# An Automatic Graph Based Text Summarizer

**Submitted in partial fulfillment of the requirements
For the award of the degree of**

**Bachelor of Technology**

**By**

**Shikhar Sharma
Nitin Agrawal
Prashant Sinha
Rahul Yadav**

**Under the guidance of**

**Dr Shobha Bagai**

**Cluster Innovation Centre
University of Delhi
Delhi 110007**

**November 2014**

## Abstract

Automatic text summarization is an important topic in the field of Natural Language Processing. It is the process of reducing a text document to a smaller format retaining all the key pieces of information that the document possesses. Formation of a coherent summary is a big challenge in the field of natural language processing. This paper aims to present an extraction based automatic text summarisation algorithm. The method constructs a weighted graph of the original text and by assuming each sentence to be a vertex. The weighted edge is determined by using a suitable distortion measure which analyses the semantic relation between two sentences. A ranking algorithm is used to compute the most important sentences in the text and that should be present in the summary. Using these techniques on a wide variety of data sets, Promising results were obtained that compared well to other successful models.

## Introduction

Automatic text summarization (ATS) is a process which enables a computer to summarize data/information automatically . With massive growth in information/knowledge summarization has become of more important for enlisting important parts of a big corpus.It provides a non redundant text from the original one.The amount of information available today is tremendous and the problem of finding the relevant pieces and making sense of these is becoming more and more essential. Nowadays, a great deal of information comes from the Internet in a textual form.Text Summarization helps in various kinds of analysis and forms a base for different Natural Language Processing Algorithms.ATS has a wide range of applications such as summarization of news/articles , in search engines to present compressed/accurate results , language translation , email thread summarization etc.Text Summarization is broadly divided into two categories.The first category is text abstraction which involves parsing the text on semantic `basis followed by a formal representation.This is followed by re-interpretation of the text into a differnet non redundant text which is a summarized version of the origanl text.The second category is text extraction wherein the process involves identification of most important/relevant aspects of text using statistical information techniques.We use the text extraction based method to summarize documents because of its less computationally intensive nature , ease of scalability and availability of various techniques for analysis.The algorithm basically involves preprocessing the words on corpus followed bygraph based text ranking based on relevance.It is an unsupervised graph based ranking algorithm . It takes in account Keywords, Frequency,Relationship between sentences and Distortion measure.The graph can briefly be described as follows.The graph is connected and undirected representing the text.Each sentence is represented by a vertex.Distortion measure / Distinctivity of sentences is thresholded to decide if an edge exists and its corresponding weight.The approach is a graph based extractive summarization . The sentences are splitted on punctuation marks followed by removal of stopwords. Distortion measure is used to form the graph as explained earlier.Distortion measure is based on Squared Error which is a statistical way of quantifying difference between values.
Finally the Text Ranking algorithm is used to check relevance and summarize the text.

## Related Work

The subfield of summarization has been investigated by the NLP community for nearly the last half century. Radev et al. [3] define a summary as “a text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that”.Earliest instances of research on summarizing scientific documents proposed paradigms for extracting salient sentences from text using features like word and phrase frequency [4], position in the text [5] and key phrases (Edmundson, 1969)[6].The study of text summarization [1] which proposed combination of sentence extraction and trainable classifier using Support Vector Machine provided good summarization but required work to be done to improve upon readability. [2] presents a sentence reduction system for automatically removing extraneous phrases from sentences that are extracted from a document for summarization purpose.
Luhn [4] stressed upon the significance of frequency of word in a text. The words were stemmed to their root forms, and stop words were deleted. Further,frequency was used to sort the words in a decreasing order. On a sentence level, a significance factor was derived that reflects the
number of occurrences of significant words within a sentence, and the linear distance between them due to the intervention of non-significant words. All sentences are ranked in order of their significance factor.Baxendale [5] provides early insight on a particular feature helpful in finding salient parts of documents: the sentence position. Towards this goal, the author examined 200 paragraphs to find that in 85% of the paragraphs the topic sentence came as the first one and in 7% of the time it was the last sentence. Thus, position feature is also important for text summarization.Edmundson [6] was credit for the development of a typical structure for an extractive summarization experiment.The two features of word frequency and positional importance were incorporated from the previous two works. Two other features were used: the presence of cue words (presence of words like significant, or hardly), and the skeleton of the document (whether the sentence is a title or heading). Weights were attached to each of these features manually to score each sentence.This methodology proved to be quite accurate in text summarization.

## Methodology

### Preprocessing

The preprocessing steps performed in automatic summarization are word stemming, stop words removal, text segmentation and query expansion. These steps are also used in various other NLP tasks such as document retrieval, information extraction and machine translation.

### Stemming

Stemming is the process of reducing the inflected forms of a word to a root form.For example words kicks , kicked , kicking all have the same root form “kick”.This enables in non redundant representation of almost same semantic.It also contributes in reduction of overall feature space. The way to perform stemming is to use a comprehensive list of inflected forms for all words in a language.Another way is to use a set of predefined language-specific rules to transform a word into its baseform. Porter stemmer is, perhaps, the most well-known algorithm of this kind. It is based on rules for suffix elimination,In certain cases it may be more rough than required eg.organizational is stemmed to organ .The viable solution is to combine the two described approaches by introducing the list of frequently used exceptions to a rule based stemmer

### Stop Words Removal

Stop words are high-frequency words of a language that don’t carry any particular information on their own. Such words are removed at the preprocessing phase to reduce the number of features. Closed class words such as pronouns, articles , prepositions and conjunctions are often included in stop words lists.

## Mathematical Analysis

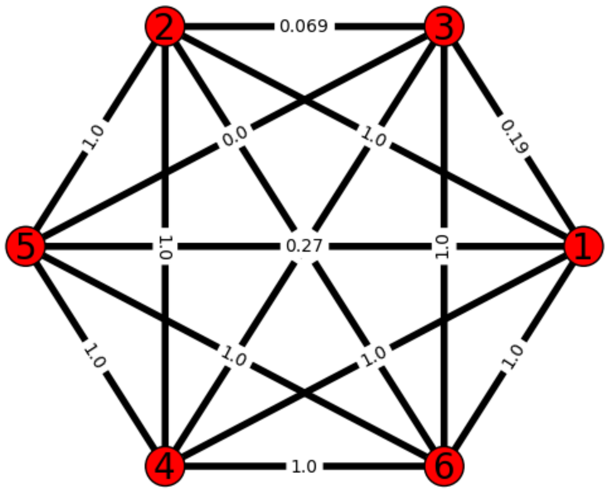
The text that is taken as input is filtered to remove any non usable characters as the algorithm concerns itself with only ASCII characters. All other characters have no significance. The input text is then split in the form of its sentences. These sentences are stemmed and have their stopwords removed. The processed sentences are now viable for a suitable ranking algorithm that can signify each sentences' importance to the entire paragraph. The algorithm suggested in this paper is a modification of the famous Page Rank algorithm (Page and Brin, 1998) [7] which was used to find the importance of a weblink in terms of its connectivity to other pages. Unlike other graph based ranking algorithms, this takes into account both the incoming and outgoing connections of a single node and then proceeds to give a set of scores governed by the pagerank equation. This equation has been modified to suit the needs of assigning scores to sentences rather than links. The scores allocated to each sentence is now governed by the weighted graph textrank equation

$$TR(V_i) = 1 - d + d\left(\sum_{V_j \in I} \frac{TR(V_j)}{Out(V_j)}\right)$$

where TR() denotes the textrank of any given sentence and Out() represents the outdegree of that sentence, d is a parameter set between 0 and 1. This parameter known as the damping factor is used to add weight to the impact that adjacents of a node have on its rank. Generally, it is taken to be 0.85. The equation runs on a set of random weights assigned initially to each edge and iterates a number of times till convergence is reached. The final values denote the rank of each node. The algorithm used in this paper takes the input as the distortion measure between each sentence as the weights of the edges, rather than taking random values. The distortion measure signifies the semantic difference between any two sentences. Every two sentences are examined by a distortion measure representing the semantic relation between them. This then represents the weight of the edge between those two nodes. The distortion measure used in this model is based on the 'Squared Error' which statistically quantifies the difference between two sentences. This is done by squaring and adding the frequency of the the words that are not common between the sentences. In case a word is common between the two sentences, its frequency is calculated in the second sentence and this is subtracted from the score of that word. The frequency is squared and added to a sum. The second sentence is then checked for its not common words with the first sentence. If the word is not common, the score is squared and added to the sum, and the number of not common words is incremented by 1. The final distortion measure can now be calculated using the equation

$$Distortion = \frac{Sum}{not\ common\ words}$$

For a simple data set of 6 sentences, the graph representing the distortion measure generated by the program is as follows



The adjacency matrix of this distortion matrix graph is then passed to the textrank equation which computes the rank of each sentence. Once the ranks have been assigned, the highest ranked sentences taken in order form the summary. The number of sentences used in the summary depend on a compression factor which takes a few of the high ranked sentences.

## Pseudocode

```
OPEN and READ file
filterASCII(text)                                        //Filter non ASCII characters
sentences = regex.split('(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', text)
FOR j in 0,length(sentences)                      //Split sentences
APPEND ([i for i in sentences[j].split() if i not in stop])
punctuationRemove(sentences)
PorterStemmer(sentences)                                 //Apply Stemmer
ComputeFrequency(word in sentences)
for i in range(0,len(sentences)):                        //Compute Adjacency
    for j in range(i+1,len(sentences)):
        adjacency[i,j] = adjacency[j,i] = find_distortion(i,j)
adjacency=1-adjacency/float(adjacency.max())
for i in range(0,10):                                    // 10 iterations for convergence
    text_rank = find_rank(text_rank,adjacency,0.85)
print sentences
```

## Implementation of Algorithm

The methodology was implemented in python with the help of NLTK module.

### Reading and Pre-Processing

The file is opened , read and natural language processing based preprocessing is done as follows

1) Replacement of new line character by spaces
2) Splitting on symbols
3) Removal of stopwords
4) Punctuation Removal
5) Stemming using Porter Stemmer

### Frequency Computation

A dictionary based on key-value pair is formed with word as the key and corresponding frequency as its value.This frequency is used in later part of the algorithm for rank computation purposes.

### Adjacency Matrix and Normalization

The distortion measure is a marker of dissimilarity between pairs of sentences.An adjacency matrix is formed with nodes being the sentence labels and edges represnting a similarity parameter which is obtained from the distortion measure.For conversion into terms of similarity the distortion measure is normalized and is subtracted from a value of 1. A value of one signifies perfect similarity and a value of zero represents perfect dissimilarity..However, a thresholding is used for representation in terms of edges of a graph.

### Rank Computation via distortion measure

The text ranking algorithm is applied which uses the graph (vertices , distortion measure) to rank sentences . The ranking is obtained and are sorted accordingly to get the summarized version of the text according to a given compression factor.

## Results & Conclusion

The summarizer is tested on different paragraphs with different compression factors. A fairly accurate summary was obtained in all the cases. The summaries contained most of the important sentences that were essential to the original text. The algorithm implemented performs well in comparison to other successful models used in summarisation. For a small data set of the following sentences

- I have a friend named Ken in England.
- We often write to each other.
- My letters are very short.
- It is still hard for me to write in English.
- I received a letter from Ken yesterday.
- In his letter he mentioned that is waiting to visit me in England.

The compression factor was kept as 1.5 and the summary that was generated is as follows

- We often write to each other.
- My letters are very short.
- It is still hard for me to write in English.

As can be seen, the important aspects of the paragraph are retained.

## References

[1] Kai ISHIKAWA et. al.; “Trainable Automatic Text Summarization Using Segmentation of Sentence”; Multimedia Research Laboratories,NEC Corporation 4-1-1 Miyazaki Miyamae-ku Kawasaki-shi Kanagawa 216-8555, 2003.
[2] Hongyan Jing; “Sentence Reduction for Automatic Text Summarization”; Proceedings of the sixth conference on Applied natural language processing, Seattle, Washington, pp.310 – 315, 2000.
[3].Radev, D. R., Hovy, E., and McKeown, K. (2002). Introduction to the special issue on summarization. Computational Linguistics., 28(4):399–408. [1, 2]
[4].Luhn, H. P. (1958). The automatic creation of literature abstracts. IBM Journal of Research Development, 2(2):159–165. [2, 3, 6, 8]
[5].Baxendale, P. (1958). Machine-made index for technical literature - an experiment. IBM Journal of Research Development, 2(4):354–361. [2, 3, 5]
[6].Edmundson, H. P. (1969). New methods in automatic extracting. Journal of the ACM, 16(2):264–285. [2, 3, 4]
[7].Page, L., and Brin, S. 1998. The anatomy of a large-scale hypertextual web search engine. In Computer Networks and ISDN systems.

## Appendix

### Python Code

```
import re
import nltk
import string
from nltk.stem.porter import *
import copy
import math
import numpy

##Function for distortion computation b/w any two indices
def find_distortion(ind1,ind2)  :
    s0=copy.deepcopy(sentences_stem[ind1])
    s1=copy.deepcopy(sentences_stem[ind2])

    not_common=0;
    sums=0
    for word in s0:
        occurences=s1.count(word)
        sums=math.pow((freq_dict[word]-occurences),2)+sums
        if(occurences==0):
            not_common=not_common+1

    for word in s1:
        if word not in s0:
            sums=sums+math.pow(freq_dict[word],2)
            not_common=not_common+1

    if(not_common==0):
        not_common=1
    distortion=sums/float(not_common)

    return distortion/float(not_common)

##Function for computation of weighted page rank
def find_rank(ranks,adjacency,d):
    size_adjacency=adjacency.shape
    ranks = numpy.array(ranks)

    for node in range(0,size_adjacency[0]):
        ranks[node] = 0
        for to_node in range(0,size_adjacency[0]):
            ranks[node] = ranks[to_node]*adjacency[node,to_node]/adjacency[to_node].sum()
        ranks[node] = ranks[node]*d+(1-d)
    return ranks
```

```
text = ''.join(open('sherlock.txt').readlines())


#text=unicode(text)

text=text.replace('\n',' ')
original_sentences=re.split('(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', text)
sentences = re.split('(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', text.lower())
adjacency = numpy.zeros((len(sentences),len(sentences)))
text_rank = numpy.random.rand(len(sentences))
compression_factor = 5
#gives compression extent

##removes stop words
from nltk.corpus import stopwords
stop = stopwords.words('english')
sentences_temp=list();


for j in range(0,len(sentences)):
    sentences_temp.append([i for i in sentences[j].split() if i not in stop])

#preprocess using nlp
##remove punction
for s_in,sentence in enumerate(sentences_temp):
    for w_in,words in enumerate(sentence):
        sentences_temp[s_in][w_in]=words.translate(None,string.punctuation)

freq_dict=dict()
sentences_stem=copy.deepcopy(sentences_temp)

##Stemming using porter stemmer
stemmer = PorterStemmer()
for s_in,sentence in enumerate(sentences_temp):
    for w_in,words in enumerate(sentence):
        sentences_stem[s_in][w_in]=stemmer.stem(words)

# frequency computation for ranking
for sentence in sentences_stem:
    for word in sentence:
        freq_dict[word]=0
for sentence in sentences_stem:
    for word in sentence:
        freq_dict[word]=freq_dict[word]+1

##adjacency matrix computation
```

```
for i in range(0,len(sentences)):
    for j in range(i+1,len(sentences)):
        adjacency[i,j] = adjacency[j,i] = find_distortion(i,j)
adjacency[adjacency<0.5]=0
#normalisation of waits and conversion of distortion score to similarity score
adjacency=1-adjacency/float(adjacency.max())
#adj mat : vertices versus distort is further normalized and converted to similarity measure

#print adjacency5

for i in range(0,10):
    text_rank = find_rank(text_rank,adjacency,0.85)


print text_rank
sorted_indices=(-text_rank).argsort()
ordered_sorted=numpy.sort(sorted_indices[1:len(sorted_indices)/compression_factor])
for i in range(0,len(ordered_sorted)):
    print original_sentences[ordered_sorted[i]]
```