possibly user-defined with input for regex and replacement plus active checkbox
  - show matches / test regex (first 5 replacements)
- group filters into AND/OR/NOT
- corpus
  - apart from word cloud
    - pie charts
  - most important words tf.idf by class/label
  - Satzlängenverteilung, Wortverteilung etc.

- topic modeling (10)
- tokenizers
  - default tokenizer to analyze corpus -> for corpus screen
    - configurable / presets
    - anschauen, was dieser Tokenizer produziert hat
  - compare tokenizers in that corpus
- Don
  - distribution, compare tokenizers
- GUI
  - upload
    - files
      - (format, encoding)
      - mini-analysis, show column titles, what is what (ID, text, label)
    - (choose default tokenizer, possible smart/clever (natural language))
  - overview/corpus
    - corpus screen
      - KISS
    - full-text search ("St. Gallen") -> view text in reader
    - click word cloud -> enter word into search
    - bar chart in distribution -> show documents matching that term/feature/etc.
  - text reader / single text
    - text screen
    - view results of corpus analysis in detail
    - metadata (ID, labels)
    - facts & figures, word cloud, sentence length, sentences
  - settings
    - default tokenizer
    - password

#### 9.4.3.3 ToDo

- user stories
  - Was ist in meinem Korpus an Texten?
  - Understand label distribution in corpus?
  - Ich will mein Preprocessing für die Classifier korrekt einstellen
  - ...
- dashboard frameworks (front- and backend)
- data source
  - swiss sentiment corpus (spinningbytes -> Jan, Don); multilabel
  - 20 news groups
  - scikit-learn

#### 9.4.4 2018-03-04

##### 9.4.4.1 Questions

- *none*

#### 9.4.4.2 Meeting

#### 9.4.4.3 ToDo

- Linus
  - click search query vuex do search
- Semesterplan mit Aufwand, Integration ML
- auto-sklearn

#### 9.4.5 2018-03-12

#### 9.4.5.1 Questions

- Timetable
- How many tokenizers? Sample size to return?
- Meetings weeks 14-16

#### 9.4.5.2 Meeting

- sample size question:
  - sentences where there are stark differences
  - 5-10 samples
- Stanford NLP Tweet tokenizer
- 3 tokenizers

#### 9.4.5.3 ToDo

- Nadina
  - tfidf instead of word frequency
  - total row of label stats in JSON

#### 9.4.6 2018-03-19

#### 9.4.6.1 Questions

#### 9.4.6.2 Meeting

#### 9.4.6.3 ToDo

- Linus
  - check if lines skippable (2 header rows)
  - show samples for errors
  - display tokenizers

#### 9.4.7 2018-03-26

#### 9.4.7.1 Questions

#### 9.4.7.2 Meeting

- flag for multi-label?
- combine labels? labels depending on one another (e.g. hate data set with label "toxic")
- auto-sklearn: use PA-style

#### 9.4.7.3 ToDo

- Linus

- api: integrate Python calls
- docker
- corpus: outline navigation
- main/randomColors: maximal differenzierbare Farben
- corpus index: change pie chart in label stats to bar chart
- upload index: add column type "ignore", "date"
- corpus index: stacked bar chart / onclick -> show bar chart for sub-labels
- Stats/Tokenizers: show vertical/col borders
- Upload: improve smart CSV design
- remove smart header option -> always on
  - display alert please double check mapping
- upload parse results: rename "Data"
- upload: improve user flow -> parse -> config -> save
  - possible save on server first and check user abandonment (to conserve space)
- cleanup "text"
- corpus: global/label filter, date/value filter
- Nadina
  - align sentence splitting by sentence start
  - word tokenizer bug "Seriously, though, we are trying to write an encyclopedia here, so don't make joke edits"
  - topic modeling
  - apply global/label filter, date/value filter to computations
  - detect binary/non-binary labels

#### 9.4.8 2018-04-04

##### 9.4.8.1 Questions

##### 9.4.8.2 Meeting

##### 9.4.8.3 ToDo

- integrate PA
- Linus
  - rename everything "job" to "project"
  - re-organize Corpus view
  - general stats:
    - number of texts/rows
    - what labels are there
  - hide table behind "show more" for bar charts
  - add output.json endpoint
  - colorize sentences just like word tokenizers
  - instance pie chart
    - show total number of rows next to "number of instances with no labels"
    - show also as table
    - inline bar plot to indicate relative frequency (progress bar?)
  - store total number of rows in file in DB
  - corpus > stats > label table: remove "Total"
  - instance distribution
- Nadina

- instance distribution (how many rows have 1 label, 2 labels etc.)
- sort instances by
  - frequency/count
  - number of labels, THEN frequency/count

#### 9.4.9 2018-04-09

##### 9.4.9.1 Meeting

- topic modeling
  - configurable number of topics
  - configurable number of words to show
  - configurable number of features
  - configurable n for n-grams
  - configurable weights (y/n)
  - default: 10
  - input[type=range]
  - it's about getting a feeling for the data; not too many details (weights etc.)
  - show topic distribution (topics/documents)
  - n-grams? (n=2,3) ("new york") -> more features
  - label distribution in topics
  - write about that in BA report
  - display in table
  - fancy chart?
- prefer fast computations

##### 9.4.9.2 ToDo

- integrate PA
- Linus
- Nadina
  - general stats about corpus; possibly also per label
    - avg. sentence length
    - distribution of sentence length
    - correlation of labels and text length
    - ...

#### 9.4.10 2018-04-16

##### 9.4.10.1 Questions

- how to deal with values in labels (buckets)
- what is desired for classification comparison? PA integration?
- Nacht der Technik? Nadina FW vice president

##### 9.4.10.2 Meeting

- no Nacht der Technik
- BA Buch
- prioritize PA integration over fine-tuning

##### 9.4.10.3 ToDo

- Linus
  - user flow



- corpus analysis: update results?
- don't use buckets for charts etc. -> use top 5 for every label
  - stacked bar chart for text labels?
- improve guessColumns()
- when setting column types, display first 3 rows next to
- data preview: sortable
- data preview: 50 rows instead of 100 -> add setting
- submit button below column mapping selects
- reset -> rename to "re-configure CSV format"
- after parse -> show corpus (or preview), NOT analysis options
- analysis options as option under corpus
- corpus: show columnMapping in right sidebar (?) including data excerpt
- row\_count without header row!
- label stats total row -> move to "general" table
- chars/words min/avg/max: boxplot
- dashboard links
- new corpus order of components:
  - general corpus stats
  - by word cloud of complete corpus (non-configurable)
  - list of topics (no details)
  - label table
  - label chart
  - instance table (histogram, number of instances)
  - instance chart
  - ...
  - sentences
  - tokenizers
- topics as word lists next to each other
- sort label chart descending by count/value
- show "no instances" disclaimer from instance chart under label chart, too
- instance distribution chart
  - sort
  - add permanent labels
- integrate PA
- use Nadina's debug flag for analysis preview
- figure out how to display labels with multiple values

9.4.11 2018-04-23

9.4.11.1 Questions

9.4.11.2 Meeting

- bad topic modeling
- settings for algorithms in classification
  - presets only for now
- compare classification?
  - samples where no classifier is correct
  - total agreement
  - different tags
  - sample  $n=10$

- ? compare à la github activity graph (color-coded boxes)
  - | label |
  - | ground-truth color | color for C1 | color for C2 |
- BA presentation
- upload test set

#### 9.4.11.3ToDo

- Linus
  - Corpus: if no jobs in split dropdown, hide split
  - increase "no. features"
  - Upload+Parse separate
    - use file\_id instead of path
    - rewrite cleanup
    - use for Corpus and Evaluate
    - "Use column mapping from... [other file]"
  - (re-)add Start for Classification
    - checkboxes for algos
    - choose tokenizers
    - no-autostart
  - same color for labels (again; due to label-values; see sentiment)
- Nadina
  - cleanup
  - take documents that have actual topics and test topic modeling
  - figure out good default for topic modeling
    - 20newsgroup
  - auto-sklearn
  - generic deep learning (to satisfy BA title)
    - Keras
    - train word embeddings beforehand / with Keras
  - output
    - models

#### 9.4.12 2018-04-30

##### 9.4.12.1Questions / Announcements

- features are frozen
- please test Docker
- backend
  - better topics
  - CNN
  - classifier comparison
- frontend/API:
  - lots of UI tweaks and polishing
  - everything is in a file hierarchy
  - display classifier comparison
  - more details for Evaluate
  - consistent label colorization
  - better column guessing
- report ideas / suggestion / outline

- code docs similar to PA?
- classifier comparison
  - footer content (absolute/relative)
  - no labels, also report?

#### 9.4.12.2 Meeting

- user test
  - Jan
  - Fernando
  - Pius von Däniken
  - Dirk von Grünigen
- report
  - classification: copy-paste from PA
  - code docs: extensive, as prep for future PA; à la <http://dreamboxx.com/mark/data/PABAs/PA16CrossDomainSentimentWeilenmann-vonGruenigen.pdf>
  - literature overview: other tools, other approaches (automl, elasticsearch, gate etc.); beyond the tellerrand
  - key results from work
  - what does it mean to understand a corpus?
  - why do we have the features we have
  - also talk about why we don't have some things
  - 1-click solution
  - talk about user stories; why they are important
  - topic modeling: sometimes good, sometimes bad

#### 9.4.12.3 To Do

- Linus
  - delete file confirmation
  - classifier comparison
    - rename "All labels" rename "Strong disagreement"
    - 731 out of 3810 labels (negative (369 / 580), neutral (105 / 1916), positive (257 / 1314))
    - NEW: All classifiers are wrong (correct / strongly disagree) for **731** texts out of **3810** texts (x%). Per labels (negative (369 / 580 / x%), neutral (105 / 1916 / x%), positive (257 / 1314 / x%))
    - duplicate above text and show above cards as summary: *alltrue*, *alldiff*, *all\_false* + weak disagreement ("other")
    - download
    - *allfalse before alldiff*
  - Evaluate single\_string: colorize labels
- Nadina
  - classifier comparison
    - download complete everything
    - all\_diif: more than half of the labels
    - for evaluate w/o label: disagreement, agreement
- are there any good topics for tweets? 10...20 topics, 1...3 n-grams

### 9.4.13 2018-05-07

#### 9.4.13.1 Questions

- copy paste PA?
- DoD?
- User test room
- BA Tool: German title is in English
- meetings?
- 9 am on June 8?
- hand-in requirements

#### 9.4.13.2 Meeting

- BA needs to be self-contained
- treat PA as "source"
- treat PA code as "Vorwissen"
- indicate code from PA
- "Classification.vue" -> quick overview, reference PA
- Report: project management / schedule (appendix)
- presentation
  - non-public (Mark, Don, Vogel, ...)
  - half-half time split
  - assume knowledgeable audience
  - motivation for BA, introduction, why this took so long to build, why we chose it etc., basics, applications, show demo (10 minutes -> prove it's working)

#### 9.4.13.3 To Do

- Linus
  - see above!!!!
  - classifier comparison: set: All incorrect / undefined (NaN%)
  - Evaluate: prevent from leaving page
- German title -> email Mark
- ask Don about Abgabe
- Nadina
  - evaluate with no labels crashes

### 9.4.14 2018-05-14

#### 9.4.14.1 Questions

- German title: Automatische Optimierung von Machine Learning- und Deep Learning-Algorithmen zur Textklassifizierung
- publication on website is okay
- title wrong?!
- tweets: no good topics b/c texts are too short -> mixups ("game" -> sport/GoT, "apple"/"netflx"+watch etc.)
- Docker

#### 9.4.14.2 Meeting

- topic question from above:

- accept as-is
- but possible with larger corpora
- greater number of topics?

#### *9.4.14.3ToDo*

- new title
- 100 topics on tweets

## 9.5 Timetable

<b>Week (semester   calendar)</b>	<b>To-Do</b>
<b>1   8</b>	Kick-off
<b>2   9</b>	foundation, login
<b>3   10</b>	foundation, libs
<b>4   11</b>	Linus: upload / import; Nadina: three tokenizers
<b>5   12</b>	Nadina: import of config+data; Linus: display tokenizers
<b>6   13</b>	Nadina: settings / config; Linus: display tokenizers
<b>7   14</b>	Nadina: auto-sklearn; Linus: settings / config
<b>8   15</b>	Nadina: auto-sklearn
<b>9   16</b>	(reserve)
<b>10   17</b>	(reserve)
<b>11   18</b>	running algos with configs
<b>12   19</b>	testing
<b>13   20</b>	write thesis
<b>14   21</b>	write thesis
<b>15   22</b>	proofreading
<b>16   23</b>	hand-in

*Table 10: Timetable for the Bachelor's thesis*

## 9.6 User Test Notes (raw)

### 9.6.1 Test 1 - Dirk

(45 min)

#### 9.6.1.1 Pre

- quick explanation
- showed files
- read the instructions

#### 9.6.1.2 Notes

- Bootstrap
- noticed already uploaded file
- wondered about instructions which hate file is which
- found correct file
- named it correctly
- liked the toast notifications
- got stuck on parse screen
- realized smart headers
- nice gui!
- looked at the complete data preview screen
- successfully deleted the correct file
- uploaded tweets successfully
- started corpus analysis for tweets and was happy about the result
- news group analysis was missing

*next step*

- successfully loaded hate speech corpus analysis
- sentence tokenizers: wonders about what the colors mean; showed hint about the radio buttons -> realized what it's all about; wondered about what nltk, spacy are; figured out it's all about agreement; started thinking about "sentence splitting"
- word tokenizers: "looks much more like tokenizers"
- asked whether "core" is our own?
- accordions + explanation texts
- started thinking about integrating own tokenizers
- went back to sentences: realized it's about sentence splitting
- correctly interpreted word cloud
- wondered about label chart w/ binary label ":0"
- looked at the corpus stats table
- instance distribution was wrong
- overlap for sentences missing

*next step*

- news groups corpus analysis
- topic analysis: correctly interpreted mini chart
- tokenizers: non-sensical text
- found sidebar nav

- asked how we choose sample
- can't do anything with topic boxes
- asked about sub-newsgroups / labels
- word cloud clickable?!
- topic mini chart: small values hardly legible; (note: log scale?)

*next step*

- often uses top nav instead of "Dashboard"

*next step*

- classification
- asked about "automatic split"
- found sklearn algos
- confused about when to "Start"
- looked at horse race chart
- individually killed algos
- when killing, show toast
- didn't notice changing color of algo after killing it
- looked at comparison: correctly interpreted table "all incorrect"
- looked at comparison table, didn't immediately understand it -> heading is about comparison but classifiers aren't show in table
- after looking at comparison sets, understood table
- looked at "Processes"
- rounding differences between classification report and F1 score
- wonders about tensorflow resource usage
- wonders why killed CNN showed epoch graph and SVM didn't show anything
- asked about integrating own algos
- didn't notice process status badge
- asked whether sequential or parallel execution

*next step*

- asked about what "Monitor" is when navigation to "Evaluate"
- noticed models are saved
- entered string input, tried mean and nice version
- left evaluation to check label distribution
- went back by using browser back button
- SVM skewed label distribution
- another mean string input with hashtag
- noticed CNN not producing output (note: it hadn't saved a model yet due to being killed)
- downloaded comparison for NB, Excel had troubles with the format
- noticed we "just add the column"
- wondered about column name "NB"
- download all classifiers?
- how do extract features? 1-hot? word embeddings? tfidf? unclear... -> important!;
- asked whether we use corpus analysis options for classification
- asked about file/project/job hierarchy



*next step*

- good for exploration, for classification still needs hyperparameters, feature engineering
- "gseht super us"
- long pages

*next step*

- didn't notice any of the split dropdown links in content buttons

*next step*

- asked whether to inspect raw csv; showed explorer + Excel

*next step*

- functionality is there
- started envisioning future work

**9.6.1.3 Post**

- further comments?
- thank you
- chocolate

**9.6.2 Test 2 - Jan**

(45 min)

**9.6.2.1 Pre**

- quick explanation
- showed files
- read the instructions

**9.6.2.2 Notes**

- trouble finding "Upload"
- "wow, fancy gui, gseht guät us"
- wondered about "Smart CSV"
- happy about guessed headers
- didn't realized data preview was scrollable horizontally
- "sehr hübschäs GUI"
- asked about "ignore"
- confused when "parse" happened
- deleted file

*next step*

- start corpus analysis
- happy to set advanced options for corpus analysis
- asked about exploding laptop for n-grams 3
- happy about toast notifications
- delighted about charts
- numbers are correct (he knows the corpus by heart)

- really likes it
- asked about instances, explained it
- "mega cool" about pie chart and toxic/other columns for hate speech
- wondered what the word cloud does
- noticed "tomorrow" is always big in word clouds
- looked in-depth into negative word cloud
- explain word cloud (frequency of words?)
- asked if word cloud label radio buttons also affect topics
- would like to know more about topics, how they were generated
- "1 topic per weekday?" topic modeling is hard
- made the link between advanced options and number of topics
- can advanced options be re-configured later? would be nice
- slightly confused about topic table
- topic mini-chart: show comparison of topic labels compared to total number of that label (positive vs. negative with tweets)
- after some time, figured out sentence splitting
- spacy tokenizes URLs
- talked about spacy speed
- asked about blank lines in sentence splitting? "Zwischensatz" -> alignment
- tokenizer overlap: what exactly does it refer to?
- asked about how we get tokenizer samples; would like an explanation text
- replace links by tokens/placeholders

*next step*

- didn't figure out how to "load a corpus analysis"
- didn't know what the corpus looked like and didn't see right sidebar
- confused about 20 instances but only 5 labels
- label distribution graph should start at 0
- was curious about topic analysis, noticed a few good ones
- happy about topic mini charts, delighted
- noticed what the different tokenizers screw up
- wondered where the sentences are
- had to show "top 5" checkboxes
- had to show right sidebar with data preview

*next step*

- started classification via Dashboard
- quickly wondered about split
- add validation file? 2 or 3 files?

*next step*

- noticed sentence, word tokenizer in monitor header
- looked at baseline
- noticed F1 scores are displayed
- wishes for other scores than F1 to compare them (e.g. pos/neg or pos/neg/neut)
- wanted to kill CNN, but wait for an epoch; checks out raw output and really likes it
- made the link between colors and process status

- asked about what hyperparameters were chosen (kernel?)
- "all incorrect" correctly realized what it's about and really liked it "super feature"
- really likes the classifier comparison!
- looked at all sets

#### *next step*

- navigated to evaluate by using top nav, not "Tools"
- positive string input, waited patiently for results, everything positive; changed it to negative, everything negative
- unchecked CNN for file
- chose correct file for evaluation
- happy about the tool in general
- delighted evaluate shows comparison, too
- correctly figured out why sets were different (no labels)
- really likes the comparison!

#### *next step*

- bad topic modeling for hate speech
- (note: hate speech seems to be wrongly analyzed)
- likes seeing the differences
- overlap is always high

#### *next step*

- really likes it!
- really like the PA part, would like more metrics, hyperparameters etc.
- sees a lot of potential in it!

#### 9.6.2.3 *Post*

- further comments?
- thank you
- chocolate

### 9.6.3 Test 3 - Pius

(45 min)

#### 9.6.3.1 *Pre*

- quick explanation
- showed files
- read the instructions

#### 9.6.3.2 *Notes*

- clicked on Files, clicked Upload
- didn't correctly name the file
- noticed defaults and used them
- successfully ignored columns
- saved files
- likes saving and toast
- successfully deleted file

*next step*

- wanted to re-upload a file instead of starting from the already uploaded file
- wanted to navigate to classification, didn't read toasts; sent him back
- word clouds: most frequent words by frequency
- playing around with word cloud checkboxes
- glanced over sentences, silent for a long time, wondered what sentences are about ("differences between tokenizers?")
- found right sidebar
- (laptop overloaded; loaded previous corpus analysis)
- wondered how samples were chosen; told him
- found topics, "interesting"
- had a laugh about topic 6
- correctly interpreted topic table
- wondered how topics were chosen for table
- asked about "core"
- standard token substitution? (URLs, usernames etc.)
- told him about space and about core bracket substitution

*next step*

- trouble between "Start" / "Load"
- click on toast -> dismiss
- more classes, looked at instance distribution
- for word/char counts: in addition to min/max: also show median, histograms (for skewed distributions)
- had to show "top 5" checkboxes after he inquired about "missing labels" in the table
- button per component for "top 5"
- "no one wants to talk about religion"
- checked word cloud for hocky terms when checkbox was active
- asked (correctly) about checking two labels for word cloud
- looked at topics, noticed religious, crypto, cars topics
- asked about the corpus ("newsgroups")
- liked topic mini charts
- noticed consistent label coloring!
- looked at religion topic mini chart
- wondered whether tokenizers would be better on news texts
- had to explain show/hide/full/... radio buttons -> he understood why
- really likes the GUI, "rächt cool"; was prepared for a "standard grauenhaft" GUI

*next step*

- chose correct file, automatic split
- started job
- identified F1 scores as "accuracies"
- SGD: why is it in there?
- killed processes through process box (not Tools)
- noticed similar execution time of AdaBoost and Naive Bayes
- asked about whether you can restart an algo?

- clicked around a bit, started looking at classifier comparison, was silent for a while
- noticed only 20 were "all correct" for "negative"
- troubles interpreting the first classifier comparison table; too many "Total"s in there; good comparison, but hard to read
- identified "all incorrect" correctly
- after looking at many exclamation sign icons, wondered why "negative" and "positive" are both read; was happy about the tooltip
- correctly identified exclamation sign and tick icons and liked it
- happy to see confusion matrix and classification report -> prefers these over the classifier comparison table, should come first
- inquired about classification report per epoch for CNN
- F1 scores: 0.64 vs 64% -> assumed accuracy because of percentage (technically speaking, F1 is not a percentage)
- different ways to average F1 score
- likes download feature of sets
- paginated viewing all samples of a set?

#### *next step*

- started evaluation through "Tools", had a bit of trouble finding it
- string "this demo is awesome", all positive, satisfied
- noticed CNN not producing any output
- string "workin as intended #meow", all neutral
- string "missed my bus #fml", SGD negative, others neutral
- evaluated using 20newsgroups
- liked the output, even though it was non-sensical, confusion matrix showed and he was slightly confused
- re-evaluated using "tweets no labels"
- sanity checked whether disagreements are the same as before

#### *next step*

- UI quirks, but overall likes it
- wouldn't have found a few things if it weren't for the instructions

#### 9.6.3.3 *Post*

- further comments?
- thank you
- chocolate

### 9.6.4 Test 4 - Fernando (30 min)

#### 9.6.4.1 *Pre*

- quick explanation
- showed files
- read the instructions

#### 9.6.4.2 *Notes*

- said he had some questions about instructions, but we'd get to them later

- asked whether to delete other files
- uploaded file successfully
- parse successful
- successfully ignored columns
- wished for batch "set to ignore"
- successfully deleted file

*next step*

- started corpus analysis through top nav
- chose correct file
- looked at advanced options; asked whether to change any
- looked at sidebar
- (loaded previous analysis for performance/time reasons)
- looked at topic cards extensively, found them to be a bit strange at times
- noticed a lot of appointments in topics
- asked whether to look extensively at corpus analysis; said yes
- talked about overlap chart
- and left

*next step*

- had to show how to load a corpus analysis
- got hung up on only 11k texts for 20 news groups which should be 20k
- knows 20 news groups really well
- went through every topic (in cards) and tried to identify it (all correctly)
- skimmed over everything else

*next step*

- loaded existing analysis instead of starting
- automatic split: percentage??
- wanted to choose spacy
- inquired about details of AdaBoost
- noticed trophy icon
- inquired about details of SVM, suggested we use linear SVM (better for texts)
- had slight troubles interpreting "all incorrect" at first
- noticed HTML entities (&lt; ;)
- surprised SVM was still running (note: he hadn't killed anything yet)
- studied "all incorrect" set and tried to come up with a reason why they were wrong (non-words, typos, capitalization etc.) -> looked extensively at the text provided, and tried to figure out why they were labelled as they were
- looked at confusion matrix
- noticed pos/neut are often confused and not as many neuts as others
- recall for pos is bad for NB
- asked whether F1 score was macro
- analyzed AdaBoost
- wondered why SVM was still running
- killed all processes via Tools when he wanted to go to Evaluate

*next step*

- used Tools to navigate to Evaluate
- single "Trump will decide over Iran today!!!!" -> all neutral
- single "FC Winterthur will win the championship" -> 2 neg, 2 pos, he was slightly confused
- single "I do not like bananas" -> 1 neg, 3 neut
- chose newsgroups for evaluation
- noticed old evaluations are still around and correctly realized (without reading the text) evaluations are lost upon reloading
- realized class mismatch of tweets and news groups
- analyzed results, nothing was classified as negative

#### *next step*

- loaded hate speech corpus analysis
- looked at word cloud; found "fuck" in word cloud
- went through topics, noticed only 10 topics, found the cuss word topic
- (??) something about the number of texts, topics, and probability
- would like to sort topic table
- had a chuckle about sentence tokenizer samples
- (still) didn't go into much depth about tokenizers

#### *next step*

- likes it as a "getting a good overview" tool
- likes visualizations
- still hung up on SVM

#### 9.6.4.3 Post

- further comments?
- thank you
- chocolate

## 9.7 License Information

Due to the nature of the programming languages involved (JavaScript and Python) we made heavy use of open source packages. As this application is not intended neither for commercial usage nor redistribution, no special considerations were made regarding whether a package might not be usable due to its license. We would like to thank the authors and contributors of these packages for their work.