Text Summarization Techniques using Linear Algebra

May 2020

1. Abstract

Automatic summarization[1] is the process of shortening a set of data computationally, to create a subset (a summary) that represents the most important or relevant information within the original content. In addition to text, images and videos can also be summarized. Text summarization finds the most informative sentences in a document; image summarization finds the most representative images within an image collection; video summarization extracts the most important frames from the video content.

Text summarization[2] refers to the technique of shortening long pieces of text. The intention is to create a coherent and fluent summary having only the main points outlined in the document. Automatic text summarization is a common problem in machine learning and natural language processing. Machine learning models are usually trained to understand documents and distil the useful information before outputting the required summarized texts.

Propelled by the modern technological innovations, data is to this century what oil was to the previous one. Today, our world is parachuted by the gathering and dissemination of huge amounts of data. In fact, the International Data Corporation (IDC)[3] projects that the total amount of digital data circulating annually around the world would sprout from 4.4 zettabytes in 2013 to hit 180 zettabytes in 2025. With such a big amount of data circulating in the digital space, there is need to develop machine learning algorithms that can automatically shorten longer texts and deliver accurate summaries that can fluently pass the intended messages. Furthermore, applying text summarization reduces reading time, accelerates the process of researching for information, and increases the amount of information that can fit in an area.

2. Keywords

Machine Learning, Natural Language Processing, Summary, Automated Text Summarization, Text Mining, Information Retrieval, Extractive Approach, Abstractive Approach, Weights, Topic Modelling, LSA, TF-IDF

3. Introduction

There are multiple approaches to text summarization but they all can be broadly classified into two categories – extractive and abstractive text summarization techniques.

1. Extractive Summarization: These methods rely on extracting several parts, such as phrases and sentences, from a piece of text and stack them together to create a summary. Therefore, identifying the right sentences for summarization is of utmost importance in an extractive method.

2. Abstractive Summarization: These methods use advanced NLP techniques such as Deep Learning and Neural Networks to generate an entirely new summary. Some parts of this summary may not even appear in the original text.

Although modern state of the art abstractive text summarization techniques is the current way to go, extractive text summarization techniques are still used today. Extractive text summarization techniques rely on concepts of linear algebra such as vector spaces and topic modelling approaches to rank sentences based on their relevance. These techniques work much faster than Deep Learning techniques and are sometimes more accurate too since there is no possibility of creating factual inaccuracies.

We will be comparing two popular approaches to extractive text summarization, namely TF-IDF score based and topic model based and compare the summaries produced by both algorithms. A good summary will contain all the keywords present in the original text document and less irrelevant features. We will be using Python's Natural Language Toolkit's[4] collection of books from Project Gutenberg[5] to attempt to summarise these books using both algorithms and compare the keyword retention factor in both approaches.

4. Summarization with TF-IDF Scores

4.1 Linear Algebra Background

Representation. *In natural language processing, documents are represented as a term-document matrix, where the relation between term* **i** *and document* **j** *is given by matrix* **(i, j)**.

$$\begin{array}{cc}
 & \begin{array}{ccccc} & & \mathbf{d}_j & & \\ & & \downarrow & & \end{array} \\
\mathbf{t}_i^T \rightarrow & \begin{bmatrix} x_{1,1} & \cdots & x_{1,j} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i,1} & \cdots & x_{i,j} & \cdots & x_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,j} & \cdots & x_{m,n} \end{bmatrix}
\end{array}$$

t_i stands for term **i** *and d_j stands for document* **j**

4.2 Mathematical Background

TF-IDF creates a vector space out of our dataset of documents and the weights or scores give us the relevance of a term in a document. The TF-IDF algorithm is made of 2 algorithms multiplied together. A high weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents.

Assume we have a matrix term-sentence **X** *with TF-IDF scores.*

The matrix **X** *will look like this.*

$$\begin{array}{cc} & \begin{array}{c}\mathbf{d}_j\\ \downarrow\end{array} \\ \mathbf{t}_i^T \rightarrow & \begin{bmatrix} x_{1,1} & \cdots & x_{1,j} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i,1} & \cdots & x_{i,j} & \cdots & x_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,j} & \cdots & x_{m,n} \end{bmatrix} \end{array}$$

t_i stands for term **i** *and d_j stands for sentence* **j**

We have another **1xj** *matrix* **S** *that holds the final aggregated scores of each sentence. To find the values of this vector, we find the sum* $\mathbf{s}_\mathbf{j}$ *of the elements for every column vector* $\mathbf{d}_\mathbf{j}$ *and store the pair* **(j,** $\mathbf{s}_\mathbf{j}$**)** *in* $\mathbf{S}_\mathbf{j}$.

We then rank these sentences based on their aggregate scores $\mathbf{s}_\mathbf{j}$ *and sort them in non-increasing order. We discard the sentences with low scores and extract the top* **k** *sentences with scores greater than the average and then rank them again in their order of precedence* **j**. *We combine these* **k** *sentences together to return the summary of the entire document.*

4.3 Implementation

We implemented TF-IDF score-based summarization by first tokenizing our document at the sentence level and then converting each sentence into a vector space model. We then implemented a function to find the score of a sentence from scratch and used these scores to rank the sentences. We finally implemented another function to sort these sentences by order and return the summarised version of the document.

4.4 Algorithm to Generate Summary

We need to first obtain the score for every sentence in our document after converting it into a vector space model. This can be done in a simple iterative matter.

Part 1: Finding the score of each sentence in the document

```
def sentence_score(X):

    vectorizer = TfidfVectorizer(smooth_idf=True)

    X = vectorizer.fit_transform(X)

    X = X.toarray()

    ss = [(pos,sum(sentence)) for pos, sentence in enumerate(X)]

    return ss
```

After finding the scores, we need to rank them and return the top scoring sentences which will form our summary.

Part 2.1: Generating summary based on sentence scores

```
def summarize(sentences,scores):
    agg = 0
    for i in scores:
        agg+=i[1]
    avg = agg/len(scores)
    scores = [i for i in scores if i[1]>=avg]
    scores.sort(key = lambda x:x[0])
    doc = [sentences[pair[0]] for pair in scores]
    return " ".join(doc)
```

We can also choose to set a threshold on the number of sentences, say **k** and rank the top **k** sentences and return the top scoring sentences which will form our summary.

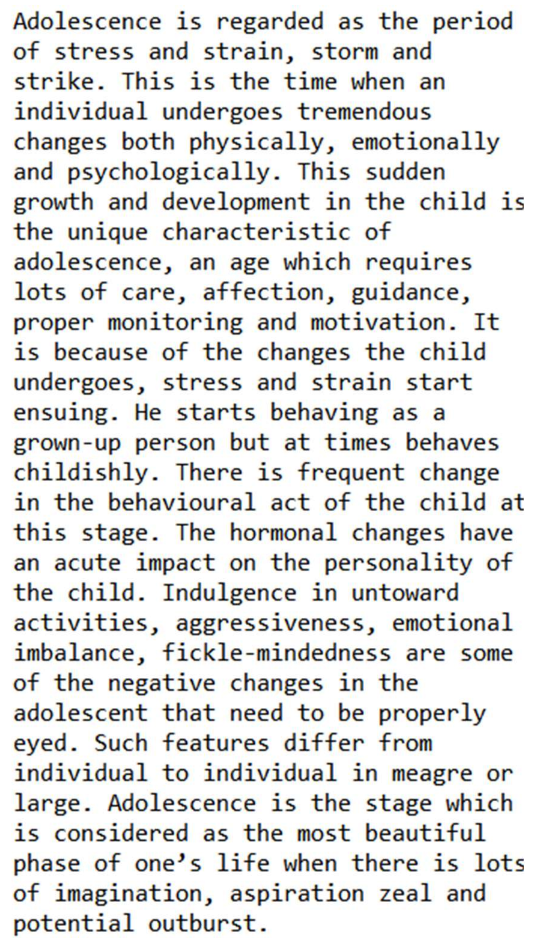
Part 2.2: Generating summary with a threshold on number of sentences

```
def summarize(k,sentences,scores):
    scores.sort(key = lambda x:x[1], reverse = True)
    scores = scores[:k]
    scores.sort(key = lambda x:x[0])
    doc = [sentences[pair[0]] for pair in scores]
    return " ".join(doc)
```

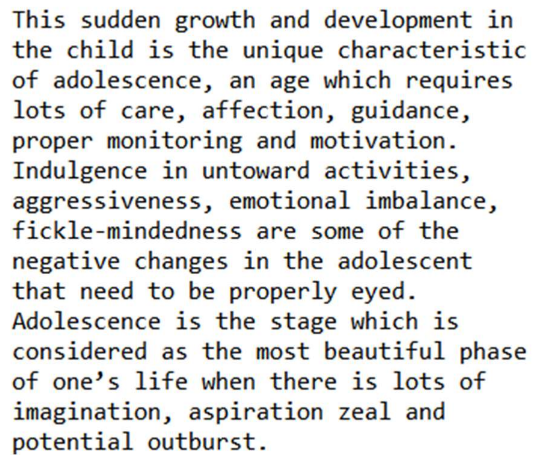
4.5 Final Implementation

We implemented the methodology from scratch to pre-process the original document and return the sentence scores after scoring the sentences based on the TF-IDF weights. We finally returned the top **k** sentences after ranking them.

Let us assume we are working with the following text document about Adolescence. After performing the above mentioned steps and setting a threshold on the value of **k** as 3, we obtain the summarised version of the original document.


Adolescence is regarded as the period of stress and strain, storm and strike. This is the time when an individual undergoes tremendous changes both physically, emotionally and psychologically. This sudden growth and development in the child is the unique characteristic of adolescence, an age which requires lots of care, affection, guidance, proper monitoring and motivation. It is because of the changes the child undergoes, stress and strain start ensuing. He starts behaving as a grown-up person but at times behaves childishly. There is frequent change in the behavioural act of the child at this stage. The hormonal changes have an acute impact on the personality of the child. Indulgence in untoward activities, aggressiveness, emotional imbalance, fickle-mindedness are some of the negative changes in the adolescent that need to be properly eyed. Such features differ from individual to individual in meagre or large. Adolescence is the stage which is considered as the most beautiful phase of one’s life when there is lots of imagination, aspiration zeal and potential outburst.

Original Document


This sudden growth and development in the child is the unique characteristic of adolescence, an age which requires lots of care, affection, guidance, proper monitoring and motivation. Indulgence in untoward activities, aggressiveness, emotional imbalance, fickle-mindedness are some of the negative changes in the adolescent that need to be properly eyed. Adolescence is the stage which is considered as the most beautiful phase of one’s life when there is lots of imagination, aspiration zeal and potential outburst.

Summary of document

4.6 Conclusion

We can thus summarize a text document of any size by ranking the sentences based on their TF-IDF weights. We can also put a limit on the number of sentences in the summary by setting a threshold **k**. Both the approaches are similar in methodology, but it has been observed that a summary accounting for sentences with scores greater than the average tends to retain more keywords and provide a more accurate gist of the original document.

4.7 Improvements

The algorithm to generate a summary based on TF-IDF scores has been through iterations of modifications over the years. It has been observed that keeping stop words (instead of removing them as the norm) and weighing nouns and parts-of-speech terms sometimes leads to better summaries. TF-IDF weighting methods that account for context such as *n*-grams and skip-grams retain parts of sentence structure to score keywords and although slower, tend to provide a better selection of sentences. However, the main drawback that remains unaddressed is that TF-IDF is prone to missing sentences which might be important to a document and may be directly related to the document's topic of discussion. This can be improved upon by using Latent Semantic Analysis (LSA).

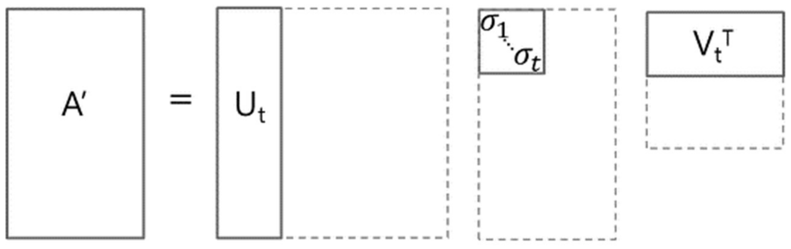
5. Summarization with Latent Semantic Analysis (LSA)

5.1 Linear Algebra Background

Theorem 1. *The singular value decomposition of an* ***m x n*** *real or complex matrix* **M** *is a factorization of the form* $\mathbf{USV}^\mathbf{T}$*, where* **U** *is an* ***m x m*** *real or complex unitary matrix,* **S** *is an* ***m x n*** *rectangular diagonal matrix with non-negative real numbers on the diagonal, and* **V** *is and* ***n x n*** *real or complex unitary matrix. If* **M** *is real,* **U** *and* $\mathbf{V}^\mathbf{T}$ *are real orthonormal matrices.*

$$\mathbf{M} = \mathbf{U}\ \mathbf{S}\ \mathbf{V}^\mathbf{T}$$

Theorem 2. *The truncated singular value decomposition of a matrix* **M** *is performed by taking only the* ***t*** *column vectors of* **U** *and* ***t*** *row vectors of* $\mathbf{V}^\mathbf{T}$ *corresponding to the* ***t*** *largest singular values in* **S** *and discarding the rest of the matrices.*

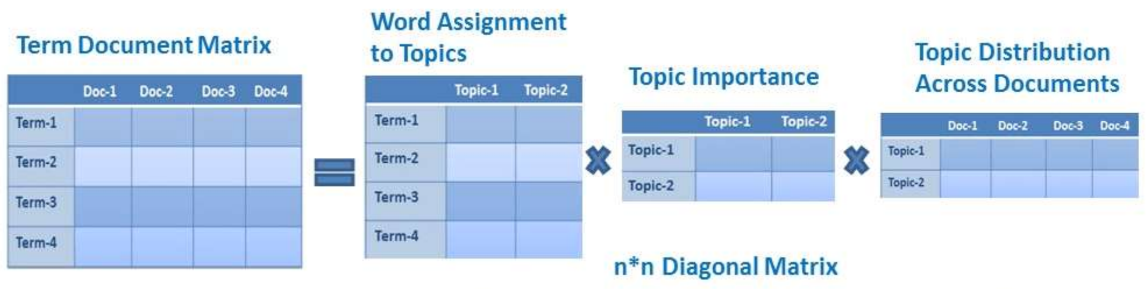


5.2 Mathematical Background

From **Theorem 2,** *it turns out that when you select the* **k** *largest singular values, and their corresponding singular vectors from* **U** *and* **V**, *you get the rank* **k** *approximation to* **X** *with the smallest error.*

$$\mathbf{X_k} = \mathbf{U_k}\,\mathbf{S_k}\,\mathbf{V_k^T}$$

This approximation has a minimal error. But more importantly we can now treat the term and document vectors as a "semantic space". We can now compare document vectors and even term vectors from the respective matrices and gain insights from the result obtained from decomposition.



5.3 Implementation

We implemented the algorithm from scratch to generate a summary by performing topic modelling using Latent Semantic Analysis and then performed sentence selection based on Murray & Renals & Carletta's approach.

From each row of $\mathbf{V}^{\mathbf{T}}$ matrix, for each topic, one or more sentences with the highest scores are selected. The number of sentences to be collected for each concept is determined by getting the percentage of the related singular value over the sum of all singular values, which are represented in the **S** matrix.

Finally, these sentences are combined, and the summary of the document is returned.

5.4 Algorithm to Generate Summary

We needed to identify the underlying topics in our document. We proceeded to first pre-process the document and break it up into sentences and then use LSA for topic modelling by fixing the value of the hyperparamter as **k** topics. We returned the decomposed matrices along with the term-topic document-topic matrices.

Part 1: Performing LSA to identify topics

```
def LSA(k, X):

    X_clean = list(map(clean,X))

    vectorizer = TfidfVectorizer(smooth_idf=True)

    X_transformed = vectorizer.fit_transform(X_clean)

    U, Sigma, VT = randomized_svd(X_transformed,
n_components=k,n_iter=300,random_state=122)

    svd = TruncatedSVD(n_components=k, algorithm='randomized', n_iter=300,
random_state=122)

    lsa = svd.fit_transform(X_transformed)

    column_names = ["Topic {}".format(str(i+1)) for i in range(k)]

    document_topic_matrix = pd.DataFrame(lsa,columns=column_names)

    document_topic_matrix["Document"] = X

    document_topic_matrix["Position"] = [i for i in range(len(X))]

    dic = vectorizer.get_feature_names()

    term_topic_matrix = pd.DataFrame(svd.components_, index = column_names,
columns = (dic)).T

    return U,Sigma,VT,lsa, document_topic_matrix, term_topic_matrix
```

We needed to find the percentage of sentences covered for every topic **k** in our document.

Part 2: Finding percentage of sentences for every topic

```
def percent_sigma(Sigma):

    agg = sum(Sigma)

    return Sigma/agg
```

We finally generated the summary from the underlying topics

Part 3: Generating the summary

```
def summarize(k,sigma, document_term):

    summary = []

    sigma*= len(sigma)
```

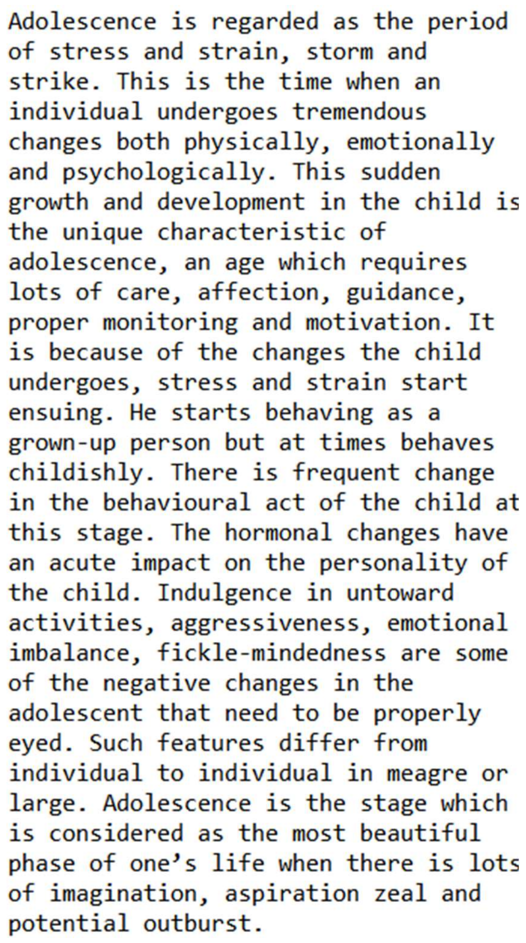
```
        sigma = sigma.astype(int)
        column_names = ["Topic {}".format(str(i+1)) for i in range(k)]
        for i in range(len(column_names)):
            num_sent = sigma[i]
            topic = column_names[i]
            document_term.sort_values(by = topic,inplace=True)
            document_term.reset_index(inplace=True)
            document_term.drop(columns = ["index"],inplace=True)
            for j in range(num_sent):
                if document_term["Document"][j] not in summary:

    summary.append((document_term["Document"][j],document_term["Position"][j]))
        summary.sort(key = lambda x: x[1])
        sent = [i[0] for i in summary]
        return " ".join(sent)
```

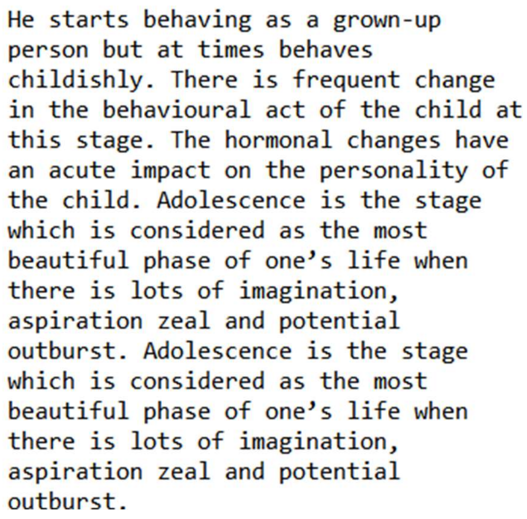
5.5 Final Implementation

We implemented the methodology from scratch to pre-process the original document and obtain the underlying **k** topics in the document. We finally returned the top contributing sentences from each topic after ranking them.

Let us assume we are again working with the text document about Adolescence. After performing the above mentioned steps and setting the value of topics **k** as 3, we obtain the summarised version of the original document.


Adolescence is regarded as the period of stress and strain, storm and strike. This is the time when an individual undergoes tremendous changes both physically, emotionally and psychologically. This sudden growth and development in the child is the unique characteristic of adolescence, an age which requires lots of care, affection, guidance, proper monitoring and motivation. It is because of the changes the child undergoes, stress and strain start ensuing. He starts behaving as a grown-up person but at times behaves childishly. There is frequent change in the behavioural act of the child at this stage. The hormonal changes have an acute impact on the personality of the child. Indulgence in untoward activities, aggressiveness, emotional imbalance, fickle-mindedness are some of the negative changes in the adolescent that need to be properly eyed. Such features differ from individual to individual in meagre or large. Adolescence is the stage which is considered as the most beautiful phase of one’s life when there is lots of imagination, aspiration zeal and potential outburst.

Original Document


He starts behaving as a grown-up person but at times behaves childishly. There is frequent change in the behavioural act of the child at this stage. The hormonal changes have an acute impact on the personality of the child. Adolescence is the stage which is considered as the most beautiful phase of one’s life when there is lots of imagination, aspiration zeal and potential outburst. Adolescence is the stage which is considered as the most beautiful phase of one’s life when there is lots of imagination, aspiration zeal and potential outburst.

Summary of document

5.6 Conclusion

We can thus summarize a text document of any size by first converting it into a vector space model of TF-IDF weights and then performing SVD to obtain the underlying topics using LSA. We also obtain three other matrices which give us an idea about the importance of each topic and how each topic is related to the different sentences and terms. We then pick the top contributing sentences from every topic.

Text summarization using LSA tends to work better than the traditional TF-IDF approach since it ensures to cover every single topic in the document. It also has weights assigned to these topics and uses these weights to select the number of sentences for each topic. This implies that topics with greater importance will have a larger number of sentences and the contribution of a topic increases with its weight in the topic-topic matrix.

5.7 Improvements

LSA is an immensely powerful linear algebraic method used for both text summarization purposes and tends to work better than just TF-IDF weights. However, its one downside is that one needs to choose the optimum value of the hyperparameter **k** to select the number of topics. This is usually done using cluster analysis by reducing the dimensions of the vector space to two and then selecting the optimum value for **k** to represent the underlying topics.

6. Final Implementation

We imported all the books from Project Gutenberg that are part of Python's Natural Language Toolkit and applied both approaches to summarization. To test the accuracy of our summarizations and their keyword retention capacities, we used Rapid Automatic Keyword Extraction (RAKE) to analyse the number of keywords returned by both summaries. We compared this to the total number of keywords present in the original document to obtain our information retention factor.

```
documents = [text1,text2,text3,text4,text5,text6,text7,text8,text9]

documents = [" ".join(i.tokens) for i in documents]

r = Rake()

keyvalues = []

for doc in documents:

    r.extract_keywords_from_text(doc)

    rake_count_total = len(r.get_ranked_phrases())


    X = sent_tokenize(doc)

    X_clean = list(map(clean,X))
    sentence_scores = SummarizationTFIDF.sentence_score(X_clean)

    summary1 = SummarizationTFIDF.summarize_avg(X,sentence_scores)

    r.extract_keywords_from_text(summary1)

    rake_count_tfidf = len(r.get_ranked_phrases())


    U,Sigma,VT,lsa, document_topic_matrix, term_topic_matrix = 
SummarizationLSA.LSA(10,X)

    percentage_topic = SummarizationLSA.percent_sigma(Sigma)

    summary2 = 
SummarizationLSA.summarize(10,percentage_topic,document_topic_matrix)

    r.extract_keywords_from_text(summary2)

    rake_count_lsa = len(r.get_ranked_phrases())



keyvalues.append((rake_count_tfidf/rake_count_total,rake_count_lsa/rake_cou
nt_total))
```

7. Conclusion

After implementation of both methodologies we observe that ____ tends to perform better than ______. We see that LSA performs better at identifying topics and selecting the optimum number of sentences related to each topic such that every concept in the document is represented based on its relevance. We also notice that as the size of the text document increases LSA scales extremely quickly compared to TF-IDF. The time taken to factorise a matrix into three decompositions and then summarise by picking the highest contributing sentences from every topic grows exponentially compared to TF-IDF. Although TF-IDF misses out on a few keywords compared to LSA, it finishes the task of summarizing a large paragraph much quicker.

Hence there is no preferred or optimal linear algebraic approach to text summarization. Both the above methodologies work extremely well and are application and use specific and depend a lot on the size of the text document and the number of text documents to be processed. It is preferable to use LSA for medium scale documents or when you are working with a smaller set of documents. However, if an extremely large set of documents is to be processed or the size of each document is immense, it is advisable to use TF-IDF for summarization purposes.

8. Scope of Future Work

There have been improvements to the TF-IDF model for text summarization in recent years which include using *n*-grams and skip grams for creating the vector space as well as retention of stop words and higher scaling of parts-of-speech and nouns. These approaches can improve the generation of the text summary at the cost of computation and speed of execution. LSA can also be improved by first finding the optimum number of topics in the document by simple cluster analysis followed by truncated Singular Value Decomposition (SVD) to give us a better idea about the topic distribution. Further, the modern methods of information extraction such as Probabilistic-LSA and LDA can be investigated for text summarization along with current state-of-the-art methods involving Deep Learning and Neural Networks that use CNN's and RNN's to generate an abstractive summary.

9. Summary

- We imported books from Project Gutenberg to analyse our text summarization methods
- We implemented TF-IDF score based and LSA topic model based approaches to text summarization
- We pre-processed each document and extracted the summary using the above mentioned methodologies
- We then calculated the keyword retention factor in both summaries and performed a simple analysis of the statistics
- We then concluded that the text summarization technique to be used depends greatly on the application and is use-specific.

References

[1] https://en.wikipedia.org/wiki/Automatic_summarization

[2] https://en.wikipedia.org/?title=Text_summarization&redirect=no

[3] https://www.idc.com/getdoc.jsp?containerId=prUS45213219

[4] https://en.wikipedia.org/wiki/Natural_Language_Toolkit

[5] https://en.wikipedia.org/wiki/Project_Gutenberg

[6] Christian, Hans & Agus, Mikhael & Suhartono, Derwin. (2016). Single Document Automatic Text Summarization using Term Frequency-Inverse Document Frequency (TF-IDF). ComTech: Computer, Mathematics and Engineering Applications. 7. 285. 10.21512/comtech.v7i4.3746.

[7] PRACTICAL APPROACH TO AUTOMATIC TEXT SUMMARIZATION JIRI HYNEK 1 , KAREL JEZEK 2 1 inSITE, s.r.o. Rubesova 29, Plzen, Czech Republic e-mail: jiri.hynek@insite.cz 2Department of Computer Science and Engineering University of West Bohemia Univerzitni 22, 306 14 Plzen, Czech Republic

[8] Using Latent Semantic Analysis in Text Summarization and Summary Evaluation by Josef Steinberger, Ježek

[9] Murray, Gabriel & Renals, Steve & Carletta, Jean. (2005). Extractive summarization of meeting recordings. 593-596.

[10] Ozsoy, Makbule & Alpaslan, Ferda & Cicekli, Ilyas. (2011). Text summarization using Latent Semantic Analysis. J. Information Science. 37. 405-417. 10.1177/0165551511408848.

[11] O. Foong, S. Yong and F. Jaid, "Text Summarization Using Latent Semantic Analysis Model in Mobile Android Platform," 2015 9th Asia Modelling Symposium (AMS), Kuala Lumpur, 2015, pp. 35-39, doi: 10.1109/AMS.2015.15.