FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Visual Comparison of Natural Language Processing Pipelines

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Patrick Hromniak
Matrikelnummer 1425731

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Dr.Techn. Waldner Manuela, Msc

Wien, 21. November 2018

Patrick Hromniak	Waldner Manuela

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Visual Comparison of Natural Language Processing Pipelines

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Patrick Hromniak
Registration Number 1425731

to the Faculty of Informatics

at the TU Wien

Advisor: Dr.Techn. Waldner Manuela, Msc

Vienna, 21st November, 2018

Patrick Hromniak

Waldner Manuela

Erklärung zur Verfassung der Arbeit

Patrick Hromniak
1150 Wien, Grimmgasse 19/9, Österreich

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

Wien, 21. November 2018

Patrick Hromniak

Danksagung

Mein Dank gilt an meine Betreuerin Manuela Waldner, die mich bei dieser Arbeit unterstützt hat und mit Ihren Treffen für sehr viel Input sorgte. Darüber hinaus möchte ich meinen Kollegen herzlichst danken die das Studium zu einem tollen Erlebnis gemacht haben und immer für Hilfe bereit waren.

Acknowledgements

I would like to thank my supervisor Manuela Waldner for supporting me at writing this thesis and always being available for meetings and discussions. Additionally, I would like to thank my friends from university who were always open for help during studying and made university a joyful experience.

Kurzfassung

In dieser Arbeit wurde ein Interface entwickelt, welches dem User ermöglicht verschiedene Natural Language Processing Pipelines zu vergleichen. Es gibt dem Benutzer die Möglichkeit die verschiedenen Output-Vektoren der Pipelines zu betrachten und die Dauer der Berechnung, sowie auch die Ähnlichkeit zu vergleichen. Da es keine klare Best-Practice gibt wie eine solche NLP korrekt konfiguriert werden kann, wurde dieses Tool entwickelt um Pipelines graphisch darzustellen. Grundsätzlich sollen Pipeline Schritte die starke Auswirkung auf das Ergebnis haben durch dieses Tool erkannt werden. Schlussendlich soll mit Hilfe dieses Tools die beste Pipeline-Konfiguration durch visuellen Vergleich der Ergebnisse von bekannten Dokumenten gefunden werden.

Abstract

Natural Language Processing comprises a variety of operations that can be applied on raw text to extract features. The sequence of operations is called NLP pipeline. However, the sequence and parameters of these individual operations differ between applications. In each step of the ongoing sequence, a single process is performed with a specialized task. Such a task can be the determination of the end of sentences or the removal of so-called stop words. There is no best-practice which combination is most effective and accurate to determine the descriptive features (key words) of a single document. The goal of this bachelor thesis is to compute the features of different variations of NLP pipelines and visualize them as basic word clouds. It is also important to know how the resulting word cloud of each pipeline is affected by varying the order of certain steps, adding steps or removing steps. The presented interface gives an overview of performance and similarity values of each computed pipeline.

Contents

CHAPTER 1

Introduction

The following thesis focuses on Natural Language Processing (NLP) and introduces a visual interface for comparing individually created NLP pipelines. NLP offers various steps to extract information from a given document. This work concentrates on human written language which is already in machine-readable format. Our language has predefined rules which make it clear for the human how certain constellations and structures of a text have to be understood. On the one hand, we have certain sets of words with their individual meaning and usage. Those words are either nouns, verbs, adjectives, adverbs or also prepositions and conjunctions. On the other hand, we have the rules of a language, herein after referred to as grammar, that are always well defined and combine all available words together to create semantic comprehension. This defined set of rules gives us the opportunity to process texts with a machine like a human reader and extract semantic information from the input.

However, NLP offers various steps to extract features from a document. These steps are combined and executed one after another. Each step has its individual impact on the resulting features. We further refer to this process as an NLP pipeline. An example of such a pipeline can be observed in Figure 1.1

Every pipeline processes either a single document or a large document set and after the process is finished, the pipeline returns a set of features which describe it. Resulting words are commonly called keywords. However, the resulting keywords are always a partial subset of all features available in the processes document. The following example

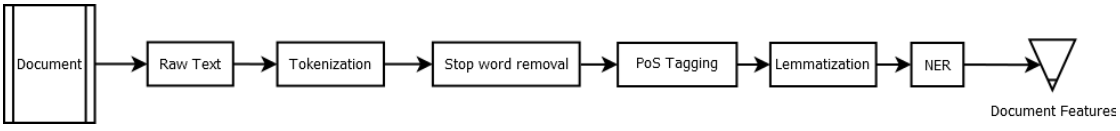

Figure 1.1: An example Natural Language Processing Pipeline

shows how a text from an anonymous Amazon comment is processed by each individual step.

Example Sentence: *Classic Tarantino film with intense scenes and well written dialogue. Christopher Waltz does a superb job of playing the evil nazi and his Oscar for it was well deserved. The story is not true to historic events and obviously doesn't claim to be.*

We assume that our text is already sanitized and in raw format. The first step is tokenization according to the sample pipeline in Figure 1.1. After tokenization our example text is split into single pieces called tokens.

Example Sentence: *['Classic' 'Tarantino' 'film' 'with' 'intense' 'scenes' 'and' 'well' 'written' 'dialogue'" "'Christopher' 'Waltz' 'does' 'a' 'superb' 'job' 'of' 'playing' 'the' 'evil' 'nazi' 'and' 'his' 'Oscar' 'for' 'it' 'was' 'well' 'deserved'" "'The' 'story' 'is' 'not' 'true' 'to' 'historic' 'events' 'and' 'obviously' 'doesn't' 'claim' 'to' 'be']*

The input text contains a lot of information that has to be processed, but does not contribute to the actual meaning of the sentence. The next step that is executed is stop word removal. We remove all unnecessary words from the sentence. Stop words that are removed are usually taken from an dictionary suiting the language we are dealing with.

Example Sentence: *['Classic' 'Tarantino' 'film' 'intense' 'scenes' 'well' 'written' 'dialogue'] ['Christopher' 'Waltz' 'does' 'superb' 'job' 'playing' 'evil' 'nazi' 'Oscar' 'well' 'deserved'] ['story' 'not' 'true' 'historic' 'events' 'obviously' 'doesn't' 'claim']*

The sample text is now free from stop words. The next step shown in the example pipeline is part-of-speech tagging, which is an essential preprocessing step for following steps such as lemmatization. We now assign each token a matching tag according to its word form. This tags are commonly shown as NN,VB,RB,ADJ. Where NN stands for nouns, RB for verb and so on.

Example Sentence: *[('Classic', 'JJ'), ('Tarantino', 'NNP'), ('film', 'NN'), ('intense', 'JJ'), ('scenes', 'NNS'), ('well', 'RB'), ('written', 'VBN'), ('dialogue', 'NN'), ('Christopher', 'NNP'), ('Waltz', 'NNP'), ('superb', 'VBD'), ('job', 'NN'), ('playing', 'VBG'), ('evil', 'JJ'), ('nazi', 'JJ'), ('Oscar', 'NNP'), ('well', 'RB'), ('deserved', 'VBD'), ('story', 'NN'), ('true', 'JJ'), ('historic', 'JJ'), ('events', 'NNS'), ('obviously', 'RB'), ('claim', 'NN')]*

After this step named entity recognition is be performed. For example we want to extract persons or organizations. An ideal output could be a concatenation of phrases, such as "Classic Tarantino Film" or "Christopher Waltz".

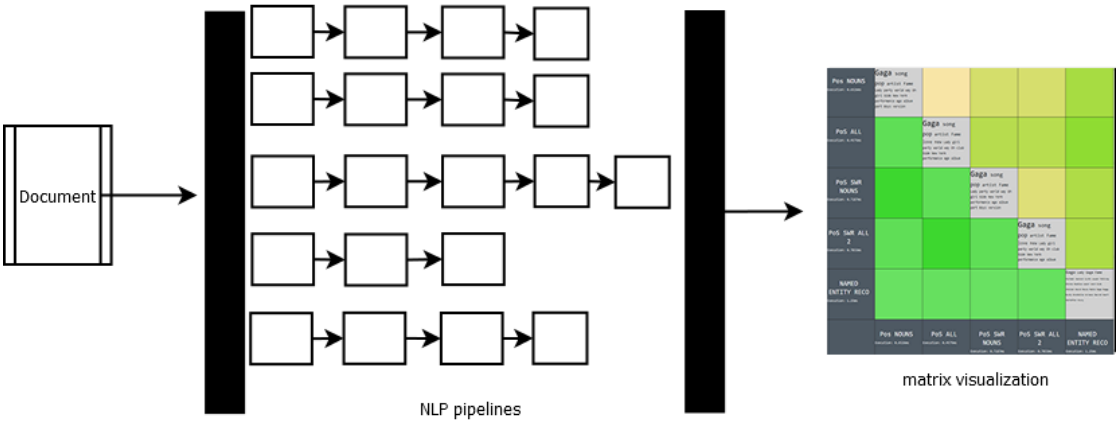

Figure 1.2: An example how the proposed system should work

Due to the various available steps, which have individual parameters that can be configured, there is no best-practice how a perfectly suiting NLP pipeline should be designed. It is open to the developer which steps to choose and combine. Each step has its advantages and disadvantages when performed on a given document. Also speed is always a factor that has to be considered. To give more insight into the differences of various NLP pipelines, a system is needed for having an overview over the output features and the resulting performance. Ideally, such a system gives a good foundation for deciding which pipeline to choose.

When comparing two pipelines with different configurations, it is possible that the observer will notice that the output can be quite similar. Due to the configurability of each single pipeline step and the individual parameters for each pipeline itself, the available variations of different pipelines can be enormous. Since it is highly individual and context-dependent which key words are descriptive for a given document, the best pipeline configuration cannot be determined automatically by a system. It rather requires the user's judgment to rate the results. Only the similarity between the output features and the performance differences can be computed automatically. What the system can offer is the recognition of expendable pipeline steps. The quality of the extracted features can not be optimized computationally, therefore the human has to remain in the process of building a pipeline for a given set of documents.

Therefore, when comparing high dimensional data, a visual presentation for examining the returned results is needed. Such a visualization of the pipelines gives direct feedback how each single pipeline performs and how the different parameters will affect their output. The interface for comparison should be easy to use and focus on state-of-the-art standards in creating web-based visualizations. Not only refers the term easy to use to the comprehension of the visualizations, it also considers the usability of creating and configuring pipelines by the user himself. The combination of usability, NLP and visualization techniques gives the user full control over comparing pipelines.

For example, with such a system the user is able to see if a single pipeline causes massive performance losses. The user is then able to decide if it is a better option to remove the step from the pipeline or to tweak its parameters. Additionally, the user should also be able to examine single steps of the pipeline and also be able to compare not only similarity, performance and keyword output – the user should also be able to get overall information about the extracted keywords shown in diagrams and other useful representations. The intended application of the tool is to analyze the same input text with different pipeline configurations and compare all variations against each other. The system is built as a web application where pipelines can be created and saved. Each pipeline can consist of various processing steps and has also parameters that can be adjusted. Documents can be uploaded as raw text. The user can choose from his previously created pipelines and can perform all pipelines simultaneously on a picked document. The resulting features are visualized as word clouds. To compare the results across pipelines, a matrix provides a pairwise comparison of feature similarities between pipeline outputs and performance differences.

Contribution

The main contribution of this work is an interface to test NLP pipelines of different documents and by providing a comparative visualization of the performance and the results. This work presents one of the first interfaces for comparing output of individually configurable NLP pipelines. Furthermore, due to the implementation as a web-application, it is possible that the application can function as a publicly accessible website. There it could function as a centralized location for observing NLP pipelines. To confirm and validate the generated output by our application, the thesis uses an empirically determined ground-truth word cloud from a document used by Chuang et. al [CMH12]. With the ground-truth it was possible to prove that the implementation works correct and gives satisfying results.

CHAPTER 2

Related Work

The related work contains previous generated knowledge about state-of-the-art text visualization and NLP techniques for automatic keyword extraction. Both areas are needed for this thesis and need to work seamlessly together to return comparative results. However, a direct comparison of different pipelines and showing their output in a visualization has not been done yet. One option is to manually compare the extracted keywords of each pipeline. Another option is to evaluate the available toolkits for NLP themselves, examine their output and compare them to each other. The importance of this work is however to visualize the output of the pipeline configurations of a single toolkit. The comparison itself is still done by a human, the interface facilitates the decision through visual comparison of the results.

When processing documents with NLP, the developer can choose from various toolkits available for doing this task. This work uses the Natural Language Tool Kit for Python [Pro18], which offers the required algorithms for various NLP tasks.

However, there are also other libraries to perform Natural Language Processing. The usually mentioned libraries NLP tasks are Apache OpenNLP [Fou11] for Java or the Stanford CoreNLP[MSB+14] which is available for C++ and C-# projects. In general, all libraries for Natural Language Processing offer a huge coverage of NLP functions, however in detail there are some differences that have to be considered. In NLTK, the developer always has to define own grammar when processing text. Stanford CoreNLP for example offers a probabilistic parser. Probabilistic parsing is using algorithms to compute the most probable parses of a given sentence, given a statistical model of the syntactic structure of a language. Additionally it offers out-of-the-box Named Entity Recognition. Standford CoreNLP can also be used in combination with Python. Additionally, there are also specialized NLP toolkits such as TwitterNLP [Bha18], which is available for the programming languages Python and Java. TwitterNLP offers several additional functions to process social-media data over standard NLP toolkits like dealing with emoticons in text. In this work we concentrate on formal text, which obeys grammar rules of English

language and only uses words available in a common dictionary. When performing NLP on other data sets, such as data sets from social media, other ways have to be found to deal with such data. Social media posts do not obey certain grammar rules and also use slang words not available in every dictionary, furthermore texts from platforms like Twitter and Facebook have also be re-formatted and sanitized because of Emojis and ASCII signs. Different NLP toolkits have been compared by Pinto et al. [PGOOA16]. When studying their results, NLTK is a good choice and has good performance for the tasks needed in this work compared to the other tested toolkits. Their result is based on precision, recall and F-measure [MKS+99]. However, in the mentioned paper only certain individual steps are compared and no visual output is given.

An important related work to this thesis is the information provided by Chuang et al. [CMH12]. It is important because it compares empirically determined features with automatically determined features. Results are represented as word clouds. We use the same representation for our features in this work. Chuang et al. focus on how the best found descriptive terms of a document can be presented. However, their paper does not focus on comparing NLP pipelines directly, instead they take a look on visualization. This bachelor thesis takes up on this part and adds comparative visualization of the different possible ways to configure certain pipelines.

Apart from NLP, this work relates also to works published about Text Visualization. There are numerous different ways of visualizing texts and document sets. One of the first examples that can be considered as word clouds may have been created by Milgram et al. in 1976 [Mil76]. People were asked by the author to name landmarks in Paris. A collective map was created based on the mentioned landmarks. By using font size it was visualized how often each place was mentioned. In the proposed interface the visual approach of emphasizing the pipeline output is done by a word cloud and different font-sizing. The output of each pipeline is an amount of n words ordered by their weight in the examined document. The most significant word gets the largest font size. Other forms of text visualization are not used in this work.

2.1 Text Visualization

As like in other fields of visualization, Text Visualization makes use of the same visual variables such as size, colour and alignment. It is possible to make use of the whole range of visualizations, such as using trees, matrices or different types of maps. The background of all mentioned variations of visualization is natural language processing.

When extracting information from document sets with NLP, not all available forms of visualization make sense. First of all, we have to consider the input and also the output of our NLP process is. In this work only one document is processed by each NLP pipeline. However, if more documents are processed, it is for example possible to categorize them into different topics. At this point interactive treemaps or heatmaps can be used. For visualizing topics, mostly graph based algorithms and approaches are used like in the work Topic Similarity Networks proposed by Maiya et al. [MR14]. Another way is shown

by LDA Explore proposed by Ganesan et al. [GBPC15], which shows the affiliation of documents to certain topics using parallel coordinates and implemented brushing. Another approach different to the other ones mentioned before is shown by TopicLens introduced by Devendorf et al. [DOH12], where also 3D plays a role in visualizing the certain topics of documents.

CHAPTER 3

Natural Language Processing

The common used processing steps in Natural Language Processing are tokenization, stopword removal, lemmatization, stemming, part of speech selection/tagging and named entity recognition. In the following paragraphs we want to discuss the mentioned steps.

NLP can be used in various fields, such as starting with extracting keywords from texts or trying to structure a document set in its matching topics. Furthermore it can also be used to create language, by first understanding a human written input and answering to it. Nowadays machine learning has had a big impact on NLP and has improved the previous process, which was mostly defined by certain hard coded rules like grammar or stemming patterns. However, not always the developer has to consider a high computational burden when thinking about extracting text from documents. Even when the given task is creating a simple tag cloud, NLP steps have to be performed. In this work NLP is the backbone for the visual comparative interface. It makes use of the most relevant features of NLP, such as: tokenization, stopword removal, lemmatization, stemming, part of speech selection/tagging and named entity recognition. In this section we want to take a closer look onto these steps.

3.1 Tokenization

For processing a document it is crucial to split it into smaller parts. Tokenization splits a given text into single fragments. There are two common approach when splitting text in NLP. Approaches to tokenization are for example the n-Gram model or the bag of words model, which also can be combined. When using the n-Gram model the text can be split into unigrams, bigrams, trigrams and higher order “n”-grams. The n-Gram model [JM14] can be defined as the following definition where $\sum$ is an finite alphabet and n is a positive integer value. An n-Gram can be defined as a word w with the length n over the alphabet $\sum$.

$$w = (w_1, \cdots, w_n) \in \sum \tag{3.1}$$

The Bag-of-Words approach splits the text into its single occurring words according to their frequency. The tokenization process outputs tokenized sentences in form of lists. Each list item represents on single token. The splitting is performed on characters like "," or ".". Tokenization is usually the first step performed in a pipeline. However, before performing tokenization a few preparations have to be done, which are not considered as pipeline steps. For exact tokenization, the input has to be sanitized from for example from mark-up languages such as HTML,XML or BBCodes.

3.2 Stop word removal

Usually written text uses a lot of short and often reoccurring words like prepositions, articles and conjunctions. They are not actually contributing to the document's topic and are called stop words. In order to get a text free from stop words we remove them. Not only that the stop words do not have any contribution to the semantics of a text, they also would generate less expressive output when generating visualizations. The stop words that have to be removed can either be obtained from a dictionary fitting the document's language or can be assigned individually. The stop word removal process has to be executed after tokenization and receives a tokenized document vector.

3.3 Lemmatization

Lemmatization plays an important part in natural language processing. In most languages a lot of word forms have different occurrences when used in different tenses or with different personal pronouns. The verb 'to drive' has regarding its context other forms, such as 'driving', 'driven or 'drove'. All this words have the same root 'drive', also called 'Lemma'. A Lemmatizer is capable of restoring the root form of verbs, adjectives or nouns and can be developed in a simple approach by just looking up a dictionary. However complex languages such as German or Arabic need more sophisticated algorithms for increasing performance and accuracy. A Lemmatizer specialized on German has been proposed by Lezius et al. [LRW98]

3.4 Stemming

In contrast to lemmatization, stemming is a more naive approach. A stemmer tries to reduce input words to their root forms by removing the ends of words. When taking a look at the example mentioned in the Lemmatization section, Stemming will reduce all forms like 'driving', 'drive' or 'driven' to 'driv'. We can see that here some problems occur. For example the past tense of 'drive' is 'drove' and is not similar to 'drive'. This means that 'drove' and 'drive' will not be mapped to the same stem. When processing English texts,

the frequently used Stemming algorithms are the Porter-Stemmer, Snowball-Stemmer and Lancaster-Stemmer. A more detailed overview about different stemming algorithms has been evaluated by Jivani et al. [J+11]

3.5 Part-of-Speech Tagging and Selection

Furthermore, Part-of-Speech selection is also an important task. Texts consist of different word forms like verbs, nouns or adjectives in various appearances. Part-of-speech selection describes the task of picking single words forms, such as nouns for example. An required task is part-of-speech tagging, which is also an important preprocessing step for lemmatization or named entity recognition. Whereas part-of-speech tagging is the process of assigning distinct grammatical categories to words in context. The order of words and used tense tell us a lot about how we understand a certain text. For example, tense syntactically changes a whole document and appearances of words. Tagging the corresponding words gives us more grip on the semantic meaning of a sentence. The result is a tokenized document vector containing tokens that are type of one of the selected word form.

There are many proposed algorithms to perform the task of part-of-speech tagging. The algorithms can be rule based, but also can be performed by statistical approaches. A rule based tagger can be observed in this early paper [Bri92] by Brill from 1992. It proposes a simple rule based tagger which estimates the most likely word from that has to be assigned to the current observed word. The algorithm is trained on the Brown Corpus [FK64], which is also used by the NLTK toolkit. The Brown Corpus contains about 1.1 million words from a variety of genres of written English.

3.6 Named Entity Recognition

Named Entity Recognition, shortly called NER, is a task that tries to classify named entities in text documents. Named entities can be person names, companies, countries, times expressions or even historical events. Named entities can be found as combined sets of words occurring in a dictionary or words that do not occur in a dictionary, such as last names. Named Entity Recognition is an important task due to the reason that named entities describe a certain document very well. In combination with words that have a high frequency, found named entities can define the overall topic of a document in a good way. Exact named entity recognition is complex task and has been pushed forward by statistical approaches such as machine learning.

3.7 Metrics

At this point we see that not all words are equal. Some words identify a document and its topic – if a certain word has high occurrence in a text, we assume that this word is important for the observed text. In natural language processing different metrics are used

such as term-frequency (tf) or term frequency–inverse document frequency (tf*idf). In this thesis only tf is used due to the reason only a single document is processed, whereas $tf * idf$ is used for document corpora with more than one document.

$$tf(t, D) = \frac{(t, D)}{\max'_t \in D(t', D)} \tag{3.2}$$

3.8 Combining Steps

The previous mentioned steps in this chapter can be combined with each other in a NLP pipeline. However, not all of the combinations are meaningful and contribute to the output. Some also do not work at all. Tokenization is usually the first step that is performed in a NLP pipeline before performing any other step. Stop word removal is not a crucial step, but improves accuracy to the end result. Not only that stop words do not contribute to the semantics, they also have a high rate in reoccuring in a document. Therefore stop words would obtain a high weight without having any actual contribution. Not all steps are capable of being carried out after eachother. For example stemming and lemmatization are such steps that can not be performed together. The user has to decide between the harsh stemming process or the lemmatization process. Lemmatization for example relies on part of speech selection. Without performing part of speech selection the lemmatizer would not be able to reduce the observed token to its actual root form. Lemmatization after the named entity recognition would not have any contribution. How text is processed by the most common steps is shown in the Introduction section. Most of the configurations that can be created are valid and will work, however not all are meaningful and return representative output.

CHAPTER 4

NLP Pipeline Comparison Interface

The user is able to choose a single document which can be processed by individually configured natural language processing pipelines. The pipelines are created by selecting subsequent operations and their parameters. To provide visualizations regarding the pipelines, the user can run all of the created pipelines at once on the input document. The system creates a comparative visualization which gives the user the ability to analyze and compare the pipeline outputs. We will now further discuss the comparison interface with detailed screenshots in the next paragraphs.

4.1 System Description

The interface used for comparing NLP pipelines in this project is designed as a web interface. At first, the application presents an overview of pipelines and documents which have been already added to the application. This is shown in Figure 4.1. Each pipeline and each document is shown as an individual card which can be clicked and opened as shown in Figure 4.1. After opening, the user is presented with all steps that have been added to the pipeline as seen in Figure 4.2

If a user clicks on a document, the application presents the user the text of a chosen document. Adding properties to the system is done by navigating to the left menu. Here two options can be found: one for adding a new pipeline and one for adding a new document. When adding a new pipeline, a new windows is opened where all information related to a pipeline can be chosen with selection fields. The user is able to select different pipeline steps and various parameters regarding the current pipeline. Additionally, the user is able to name each pipeline to give them describing identifiers for later visualization. Each pipeline can consist out of six different pipeline steps. Due to the reason that

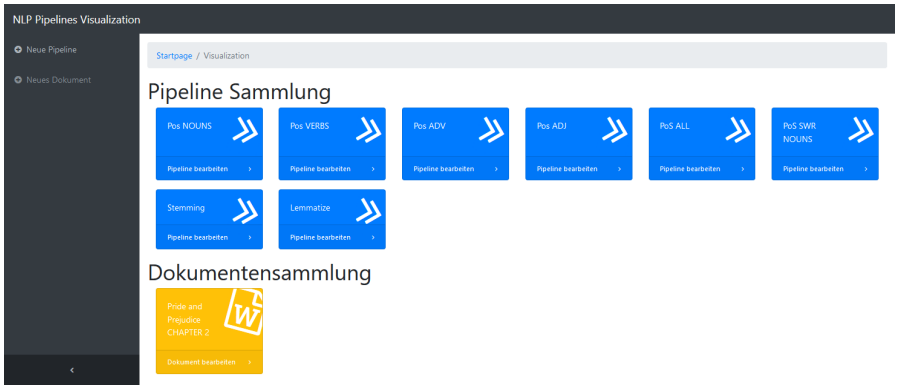

Figure 4.1: List of added pipelines and documents.

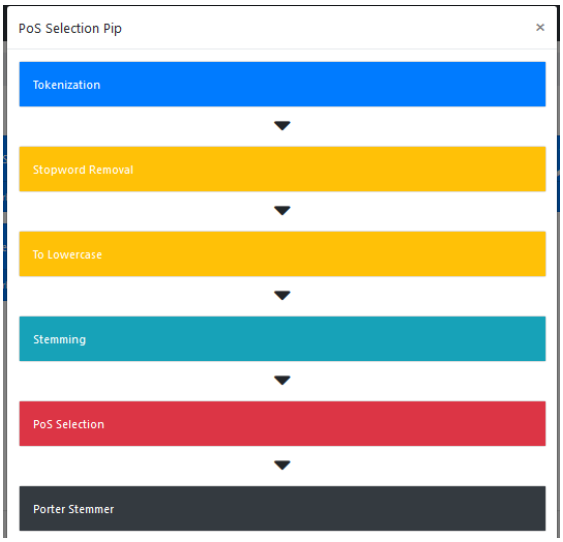

Figure 4.2: Viewing an added pipeline.

not all steps can be performed after another, the application automatically checks if a certain step is useful in the current context. If the step does not have any application as a following step, the application prevents the user from selecting it. The rules are based on the information observed in the previous chapter. The user has to decide between processes like stemming and lemmatization and is not able to perform steps that already has been chosen. Therefore the user is not able to perform steps like stemming, lemmatization or stop word removal after named entity recognition. One step is already selected at the beginning of creating a new pipeline. This is the tokenization step - it has been defined as a fixed step due to the reason that tokenization is needed for every following process in a pipeline. Without tokenization further selected steps would not work efficiently. All other steps can be chosen freely by the user with respect to the automatic check by the application itself. Notably, not all six steps have to be chosen.

If the user wants to test a pipeline with only one or two steps, the user is able to test pipelines in this minimal constellation. After selecting the several steps of the pipeline, the user is able to choose between stemmers for the stemming process. There are two stemmers of NLTK - the porter stemmer and the snowball stemmer. The last available option of adding a pipeline is selecting the word types which should be extracted by PoS-Selection, this can be nouns, adjectives, adverbs or verbs.

Pipeline steps the user can choose from:

- Tokenization
- Stopword removal
- Lemmatization
- Stemming
- PoS Selection
- Named Entity Recognition

Parameters the user can alter:

- Selected words by PoS-Selection
  - nouns
  - verbs
  - adjectives
  - adverbs
- Stemmers
  - Porterstemmer
  - Snowballstemmer

For adding a document, the user just has to provide a raw unformatted text. However, text is sanitized by internal functions. A document is defined as text part and a title, which should describe the document's actual text or topic. It will appear in the list of pipelines and documents. Further for processing added pipelines and documents, the user is able to choose the created pipelines per multiple selection. All selected pipelines will be executed one after another with the same document. The document is also chosen via a selection field. For having exact amounts of keywords that will be sent back by each individual pipeline, the user has to insert a certain number of keywords. The user is able to choose a value that fits best. However, the number of keywords should be selected wisely. A too large amount of keywords affects on the one hand the readability of the generated visualization.

After the user has started the process, a visualization is generated in form of a $n * n$ matrix, whereas n is the number of pipelines that have been previously selected. A first

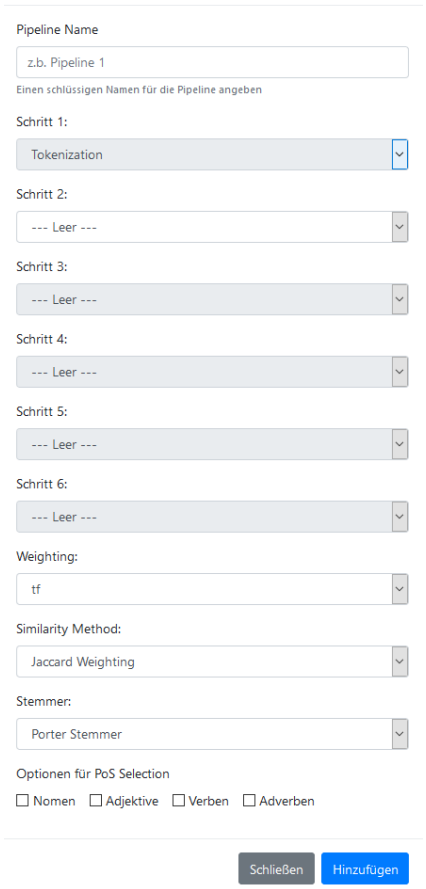

Figure 4.3: Adding a pipeline

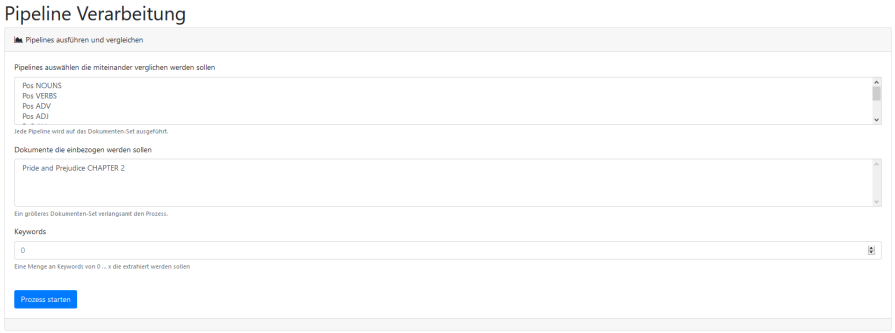

Figure 4.4: Choosing pipelines and keywords

Figure 4.5: First version of the comparative matrix

version of the implemented matrix can be seen in figure 4.5. All diagonal cells of the matrix contain a word cloud with the returned words of each pipeline. The single words of the word clouds are ordered by their frequency occurring in the submitted document. Words with high frequency are displayed with larger font size, words with lower frequency are displayed with smaller font size. A minimal font size of 7pt has been chosen for optimal legibility. With displaying a word cloud for each pipeline the user is able to see the most descriptive words processed by the according pipeline. Due to the arrangement in a matrix, the viewer is able to compare the output of the different pipelines. To overcome the problem of splitting words randomly, the interface splits words according to their correct hyphenation with '-' as a word boundary character.

The displayed matrix is split into two halves according to its diagonal line. The left side shows a comparison by similarity, the right half shows a comparison regarding the speed of each pipeline. The similarity visualization is a comparison measured on how similar the output of two pipelines is. How similarity and speed is exactly calculated is shown in the Chapter 5. To the left of the matrix is a row containing the names of each pipeline

that has been processed, also on the bottom there can also be a row found with the named of each single pipeline. The user starts reading a comparison by starting at the left side, this pipeline is mentioned as pipeline A, the pipeline on the bottom is referred as pipeline B. When comparing two pipelines, the user navigates to a certain cell which either is on the right or left side of the diagonal line. Both areas encode the differences through color. Furthermore, they also contain textual information about each pipeline pair, such as the computed similarity or the performance difference as shown in Figure 4.9. The performance difference comparison uses an interpolation between green and red, whereas green means pipeline A does not differ in terms of performance compared to pipeline B. A colouring moving towards red means slower performance compared to pipeline B. To see the actual performance value. Each cell on the right side shows the execution times of the two pipelines belonging to a single cell. The similarity comparison uses a different color scheme to give visual feedback. It uses white to show that no similarity can be found between two pipelines, whereas a tone turning green means more similarity between two pipelines can be found. In addition, the similarity is shown with three decimals as textual feedback.

One major disadvantage of the shown matrix in Figure 4.5 is the usage of fixed sized div containers. The matrix can break apart when changing the screensize to a smaller resolution or when adding too many pipelines to the matrix. To overcome this problem, a second matrix is presented.

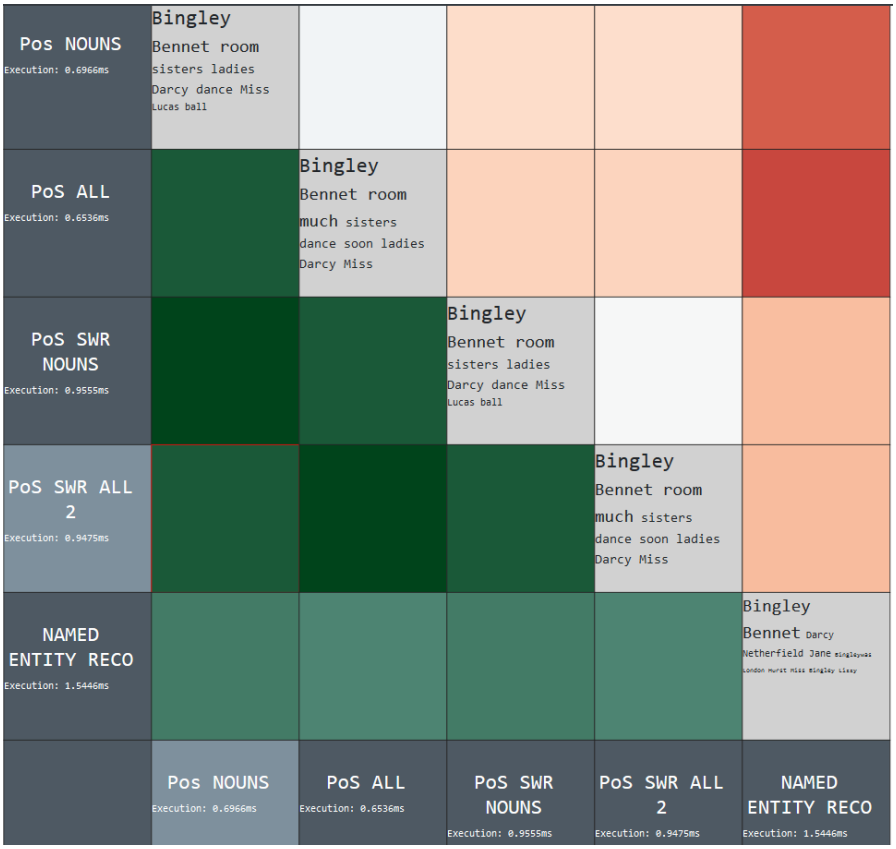

Figure 4.6: Reworked version of the comparative matrix.

The new matrix as shown in Figure 4.6 has now different color coding according to the color scheme shown in Figure 4.7. The gathering of information by reading the matrix has also changed in the second approach. As the reader can observe, the diagonal column has switched its position. The reading direction for performance and similarity always starts at the left column. The performance is now colored in red and blue with a diverging color scheme. A color moving towards red means that pipeline A on the left side has better performance, a color moving towards blue shows that pipeline B in the bottom has better performance. To gain better readability, the cells on the bottom and left side of the matrix are highlighted. The similarity is coded into green values, whereas a white field means there is no similarity. A very green colored cells shows high similarity between both evaluated pipelines.

To give insight into the execution time and calculated features, the left column has been added with the individual execution time (Figure: 4.8) and a tooltip (Figure 4.9) which shows the exact calculated features.

Figure 4.7: Colorschema of the improved matrix.

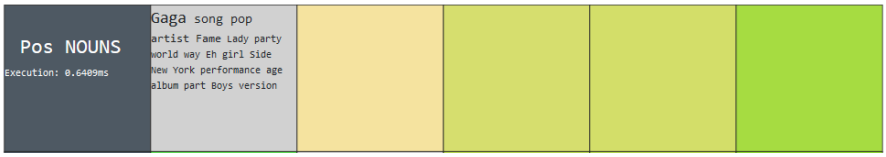

Figure 4.8: Exact execution time of the pipeline.

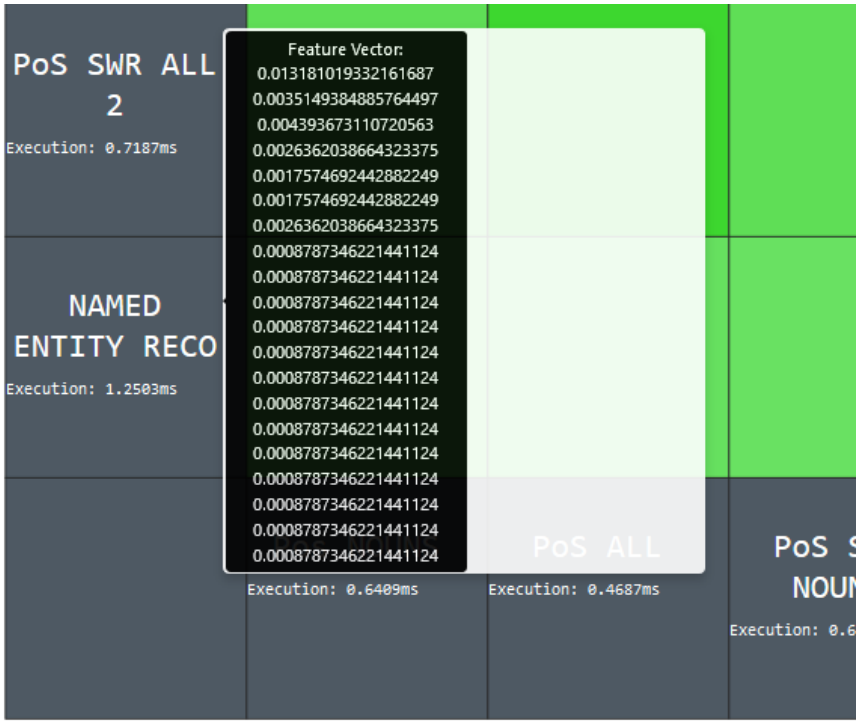

Figure 4.9: Featurevector tooltip.

4.2 Similarity and Performance

There are two common metrics for calculating the similarity between two documents: The Jaccard Index and Cosine Similarity. The implementation in this thesis uses the Cosine Similarity with x and y defined in the following formulas. x and y describe the calculated values of features in form of a vector. Each pipeline returns a feature vector. When comparing two individual pipelines, both return feature vectors which can be compared used the cosine similarity.

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||} \tag{4.1}$$

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2} \cdot \sqrt{\sum_{i=1}^{n} (y_i)^2}} \tag{4.2}$$

The calculation of the feature vectors is done by the tf metric which is calculated by the following formula. t describes a single feature found in the document D.

$$tf(t, D) = \frac{(t, D)}{\max_t' \in D(t', D)} \tag{4.3}$$

CHAPTER 5

Implementation

The proposed system was realized as a client-server setup. The following technologies have been used for implementation. The reason for developing the visual comparison interface as a client-server application is that tasks that have larger computational costs can be outsourced to a high-performance server and data can be accessed by more than only one user. It is possible designing the application as a website which also is capable of natural language processing tasks, however Python and its NLTK offers excellent performance.

Used Technologies and Frameworks

- Frontend Technologies
  - Language: JavaScript
  - Bootstrap V3.0
  - JQuery
  - d3.js
- Backend Technologies
  - Language: Python
  - Flask Webserver
  - Natural Language Processing Toolkit NLTK
  - SQLite Database

5.1 Frontend

To achieve high usability and developing an interface that is compatible with all recent browsers, Bootstrap [Boo18] in its fourth version in combination with JQuery [Fou18]

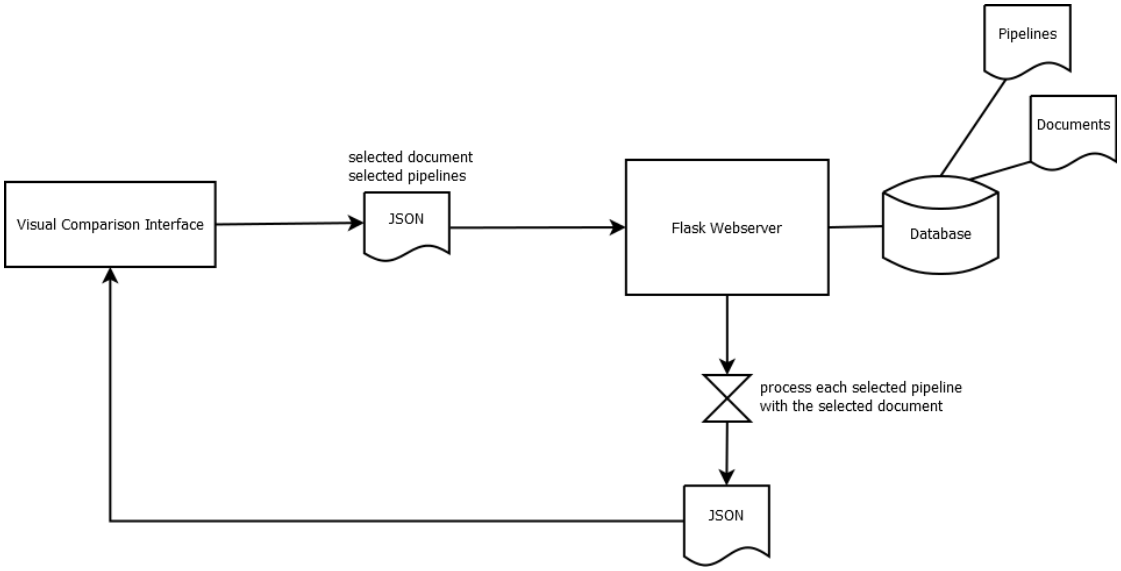

Figure 5.1: Implementation architecture overview.

has been chosen. Twitter Bootstrap is one of the widely accepted frameworks for user interfaces in web development. It utilizes responsive web design and follows accepted design principles. Bootstrap's grid layout is suitable for various applications, however it was only chosen for the general user interface. The visualization itself is generated by plain HTML div-containers which do not belong to any bootstrap classes. In combination with JQuery, the interface provides fast feedback and is able to send requests to the tied backend immediately. The template chosen for the application is the bootstrap template SB Admin 2 offered by startbootstrap.com [Sta18]. It is an open source template and MIT licensed.

The application is designed as a REST system and structures itself into three parts: the frontend, the backend and the database. A schematic overview of the system can be seen in Figure 5.1 The frontend ties itself to the backend by sending JSON requests to the Flask Server. There are three different requests that can be sent to the webserver, such as one of adding a new pipeline, one for adding a new document and one for starting the process. Starting a process also needs a JSON response from the server. This response is processed by the frontend's JavaScript code and immediately displayed in the frontend.

The JavaScript code packs all relevant information from the frontend's form fields into a well-formatted JSON string. This string is afterwards sent to the Flask Webserver via a POST request. The webserver receives the JSON string and formats it into a by Python readable notation for JSON, also called JSON-Object. The same process happens when the user sends a new document to the system, only that the POST request is differently formatted and contains less information. Each entity is depicted as a class. There is a certain class for a pipeline and a certain class for a document, containing variables matching the JSON format and the database format. The pipeline class contains the

name and also the steps encoded as integers. When the system receives a request, the JSON-Object is converted into an object corresponding to its suitable class. Afterwards the entity is stored into the SQLLite database [SQL18]. Whenever the user opens the start page of the application, all entities are fetched from the database and displayed on the landing page. There is no specific managment of users included into the application, all added pipelines can be used by any person who has access to the application. When starting a process, also a REST request is performed, containing the chosen pipelines, documents and the desired amount of keywords. Before processing, each pipeline is fetched from the database and stored into a class object of the type Pipeline. Also the document is fetched from the database and moved into a document class type object. Now, every pipeline is gone through step by step, where one step of the pipeline corresponds to an integer type. Each integer type stands for a fixed NLTK procedure solved as a method in the core system of the backend application. The current procedures and their parameters that can be used are the following which have been described in more detail in chapter **??**

The following listings show the data in its JSON format in which it is transmitted from and to the server. In Listing 5.1 we can see the JSON string which is sent to the server when starting a process. It contains all IDs of the documents and pipelines. However, we currently only process one document. In Listing 5.2 we see a by the client received JSON string format. It contains the name, the extracted keywords, the perfomance and also the featurelist. The featurelist is a float array containing all weights of the found keywords.

Listing 5.1: JSON string sent to Flask

```
"sending": {
        "documents": [documentlist] | integer,
        "keywords": number of keywords | integer,
        "pipelines": [pipelinelist] | integer
}
```

Listing 5.2: JSON string received from Flask

```
"nodes": {
 "node1": {
        "name": "Example Name",
        "keywords": [keywordlist] | string,
        "performance": | float,
        "feature": [featurelist] | float
 }
  "node2": {
        "name": "Example Name",
        "keywords": [keywordlist] | string,
        "performance": | float,
        "feature": [featurelist] | float
 }
  "node3": {
        "name": "Example Name",
        "keywords": [keywordlist] | string,
        "performance": | float,
        "feature": [featurelist] | float
 }
}
```

5.2 Visualization

The matrix visualization is generated by JavaScript. After the Flask Server has processed the pipelines and has sent back the JSON result string, the frontend brings the result into readable format by JQuery. The first approach by rendering the matrix as SVG has been abandoned at the beginning of the thesis due to the reason that it was not possible to handle our generated tag clouds and suitable word wraps. At this point it was necessary to balance the reasons of choosing an own generated matrix over a matrix generated by D3.js. Therefore the matrix has been rendered with DIVs instead, because it offered fast and simple rendering. The SVG approach offers smoother rendering and invariant displaying regarding the display size or the current window size. The approach using DIVs however is not screen size invariant and too many pipelines can disrupt the layout of the matrix. As soon it was possible to find a solution for wordwrapping in SVGs, the matrix was implemented with d3.js.

In a first approach, the matrix has been designed with regular DIV containers. After testing the usability of the generated matrix, which gave not satisfying results, the implementation was as scalable SVG reworked using D3.js. Due to the reason that the diagonal cells are filled with word clouds and each side of the diagonal has contains

different information, the matrix can't be filled in one iterative loop. The matrix is gradually filled with information, however the matrix grid itself is generated in one loop. Each cell is given an HTML id attribute so it can be addressed directly with JQuery for further processing. At first only the diagonal elements are filled with information since one diagonal cell belongs to one executed pipeline. Each other cell belongs to one pair of pipelines and also has to be colored according to the colouring scheme. After generating the matrix and filling the diagonal elements, two loops follow which fill the remaining elements. At this point, the performance difference and also the similarity is calculated.

5.3 Constraints

CHAPTER 6

Use Cases

6.1 Testdocument 1

Testing the solution with real world input and comparing it to papers which also have used the same input is crucial to get an understanding how exact the implemented project works. The input that has been chosen for comparison is a biographic text of Lady Gaga [Sup10] which was also used by Jason Chuang et al. [CMH12] to render their word clouds. One approach is trying to build pipelines that provide similar results to the given word cloud in the mentioned paper. The words shown in the word cloud extracted from the biography are mostly nouns. Therefore, the first approach was to define a pipeline which only focuses on nouns. The pipelines we created consist out of the steps: Tokenization, stop word Removal, lowercasing, part of speech selection. The other defined pipelines were set to finding adverbs, adjectives and verbs. The output of each individual pipeline was extracted and saved. The reason for extracting each word type on its own was to find out which word types are recognized as important before creating a pipeline which extracts all word forms. The words we are taking into comparison are the results of the referenced paper and shown in Listing 6.1. The following words in the listings are ordered by their weighting. For better readability and comparability we chose to extract the words from the resulting matrix. The wordcloud used in Chuang et al. [CMH12] can be found in Figure 6.1

Listing 6.1: Chuang et. al - Lady Gaga Biography

```
Gaga pop song want Fame Lady artist album club Dirty love
disco birthday Boy Girls beautiful pants eh Rich get sing
art get Brown just birthday nothing sequin rock fame
TV dance-pop rock pants write everyone piano
really famous ham girl icon Lower
```

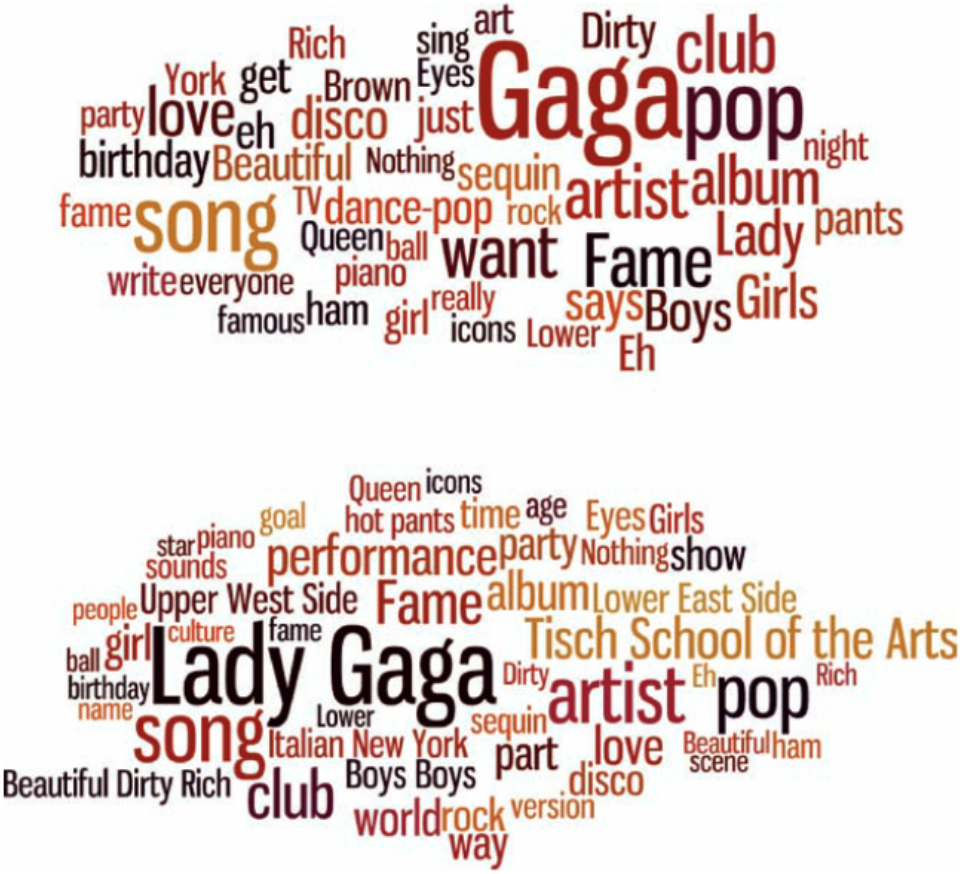

Figure 6.1: Wordcloud used in Chuang et al.

The detection of nouns has been very accurate. Almost all nouns that are present in the referenced text have been detected with our simple pipeline. Verbs have been also detected accurately except from some outliers. For example the found word *tisch* has been detected as a word. Actually it is a school name and the full word is "Tisch School of Arts at NYU" and is an named entity. The first detected verbs like *get*,*feel*,*love* and *dance* correspond with the found words in the referenced paper. Moving on to the adjectives we find a lot of outliers like *gaga*,*lady*,*queen*,*cyndi* or the word *daddy*. Lastly, the adverbs are free from outliers but do not correspond in a high grade to the referenced word cloud provided by the authors. The only adverbs which also can be found in the text are *just* and *really*. We have to consider that we now have performed four simple pipelines individually on the text and a lot of words are still missing. The overall coverage is however accurate. Some outliers can not be explained in this context, for example why so many proper nouns are considered as adverbs. When exploring the text one can find a lot sentences starting with *I* due to reason the text is in interview form. The *I* is considered as noun, verb and adverb. Our next step was to evaluate pipelines with more

parameters set. Our system allows PoS Selection on multiple word forms.

Listing 6.2: Pipeline 1 - PoS Selection NOUNS

```
Gaga song pop artist Fame Lady party world way Eh girl
Side New York performance age album part Boys version
Girls time show club birthday family today name
```

Listing 6.3: Pipeline 2 - PoS Selection VERBS

```
get write be i feel love michael dance greet strip strike do
seem tisch reach funnel trick make hate invite
```

Listing 6.4: Pipeline 3 - PoS Selection ADJECTIVES

```
new i little first such gaga more eh lady lower east rich hot
queen famous pop brown cyndi daddy
```

Listing 6.5: Pipeline 4 - PoS Selection ADVERBS

```
just always really not almost more then along innocently often
down together even very generously equally once
completely seriously also
```

Listing 6.6: Pipeline 5 - PoS Selection ALL

```
Gaga song pop artist Fame love new Lady girl party
world way Eh club Side New York performance age album
```

The next pipeline evaluated with our system considers nouns,verbs,adjectives and adverbs. The results can be found in the listing above. For example, the pipeline finds ‘gaga’ as a word with high frequency, whereas ‘lady’ has way lesser frequency. However, ‘lady’ has to occur as often as the word ‘gaga’. This matches also to the shown word cloud in the mentioned paper, where ‘Gaga’ has not the same frequency as ‘Lady’. The second listing below shows the results of performed named entity recognition. Here it was possible to find "Lady Gaga" as connected entity, also words as "New York" or for example "East Side".

Listing 6.7: Pipeline 5 - Named Entity Recognition

```
Gaga Lady Gaga Fame New York Lower East Side Beautiful
Dirty Rich Queen Michael Jackson Cyndi Lauper Rolling
Stones Beatles Upper West Side Italian David Bowie
Radio Gaga Peggy Bundy Donatella Versace
```

The now obtained features have good accuracy compared to the features listed in the referenced word cloud and proof the functionality of the proposed system. Significant issues in performance were not recognizable between the different pipeline structures. The resulting matrix with the most important pipelines can be viewed in Figure 6.2.

Figure 6.2: Matrix with Lady Gaga biography input

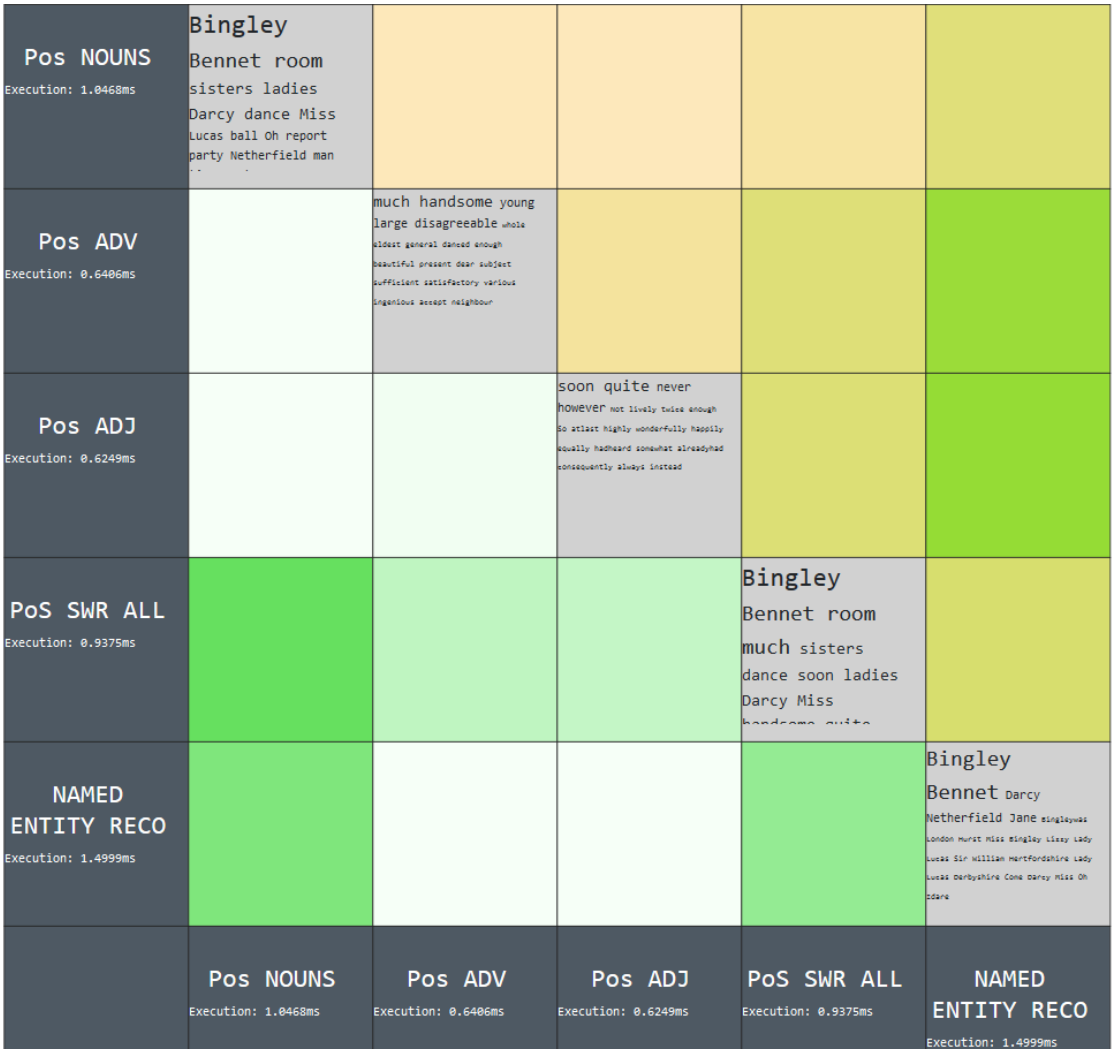

Figure 6.3: Resulting Matrix.

6.2 Testdocument 2

Another document that has been chosen is the third chapter of "Pride and Prejudice". Here we do not have any ground truth, the following tests are based on the results offered by the matrix visualization. The result is shown in Figure 6.3. A brief examination of the matrix shows that named entity recognition offers the same proper nouns than the other pipelines with higher execution time. Interestingly, the execution time of the second pipeline selecting all available word forms such as nouns, verbs, adjectives and adverbs took less time than only selecting nouns. The shown example shows the correct working calculation of similarity and performance difference.

CHAPTER 7

Future Work

The presented work offers capabilities to explore different variations of pipelines using the NLTK framework and generating a visual comparison. Insight is given into the performance of individual pipelines and their output vectors. However, the work can be improved by tasks such as different types of visualization, metrics or the ability to process more than only one document.

7.1 Visualization

The visualization can be designed more interactively with additional information about the extracted features. Currently it shows similarity, performance differences, the execution time and a tooltip with the weights of each single feature. The visualization could offer overall insight such as comparative barcharts of found features regarding all pipelines. A crucial point to mention is the tagcloud, which is not capable of displaying result sets with a large number of extracted features. In such a case, words can not be observed. One approach would be only showing the most relevant keywords and displaying others in a tooltip.

7.2 Implementation

The current application only saves pipelines and documents. It does not save any explored data about performance, similarity or generated keywords. At the current stage the user is able to reproduce his results, however it is not possible to save the results for later examination. Due to the scientific use of this application, it is crucial for being able to examine obtained results at any time. Saving explored data would also give the ability to generate statistics about the obtained results overall explored pipeline. For example a ranking of pipelines and their performances on different document sets. When talking about documents, we have to consider that the current implementation does not offer

the ability to process more than one document. Allowing more documents brings more parameters that can be set, such as a tf*idf metric. It also can give more functions to explore, such as topic modeling for example.

CHAPTER 8

Conclusion

The presented work is a useful tool for exploring the generated output of different NLP pipelines. Furthermore, it also gives the user the ability to gain an overview of different pipelines at the same time due to the implemented matrix view. The interface with its comparison interface shows performance differences and similarity values at one sight. During the evaluation, which was done by comparing the obtained results to a word cloud in a referenced paper, we found out that our implemented system is able to gain good accuracy with very basic NLTK methods. However, the system is not able to process document corpora and there does not support other metrics than tf. In conclusion, the work offers good insight into the functionality of various NLP steps and their resulting output.

List of Figures

Listings

Bibliography

[Bha18] Archna Bhatia. Twitternlp. http://www.cs.cmu.edu/~ark/TweetNLP/, 2018. [Online; accessed 26-August-2018].

[Boo18] Bootstrap. Bootstrap CSS. https://getbootstrap.com/, 2018. [Online; accessed 05-Mai-2018].

[Bri92] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics, 1992.

[CMH12] Jason Chuang, Christopher D Manning, and Jeffrey Heer. "without the clutter of unimportant words": Descriptive keyphrases for text visualization. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 19(3):19, 2012.

[DOH12] Laura Devendorf, John O'Donovan, and Tobias Höllerer. Topiclens: An interactive recommender system based on topical and social connections. In *First International Workshop on Recommendation Technologies for Lifestyle Change (LIFESTYLE 2012)*, page 41, 2012.

[FK64] W Nelson Francis and Henry Kucera. Brown corpus. *Department of Linguistics, Brown University, Providence, Rhode Island*, 1, 1964.

[Fou11] Apache Software Foundation. OpenNLP. http://opennlp.apache.org, 2011. [Online; accessed 19-Nov-2018].

[Fou18] The Jquery Foundation. Jquery Javascript Library. https://jquery.com/, 2018. [Online; accessed 07-Mai-2018].

[GBPC15] Ashwinkumar Ganesan, Kiante Brantley, Shimei Pan, and Jian Chen. Lda-explore: Visualizing topic models generated using latent dirichlet allocation. *arXiv preprint arXiv:1507.06593*, 2015.

[J+11] Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

[JM14] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

[LRW98] Wolfgang Lezius, Reinhard Rapp, and Manfred Wettler. A freely available morphological analyzer, disambiguator and context sensitive lemmatizer for german. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 743–748. Association for Computational Linguistics, 1998.

[Mil76] Jodelet Milgram. Psychological maps of paris. In *Environmental psychology*, pages 104–124, 1976.

[MKS+99] John Makhoul, Francis Kubala, Richard Schwartz, Ralph Weischedel, et al. Performance measures for information extraction. In *Proceedings of DARPA broadcast news workshop*, pages 249–252. Herndon, VA, 1999.

[MR14] Arun S Maiya and Robert M Rolfe. Topic similarity networks: visual analytics for large document sets. *arXiv preprint arXiv:1409.7591*, 2014.

[MSB+14] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[PGOOA16] Alexandre Pinto, Hugo Gonçalo Oliveira, and Ana Oliveira Alves. Comparing the performance of different nlp toolkits in formal and social media text. In *OASIcs-OpenAccess Series in Informatics*, volume 51. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[Pro18] NLTK Project. Natural language toolkit 3.4 for python. `https://www.nltk.org/`, 2018. [Online; accessed 25-Mai-2018].

[SQL18] SQLite. SQLite Database. `https://www.sqlite.org/index.html`, 2018. [Online; accessed 05-July-2018].

[Sta18] Startbootstrap. Startbootstrap Admin Template SB-Admin. `https://startbootstrap.com/template-overviews/sb-admin/`, 2018. [Online; accessed 10-Mai-2018].

[Sup10] Superadrianme. Lady Gaga Biography. `https://www.superadrianme.com/music/lady-gaga-bio/`, 2010. [Online; accessed 19-Nov-2018].

Appendix

Lady Gaga Biography

When Lady Gaga was a little girl, she would sing along on her mini plastic tape recorder to Michael Jackson and Cyndi Lauper hits and get twirled in the air in daddy's arms to the sounds of the Rolling Stones and the Beatles. The precocious child would dance around the table at fancy Upper West Side restaurants using breadsticks as a baton. And she would innocently greet a new babysitter in nothing but her birthday suit.

It's no wonder that little girl from a good Italian New York family turned into the multi-talented singer-songwriter with a flair for theatrics that she is today: Lady Gaga.

"I was always an entertainer. I was a ham as a little girl and I'm a ham today," says Lady Gaga, 22, who made a name for herself on the Lower East Side club scene with the infectious dance-pop party song "Beautiful Dirty Rich," and wild, theatrical, and often tongue-in-cheek "shock art" performances. Gaga – who designs and makes many of her stage outfits — would strip down to her hand-crafted hot pants and bikini top, light cans of hairspray on fire, and strike a pose as a disco ball lowered from the ceiling to the orchestral sounds of A Clockwork Orange.

"I always loved rock and pop and theater. When I discovered Queen and David Bowie is when it really cam together for me and I realized I could do all three," says Gaga, who nicked her name from Queen's song "Radio Gaga" and who cites rock star girlfriends Peggy Bundy and Donatella Versace as her fashion icons.

"I look at those artists as icons in art. It's not just about the music. It's about the performance, the attitude, the look; it's everything. And that is where I live as an artist, and that is what I want to accomplish." That goal might seem lofty, but consider the artist: Gaga is the girl who at age four learned piano by ear.

By age 13, she had written her first piano ballad. At 14, she played open mike nights at clubs such as New York's the Bitter End by night and was teased for her quirky, eccentric style by her Convent of the Sacred Heart School (the Manhattan private school Nicky and Paris Hilton attended) classmates by day.

At age 17, she was one of 20 kids in the world to get early admission to Tisch School of the Arts at NYU. Signed by her 20th birthday and writing songs for other artists (such as the Pussycat Dolls, and has been asked to write for a series of Interscope artists) before her debut album was even released, Lady Gaga has earned the right to reach for the sky.

"My goal as an artist is to funnel a pop record to the world in very interesting way," says Gaga, who wrote all of her lyrics, all of her melodies, and played most of the synth work on her album, "The Fame" (Streamline/Interscope/KonLive). "I almost want to trick people into hanging with something that is really cool with a pop song. It's almost like the spoonful of sugar, and I'm the medicine."

On "The Fame", it's as if Gaga took two parts dance-pop, one part electro-pop, and one part rock with a splash of disco and burlesque and generously poured it into the

figurative martini glasses of the world in an effort to get everyone drunk with her Fame. "'The Fame' is about how anyone can feel famous," she explains.

"Pop culture is art. It doesn't make you cool to hate pop culture, so I embraced, it and you hear it all over "The Fame". But it's a sharable fame. I want to invite you all to the party. I want people to feel a part of this lifestyle."

The CD's opener and first single, "Just Dance," gets the dance floor rocking with its "fun, L.A., celebratory vibe." As for the equally catchy "Boys Boys Boys," Gaga doesn't mind wearing her influences on her sleeve.

"I wanted to write the female version of Motley Crue's 'Girls Girls Girls,' but with my own twist. I wanted to write a pop song that rockers would like."

"Beautiful Dirty Rich" sums up her time of self-discovery, living in the Lower East Side and dabbling in drugs and the party scene. "That time, and that song, was just me trying to figure things out," says Gaga. "Once I grabbed the reigns of my artistry, I fell in love with that more than I did with the party life." On first listen, "Paparazzi" might come off as a love song to cameras and in all honesty, Gaga jokes "On one level it IS about wooing the paparazzi and wanting fame. But, it's not to be taken completely seriously. It's about everyone's obsession with that idea. But it's also about wanting a guy to love you and the struggle of whether you can have success or love or both."

Gaga shows her passion for love songs on such softer tracks as the Queen-influenced "Brown Eyes" and the sweet kiss-off break-up song "Nothing Else I can Say (Eh Eh)." "'Brown Eyes' is the most vulnerable song on the album," she explains. "'Eh Eh' is my simple pop song about finding someone new and breaking up with the old boyfriend."

With the new tour for this album, fans will be treated to a more polished version of what they saw (and loved) at her critically acclaimed Lollapalooza show in August 2007 and Winter Music Conference performance in March 2008. "This new show is the couture version of my handmade downtown performances, of the past few years. It's more fine-tuned, but some of my favorite elements to my past shows – the disco balls, hot pants, sequins, and stilettos – will still be there. Just more fierce, and more of a conceptual show, with a vision for pop performance art."

It's been a while since a new pop artist has made her way in the music industry the old-fashioned/grass roots way by paying her dues with seedy club gigs and self-promotion. This is one rising pop star who hasn't been plucked from a model casting call, born into a famous family, won a reality TV singing contest, or emerged from a teen cable TV sitcom. "I did this the way you are supposed to. I played every club in New York City and I bombed in every club and then killed it in every club and I found myself as an artist. I learned how to survive as an artist, get real, and how to fail and then figure out who I was as singer and performer. And I worked hard."

Gaga adds with a twinkle in her eye, "And now, I'm just trying to change the world, one sequin at a time."

Not all that Mrs. Bennet, however, with the assistance of her five daughters, could ask on the subject was sufficient to draw from her husband any satisfactory description of Mr. Bingley.

They attacked him in various ways; with barefaced questions, ingenious suppositions, and distant surmises; but he eluded the skill of them all; and they were at last obliged to accept the second-hand intelligence of their neighbour Lady Lucas. Her report was highly favourable. Sir William had been delighted with him. He was quite young, wonderfully handsome, extremely agreeable, and, to crown the whole, he meant to be at the next assembly with a large party. Nothing could be more delightful!

To be fond of dancing was a certain step towards falling in love; and very lively hopes of Mr. Bingley's heart were entertained.

"If I can but see one of my daughters happily settled at Netherfield," said Mrs. Bennet to her husband, "and all the others equally well married, I shall have nothing to wish for."

In a few days Mr. Bingley returned Mr. Bennet's visit, and sat about ten minutes with him in his library. He had entertained hopes of being admitted to a sight of the young ladies, of whose beauty he had heard much; but he saw only the father.

The ladies were somewhat more fortunate, for they had the advantage of ascertaining, from an upper window, that he wore a blue coat and rode a black horse.

An invitation to dinner was soon afterwards dispatched and already had Mrs. Bennet planned the courses that were to do credit to her housekeeping, when an answer arrived which deferred it all. Mr. Bingley was obliged to be in town the following day, and consequently unable to accept the honour of their invitation, c. Mrs. Bennet was quite disconcerted. She could not imagine what business he could have in town so soon after his arrival in Hertfordshire; and she began to fear that he might be always flying about from one place to another, and never settled at Netherfield as he ought to be. Lady Lucas quieted her fears a little by starting the idea of his being gone to London only to get a large party for the ball; and a report soon followed that Mr. Bingley was to bring twelve ladies and seven gentlemen with him to the assembly. The girls grieved over such a large number of ladies; but were comforted the day before the ball by hearing that, instead of twelve, he had brought only six with him from London, his five sisters and a cousin. And when the party entered the assembly room, it consisted of only five altogether; Mr. Bingley, his two sisters, the husband of the oldest, and another young man.