Visual Comparison of NLP Pipelines

Muhammad Samoul, 11833154
e11833154@student.tuwien.ac.at

April 22, 2019

1 Abstract

Natural Language Processing (NLP) is a sub-field of artificial intelligence (AI). It enables computers to understand, process and analyze large amounts of unstructured natural language data (raw text). Nowadays with the new techniques of machine learning, we got good performance and brings us closer to unfolding the semantic meaning of the text. However, it is far from perfect. Therefore, an alternative approach to helping humans understand a text corpus is to provide a visualization of the content. To generate such a visualization, several NLP steps are necessary to convert the raw text into features, such as weighted keywords or phrases, that can be visualized. The words to be visualized and their weights strongly depend on which NLP steps are performed, in which order, and with which parameters. However, there is currently no standard how to set up such an NLP pipeline and NLP pipeline configurations vary significantly across visualizations and input texts. Our project consists of visualizing high dimensional data with different pre-processing steps with a different order. To compare the results, we choose a well-known and wide-spread overview visualization technique: word clouds. Word clouds are composed of words used in a particular text or subject, in which the size of each word indicates its weight computed in the course of the NLP pipeline.

2 Introduction

We know that NLP prepossessing steps is very important in machine learning, but there is not standard for using them or the best order of the steps some are optional some are required and those pipeline differ from corpus to another so Visualization will help by turns cognitively demand tasks into perceptual tasks since humans are very effective in quickly grasping visualized data. so creating such powerful Visualization tool will keep the human in the loop.

The goal of this work is to visually judge the difference of the outcome if we vary the pipeline (Preprocessing steps) so that we can easily judge, for instance, how the weighting scheme really influences the output. To generate such graphs, we will use the classic bag-of-words approach as a features representation, which is not really state-of-the-art anymore, but very easy and fast to compute and still very wide-spread in the text visualization field. We used different weighting schemes Counting terms and TF*IDF.However, we need to do some preprocessing steps on the text corpus before we weight the features,

3 NLP Preprocessing Pipeline

3.1 Word Tokenization

One common task in NLP (Natural Language Processing) is tokenization. "Tokens" are usually individual words (at least in languages like English) and "tokenization" is taking a text or set of text and breaking it up into its individual words. perhaps at the same time throwing away certain characters, such as punctuation. These tokens are then used as the input for other types of analysis or tasks,

3.2 Stopword removal

A stop word is a commonly used words (such as "the", "a", "an", "in") this words have little value and don't really contribute to the document One of the major forms of Preprocessing is to filter out useless data and excluded them from the vocabulary entirely.

3.3 Capitalization

Case-folding is reducing all letters to lower case. Often this is a good idea: it will allow instances of "Technology" at the beginning of a sentence to match with a query of "technology". On the other hand, such case folding can equate words that might better be kept apart. Many proper nouns are derived from common nouns and so are distinguished only by case, including companies (General Motors, The Associated Press), and person names (Bush, Rose).

3.4 Lemmatization

In many languages, words appear in several inflected forms. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', that one might look up in a dictionary, is called the lemma for the word. The association of the base form with a part of speech is often called a lexeme of the word. this reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is $\Rightarrow$ be
car, cars, car's, cars' $\Rightarrow$ car

3.5 Stemming

Stemming is closely related to Lemmatisation. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words that have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster. The reduced "accuracy" may not matter for some applications.

4 Features Weighting Approaches

4.1 Counting terms

It's the simplest Weighting Approach, it's simply counting the number of times each term occurs in each document; the number of times a term occurs in a document is called its term frequency

4.2 Term frequency–Inverse Document Frequency (TF-IDF)

TF-IDF weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log(total number of documents / number of documents with term t in it)

5 Word-Cloud

Word cloud(tag cloud) is a novelty visual representation of text data, as an image or interactive image composed of terms used in a particular text or subject, each terms are usually single words, and the importance of each term is shown with font size and/or color. This format is useful for quickly perceiving the most prominent terms and for locating a term alphabetically to determine its relative prominence. Each term Weights is computed from the the different Weights Approaches above, then we will build a two different word cloud (one from the Term frequency and the other from the TF-IDF)

We found interesting thing after we build both clouds the term frequency and the tfidf at first for the TFIDF we were taking the average from the output matrix to got a value for each rank but we found the output cloud becoume similar to the term frequency cloud so we decided to take the maximum value for each term

6 Interface

We introduce a web application with easy and simple interface that make the user able to select the text corpus (text files) and select two different pipeline which differ in order and/or steps then the we compute the selected Weighting scheme then the output result will be visualized in another page as a wordcloud as we will see in the next sections

6.1 Setting

We Designed a simple user interface that allow the user to configure it in an easy way as showen in fifigure 1, first the user should choose a text corpus (one of the pre-uploaded) or upload new one, if the user decided to upload a new corpus, all the text corpus should be in one directory (the user can only choose to upload a directory not files).

Next the user can choose two different sets of the Preprocessing steps in any order to compare between them, the user add all the steps at once by selecting add all or remove any step by clicking the small x or remove all the steps by selecting the clear all button, removing the HTML Tags is selected by defaults as a first step

We give the user the ability to deiced the density of each word cloud he must choose the number of the terms to be shown in the cloud we choose the highest ranked terms after we process all the corpus

Also we give the user the ability to choose from the Weighting Approaches.

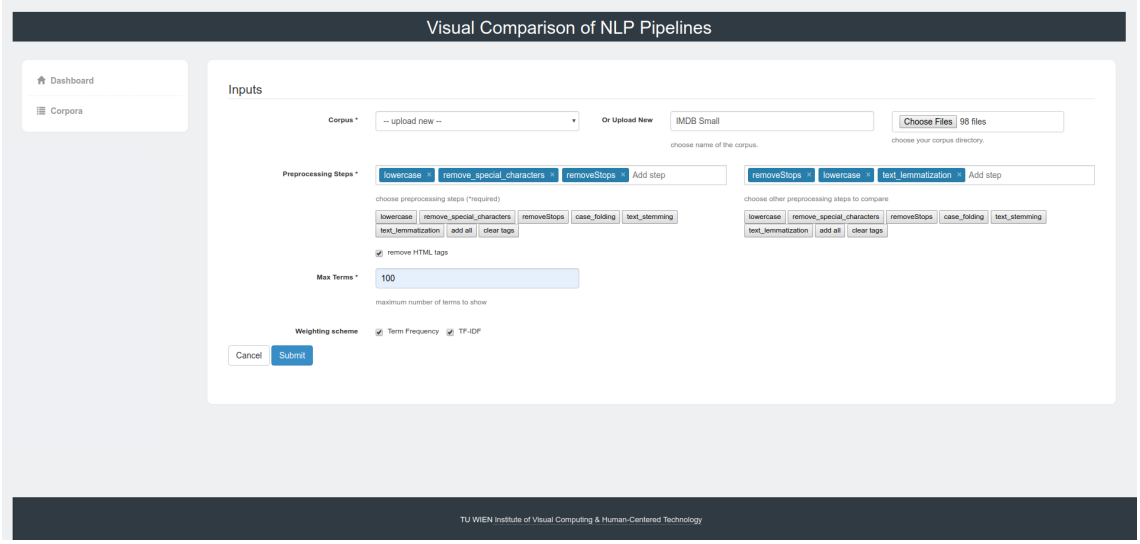

Figure 1: Setting page

6.2 Database

we save every uploaded corpus so it be easier if the user decided to use this corpus again the database page show a list of the saved corpus with the name and the corpus size, the user can delete it any time from the red delete button as shown in figure 2

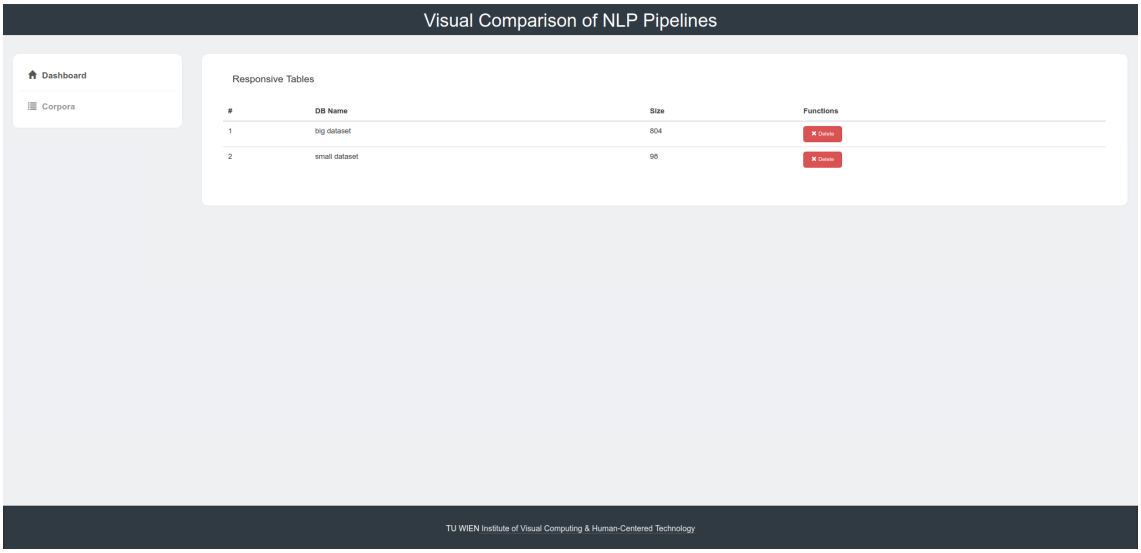

Figure 2: database page

6.3 Output and Visualization

Next we apply the chosen processing steps and the settings to the uploaded/chosen database after that we get the weights for each term and using those values we build the wordclouds, we use the color gradient and the word size to show the values of each term , the bigger the word the highest value for this term(word) also we added an red color when hover over the word to show or work as a link between the same word in the different graphs as shown in figure 3, in the last row of this page you can find info about the different processing steps and the time it takes for each one to apply the processing steps and to compute the wights.

Figure 3: output and visualization page

7 Implementation

The proposed system is a web-application system that can be installed as a client-server in any machine. we considered the web application because of the easy development and installation , also we can install it on a remote server and access it from different machines folloing is the technologies we used for the development:

For Backend we used :

- Python3 [5]
- Flask framework [2]
- Scikit-learn [6]
- Natural Language Processing Toolkit - NLTK [4]

For Frontend we used:

- Javascript
- d3.Js [1]
- Bootstrap framework
- Bootstrap admin template
- Jinja template engine [3]

7.1 Code Architecture

We tried to make the code as simple as we could to make adding features and adding any additional processing step easy for the developer following is the code architecture

```
main/
    __init__.py
    main.py
    pipeline.py

    utiles/
        helper.py
        preprocessing.py
        weighting.py

    templates/
        404.html
        505.html
```

```
        ..
        index.html

    static/
        css/
        ..
        js/

uploads/
    dataset1/
    ..
    dataset2/
```

The main files in our code are:

- main.py : where we defined our routes urls
- preprocessing.py: where we defined our preprocessing function like (removeStops, lowercase, lemmatization, stemming, etc ..) with help of nltk lib each function return a dict filename : [term1 ,term2 , ...] , ..
- weighting.py: here we defined our weighting function (t_frequency, tfidf) with help of sklearn.feature_extraction lib we used CountVectorizer function to compute the term_frequency and TfidfVectorizer to compute the tfidf, as we said before becouse the tfidf matrix which is the value or the weight for each term in each document and because the averages give the same distribution as the term frequency only that is why we choosed the maximum value for the term in all the documents

7.2 installation

To install the system we need a python environment with all the needed libraries installed, to do so:

Create virtual environment:
_$ virtualenv -p python3 venv/py36web

Activate the environment:
_$ source venv/py36web

Install the needed libs:
_$ cd PATH/TO/OUR/APP
_$ pip3 install -r requirements.txt

To run the app:
_$ export FLASK_APP=main
_$ export FLASK_ENV=development
_$ flask run

Note: we suggest to use Firefox for large corpus.

8 conclusion

As we explained above the proposed system is very easy to use and help to see what is inside the given corpus from the terms view with two different weighting scheme the term frequency and the term frequency–inverse document frequency the output wordcloud have a nice color gradient and size to describe the value of the term's wight and hover any term will show all the terms with the same word root to see the different wights in each wordcloud

9 Future work

we can add another weighting schemes like the inverse document frequency only to show the uniqueness of the terms it will be a good idea also to show the Chi-Square and compare it with the different weighting scheme also we can add more visualize techniques like Parallel Tag Clouds

References

[1] D3. *Data-Driven Documents*. URL: `https://d3js.org/`.

[2] Flask. *Micro web framework written in Python*. URL: `http://flask.pocoo.org/`.

[3] Jinja. *Full featured template engine for Python*. URL: `http://jinja.pocoo.org/`.

[4] NLTK. *Natural Language Toolkit*. URL: `https://www.nltk.org/`.

[5] Python. *An interpreted, high-level, general-purpose programming language*. URL: `https://www.python.org/`.

[6] scikit-learn. *Machine Learning in Python*. URL: `https://scikit-learn.org/`.