Voice and Text Summary Generation using ML and NLP

Mini Project Report

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Engineering (Computer Engineering)

by:

Aditya Singh	TU3F1819005
Prathamesh Chaskar	TU3F1819006
Shruti Rathod	TU3F1819029

Under the Guidance of

Dr. Mire Archana Vasant

Department of Computer Engineering

TERNA ENGINEERING COLLEGE

Nerul (W), Navi Mumbai 400706

(University of Mumbai)

(2020-21)

Internal Approval Sheet

TERNA ENGINEERING COLLEGE, NERUL

Department of Computer Engineering

Academic Year 2020-21

CERTIFICATE

This is to certify that the mini project entitled **"Voice and Text Summary Generation using ML and NLP"** is a bonafide work of

Aditya Singh	TU3F1819005
Prathamesh Chaskar	TU3F1819006
Shruti Rathod	TU3F1819029

submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the Bachelor of Engineering (Computer Engineering).

Guide **Head of Department** **Principal**

Approval Sheet

Project Report Approval

This Mini Project Report – entitled "**Voice and Text Summary Generation using ML and NLP**" by following students is approved for the degree of ***B.E. in "Computer Engineering"***.

Submitted by:

Aditya Singh	TU3F1819005
Prathamesh Chaskar	TU3F1819006
Shruti Rathod	TU3F1819029

Examiners Name & Signature:

1.----------

2.----------

Date: ----------

Place: ----------

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Aditya Singh	TU3F1819005	----------
Prathamesh Chaskar	TU3F1819006	----------
Shruti Rathod	TU3F1819029	----------

Date: _____

Place: _____

Acknowledgement

We would like to express our sincere gratitude towards our guide **Dr. Mire Archana Vasant** for their help, guidance and encouragement, she provided during the project development. This work would have not been possible without her valuable time, patience and motivation. We thank her for making our stint thoroughly pleasant and enriching. It was great learning and an honor being their student.

We are deeply thankful to **Dr. Mire Archana Vasant (H.O.D Computer Department)** and entire team in the Computer Department. They supported us with scientific guidance, advice and encouragement, they were always helpful and enthusiastic and this inspired us in our work.

We take the privilege to express our sincere thanks **to Dr. L. K. Ragha** our Principal for providing the encouragement and much support throughout our work.

Aditya Singh	TU3F1819005	----------
Prathamesh Chaskar	TU3F1819006	----------
Shruti Rathod	TU3F1819029	----------

Date: _____

Place: _____

Abstract

Text summarization is a subdomain of Natural Language Processing (NLP) that deals with extracting summaries from huge chunks of texts. It is a process of generating a concise and meaningful summary of text from multiple text resources such as books, news articles, blog posts, research papers and emails or more. Machine learning algorithms can be trained to comprehend documents and identify the sections that convey important facts and information before producing the required summarized texts. This project uses machine learning and NLP to generate condensed form of the text and also give you the entities that was in the provided input.

The number of sentences picked for compression may depend on the compression ration of the summary.

Summarization systems often need to have additional evidence that they can utilize in order to specify the most important topics of the documents. For example, a scientific paper or a resume will contain considerable amount of information which needs to be summarized and need to be noted. An aspect of machine learning known as Named Entity Recognition is used to fulfill this purpose to provide the crucial data and information present in the document.

A trained Named Entity Recognition model helps to recognize the entities in a document and can be very useful for getting valuable information from information centric documents like applications, resumes, research papers etc. In this report machine learning is used to generate the summaries and also collect the named entities in a document and save it using a file system.

Table of Contents

List of Figures

CHAPTER - 1
Introduction

Automatic text processing is a research field which is currently extremely active. The important task in this field is automatic summarization which consists of reducing the size of a text document while preserving its crucial information content. A summarization is a system that produces a condensed representation of the input given by the user for their consumption. This application has been researched a lot and still it is a nascent stage as compared to manual summarization. It is very useful to condense unstructured text of an article into a summary automatically.

The Abstractive Summarization selects the words based on semantic understanding. They interpret and examine the text using advanced natural language techniques. It can be in a similar way the human reads a text article and then summarizes in their own way. The extractive methods attempt to summarize the articles by selecting a subset of words that retain the most important points. Purely extractive summaries sometimes give better summaries.

In addition to a wholesome summary of a text document, other special documents like a resume, scientific, research paper need to have additional evidence that can be utilized for much better understanding in the condensed form of the document. For this purpose, a Named Entity Recognition model is trained to recognize the words of special importance like organizations, location, name, currency, date, time, etc. and provide the user with this additional information to cover all important aspects of the document summary.

1.1 Aim and Objectives of Project

In order to automate the manual process of generation of meaningful condensed form of a text document we aim to generate the summaries of a given text document and extract the important entities mentioned in those documents using machine learning and natural language processing algorithms of GloVe and training and using of Named Entity Recognition model. The objective of this project is to use the machine learning algorithm known as the Global Vectors for Word Representation (GloVe) as the statistics of word occurrence with that primary source of data and to train a Named Entity Recognition (NER) system to use linguistic grammar-based techniques to generate summaries of provided input.

1.2 Scope

The scope of this project is to make computer based automated production of condensed versions of documents. This automation is necessary for the information driven society. Its applications lies in areas of speech summaries, commentary, sports, news, scientific, business reports, human resource, resume filtering etc. Also with the information about the entities present in the original document such as the name, locations, organizations, time, date, currency, numbers and more can be used to evaluate some complex documents.

1.3 Organization of the report

Chapter 1 gives a brief overview about the aim for developing this project. The problem definition tells us about the expected outcome of the project for the application.

Chapter 2 of the report includes the literature survey on the existing system.

Chapter 3 shows the software model used to design is described, along with this our proposed system is also described.

Chapter 4 shows the Flow diagrams that give us an abstract view of the system. This chapter gives a detailed explanation about the technologies used and the techniques used in the development of the project.

Chapter 5 shows the project module that are designed and used in our proposed system.

Chapter 6 shows the overall working of the system.

Chapter 7 evaluates the performance of the project.
Chapter 8 describes the tasks that are achieved through this project. It describes the applications of the project.

Chapter 9 is conclusion. This chapter gives a summary of the entire project. It also gives the future scope for research and development in this project.

Chapter 2

Literature Survey

1) Automatic Text Summarization Survey[1]
   Year published: 2017
   The text summarization is based on the query generic approach and Neural networks based. The neural network is trained on a corpus of articles and then that neural network is modified through fusion to produce a summary of the most ranked sentences of the article. Through the feature fusion, the network discovers the importance of various features used to determine the summary worthiness of each sentence.
   Parts of Speech Tagging is the process of grouping of the words according to the speech category such as noun, verb, adverbs, adjectives, etc.
   Stop word filtering, stopwords are words which are filtered out before or after processing of corpus. It is fully non-objective depends on the needs. For example, a, an, in, by can be considered stop words from a plain text document.
   The query-based text summarization, the scoring of sentences of a given document is based on the frequency counts of words. The sentences with the highest scores are then extracted for the output summary together with their structural context.

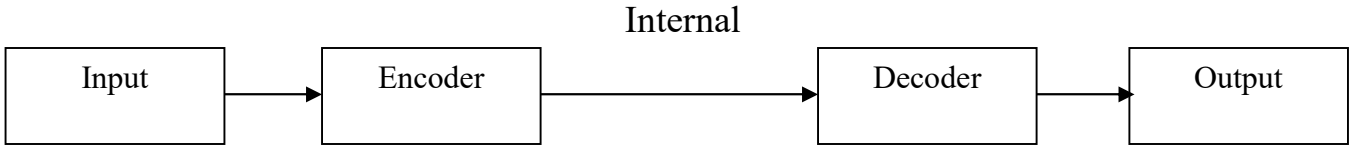
2) GloVe: Global Vectors for Word Representation[2]
   Year published: 2014
   The statistics of word occurrences in a corpus is the primary source of information available to all the unsupervised methods for learning word representations. Semantic vector space models of language represent each word with a real valued vector. These vectors can be used as features in variety of applications like document classification, question answering, named entity recognition, determining the similarity between multiple documents, etc. Most word vector methods rely on the distance or angle between pairs of words vectors as the primary method of evaluating the intrinsic quality of such set of words.

3) Named Entity Recognition[3]

Year published: 2011
NER systems have been created that use linguistic grammar-based techniques as well as the statistical models with aspects to machine learning. There are predefined NER models available but an NER model could be trained as well to perform better or as to our needs. Multiple NER models can be created for specific needs. These models should be robust across multiple domains as it is expected to be applied on a diverse set. It is the fundamental task and is the core of Natural Language Processing systems. NER has basically two tasks, which is firstly the identification of proper names in the text and secondly the classification of these names into a set of trained categories of interest such as persons, names, organizations, etc.

2.1 Existing System in market use:

Abstractive Summarizers are called these because they do not select sentences from the originally give text passage to create summary. Instead, they produce a paraphrasing of the main contents of the given text, using a vocabulary set different from the original document. This is very similar to as a manual summarization process. We create a semantic representation of the document, then we pick words from our general vocabulary and create short summary that represents all the points of an original document.
The basic structure of implementing this type of system uses the application of sequence-to-sequence RNNs. The models are designed to create an output sequence of words from an input sequence of words.
The Encoder-Decoder Networks



The encoder is responsible for taking in the input and generate a final state vector. The encoder may contain LSTM layers, RNN or GRU layers. Mostly LSTM layers are used due to the removed Exploding and vanishing gradient problem. It also uses the attention mechanism which aims to focus on some specific sequences from the input only rather than the entire input sequence to predict a word. This approach is very similar to the human approach to solve the problem.

Chapter 3

Software Analysis

The software we are using with their purposes are listed as follows

1) Natural Language Toolkit
   The natural language toolkit is a suit of libraries and programs for symbolic and statistical natural language processing for English written in Python programming language. The summary generation and named entity recognition is a subtask of information extraction. We use the NLTK in this project for multiple purposes.
   From NLTK we use the Punkt Sentence Tokenizer. This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations and words that start sentences. The NLTK data package includes a pre-trained Punkt tokenizer for English language.
   Another is the stopwords. The stopwords are the English words which do not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For example, the, he, have, etc. We use the NLTK to download the stopwords. It as available for various languages as well.

2) spaCy
   spaCy is an open-source software library for advanced natural language processing, written in Python programming language and Cython. It focuses on providing the software for production useage. It supports the deep learning workflows with connections of statistical models training with machine learning libraries. spaCy provides a good features for named entity recognition, text categorizing, parts of speech tagging, etc.
   spaCy provides some small pipelines like en_core_web_sm is a small English pipeline trained on written web texts like blogs, news, comments that includes vocabulary, vectors, syntax and entities.

3) Scikit-learn
   It is a free machine learning library for Python. It provides various features like classification, regression and clustering algorithms. In our project we use a metric of sklearn to compute the similarity between the texts and

sentences. We the metric called as cosine similarity. The cosine similarity is the measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them which is also same as the inner product of the same vectors normalized to both have length 1.

4) Networkx
   Networkx is a python library for studying graphs and networks. It is used for creation, manipulation and study of structure, dynamics and functions of complex networks.[9] It is used for pagerank the nodes of the graph created using the similarity matrix. The pagerank algorithm outputs a probability distribution used to represent the likelihood for that text to be worthy in a summary.

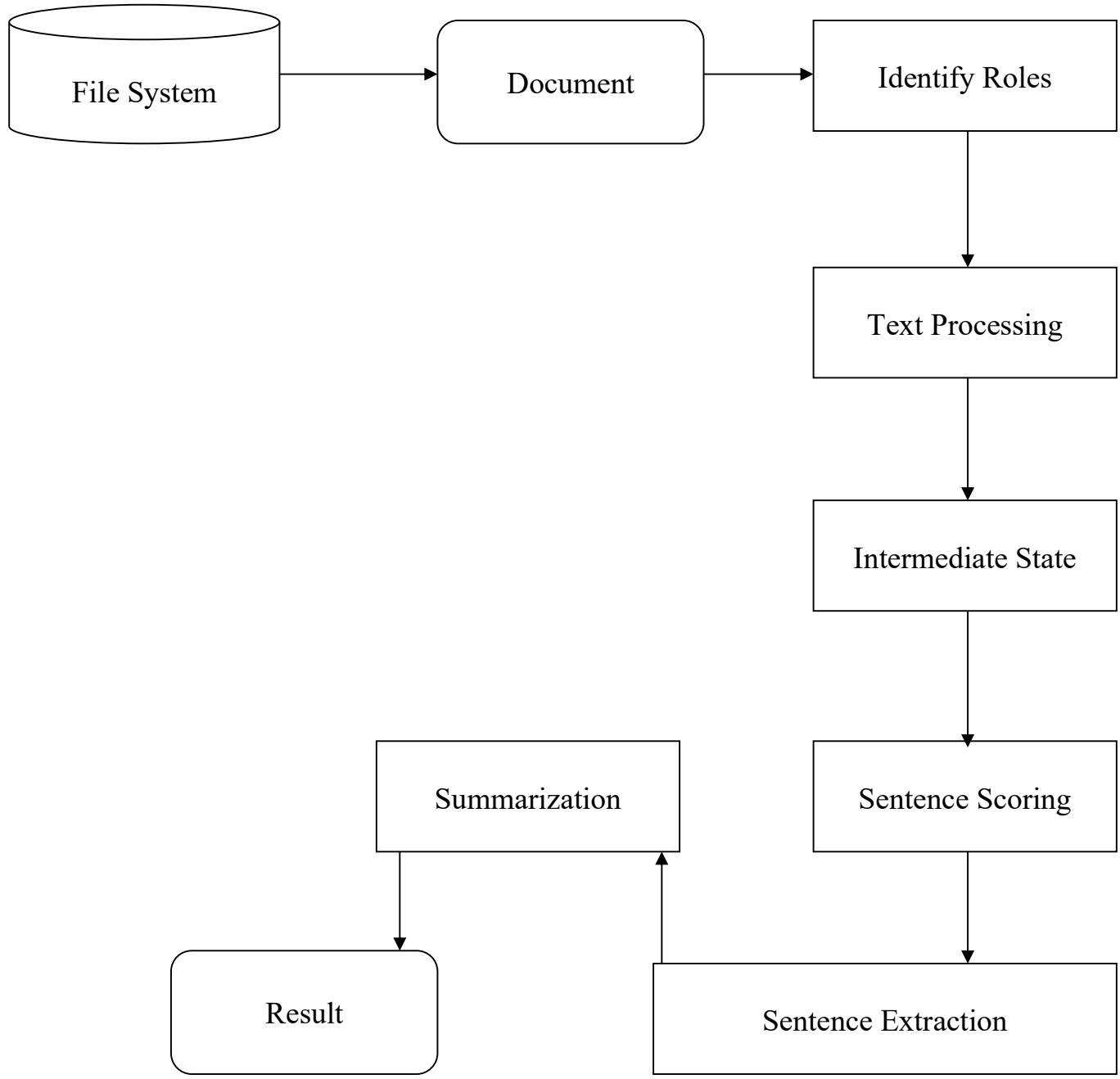
3.1 Proposed System

This project uses the extractive summary generation techniques. Extractive summarization is the extraction of the most important sentences from the whole document which would coherently represent the document. There can be multiple approached depending on the summarization application like for summarizing the reviews we may want to pick the highly positive and negative sentences as summary. Another can be some sentences which contains some objects and entities such as name, location, person, date, time, etc.
If we want to summarize a report or an article, we want to pick the important or main sentences from the article. Here to address the problem of main and important sentences we use ranking algorithm on different nodes of a generated tree and one such is the Page Rank algorithm which has an unsupervised approach.
To get the additional information such as the named entities we use a custom Named Entity Recognition using spaCy. It is a subtask of information extraction and seeks out categories specified entities in a body.

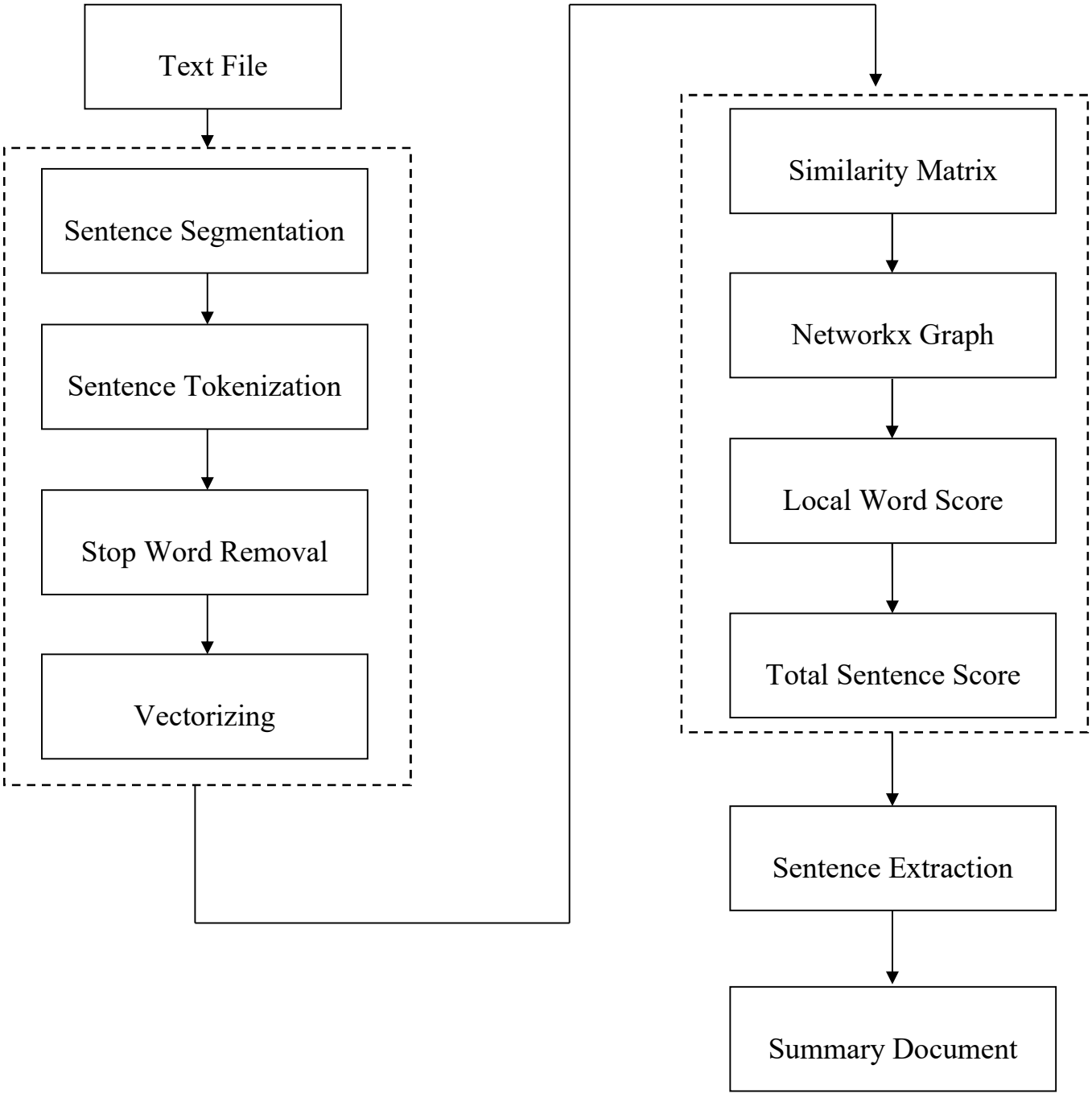
Chapter 4

Design and Implementation

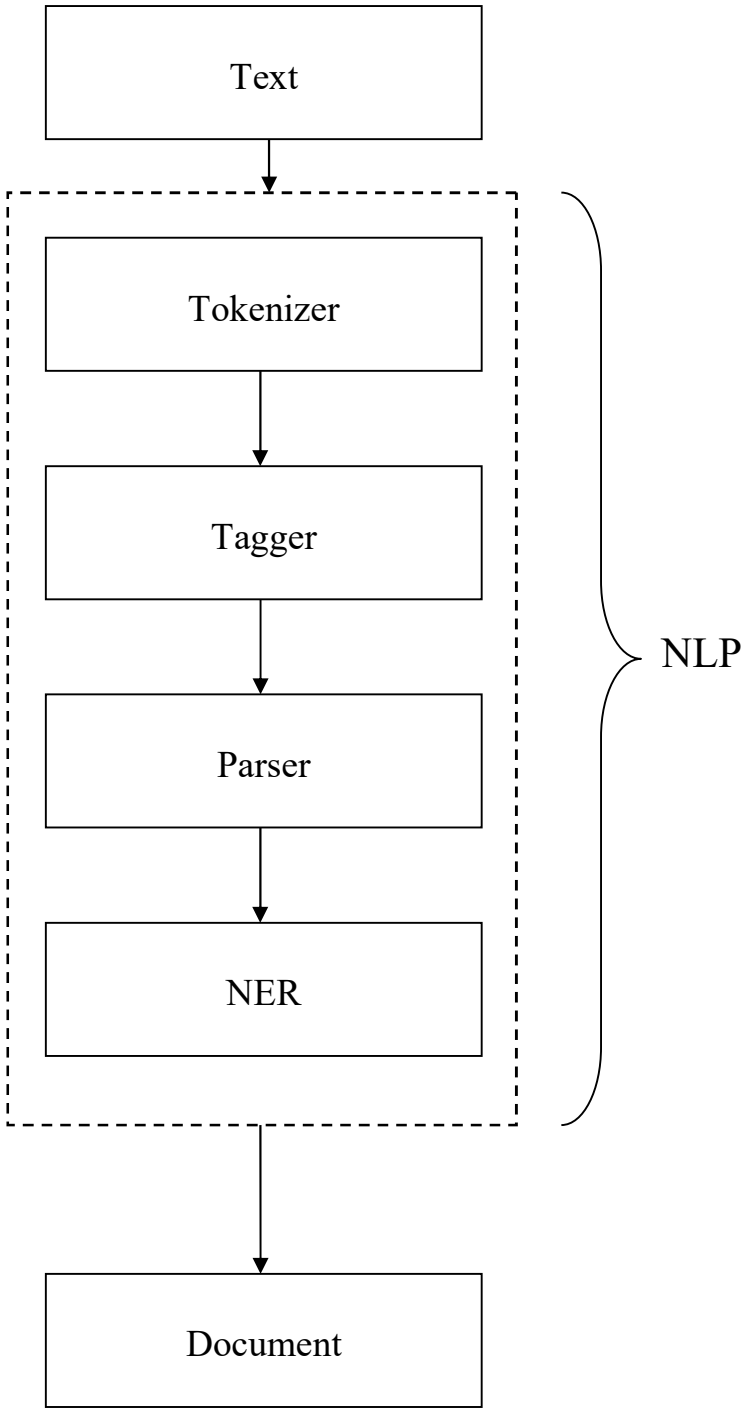
4.1 System Architecture



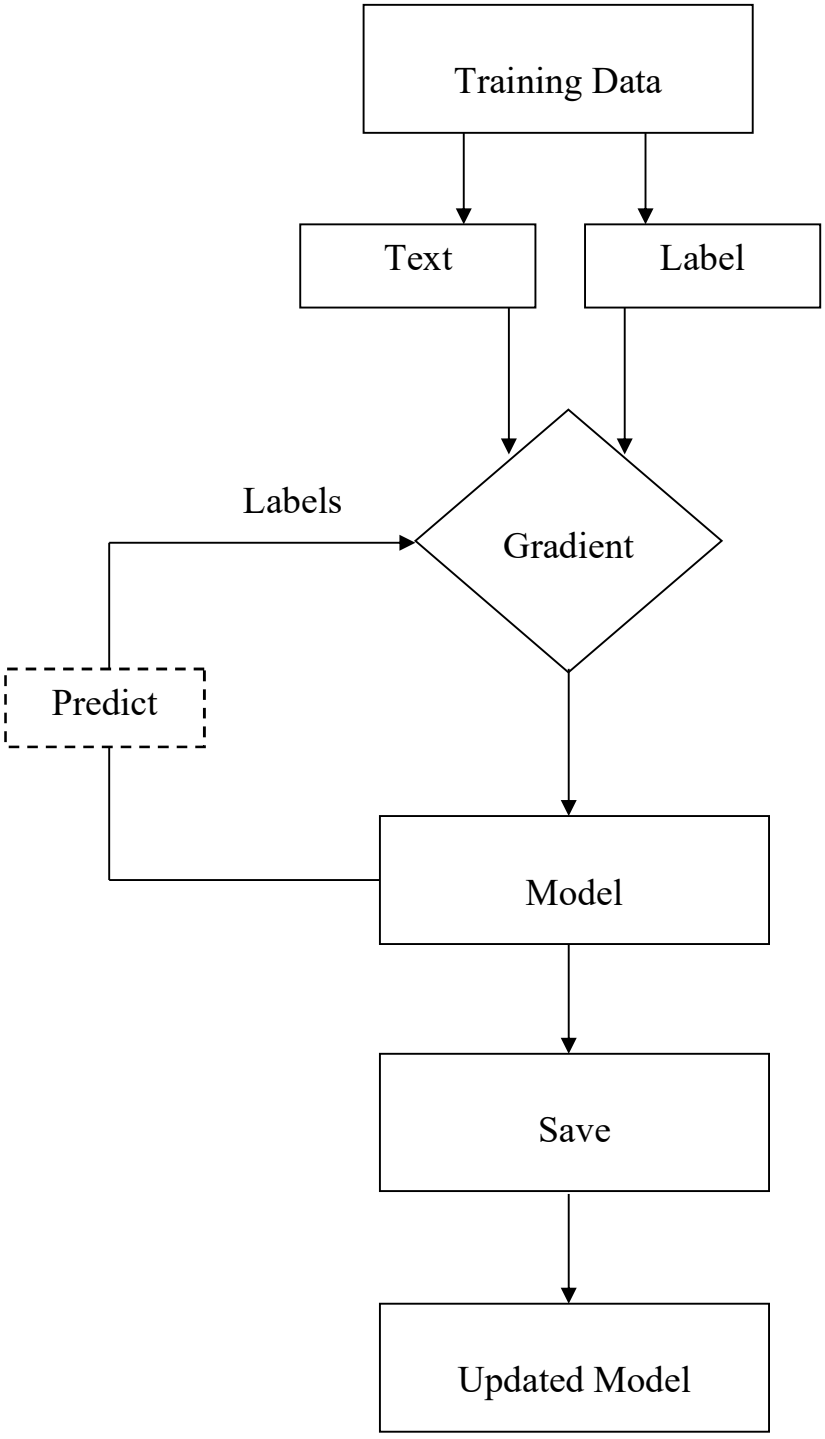
4.2 Summarization Flow diagram



4.3 spaCy NER Architecture



4.4 NER Flow Diagram



Chapter 5

Methodology

5.1 Approach

The type of approach we are using is called Extractive based approach. In this approach a subset of words that represent the most important points is pulled from a piece of the text and combined to make a summary. The dataset we will be using consists of huge samples of text with valid sources.
These extracted features can be of two kinds statistical which are based on frequency of some elements in the text and linguistic which are extracted from a simplified argumentative structure of the text.
First, we implement the punkt and stopwords from NLTK for specific purposes.
We load the Spacy for its parts of speech tagging process which is listed in a spacy module called en_core_web_sm. In corpus linguistics, part of speech tagging is also called grammatical tagging is the process of making up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. In simplified form it can be identification of words as nouns, verbs, adjectives, adverbs, etc.
Unsupervised tagging techniques use an untagged corpus for their training data and produce the tagset by induction. That is they observe the patters in the words used and derive part of speech categories themselves. For example, the statistics readily reveal that “the”, “a” and “an” occur in a similar context while “eat” occurs in a very different context. With sufficient iteration similar classes of words emerge that are very similar to those of human linguists would need. It converts the sentences to desired forms like a list of tuples where each tuple has a word and a tag element in it.

5.2 Preprocessing

In addition to the parts of speech tagging we need to score the individual words and as well as the combined individual sentences in the text. For this we use the Punkt sentence tokenizer, it is an NLTK project. This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations and words that start sentences. The NLTK data package includes a trained Punkt tokenizer for English language.

Before going with further process, we need to do some preprocessing of the text data that we have. We have to filter out the useless data as to process further. In the natural language processing the useless words or data is termed as the stop words.
A stop word is a very commonly used word such as “the”, “a”, an”, “in”, etc. that an engine has been programmed to ignore both when indexing entries for searching and retrieving them as the result. We do not want these words to take up space in the system and take valuable processing time. For this reason, we can remove them easily by storing a list of words that you consider to stop words. Natural Language Toolkit in python has a list of stopwords stored in multiple different languages. So, these words are not important for the algorithm and need to be removed.
In addition to remove the stop words we also clean the sentences:

1) Converting to lowercase
2) Remove non-alphabetic characters
3) Removing extraneous characters
4) Removing stop words
5) Lemmatizing texts

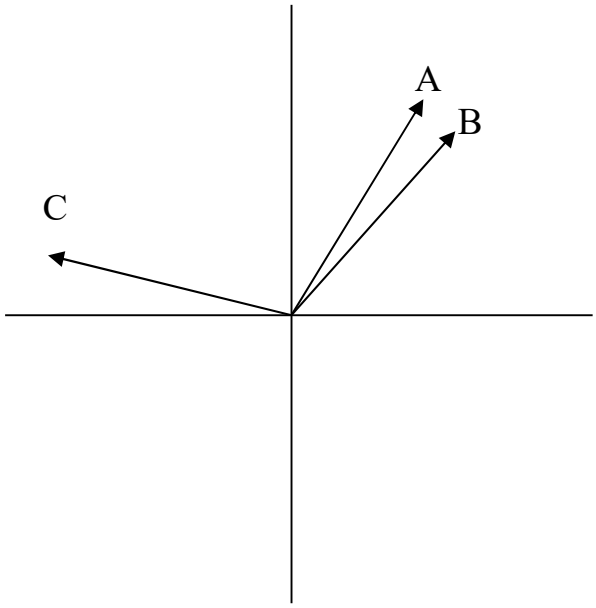
5.3 Vectorizing

The GloVe is used for obtaining vector representations of the words. It is an unsupervised learning algorithm for obtaining vector representations for words. Word vectors are simple vectors of numbers that represent the meaning of a word. Traditional approaches of machine learning such as one-hot encoding and bag-of-words models which are used as dummy variables to represent the presence or absence of a word in observation or a sentence can be useful for some machine learning tasks but it does not capture the information about a word meaning or context. This means potential relationships such as contextual closeness are not captured across collections of words.
Such encodings often provide insufficient baseline for complex natural language processing and lack the sophistication for more complex tasks such as translation and summary generation or categorization. The words vectors represent as multidimensional continuous floating-point numbers where semantically similar words are mapped to proximate points in geometric space.[5] A word vector is a row of real valued number s as opposed to dummy numbers, where each point captures a dimension of the word’s meaning and where semantically similar words have similar vectors.

5.4 Cosine Similarity

After generating the word vectors, we generate the similarity matrix. We generate this matrix using the cosine similarity. It computes the cosine similarity between the samples X and Y. It computes the similarity as the normalized dot product of X and Y. It is a metric used to determine how similar two entities are irrespective to their size. Mathematically it measures the cosine of the angle between two vectors projected in a multi-dimensional space.[6]
So mathematically if "X" and "Y" are two vectors, the cosine equation gives the angle between them. If the angle between those vectors is close to 0 degree or the cosine of the angle is close to 1 then those two vectors are very similar to each other, otherwise if the angle between them is closer to 90 degree or the cosine of the angle formed is closer to 1 or -1 then those two vectors can be orthogonal or completely opposite which makes them very different from each other.[6]

Similar vectors



The advantage of using cosine similarity over other similarity metric methods such as Euclid distance method is that, cosine similarity does not take the size of the sample into account, in other words it does not take the magnitude of the vectors into consideration instead it takes the angle formed between the two vectors which is very beneficial for comparison of similar words but with different lengths.

5.5 Ranking

In this part we rank the features of the text such as the preprocessed words and then the sentences as a whole so to combine them to form an extractive summary. For this we make the use of similarity matrix to construct a graph on which the ranking algorithms will be performed. The NetworkX package provides for the creation, manipulation and study of the structure, dynamics and functions of complex networks and graphs.
We use the Numpy ndarray property to represent the graph of the similarity matrix as an adjacency matrix. It gives the adjacency matrix representation of a graph. On this adjacency matrix we then implement a ranking algorithm which is called the pagerank algorithm. The pagerank algorithm returns the page rank of the nodes in the adjacency matrix which represents the graph. It computes a ranking of the nodes in the graph based on the structure of the incoming links.[7] Originally it was designed as an algorithm to rank web pages. It works by counting the number and quality of links to a word to determine a rough estimate of how important the node is. The underlying assumption is that the more important nodes are likely to receive more references in multiple contexts.
The algorithm of PageRank is that is outputs a probability distribution used to represent the likelihood that a person randomly referring to a node. It can be calculated for collections of documents of any size.[8]

5.6 Named Entity Recognition

This process is used to extract the additional information from the document that is very specific. This additional information extraction is very useful for some documents like scientific, research papers, resume or a job application, etc.
Using SpaCy we get the language processing pipelines. When we call the functions on the text, spaCy first tokenizes the text to product a Doc object. The Doc is then processed in several different steps, this is also known as the processing pipeline.[3]
The pipeline used by the training pipelines typically includes a tagger, a lemmatizer, a parser and the entity recognizer. Each pipeline component returns the processed Doc which is then passed to the next component.
SpaCy does provide the NER pipeline component, but we train our own NER model and put it into that NLP pipeline. For the training dataset the model is provided with over 200 resume samples with marked entities and randomized after each training iteration so it does not generalize the outcome. Using these we calculate the rankings of the multiple nodes and features. Them we calculate some extra parameters such as sentence position, sentence length, sentence weights, position taggers, word weights, inverse sentence frequency, token weights, token frequencies and using these we generate the extractive summaries.

Chapter 6

Implementation Details

The implementation of this project is done using the following sources in addition to the python packages imported.

1) Glove.6B.100d.txt
   The whole glove algorithm is used for obtaining the vector representations for words. The complete glove algorithms set consists of four files for four embedding representations which are[5]:
   a) Glove.6B.50d.txt for 6 billion tokens and 50 features
   b) Glove.6B.100d.txt for 6 billion tokens and 100 features
   c) Glove.6B.200d.txt for 6 billion tokens and 200 features
   d) Glove.6B.300d.txt for 6 billion tokens and 300 features
   We are using the one with 100 features as our project's current need.
   The contents of this glove file is the complete numerical representation of words in almost every context with continuous numbers and also for special characters which are commonly used in English language text.
   An example of the content in the file is given as:
   the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953
   , -0.10767 0.11053 0.59812 -0.54361 0.67396 0.10663
   of -0.1529 -0.24279 0.89837 0.16996 0.53516

2) Pipeline en_core_web_sm
   It is a pipeline model trained for English and written on web texts such as blogs, news, comments, articles etc.
   This machine learning model is used for parts of speech tagging in our project. It comes with three variants of small, medium and large. The difference is mostly about the size and statistical aspects. The larger models take much longer to load as compared to small model.

3) Data for Training NER model
   The custom NER model is made and trained on a dataset consisting of documents which contain high amounts of additional information only. The current choice of documents are resumes and job applications in the training dataset. The dataset provided is in a .plk file form. This type of file is created by pickle which is a python module that enables objects to be serialized to

files on a disk and deserialized back into the program at runtime. It contains the byte stream which represents the objects in the file.

6.1 Working of the Code

1) Loading the SpaCy pipeline model for parts-of-speech tagging, tokenizer and stopwords

```
nltk.download('punkt')
nltk.download('stopwords')

nlp_spacy = spacy.load('en_core_web_sm')
```

2) Removing the extra spaces from an article.
It takes the article or the document as an input and removes the extra spaces and some special characters from the body.

```
def remove_extraneous_text(sentence:str)->str:

        # Remove multiple spaces
        sentence = re.sub(" +", " ", sentence)

        if ") --" in sentence:
          sentence = sentence.split(") --")[-1]
```

3) Removing StopWords
This function takes the sentences as the input and removes all the stopwords from the sentence. It returns the same sentence without any stopwords.

```
def remove_stopwords(sentence:str)->str:

        sentence = " ".join([word for word in sentence.split() if word not
in stop_words])

        return sentence
```

4) Text Lemmatization

This function takes a sentence as input and uses SpaCy to convert each word into its lemma. The lemma is a informal logic and argument mapping, a general minor proven proposition which is used as a stepping stone for a larger result. We use the nlp_spacy class defined when loading the spacy pipeline for this purpose.

```
def lemmatize_text(sentence:str)->str:

        sentence = nlp_spacy(sentence)
        sentence = ' '.join([word.lemma_ if word.lemma_ != "-PRON-"
else word.text for word in sentence])
        return sentence
```

5) Cleaning the Text

This function does some cleaning, preprocessing of the text data which is given to it as the input. Also calls multiple functions to clean the sentences by converting to lowercase, remove non alphabetic characters, removing extraneous characters, removes the stopwords, and lemmatizing words.

```
def clean_text(sentence:str)->str:

        sentence = sentence.lower()
        sentence = re.sub("[^a-zA-Z]", " ", sentence)
        sentence = remove_extraneous_text(sentence)
        sentence = remove_stopwords(sentence)
        sentence = lemmatize_text(sentence)
        return sentence
```

6) Total terms

This function takes a sentence as input and returns the total number of terms or tokens in that sentence.

```
def get_total_terms(cleaned_sentences:list)->int:

        total_terms = 0
        for sentence in cleaned_sentences:
          total_terms += len(sentence.split())
```

```
return total_terms
```

7) Term Frequencies
   This function takes a list as an input and returns a dictionary containing tokens as keys and their frequencies as values.

```
def get_term_frequencies(cleaned_sentences:list)->dict:

    freq_dict = {}

    for sentence in cleaned_sentences:
      for word in sentence.split():
            freq_dict[word] = freq_dict.get(word, 0) + 1

    return freq_dict
```

8) Term weights
   This function takes a list of sentences as input and returns a dictionary containing Tokens as keys and their weightage as values. The weight of everything in the list is calculated by using the formula:
   TW(i) = (TF(ti) * 1000) / (Nt)
   Where "ti" is each token, "TW" is the term weight, "TF" is the term frequency and "Nt" is the total number of terms.

```
def get_term_weights(cleaned_sentences:list)->dict:

    total_terms = get_total_terms(cleaned_sentences)
    term_freq_dict = get_term_frequencies(cleaned_sentences)
    term_weights = dict()

    for key, value in term_freq_dict.items():
      term_weights[key] = (value * 1000) / total_terms

    return term_weights
```

9) Inverse sentence Frequency

This function takes in a list of sentences and returns a dictionary containing Tokens as keys and their inverse sentence frequency as values. The inverse sentence frequency is calculated as:
ISF(ti) = log((Ns) / Nti)
where “ti” is each token, “ISF” is inverse sentence frequency, “Ns” is total number of sentences in paragraph and “Nti” are the total number of sentences in which “ti” appeared in that paragraph.

```
def inverse_sentence_frequency(cleaned_sentences:list)->dict:

        vocabulary = set()

        for sentence in cleaned_sentences:
          vocabulary = vocabulary.union(set(sentence.split()))

        isf = dict()
        number_of_sentences = len(cleaned_sentences)

        for word in vocabulary:
          number_of_appearances = 0

          for sentence in cleaned_sentences:
                if word in sentence:
                      number_of_appearances += 1


        isf[word]=np.log(number_of_sentences/number_of_appearance
s)

        return isf
```

10) Word weights
Takes in a list of sentences and returns a dictionary containing Tokens as keys and their resultant weightage as values. The weightage is calculated as:
RW(ti) = ISF(ti) * TW(ti)

where “ti” is each token, “RW” is resultant weightage, “ISF” is inverse sentence frequency and “TW” is term weightage.

```
def word_weights(cleaned_sentences:str)->dict:

                    term_weights = get_term_weights(cleaned_sentences)
                    inverse_sentence_freq=
inverse_sentence_frequency(cleaned_sentences)

                    resultant_weights = dict()

                    for word in term_weights.keys():
                     resultant_weights[word]      =      term_weights[word]      *
inverse_sentence_freq[word]

                    return resultant_weights
```

11) Parts of speech tagging

Takes in a list of sentences and returns a list of lists, where each Token is represented as a tuple of the form (Token, POS tag).

```
def pos_tagging(cleaned_sentences:list)->list:

                    tagged_sentences = []

                    for sentence in cleaned_sentences:
                     sentence_nlp = nlp_spacy(sentence)

                     tagged_sentence = []

                     for word in sentence_nlp:
                            tagged_sentence.append((word, word.pos_))

                     tagged_sentences.append(tagged_sentence)

                    return tagged_sentences
```

12) Sentence Weights

Takes in a list of POS tagged sentences and total number of terms. Returns a list containing the sentence weight of each sentence. The sentence weight is calculated as:

SW(si) = Number of nouns and verbs in sentence / total number of terms in paragraph.

```
def sentence_weights(tagged_sentences:list, total_terms:int)->list:

                    sent_weights = []

                    for sentence in tagged_sentences:
                     relevance_count = 0

                     for word, tag in sentence:
                             if tag == 'NOUN' or tag == 'VERB':
                                     relevance_count += 1

                     sent_weights.append(relevance_count / total_terms)

                    return sent_weights
```

13) Sentence Position

Takes in a list of sentences and returns weight for each sentence based on it's position. For this we have defined some weight checkpoints in the list named weights.

```
def sentence_position(cleaned_sentences:list)->list:

                    sent_position = []
                    number_of_sentences = len(cleaned_sentences)

                    weights = [0, 0.25, 0.23, 0.14, 0.08, 0.05, 0.04, 0.06, 0.04, 0.04,
0.15]

                    for i in range(1, len(cleaned_sentences)+1):
                     sent_position.append(weights[int(np.ceil(10 * (i /
number_of_sentences)))])
```

```
        return sent_position
```

14) Sentence Length

Takes in a list of sentences and returns a list containing length of each sentence.

```
def sentence_length(cleaned_sentences:list)->list:

        sent_len = []

        for sentence in cleaned_sentences:
          sent_len.append(len(sentence.split()))

        return sent_len
```

15) Text ranking

Takes a list of sentences and Glove word embeddings as input and returns a dictionary containing sentences index as key and rank as value. The ranking is done based on the PageRank algorithm.

```
def text_rank(sentences:list, word_embeddings:dict)->dict:
        # Clean sentences for PageRank algorithm.
        clean_sentences = pd.Series(sentences).str.replace("[^a-zA-Z]", "
")
        clean_sentences = [s.lower() for s in clean_sentences]
        clean_sentences = [remove_stopwords(r) for r in
clean_sentences]

        # Replace each word with Glove embeddings. The Sentence
vector is the average of the sum of embeddings of all words in that
        # sentence.
        sentence_vectors = []
        for i in clean_sentences:
          if len(i) != 0:
                v = sum([word_embeddings.get(w, np.zeros((100, ))) for
w in i.split()]) / (len(i.split()) + 0.001)
          else:
                v = np.zeros((100, ))
          sentence_vectors.append(v)
```

```
                # Initialize a similarity matrix for pair of sentences
                sim_mat = np.zeros([len(sentences), len(sentences)])

                # Calculate cosine similarity for each pair of sentences
                for i in range(len(sentences)):
                  for j in range(len(sentences)):
                        if i != j:
                              sim_mat[i][j]=
cosine_similarity(sentence_vectors[i].reshape(1,100),
sentence_vectors[j].reshape(1, 100))[0, 0]

                # Create a PageRank graph using similarity matrix
                nx_graph = nx.from_numpy_array(sim_mat)
                scores = nx.pagerank(nx_graph)

                return scores
```

16) Ranking Features

Takes a list of sentences as input and returns a dictionary containing ranking of each sentence. The ranking is calculated using word and sentence level features.

```
def feature_rank(sentences:list)->dict:

                cleaned_sentences = [clean_text(sentence) for sentence in
sentences]

                term_weights = word_weights(cleaned_sentences)
                tagged_sentences = pos_tagging(cleaned_sentences)
                total_terms = get_total_terms(cleaned_sentences)
                sent_weights = sentence_weights(tagged_sentences,
total_terms)
                sent_position = sentence_position(cleaned_sentences)
                sent_len = sentence_length(cleaned_sentences)

                sentence_scores = []

                for index, sentence in enumerate(cleaned_sentences):
```

```
        score = 0

        for word in sentence.split():
                score += term_weights[word]

        score *= sent_weights[index]
        score += sent_position[index]

        if sent_len[index] != 0:
                score /= sent_len[index]
        else:
                score = 0

        sentence_scores.append(score)

    sentence_scores = sentence_scores / np.sum(sentence_scores)

    final_scores = dict()

    for i in range(len(sentence_scores)):
        final_scores[i] = sentence_scores[i]

    return final_scores
```

17) After this the extractive summary generation takes by calling all these functions with their respective parameters.

18) NER
Open the training data to train the NER model for spacy.

```
train_data = pickle.load(open('train_data.pkl', 'rb'))
```

19) Create a new blank pipeline for model in spacy to further train it using training dataset.

```
nlp = spacy.blank('en')
```

20) Function to train the model.

```
def train_model(train_data):
  if 'ner' not in nlp.pipe_names: # Checking if NER is present in pipeline
```

```
    ner = nlp.create_pipe('ner') # creating NER pipe if not present
    nlp.add_pipe(ner, last=True) # adding NER pipe in the end

  for _, annotations in train_data: # Getting 1 resume at a time from our training data of 200 resumes
    for ent in annotations['entities']: # Getting each tuple at a time from 'entities' key in dictionary at index[1] i.e.,(0, 15, 'Name') and so on
      ner.add_label(ent[2]) # here we are adding only labels of each tuple from entities key dict, eg:- 'Name' label of (0, 15, 'Name')

  # In above for loop we finally added all custom NER from training data.

  other_pipes = [pipe for pipe in nlp.pipe_names if pipe != 'ner'] # getting all other pipes except NER.
  with nlp.disable_pipes(*other_pipes): # Disabling other pipe's as we want to train only NER.
    optimizer = nlp.begin_training()

    for itn in range(10): # trainig model for 10 iteraion
      print('Starting iteration ' + str(itn))
      random.shuffle(train_data) # shuffling data in every iteration
      losses = {}
      for text, annotations in train_data:
        try:
          nlp.update(
            [text],
            [annotations],
            drop=0.2,
            sgd=optimizer,
            losses=losses
          )
          print(losses)
        except Exception as e:
          print('Pass')
          pass

train_model(train_data)
```

21) Saving the model and using it.

```
# Saving our trained model to re-use.
nlp.to_disk('nlp_model')
nlp_model = spacy.load('nlp_model')
# Checking all the custom NER created
print(nlp_model)
```

22) Sample form of training data for Named Entity Recognition.

```
TRAIN_DATA = [
        ("Walmart is a leading e-commerce company", {"entities": [(0, 7, "ORG")]}),
        ("I reached Chennai yesterday.", {"entities": [(19, 28, "GPE")]}),
        ("I recently ordered a book from Amazon", {"entities": [(24,32, "ORG")]}),
        ("I was driving a BMW", {"entities": [(16,19, "PRODUCT")]}),
        ("I ordered this from ShopClues", {"entities": [(20,29, "ORG")]}),
        ("Fridge can be ordered in Amazon ", {"entities": [(0,6, "PRODUCT")]}),
        ("I bought a new Washer", {"entities": [(16,22, "PRODUCT")]}),
        ("I bought a old table", {"entities": [(16,21, "PRODUCT")]}),
        ("I bought a fancy dress", {"entities": [(18,23, "PRODUCT")]}),
        ("I rented a camera", {"entities": [(12,18, "PRODUCT")]}),
        ("I rented a tent for our trip", {"entities": [(12,16, "PRODUCT")]}),
        ("I rented a screwdriver from our neighbour", {"entities": [(12,22,
"PRODUCT")]}),
        ("I repaired my computer", {"entities": [(15,23, "PRODUCT")]}),
        ("I got my clock fixed", {"entities": [(16,21, "PRODUCT")]}),
        ("I got my truck fixed", {"entities": [(16,21, "PRODUCT")]}),
        ("Flipkart started it's journey from zero", {"entities": [(0,8, "ORG")]}),
        ("I recently ordered from Max", {"entities": [(24,27, "ORG")]}),
        ("Flipkart is recognized as leader in market",{"entities": [(0,8, "ORG")]}),
        ("I recently ordered from Swiggy", {"entities": [(24,29, "ORG")]})
        ]
```

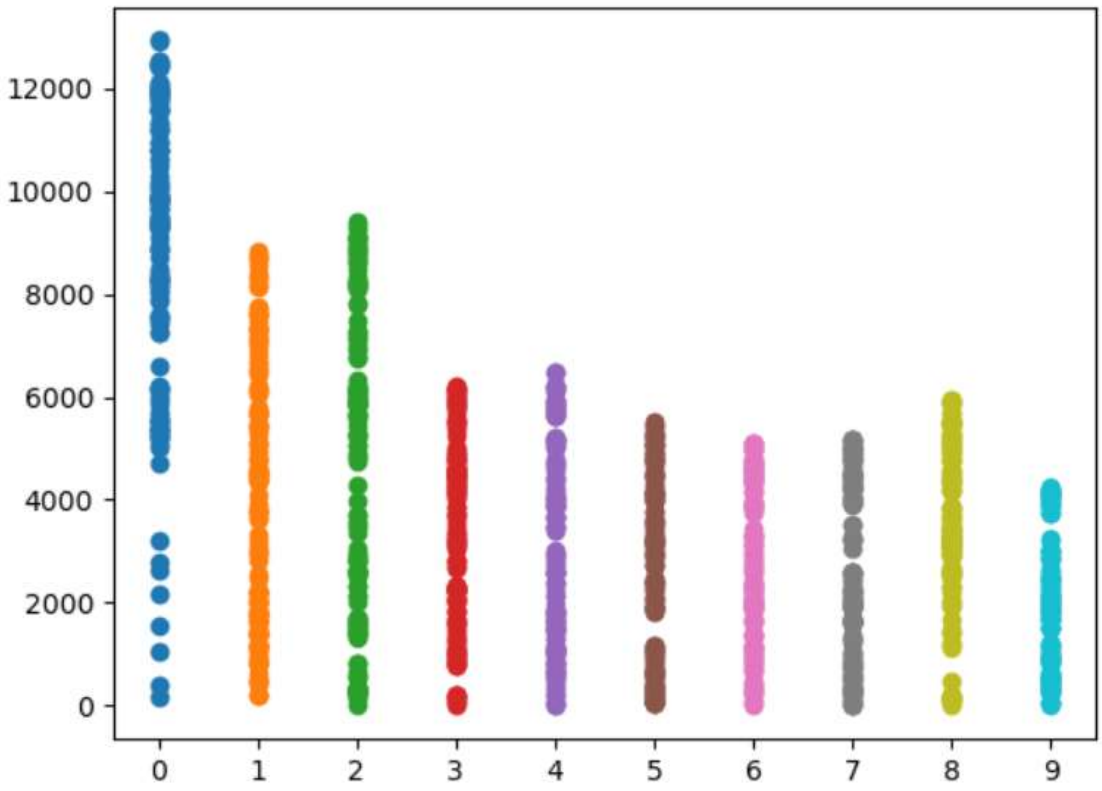
Chapter 7

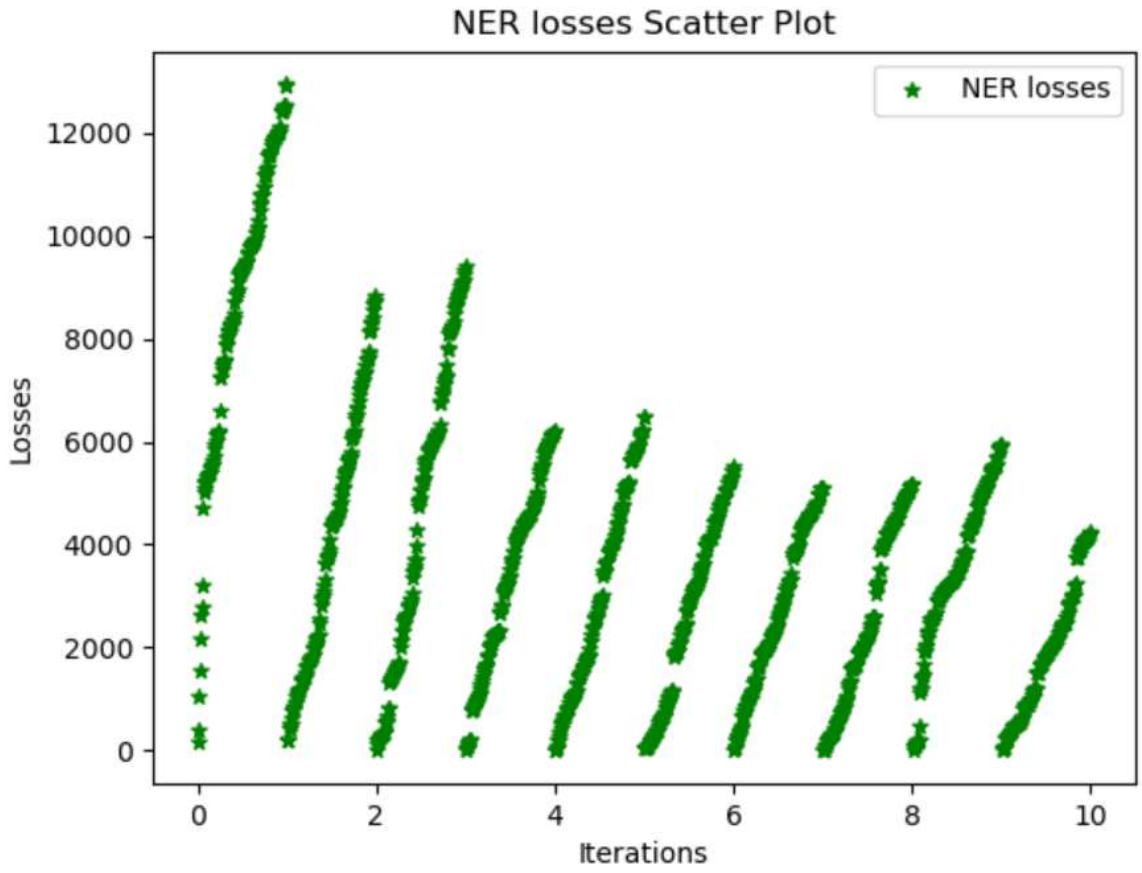
Performance Evaluation

The performance evaluation of the summary generation is done on the basis of the length of the summary generated as compared to the original text. For most of the plain text documents the program has successfully generated the summaries with successfully preservation of meaningful important words and as well as the meaning in the context of each word.
The output of extractive summary generation is a summary of about 20% to 30% of the total length of the original article or document. For a use case we took a text document which has an article regarding Ganesh Chaturthi festival of the total length of 3304 characters. After processing it through the program the resulted summary contains 666 characters which is approximately 20% of the original document.

The training of NER model pipeline is evaluated on the bases of pipeline losses for each data entity after each iteration of training in the model. For a short dataset the losses will be small. Out dataset contains 200 data points and randomly trained.
The initial losses of this for the first iteration was above 10000 and with subsequent iterations the losses came down.
{'ner': 205.58794659748673}
{'ner': 216.2805469001487}
{'ner': 421.50882047068626}
{'ner': 431.6458966203809}
Pass
{'ner': 523.0486146875501}
{'ner': 533.3927916462893}
Pass
Pass
{'ner': 618.948973077106}
Pass
{'ner': 779.149585860538}

Scatter plot for NER training losses




NER losses Scatter Plot
NER losses
Losses
Iterations
0
2000
4000
6000
8000
10000
12000
0
2
4
6
8
10

Chapter 8

Results and Conclusion

This project is now able to generate the extractive summaries of the given text document and the size of the generated summary is about 20% of the original document. Also enables to extract the valuable additional information for the documents such as the entities of names, locations, organizations etc. from special documents.

NER systems have been created that use linguistic grammar-based techniques as well as statistical models such as machine learning. Hand-crafted grammar-based systems typically obtain better precision in reading and understanding the contexts of the documents.

8.1 Screenshots

Input Text

```
Ganesh Chaturthi is a religious festival observed by Hindus to celebrate the birth of deity Lord Ganesh
(the son of Lord Shiva and Parvati). According to the mythological anecdotes, Lord Ganesh is worshipped
as the god of wisdom, good fortune and prosperity. This Hindu festivity, also popularly known as Vinayaka
Chavithi or Vinayaka Chaturthi in many places, commences in the month of Bhaadrapada that kicks off on
the shukla chaturthi and last for 10-12 days. Celebration of Ganesh Chaturthi usually falls between 20
August and 15 September every year. Ganesh Chaturthi 2016 will be celebrated on Monday, September 5
across the nation. It is an important festival and observed with fervour in the states of Karnataka,
Maharashtra, Tamil Nadu and Andhra Pradesh.Lord Ganesh is easily recognized for his elephant head
irrespective of his other attributes. About his birth there are numerous myths and traditional stories.
Ganesh Chaturthi is celebrated with great enthusiasm by the devotees across India by offering prayers and
preparing sweet dishes. Markets are embellished and skilled artisans start crafting clay models of Lord
Ganesh in various poses, a few months before the mega-event. People install Ganesh idols in
well-decorated mandaps and mount fascinating Ganesh Chaturthi photos or pictures in their houses or
localities to offer their prayers and celebrate the festival in the best way possible. People and
devotees also install images and statues at their homes and offer prayers during this auspicious
festival. At mandaps, the celebration kicks off with the chanting of mantras and prayers offered by
priest clad in white or red dhoti. Loads of sweets, jaggery, modakas, flowers are offered to Lord Ganesh.
Throughout the auspicious ceremony, priests chant hymns from the Vedas and Upanishads as an act to show
reverence. Statues of Lord Ganesh are decorated with glittering lights, colourful flower garlands and
anointed with kumkum on the occasion of Ganesh Chaturthi. Chanting hymns from the Vedas and Ganapati
Atharva Shirsha Upanishad is also considered as an inevitable traditional sacrament to honour and praise
the Lord. At the end of the puja ceremony, 'prasad' is distributed to the devotees as it is believed to
have deity's blessings. The festivity concludes after a continuous celebration of more than a week by
submerging the statue in rivers. The joy of Ganesh Chaturthi is celebrated in many ways by ardent
```

Output summary

```
People and devotees also install images and statues at their homes and offer prayers
during this auspicious festival. Believers of Lord Ganesh leave no stone unturned to
share the significance and excitement of the celebration.To mark the quintessence of
the religious festival, people send Ganesh Chaturthi wishes to their friends and family
members in form of fancy gifts and devotional text messages. Well-written inspirational
Ganesh Chaturthi SMSes are also considered a powerful medium to share the enthusiasm
and significance of the celebration as these short and precise messages allow senders
to edit and insert images accordingly to give a personal touch.
```

Snippet of Training losses of NER model:

```
{'ner': 4206.115725940801}
{'ner': 4208.7205864101015}
{'ner': 4215.893974960625}
{'ner': 4219.1757511729065}
{'ner': 4232.626771574051}
{'ner': 4239.095517908341}
Pass
{'ner': 4273.658082501973}
{'ner': 4316.602687404736}
{'ner': 4318.459301545825}
{'ner': 4407.628424822893}
Pass
Pass
Pass
{'ner': 4410.651899351631}
{'ner': 4415.515678114218}
{'ner': 4445.1222989689995}
Pass
{'ner': 4478.317854132024}
{'ner': 4495.119600750643}
{'ner': 4515.310396992563}
{'ner': 4521.433215256902}
Pass
```

Sample Input to NER model
Resume

Alice Clark

AI / Machine Learning

Delhi, India Email me on Indeed

- 20+ years of experience in data handling, design, and development
- Data Warehouse: Data analysis, star/snow flake scema data modelling and design specific to data warehousing and business intelligence
- Database: Experience in database designing, scalability, back-up and recovery, writing and optimizing SQL code and Stored Procedures, creating functions, views, triggers and indexes. Cloud platform: Worked on Microsoft Azure cloud services like Document DB, SQL Azure, Stream Analytics, Event hub, Power BI, Web Job, Web App, Power BI, Azure data lake analytics(U-SQL)

Willing to relocate anywhere

WORK EXPERIENCE

Software Engineer

Microsoft – Bangalore, Karnataka

January 2000 to Present

1. Microsoft Rewards Live dashboards:

January 2000 to Present

1. Microsoft Rewards Live dashboards:

Description: - Microsoft rewards is loyalty program that rewards Users for browsing and shopping online. Microsoft Rewards members can earn points when searching with Bing, browsing with Microsoft Edge and making purchases at the Xbox Store, the Windows Store and the Microsoft Store. Plus, user can pick up bonus points for taking daily quizzes and tours on the Microsoft rewards website. Rewards live dashboards gives a live picture of usage world-wide and by markets like US, Canada, Australia, new user registration count, top/bottom performing rewards offers, orders stats and weekly trends of user activities, orders and new user registrations. the PBI tiles gets refreshed in different frequencies starting from 5 seconds to 30 minutes.
Technology/Tools used

EDUCATION

Indian Institute of Technology – Mumbai

2001

SKILLS

Machine Learning, Natural Language Processing, and Big Data Handling

Configuration of NER model:

```
{
  "beam_width":1,
  "beam_density":0.0,
  "beam_update_prob":1.0,
  "cnn_maxout_pieces":3,
  "nr_feature_tokens":6,
  "nr_class":46,
  "hidden_depth":1,
  "token_vector_width":96,
  "hidden_width":64,
  "maxout_pieces":2,
  "pretrained_vectors":null,
  "bilstm_depth":0,
  "self_attn_depth":0,
  "conv_depth":4,
  "conv_window":1,
  "embed_size":2000
}
[paths]
train = null
dev = null
vectors = null
init_tok2vec = null

[system]
seed = 0
gpu_allocator = null

[nlp]
lang = "en"
pipeline = ["ner"]
disabled = []
before_creation = null
after_creation = null
after_pipeline_creation = null
batch_size = 1000
tokenizer = {"@tokenizers":"spacy.Tokenizer.v1"}

[components]

[components.ner]
```

Meta data of NER model:


```
{
  "lang":"en",
  "name":"model",
  "version":"0.0.0",
  "spacy_version":">=2.3.5",
  "description":"",
  "author":"",
  "email":"",
  "url":"",
  "license":"",
  "spacy_git_version":"1d4b1dea2",
  "vectors":{
    "width":0,
    "vectors":0,
    "keys":0,
    "name":"spacy_pretrained_vectors"
  },
  "pipeline":[
    "ner"
  ],
  "factories":{
    "ner":"ner"
  },
```


Output using NER model

```
(mini) C:\Users\Admin\Desktop\Test\Text-Summarization>python spacy2.py
<spacy.lang.en.English object at 0x000001B6D6769490>
a
NAME                - Alice Clark
a
LOCATION            - Delhi
a
DESIGNATION         - Software Engineer
a
COMPANIES WORKED AT  - Microsoft –
a
COLLEGE NAME        - Indian Institute of Technology
a
GRADUATION YEAR     - 2001
```

Glove data snippet:

```
the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545 0.087459 0.28787 -0.06731 0.30906 -0.26384 -0.13231 -0.20757 0.33395 -0.33
, -0.10767 0.11053 0.59812 -0.54361 0.67396 0.10663 0.038867 0.35481 0.06351 -0.094189 0.15786 -0.81665 0.14172 0.21939 0.58505 -0.52158 0.22783 -0.16642 -0.
. -0.33979 0.20941 0.46348 -0.64792 -0.38377 0.038034 0.17127 0.15978 0.46619 -0.019169 0.41479 -0.34349 0.26872 0.04464 0.42131 -0.41032 0.15459 0.022239 -0
of -0.1529 -0.24279 0.89837 0.16996 0.53516 0.48784 -0.58826 -0.17982 -1.3581 0.42541 0.15377 0.24215 0.13474 0.41193 0.67043 -0.56418 0.42985 -0.012183 -0.1
to -0.1897 0.050024 0.19084 -0.049184 -0.089737 0.21006 -0.54952 0.098377 -0.20135 0.34241 -0.092677 0.161 -0.13268 -0.2816 0.18737 -0.42959 0.96039 0.13972
and -0.071953 0.23127 0.023731 -0.50638 0.33923 0.1959 -0.32943 0.18364 -0.18057 0.28963 0.20448 -0.5496 0.27399 0.58327 0.20468 -0.49228 0.19974 -0.070237 -
in 0.085703 -0.22201 0.16569 0.13373 0.38239 0.35401 0.01287 0.22461 -0.43817 0.50164 -0.35874 -0.34983 0.055156 0.69648 -0.17958 0.067926 0.39101 0.16039 -0
a -0.27086 0.044006 -0.02026 -0.17395 0.6444 0.71213 0.3551 0.47138 -0.29637 0.54427 -0.72294 -0.0047612 0.040611 0.043236 0.29729 0.10725 0.40156 -0.53662 0
" -0.30457 -0.23645 0.17576 -0.72854 -0.28343 -0.2564 0.26587 0.025309 -0.074775 -0.3766 -0.057774 0.12159 0.34384 0.41928 -0.23236 -0.31547 0.60939 0.25117
's 0.58854 -0.2025 0.73479 -0.68338 -0.19675 -0.1802 -0.39177 0.34172 -0.60561 0.63816 -0.26695 0.36486 -0.40379 -0.1134 -0.58718 0.2838 0.8025 -0.35303 0.30
for -0.14401 0.32554 0.14257 -0.099227 0.72536 0.19321 -0.24188 0.20223 -0.89599 0.15215 0.035963 -0.59513 -0.051635 -0.014428 0.35475 -0.31859 0.76984 -0.08
- -1.2557 0.61036 0.56793 -0.96596 -0.45249 -0.071696 0.57122 -0.31292 -0.43814 0.90622 0.06961 -0.053104 0.25029 0.27841 0.77724 0.26329 0.56874 -1.1171 -0.
that -0.093337 0.19043 0.68457 -0.41548 -0.22777 -0.11803 -0.095434 0.19613 0.17785 -0.020244 -0.055409 0.33867 0.79396 -0.047126 0.44281 -0.061266 0.20796 0
on -0.21863 -0.42664 0.5196 0.0043103 0.58045 -0.10873 -0.37726 0.4566 -0.60627 -0.075773 0.11306 0.17703 0.1605 0.074514 0.63649 -0.078852 0.75268 -0.24962
is -0.54264 0.41476 1.0322 -0.40244 0.46691 0.21816 -0.074864 0.47332 0.080996 -0.22079 -0.12808 -0.1144 0.50891 0.11568 0.028211 -0.3628 0.43823 0.047511 0.
was 0.13717 -0.54287 0.19419 -0.29953 0.17545 0.084672 0.67752 0.098295 -0.035611 0.21334 0.51663 0.20687 0.44082 -0.33655 0.56025 -0.6879 0.51957 -0.21258 -
said -0.13128 -0.452 0.043399 -0.99798 -0.21053 -0.95868 -0.24609 0.48413 0.18178 0.475 -0.22305 0.30064 0.43496 -0.3605 0.20245 -0.52594 -0.34708 0.0075873
with -0.43608 0.39104 0.51657 -0.13861 0.2029 0.50723 -0.012544 0.22948 -0.6316 0.21199 -0.018043 -0.39364 0.74164 0.30221 0.51792 -0.25191 0.25373 -0.65184
he 0.1225 -0.058833 0.23658 -0.28877 -0.028181 0.31524 0.070229 0.16447 -0.027623 0.25214 0.21174 -0.059674 0.36133 0.13607 0.18755 -0.1487 0.31315 0.13368 -
as -0.32721 0.096446 0.34244 -0.44327 0.30535 -0.042016 -0.071235 -0.31036 -0.22557 -0.181 -0.29088 -0.61542 0.29751 0.030491 0.41504 -0.51489 0.68628 -0.020
it -0.30664 0.16821 0.98511 -0.33606 -0.2416 0.16186 -0.053496 0.4301 0.57342 -0.071569 0.36101 0.26729 0.27789 -0.072268 0.13838 -0.26714 0.12999 0.22949 -0
by -0.20875 -0.1174 0.26478 -0.28339 0.19584 0.7446 -0.03887 0.028499 -0.44252 -0.30426 0.27133 -0.51907 0.52183 -0.76648 0.28043 -0.48344 -0.15626 -0.49705
at 0.1766 0.093851 0.24351 0.44313 -0.39037 0.12524 -0.19918 0.59855 -0.82035 0.28006 0.54231 0.023079 0.12837 -0.044489 0.3837 -0.75659 0.40254 -0.4462 -0.8
( 0.19247 0.36617 0.52301 -0.79857 -0.2592 0.18267 0.19564 0.83148 -0.67636 -0.84648 1.4429 -0.84978 -0.023986 1.328 0.74061 0.039546 0.61659 -0.075604 -0.59
) -0.13797 0.27084 0.84036 -0.45668 -0.49429 0.35777 0.077772 0.42481 0.0076481 -0.50942 1.4008 -0.79993 0.053011 1.2054 0.3783 0.0842 0.91317 -0.35173 -0.30
from 0.30731 0.24737 0.68231 -0.52367 0.44053 0.42044 0.0002514 0.15265 -0.61363 0.22631 0.083071 0.070425 0.017683 0.56807 1.0067 -0.46206 0.44524 -0.50984
his 0.12883 -0.82209 0.27438 -0.069014 0.17989 0.72605 -0.15112 0.0085541 -0.95122 0.77243 -0.28375 0.28329 0.14825 -0.01223 -0.019267 -0.03446 0.31506 -0.16
'' 0.16478 0.17071 0.62111 -1.2101 -0.84063 0.21893 0.48123 -0.15044 0.36701 -0.20857 -0.23385 0.019356 -0.045098 0.18001 0.11995 -0.25622 -0.026299 0.28473
`` 0.092672 0.20241 0.69394 -0.50775 -0.097297 0.045522 -0.14156 0.30736 -0.35448 -0.20612 -0.21092 -0.0026685 -0.11537 0.052913 0.02908 0.0067036 0.47268 0.
an -0.4214 -0.18797 0.46241 -0.17605 0.36212 0.36701 0.27924 0.14634 -0.054227 0.45834 0.065416 -0.33725 0.067505 -0.36316 0.50302 -0.010361 0.72826 -0.17564
be -0.46953 0.38432 0.54833 -0.63401 0.010133 0.11364 0.10612 0.58529 0.032302 -0.12274 0.030265 0.52662 1.0398 -0.082143 0.19118 -0.83784 0.50763 0.44488 -0
has 0.093736 0.56152 0.48364 -0.45987 0.56067 -0.1694 0.018687 0.45529 0.065615 0.25181 -0.14251 0.10532 0.77865 0.1428 -0.08114 -0.069555 0.32433 0.019611 -
are -0.51533 0.83186 0.22457 -0.73865 0.18718 0.26021 -0.42564 0.67121 -0.31084 -0.61275 0.089526 -0.24011 1.1878 0.67609 -0.022885 -0.92533 0.071174 0.38837
have 0.15711 0.65606 0.0021149 -0.65144 -0.28427 -0.20369 -0.077596 0.40798 -0.03447 -0.1639 -0.21597 0.34178 1.196 0.33639 -0.21076 -0.56015 0.1507 0.34912
```

Chapter 9

References

1. A Survey Automatic Text Summarization, http://pressacademia.org/archives/pap/v5/29.pdf
2. GloVe, https://www.aclweb.org/anthology/D14-1162.pdf
3. NER, https://www.aclweb.org/anthology/D11-1141.pdf
4. SpaCy Language Processing, https://spacy.io/usage/processing-pipelines
5. GloVe: Global Vectors for Word Representation Jeffrey Pennington, Richard Socher, Christopher D. Manning Computer Science Department, Stanford University, Stanford, CA 94305
6. Learning similarity with cosine similarity ensemble, Information Sciences, volume 307, https://www.sciencedirect.com/science/article/abs/pii/S0020025515001243
7. PageRank Algorithm, https://ieeexplore.ieee.org/abstract/document/1344743
8. NetworkX, https://www.osti.gov/biblio/960616, Exploring network structure, dynamics, and function using network
9. https://networkx.org/documentation/stable/index.html