**Individual Report on Project: TEXT SUMMARISATION**

GROUP NUMBER: 12

Name: PREKSHA RAJ SHIMOGA BASAVARAJA

Email address: rajshimo@usc.edu

Rest of my group: ISHA ARORA, SHIVALIK NARAD, AKSHAY CHOPRA

URL of GitHub repository: <https://github.com/iarora7/lex-rank>

**1. Project Overview**

Too long; didn't read (tl;dr)

Most of the news articles are too long and provide in depth information of the context. These long articles and blog posts which interests a person to read are very time consuming and people tend to lose interest midway of the article due to detailed description.

In such cases the normal tendency is to start skimming through the contents and just picking up certain sentences.

So we proposed a method to provide a summary of the article because sometimes you just need a gist and a quick view of the article to understand what is the entire article about. So hence we have worked to implement algorithms for text summarization.

Automatic document summarization can be done in two ways. We have chosen extractive summarization, which identifies important words/phrases from the source document and extracts them and creates the summary.

We have worked on two prime algorithms:

Text Rank

Lex Rank

**DATA COLLECTION**

Data is collected by the following sources:

<https://github.com/kylehg/summarizer/tree/master/input>

The corpus is a collection of news articles consolidated from New York times and APW papers.

The corpus has 67 folders of articles and each folder containing 10 articles each. Each text file contains the text of a news article.

We also used another dataset called Timeline 17 to test our summary generation.

[http://www.l3s.de/~gtran/timeline/Timeline17.zip](http://www.google.com/url?q=http%3A%2F%2Fwww.l3s.de%2F~gtran%2Ftimeline%2FTimeline17.zip&sa=D&sntz=1&usg=AFQjCNE4IVx_o9rJKTJ86UWSYZl0WaFHtw)

Since it is difficult to find manually summarized text of any particular corpus, we have not used any input data which already have the associated summaries with the articles and hence used the online resource called SUMY to get reference summary. And also we have not annotated any of our data.

**TEXT RANK**

TextRank is a graph-based ranking algorithm which runs PageRank on a graph of sentences. For key phrase extraction, it builds a graph using each line as a vertex. And edges are going to be lexical similarity between these sentences.

Unlike PageRank, the edges are typically undirected and can be weighted to reflect a degree of similarity. Once the graph is constructed, it is used to form a stochastic matrix, combined with a damping factor to find the rank of each node(sentence). The top ranked sentences are extracted as the summary of the text.

We have implemented 2 approaches of Textrank:

1.Sentence Extractive summary: Each sentence is marked as a node and the similarity between these sentences based on cosine similarity is marked as the weight of the edges between the nodes. Then Pagerank is applied on this graph to find top ranked sentences as the summary.

2.Keyword Extractive summary: Each word in each sentence is marked as the nodes and the similarity between each word is determined based on Levenstein’s edit distance as these similarities are marked as the weights of the edges. Then pagerank is applied to this graph to find top ranked words as the summary.

**LEX RANK**

Text Rank implements page rank algorithm but it doesn’t take into consideration the important of a word in each sentence.

Lex rank features:

• Word importance taken into consideration while ranking (tf-idf scores)

• Calculate tf-idf metrics for words in the document corpus

• Rank the sentences using the above metrics based on the words that it contains

• Produce summary based on the sentence ranking

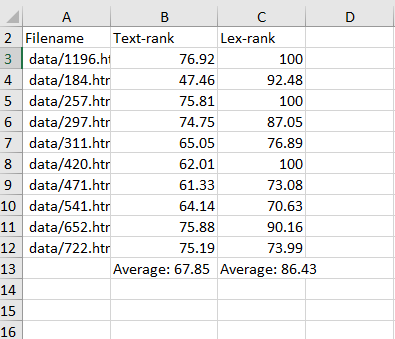
• Lex-Rank algorithm can work for multi-document summarization

Another Algorithm called LSA was also implemented for text summarization to generate reference summary to compare the different generated summary. It gives importance to every word in each sentence.

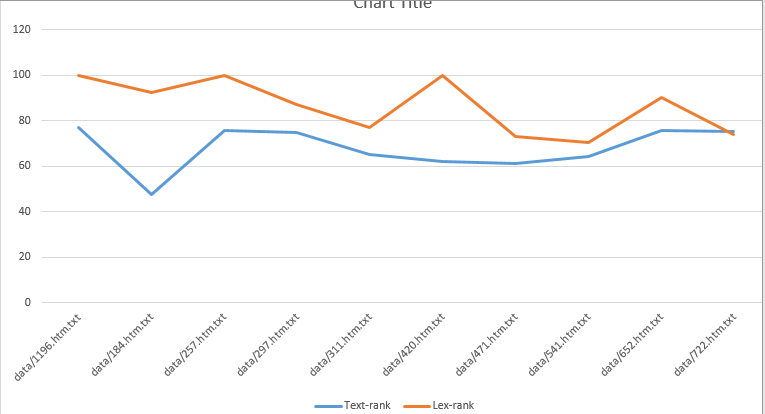
**SUMMARY EVAUATION**

We have taken into consideration the python package called SUMY which generates summary of documents. The SUMY package has a set of implemented algorithms like Text rank, Lexrank, Luhn and LSA. We reference these algorithms by making an API call to the SUMY Package and got the reference summary from the SUMY Text Rank and SUMY Lex Rank.

The summary obtained from the Text rank implementation is compared to a reference summary that is generated by the SUMY package to determine the accuracy of our implementation with respect to the reference. The similar approach is done for Lex Rank, our Lex Rank summary has been compared to the SUMY implementation of Lex Rank. The evaluation is done using the metric called Rouge which gives a graph of the comparison and also gives the stats of comparison. Based on the evaluation the Textrank gives similarity of 67.85% with the reference summary and Lex rank gives better similarity with reference summary of 86.43%



Evaluation result table



**2. My Primary Responsibility**

I was responsible for the following Tasks:

1.Research and preprocessing of Data

2.Implementation of Textrank- Sentence based Extraction

3. LSA text summarization.

4.Extracting Summaries from SUMY package to create reference summary for comparison

I did research on the Extractive Text summarization and read the papers listed in the references for understanding the Text rank implementation.

I also collected the data from the internet which is a set of consolidated news articles from New York time and APW.

1. **PRE PROCESSING OF DATA**

The Data that I collected was a folder of text files which needed to be processed to be used as inputs for the algorithm.

The steps of Preprocessing followed are:

1. Remove the trailing white spaces
2. Split the article text into lines(sentences)
3. Sentence extraction from paragraphs
4. Tokenization of sentences to words
5. Stemming the words
6. Stop words elimination
7. Remove the special symbols and digits and from the lines

I wrote the script called preprocess.py which preprocessed all the data and returned a list of sentences for the next step of Textrank algorithm-approach 1.

1. **TEXTRANK IMPLMENTATION**

TextRank is a graph-based ranking algorithm which runs PageRank on a graph of sentences.

The steps that was followed in the code are:

1. After the article was preprocessed and returned a list of sentences. A class for Node is created and initialized with a sentence at each node.
2. A graph class is created to accommodate the nodes and connect these nodes.
3. The nodes are added to the graph. Sentence mark each node. Each node has a score and a sentence.
4. The set\_graph() methods is called to set the nodes and set the edges between them. This forms an undirected weighted graph.
5. Then from each node to every other node I found the similarity between a pair of nodes by calling the get\_similarity() on a pair of sentences.

Eg: get\_similarity(sentence1, sentence2)

1. This get\_similarity method uses the Cosine Similarity to compute the similarity between each pair of sensetences(each pair of nodes) based on the number of similar words in both sentences.
2. The sentences are first converted into vectors and mapped on a graph and the cosine of angle between these vectors are determined and then the vectors are converted back to text and the cosine angle is marked as the cosine similarity.
3. These similarities are mapped as the weights of the edges between similar nodes.
4. Then I implement Pagerank on this graph to determine the rank of each sentence. Max iteration:1000
5. The Pagerank calculated the rank of each node first by counting all the nodes that are linked to the current node and then summing the score of each inked node including the weighted edges to those related nodes.
6. Then all nodes are sorted in the reverse order based on their Pagerank score to give the sentences with top most ranks. These are sentences are the most important in the article and hence would be a part of the summary.
7. We select the number of lines that we would want our summary in. So based on the sentence count the top ranked sentences are returned as the summary.

EVALUATION: The generated summaries are compared with the reference summaries from SUMY implementation of Textrank using the Rogue metric. It gave a similarity of 67.85% because it is based on the sentence as a whole and each word is not given importance.

1. **LSA TEXT SUMMARIZATION**

Latent Semantic Analysis is a technique based on analyzing relationship between a set of sentences and terms that they contain.

The steps followed were:

1. Split the text into sentences.
2. Created a dictionary of all the words in the sentences after removing punctuations, stop words and stemming.
3. Created a sparse matrix of shape |unique words|\*|sentences| where cells contained number of occurrences of the words(rows) in sentences(columns) filled with zeros.
4. Computed the TF metric for each sentence(column) in the matrix. Updated the matrix with count of each word in each sentence.
5. Normalized each word with tf-idf
6. Transformed each sentence to a vector in multidimensional conceptual space using Single Value Decomposition.
7. Computed ranks of each sentence based on absolute values of their sentence vectors and picked the top ranked sentences as the summary.
8. These we the additional project work that I worked on to provide the reference summary by another approach named LSA summarization.

**D.REFERRENCE SUMMARIES**

Since we do not have reference summaries in our input data to evaluate our results against, we have used the SUMY API to get reference summaries for evaluation.

The SUMY package contains simple evaluation framework for text summaries.

1. I have written a script to use SUMY package as a library to get the summaries from already implemented algorithms like TextRank and LexRank in the package.
2. I created a method to pass the number of sentences required for our summary, then the parser is called on the input file.
3. Then all the sentences from the input files are stemmed using the stemmer method defined in the SUMY package.
4. Stop words are removed from all the sentences
5. Then these sentences are passed to the SUMY Textrank Summarizer to get the top ranked sentences from the algorithm.
6. These sentences are then added to a list in order to process for evaluation.

So summary.py calls the SUMY Textrank and SUMY Lexrank summarizers and return the summary sentences to be considered as the reference summary for Rogue evaluation.

**3. Other Project Work**

A. I also researched on Lex Rank and LSA algorithm to compare which approach is better to extract a summary from the text.

B. I prepared the slides for the presentation. Included the content and summarized all the work done on all algorithm step by step for easy deliver to the class.

C. I also worked on implementing the LSA algorithm. This algorithm is based on term frequency in each sentence and such sentences are given more importance with greater tf-idf score.

D. I tried to find the similarity between each sentences in Textrank implementation with Levenstein’s similarity. But that did not give a good result So hence used Cosine similarity.

**4. Online Resources**

The online resource that we used are SUMY python package that we called to retrieve a set of summaries from the Textrank and Lex rank implementations.

1. These were used as the reference summaries to compare our results to.

SUMY: <https://pypi.python.org/pypi/sumy>

1. Also followed video tutorials from youtube to learn more Textrank implementation.

Youtube Link: <https://www.youtube.com/watch?v=GX4RTIIuxy8>

1. Data for running our text summarization algorithm was used from the following sources:

[http://www.l3s.de/~gtran/timeline/Timeline17.zip](http://www.google.com/url?q=http%3A%2F%2Fwww.l3s.de%2F~gtran%2Ftimeline%2FTimeline17.zip&sa=D&sntz=1&usg=AFQjCNE4IVx_o9rJKTJ86UWSYZl0WaFHtw)

and

<https://github.com/kylehg/summarizer/tree/master/input>

1. To create graph we used the following resource of networkx: <https://networkx.github.io/>
2. To computer similarity we used the following packages: <https://pypi.python.org/pypi/editdistance>

**5. References**

[1] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. Association for Computational Linguistics, 2004

[2] <https://github.com/miso-belica/sumy>

[3] <http://megaslides.com/doc/3058979/graph-based-nlp>

[4] <https://joshbohde.com/blog/document-summarization>

[5] <http://www.slideshare.net/innovationengineering/lsa-getting-started>

[6] <https://www.youtube.com/watch?v=GX4RTIIuxy8>

[7] <https://www.youtube.com/watch?v=GX4RTIIuxy8>