MINI PROJECT REPORT

ON

LINEAR ALGEBRA TECHNIQUES IN NATURAL LANGUAGE PROCESSING

Submitted by

1.   **ADITEYA BARAL**   PES1201800366

2.   **VINAY V KIRPALANI**   PES1201800218

3.   **SAARTHAK AGARWAL**   PES1201801811

Branch & Section: **CSE D**

PROJECT EVALUATION

(For Official Use Only)

Sl.No.	Parameter	Max Marks	Marks Awarded
1	Background & Framing of the problem	4	
2	Approach and Solution	4	
3	References	4	
4	Clarity of the concepts & Creativity	4	
5	Choice of examples and understanding of the topic	4	
6	Presentation of the work	5	
	Total	25	

Name of the Course Instructor   : P RAMA DEVI

Signature of the Course Instructor   :

Linear Algebra Techniques
In
Natural Language Processing

May 2020

Contents

Introduction
To
Natural Language Processing
May 2020

1. Introduction

Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software. Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyse large amounts of natural language data.

Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation. The study of natural language processing has been around for more than 50 years and grew out of the field of linguistics with the rise of computers.

2. Applications of Natural Language Processing

Natural language processing helps computers communicate with humans in their own language and scales other language-related tasks. For example, NLP makes it possible for computers to read text, hear speech, interpret it, measure sentiment, and determine which parts are important.

Today's machines can analyze more language-based data than humans, without fatigue and in a consistent, unbiased way. There are several applications of NLP such as

1. Content categorization: A linguistic-based document summary, including search and indexing, content alerts and duplication detection.
2. Topic discovery and modeling: Accurately capture the meaning and themes in text collections, and apply advanced analytics to text, like optimization and forecasting.
3. Contextual extraction: Automatically pull structured information from text-based sources.
4. Sentiment analysis: Identifying the mood or subjective opinions within large amounts of text, including average sentiment and opinion mining.
5. Speech-to-text and text-to-speech conversion: Transforming voice commands into written text, and vice versa.
6. Document summarization: Automatically generating synopses of large bodies of text.
7. Machine translation. Automatic translation of text or speech from one language to another.

Vector Space Models

May 2020

1. Abstract

Vector space model[1] or term vector model in natural language processing is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. These vectors are also known as Word Embeddings[2]. Its first use was in the SMART Information Retrieval System.

Machine learning algorithms operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features. Natural Language Processing deals with the study and analysis of natural languages in their raw text form. To perform machine learning on text, we need to transform our documents into vector representations such that we can apply numeric machine learning. This process is called feature extraction or more simply, vectorization, and is an essential first step toward language-aware analysis.

It is used in information filtering, information retrieval, indexing and relevancy rankings. Word Embeddings help us to perform basic NLP tasks such as text classification, document clustering and feature extraction along with other higher degree tasks such as text summarization and similarity metrics.

2. Keywords

Vector Space Model, Machine Learning, Natural Language Processing, Word Embeddings, Feature Space, Information Retrieval, Text Ranking, Keyword Extraction, Text Classification, Text Clustering, Feature Clustering, Bag of Words, Smoothening Factor, TF-IDF

3. Introduction

The advantages of using a vector space model over the primitive Boolean (one encoding) methods are many, such as it eliminates binary weights and the vector space models are based on linear algebra, which implies that operations in linear algebra also apply to these vector spaces. This also allows us to rank documents based on relevance and give us the ability to match documents based on similarity.

We have used a dataset[3] compiled by IMDB (International Movie Database) for their reviews which has also been published on Kaggle. This is a very well-known dataset for text analytics and have more than 50,000 rows of movie reviews. It is a very structured dataset that contains train and test data along with labelled reviews for classification purposes.

Our objective is to find out how vector space models are built out of raw text data and to understand how *n*-dimensional vectors translate to documents. We will also look at how a vector space model can be improved upon using other mathematical approaches and how using a TF-IDF[4] normalization over the simple Bag of Words[5] model can scale and tweak the

vector space created by turning it into a close representation of the document and hence provides very meaningful insights to the document.

4. Bag of Words

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)**. *In Bag of Words, this relationship is given by the frequency of term* **i** *in document* **j.**

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

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

4.2 Mathematical Background

In Bag of Words vector space model, since we have *m* features(terms), we represent each *m*-dimensional document vector with the frequency of its terms.

$$
M_{ij} = frequency\ of\ term\ \mathbf{i}\ in\ document\ \mathbf{j}
$$

4.3 Implementation

We have implemented Bag of Words to build our vector space model. We did this by first finding the vocabulary of our documents and then used it to construct a vector for every such document. We then combined these vectors to form a term-document matrix.

4.4 Algorithm

We need a function to return the count of a word in a document

Part 1: Function to return term frequencies

```
def TF(word, document):

    return document.count(word)
```

We now need to call this function for term **i** in document **j**.

```python
def BagOfWords(documents):

    preprocessed = list(map(clean, documents))

    vocabulary = list(set(" ".join(preprocessed).split()))

    vocabulary.sort()

    X = np.zeros([len(vocabulary), len(documents)],

    dtype = 'float')

    for i in range(X.shape[0]):

        for j in range(X.shape[1]):

            X[i,j]+= TF(vocabulary[i], preprocessed[j])

    return X
```

4.5 Final Implementation

Part 1: Text Pre-processing

Raw text carries a lot of noise with it[6], in the form of redundant or unnecessary words, symbols and other lexicons. These must be removed to make our feature set more concise and reduce the dimensions of each document.

1. The first step in text pre-processing is to convert every text to lowercase and remove lexicons such as symbols and punctuations.
2. We then remove useless words that add no additional information to the sentence. These are known as stop words[7] and they include:
   a. Determiners – Determiners tend to mark nouns where a determiner usually will be followed by a noun examples: the, a, an, another
   b. Coordinating Conjunctions – Coordinating conjunctions connect words, phrases, and clauses examples: for, an, nor, but, or, yet, so
   c. Prepositions – Prepositions express temporal or spatial relations examples: in, under, towards, before
3. Certain words exist in their inflectional or derivative forms. All these words need to be transformed into their base or root form. This process of converting words to a common base form is known as Stemming[8]. Example, stemming converts 'playing', 'playful' and all other forms to 'play'.
   a. Stemming follows a rule-based approach and comes under the umbrella of rule-based NLP.
   b. It uses a set of hard and defined rules to reduce a word to its base form. These rules are based on the suffixes of the words. The rules for the Snowball Stemmer can be found at here[9].

Example, if we have the following documents, we can pre-process as

Before pre-processing	After pre-processing
First came the Golden Age and the Silver Age.	first came golden age silver age
Then came the Bronze Age.	came bronz age
Then was the age of Revolution with the Iron Age.	age revolut iron age
Now we are in the Digital Age	digit age

Before pre-processing   *After pre-processing*

Part 2: Preparing Vocabulary

The vocabulary of a vector space model is defined as the unique set of all words present across each document in our dataset. Each word is taken only once regardless of its frequency in the sample of documents.

Our vocabulary in the previous example is

0	1	2	3	4	5	6	7	8
age	bronze	came	digital	first	golden	iron	revolution	silver

Part 3: Generating the Bag of Words

A Bag of Words is a simple word embedding model where words are mapped to their frequency in each document. It is called a "bag" of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document. We can finally convert our set of documents into a Bag of Words by find the frequency of each term in the pre-processed documents.

	0	1	2	3
0	2	1	2	1
1	0	1	0	0
2	1	1	0	0
3	0	0	0	1
4	1	0	0	0
5	1	0	0	0
6	0	0	1	0
7	0	0	1	0
8	1	0	0	0

Bag of Words model for the example

4.6 Conclusion

We can thus convert any set of documents into their respective vector space models using a Bag of Words. This will allow us to use this model for further machine learning applications which require a two-dimensional numerical vector space. However, the Bag of Words does have a disadvantage since it tends to overscale common words that occur in multiple documents, this leading to form of bias. These words are often non important to the document and their appearances in multiple documents leads to a sense of false relevancy.

4.7 Improvements

The term frequency of a document does not always give us a better understanding of the features. It is required to associate these features with an importance factor that helps us gauge how essential or relevant these features are in a document.

The Bag of Words model is too simple and cannot represent these weights. Common words that occur across all documents will have an extremely high term frequency (like the word "age" in the above example). We want words that occur rarely to have a higher importance since it is these words that help us differentiate one document from another (like the words "iron" or "golden").

This can be done by converting the matrix into a TF-IDF (Term Frequency – Inverse Document Frequency) matrix.

5. Term Frequency – Inverse Document Frequency (TF-IDF)

5.1 Inverse Document Frequency

IDF[9] normalizes the TF or Bag of Words matrix by scaling down the frequencies of words that commonly occur across all documents and scaling up the frequencies of words that occur less frequently. Put simply, the higher the TF-IDF score, the rarer the term and vice versa.

TF-IDF was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they do not mean much to that document. This method can be used to answer questions like how important a term is in a document and which terms have the highest relevance in a document.

5.2 Mathematical Background

The TF-IDF matrix is obtained by performing an element wise multiplication of the TF matrix with the IDF matrix. The corresponding TF and IDF matrices can be obtained as

$$TF - IDF_{ij} = tf_{ij} \; x \; idf(i)$$

$$tf_{ij} = \frac{frequency \; of \; term \; \mathbf{i} \; in \; document \; \mathbf{j}}{number \; of \; words \; in \; document \; \mathbf{j}}$$

$$idf(i) = log \frac{n + 1}{1 + df(d, i)} + 1$$

$$n = total\ number\ of\ documents$$

$$df(d, i) = number\ of\ documents\ containing\ term\ \mathbf{i}$$

5.3 Implementation

We implement TF-IDF by first modifying our TF function to return a scaled down term frequency. We multiply each term in the resultant matrix with its IDF value to finally obtain the TF-IDF scores for the term-document matrix.

5.4 Algorithm

We first need a modified term frequency function to scale the values.

Part 1: Modified TF function

```
def TF(word, document, documents):
    return document.count(word)/len(documents)
```

We also need a function to find the document frequency, df

Part 2: Calculating document frequency, DF

```
def df(documents,term):
    ctr = 0
    for doc in documents:
        if term in doc:
            ctr+=1
    return ctr
```

We must convert the document frequency values into an inverse document frequency, so we need another function to do this.

Part 3: Calculating inverse document frequency, IDF

```
def idf(documents,term):
    n = len(documents)
    return 1+np.log((1+n)/(1+df(documents,term)))
```

Finally, we put all these together and find the TF-IDF matrix

Part 4: Finding the TF-IDF matrix

```
def TFIDF(documents):

    preprocessed = list(map(clean, documents))

    vocabulary = list(set(" ".join(preprocessed).split()))

    vocabulary.sort()

    X = np.zeros([len(vocabulary), len(documents)],dtype = 'float')

    for i in range(X.shape[0]):

        for j in range(X.shape[1]):

            X[i,j] = TF(vocabulary[i], preprocessed[j],
preprocessed)*idf(preprocessed,vocabulary[i])

    return X
```

5.4 Final Implementation

Part 1: Performing text pre-processing and finding the new TF matrix

On performing the same text pre-processing as above and finding the new term frequencies, we obtain the TF matrix.

	0	1	2	3
0	0.0625	0.0714286	0.1	0.111111
1	0	0.0714286	0	0
2	0.03125	0.0714286	0	0
3	0	0	0	0.111111
4	0.03125	0	0	0
5	0.03125	0	0	0
6	0	0	0.05	0
7	0	0	0.05	0
8	0.03125	0	0	0

TF matrix for the previous example

Part 2: Converting the TF matrix to a TF-IDF matrix

We finally multiply the above obtained values with the IDF scores of each term to obtain the TF-IDF matrix

	0	1	2	3
0	0.480859	0.379192	0.5938	0.462637
1	0	0.726641	0	0
2	0.363247	0.572892	0	0
3	0	0	0	0.886548
4	0.460733	0	0	0
5	0.460733	0	0	0
6	0	0	0.568947	0
7	0	0	0.568947	0
8	0.460733	0	0	0

Final TF-IDF matrix

5.5 Conclusion

On converting the term-document Bag of Words model to a TF-IDF matrix, we see that the terms have now more meaningful weights. Although the example uses small sentences, we can clearly see the scaling of frequent words like "age" compared to the scaling of rare words like "digital", thus providing a closer and more accurate representation of the document.

Index	0	1	2	3
age	2	1	2	1
bronz	0	1	0	0
came	1	1	0	0
digit	0	0	0	1
first	1	0	0	0
golden	1	0	0	0
iron	0	0	1	0
revolut	0	0	1	0
silver	1	0	0	0

Bag of Words Model

Index	0	1	2	3
age	0.480859	0.379192	0.5938	0.462637
bronz	0	0.726641	0	0
came	0.363247	0.572892	0	0
digit	0	0	0	0.886548
first	0.460733	0	0	0
golden	0.460733	0	0	0
iron	0	0	0.568947	0
revolut	0	0	0.568947	0
silver	0.460733	0	0	0

TF-IDF model

6. Implementation

The input to our problem is the IMDB reviews dataset. We use the unclassified training set with 50,000 reviews to create our vector space models. We build a model using both the Bag of Words as well as TF-IDF models and we limit our number of features to 10,000 since preparing a model with the full set of over 1,00,000 features is computationally expensive and the first few features provide a good comparison.

While implementing the following algorithm, we ensure to perform

1. Effective text pre-processing to remove lexicons such as symbols and punctuations
2. Remove stop words to reduce noise and dimensions
3. Limit the number of dimensions to 10,000
4. Add +1 smoothening to IDF to reduce chances of 0 division

After creating both models, we compare the weights of the features obtained in each.

7. Conclusion

On implementing both vector space models we see that TF-IDF does a better job at representing word embeddings compared to the primitive Bag of Words model. We also see that TF-IDF has scaled down the scores of frequent words like "actor" and has scaled up the scores of more rare words such "act". This gives us an idea about the relevance of such words in our documents and it also provides insights about the topics and words spread across all documents. We can also conclude that terms with higher scores have more relevance or importance in a document and can be used to rank the document too, and these weights and associated terms can be used for other language processing applications such as topic modelling, information retrieval and text classification.

Index	0	1	2	3	4	5
abil	0	0	0	1	0	0
abl	0	0	0	0	0	0
absolut	0	1	0	0	0	0
accent	0	0	0	0	0	0
accept	0	0	0	0	0	0
achiev	0	0	1	0	0	0
across	0	0	0	0	0	0
act	0	0	0	3	0	0
action	0	0	0	0	0	0
actor	1	1	0	0	1	0
actress	0	0	0	0	0	0
actual	0	0	1	0	0	0
ad	0	0	0	0	0	0
adapt	0	0	0	0	0	0
add	0	0	1	0	0	0
admit	1	0	0	0	0	0

Bag of Words model for IMDB Dataset

Index	0	1	2	3	4	5
abil	0	0	0	0.182888	0	0
abl	0	0	0	0	0	0
absolut	0	0.131015	0	0	0	0
accent	0	0	0	0	0	0
accept	0	0	0	0	0	0
achiev	0	0	0.120215	0	0	0
across	0	0	0	0	0	0
act	0	0	0	0.265073	0	0
action	0	0	0	0	0	0
actor	0.0947599	0.0926136	0	0	0.0736363	0
actress	0	0	0	0	0	0
actual	0	0	0.0691768	0	0	0
ad	0	0	0	0	0	0
adapt	0	0	0	0	0	0
add	0	0	0.0986622	0	0	0
admit	0.159039	0	0	0	0	0

TF-IDF model for IMDB Dataset

8. References

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

[2] https://en.wikipedia.org/wiki/Word_embedding

[3] https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

[4] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[5] https://en.wikipedia.org/wiki/Bag-of-words_model

[6] https://en.wikipedia.org/wiki/Text_mining

[7] https://en.wikipedia.org/wiki/Stop_words

[8] https://en.wikipedia.org/wiki/Stemming

[9] http://snowball.tartarus.org/algorithms/porter/stemmer.html

[10] Vectorization of Text Documents for Identifying Unifiable News Articles, Anita Kumari Singh, Mogalla Shashi Department of Computer Science & Systems Engineering, Andhra University, India, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 10, No. 7, 2019

[11] Term Weighting Approaches in Automatic Text Retrieval by Gerald Salton, Chris Bukley, 87-881 November 1987, University of Computer Science, Cornell University, New York

Document Similarity

May 2020

1. Abstract

Semantic similarity[1] is a metric defined over a set of documents or terms, where the idea of distance between items is based on the likeness of their meaning or semantic content as opposed to lexicographical similarity. These are mathematical tools used to estimate the strength of the semantic relationship between units of language, concepts, or instances, through a numerical description obtained according to the comparison of information supporting their meaning or describing their nature.

Based on text analyses, semantic relatedness between units of language (e.g., words, sentences) can also be estimated using statistical means such as a vector space model[2] to correlate words and textual contexts from a suitable text corpus. The evaluation of the proposed semantic similarity measures is evaluated through two main ways. The former is based on the use of datasets designed by experts and composed of word pairs with semantic similarity/relatedness degree estimation. The second way is based on the integration of the measures inside specific applications such the information retrieval, recommender systems, natural language processing. For example, in natural language processing, knowing one information resource in the internet, it is often of immediate interest to find similar resources.

2. Keywords

Machine learning, Natural Language Processing, Semantic Similarity, Mathematical Tools, Vector Space Model, Normalization, Cosine Law, Cosine Similarity, Linear Algebra, Inner Product, Information Retrieval

3. Introduction

Plagiarism[3] is the representation of another author's language, thoughts, ideas, or expressions as one's own original work. Plagiarism involves submitting someone's work as their own, taking passages from their own previous work without adding citations, re-writing someone's work without properly citing sources and others.

The basis of a plagiarism checker lies in checking for similar contexts, topics, and word usages in a set of documents. We need a robust algorithm that can maximize on this concept and use words present in a document as features to compare their usage and find the distance or similarity in their context and then convert these similarity scores into plagiarism measures so we can understand how much of the documents share the same content.

We are trying to use concepts of linear algebra – namely vector space models and relationships between them to measure the plagiarism content between two documents. To do this, we need to first pre-process our document and then convert them into document-term matrices so they can be interpreted as a vector space model.

We will be using mathematical vector distance measures such as cosine similarity to find the similarity between the given documents and understand how distance between vectors translates to plagiarism content.
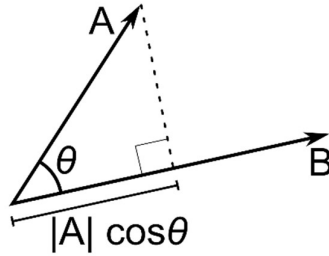
4. Cosine Similarity

4.1 Linear Algebra Background

Theorem 1. *Given two vectors $\vec{a}$ and $\vec{b}$, the inner product between these vectors is given as $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + (a_n b_n)$ or $\vec{a} \cdot \vec{b} = a^T b$. Further, the cosine of the angle between these vectors is given as*

$$\cos \theta = \frac{a^T b}{\|a\| \|b\|}$$

4.2 Proof

Assume two vectors $\vec{A}$ and $\vec{B}$ with angle θ between them and $\vec{p}$ is the projection of $\vec{A}$ on $\vec{B}$.

$$\vec{p} = \vec{A} \cos \theta$$

Taking norm on both sides,

$$\|\vec{p}\| = \|\vec{A}\| \|\cos \theta\|$$

$$\vec{p} = k\vec{B}$$

$$k = \frac{B^T A}{B^T B}$$

$$k = \frac{B^T A}{\|B\|^2}$$

Substituting this value of k, we get

$$\vec{p} = \frac{B^T A \vec{B}}{\|B\|^2}$$

$$\|\vec{p}\| = \frac{B^T A}{\|B\|}$$

$$\|\vec{p}\| = \|A\| \cos \theta$$

16

$$\|A\| \cos \theta = \frac{B^T A}{\|B\|}$$

$$\mathbf{\cos \theta} = \frac{\mathbf{B^T A}}{\mathbf{\|A\|\|B\|}}$$

4.4 Implementation

We implemented cosine similarity, we first pre-processed the documents and then converted them into a vector space model. We finally returned the inner product between them.

4.3 Algorithm

We first find the TF-IDF vector space models of the two documents and then just find the inner product between them.

```
def cosine_similarity(documents):

    clean_documents = list(map(clean,documents))

    vectorizer = TfidfVectorizer(smooth_idf=True)

    X = vectorizer.fit_transform(clean_documents)

    X = X.toarray()

    return np.dot(X[0],X[1])
```

5. Implementation

The input to our problem is a set of 2 documents in raw format. We first perform text pre-processing on these documents to eliminate noise and reduce features and perform dimensionality reduction. We then transform this data into a Bag of Words (BoW) vector space model and further scale these values by performing TF-IDF. We finally use the cosine similarity formula and obtain the distance between these vectors which translates to our percentage of plagiarism content.

While implementing the following algorithm, we ensure to perform

1. Effective text pre-processing to remove lexicons such as symbols and punctuations
2. Remove stop words to reduce noise and dimensions
3. Add +1 smoothening to IDF to reduce chances of 0 division

The two documents we used to test our implementation were definitions of machine learning from two different sources. Using these sources, we performed two tests

1. Compared one source with another: On comparing the definitions with each other, we obtained a similarity of 0.45, indicating that they did cover the same context and topics but were not similar enough to be the same.
2. Compared a portion of one source with itself: We performed this test to check for self-plagiarism where one of the sources also included the other source's contents. We obtained a score of more than 0.8, indicating a clear plagiarism between both documents.

To visualize the vector space models, we need to reduce the dimensions of the *n*-dimensional vector space to two dimensions. We used truncated SVD to perform dimension reduction and reduced the vector space and then plotted both the test cases.
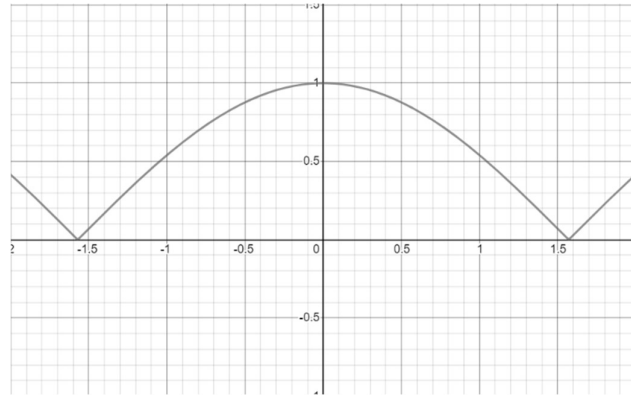
6. Visualizations

After dimensionality reduction has been performed, we can plot our 2-dimensional vectors and observe the results.
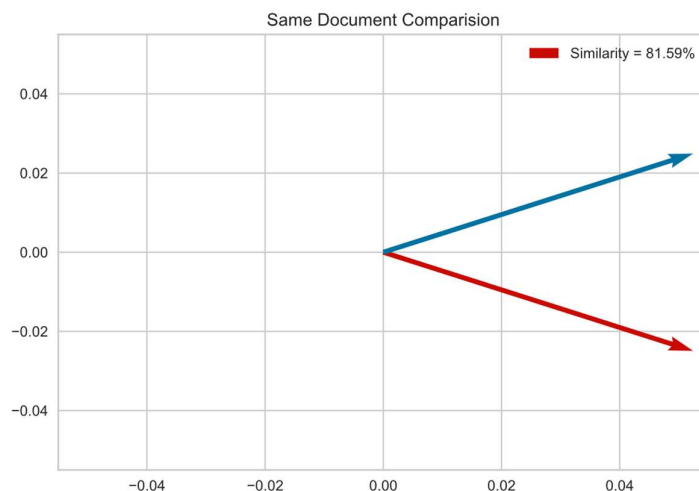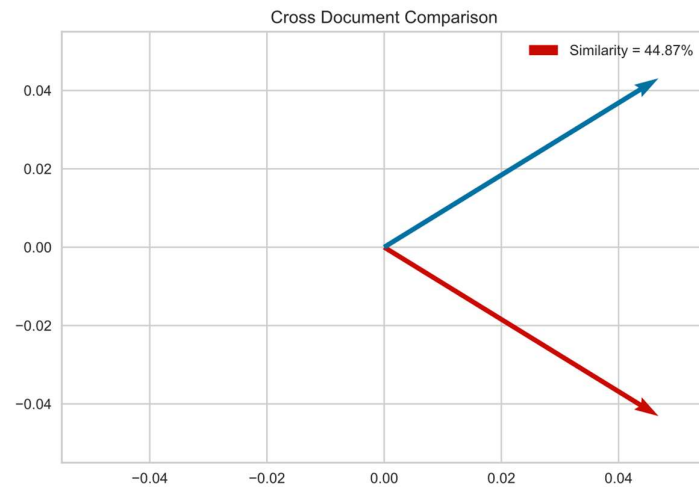
We can conclude that as the similarity between the documents increases, the vector space models tend to get closer until they eventually converge when both documents are the same.

7. Graphical Analysis

Plot of |cos x|

For any two vector space models, since the magnitude of each dimension is a scaled factor that depends on the term frequency, it will always take values within [0,1] since the term frequency can never be negative. When we apply the inner product between two such vectors, we hence infer that the possible similarities will follow the plot of $|cos\ x|$. We can also verify that if $\theta_1\ is\ lesser\ than\ \theta_2$ in $[-\frac{\pi}{2},\frac{\pi}{2}]$ then $cos\ \theta_1\ will\ be\ greater\ than\ cos\ \theta_2$. Therefore, lesser the distance, higher the cosine similarity value and higher the plagiarism.

8. Conclusion

On finding the cosine similarity between sets of documents we see that a higher score indicates a higher level of plagiarism and vice versa. Since the cosine similarity measure only uses terms or words as features, it is safe to assume that the checking is performed on a contextual level and not a semantic level. However, since plagiarism majorly depends on the context of the language and according to distributional hypothesis[3] in natural language processing which states that words occurring together have similar meaning, we can say that cosine similarity is an accurate distance metric to find document similarity.

9. Scope of Future Work

Cosine Similarity is a linear algebraic distance measure that is used to find the similarity between two text documents, provided they have been converted into their vector space models. Using more complex word embedding models such as GloVe will help us in storing the semantics as well as context in the document vector models and hence lead to far more accurate scores while estimating the distance between them. We could also use a neural network approach using complex contextual vector models such as Word2Vec, Doc2Vec and even ELMo and BERT to enchance our word embeddings which will give us state of the art distance scores too. Since the algorithm depends vastly on the type of word embeddings used, any model that enhances the vector spaces generated will automatically increase the performance.

10. Summary

- We took a set of 2 documents to find the plagiarism content between them.
- We first performed text pre-processing by removing stop words and unnecessary lexicons.
- We converted these pre-processed documents into word vector models by converting them into TF-IDF vectors.
- We then wrote a function to find the cosine similarity between two vectors
- We passed the vector space models to the above function and found the distance between them.
- Post this we quantified plagiarism mathematically.

References

[1] https://en.wikipedia.org/wiki/Semantic_similarity#Natural_language_processing

[2] https://en.wikipedia.org/wiki/Vector_space_model

[3] https://en.wikipedia.org/wiki/Distributional_semantics#Distributional_hypothesis

[4] R. Lahitani, A. E. Permanasari and N. A. Setiawan, "Cosine similarity to determine similarity measure: Study case in online essay assessment," 2016 4th International Conference on Cyber and IT Service Management, Bandung, 2016, pp. 1-6, doi: 10.1109/CITSM.2016.7577578.

[5] A. Madylova and S. G. Oguducu, "A taxonomy based semantic similarity of documents using the cosine measure," 2009 24th International Symposium on Computer and Information Sciences, Guzelyurt, 2009, pp. 129-134, doi: 10.1109/ISCIS.2009.5291865

[6] Distance Weighted Cosine Similarity Measure for Text Classification by Baoli LiLiping Han

[7] Similarity Measures for Text Document Clustering by Anna Huang, Department of Computer Science, The University of Waikato, Hamilton, New Zealand

[8] Comparison Jaccard similarity, Cosine Similarity and Combined Both of the Data Clustering with Shared Nearest Neighbour Method, Lisna Zahrotun

Topic Modelling Techniques
using
Linear Algebra

May 2020

1. Abstract

In machine learning and natural language processing[1], a topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents. Topic modelling[2] is a frequently used unsupervised text-mining tool for discovery of hidden semantic structures in a text body. Intuitively, given that a document is about a particular topic, one would expect particular words to appear in the document more or less frequently: "dog" and "bone" will appear more often in documents about dogs, "cat" and "meow" will appear in documents about cats, and "the" and "is" will appear equally in both. A document typically concerns multiple topics in different proportions; thus, in a document that is 10% about cats and 90% about dogs, there would probably be about 9 times more dog words than cat words.

The "topics" produced by topic modelling techniques are clusters of similar words. A topic model captures this intuition in a mathematical framework, which allows examining a set of documents and discovering, based on the statistics of the words in each, what the topics might be and what each document's balance of topics is. In the age of information, the amount of the written material we encounter each day is simply beyond our processing capacity. Topic models can help to organize and offer insights for us to understand large collections of unstructured text bodies. Originally developed as a text-mining tool, topic models have been used to detect instructive structures in data such as genetic information, images, and networks. They also have applications in other fields such as bioinformatics.

2. Keywords

Machine learning, Natural Language Processing, Topic Modelling, Text Mining, Unsupervised Learning, Statistical Model, Semantic Structure, Dimensionality Reduction, Feature Extraction, Information Retrieval, Eigenvectors, Eigenvalues, Orthogonality, LSA, SVD, NMF

3. Introduction

We have used the 20 News Group Dataset[3], originally made by Ken Lang for his paper on filtering news from the net. It is an extremely popular dataset using for a variety of natural language processing tasks from text classification to text clustering and even topic modelling. The dataset, which has also been published on Kaggle as well as other renowned machine learning repositories such as the UCI Machine Learning Repository and is even a part of

scikit-learn, contains almost 20,000 news documents across each of its 20 news categories. Some of the topics include hardware, motorcycles, electronics, space, sales, politics, and sports.

Our goal is to find the underlying topics within this dataset and try and obtain all the words corresponding to each topic in each news group, which will then be compared to the group's topic for accuracy measures. To perform topic modelling, we need to extract appropriate features so we can pick the words that offer most relevance to each topic and these words will indicate the underlying topics in each set of documents. Our goal is to use the two most powerful linear algebraic techniques in topic modelling namely, Latent Semantic Analysis (LSA) and Non-Negative Matrix Factorization (NMF) and compare the results of these two approaches, view the topics identified by each algorithm and perform a simple cluster analysis. Both algorithms work on the principle of matrix factorization. LSA is a powerful statistical technique that performs matrix factorization on a document-term or term-document matrix using truncated Singular Value Decomposition (SVD) to reduce its dimension and obtain the features which provide us with the most singular values. NMF factorizes a matrix into two non-negative matrices where each of these matrices hold the relevance of words to topics and features.

4. Latent Semantic Analysis – LSA

4.1 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* **USV^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* **V^T** *are real orthonormal matrices.*

$$M = U\ S\ V^T$$

Theorem 2. *The diagonal entries* **S** *are known as the singular values of* **M***.*

Theorem 3. *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* **V^T** *corresponding to the* **t** *largest singular values in* **S** *and discarding the rest of the matrices.*
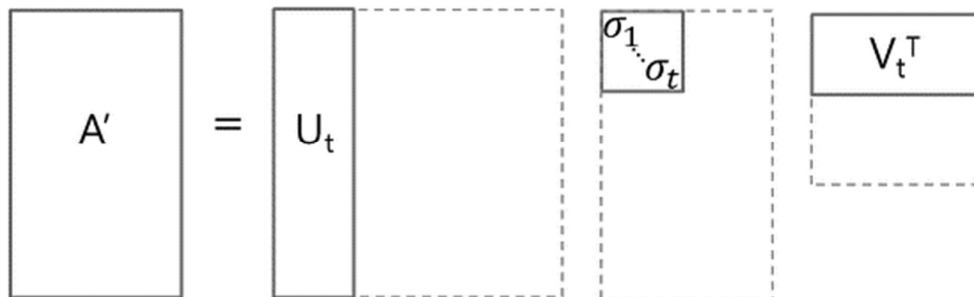
4.2 Mathematical Background

Let $\mathbf{X}$ *be a matrix where* $\mathbf{X_{ij}}$ *describes the occurrence of term* $\mathbf{i}$ *in document* $\mathbf{j}$ *(this can be, for example, the frequency).* $\mathbf{X}$ *will look like this.*

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

$\mathbf{t_i}$ *stands for term* $\mathbf{i}$ *and* $\mathbf{d_j}$ *stands for document* $\mathbf{j}$

Now a row in this matrix will be a vector corresponding to a term, giving its relation to each document.

$$
\mathbf{t}_i^T = \begin{bmatrix} x_{i,1} & \cdots & x_{i,j} & \cdots & x_{i,n} \end{bmatrix}
$$

Likewise, a column in this matrix will be a vector corresponding to a document, giving its relation to each term.

$$
\mathbf{d}_j = \begin{bmatrix} x_{1,j} \\ \vdots \\ x_{i,j} \\ \vdots \\ x_{m,j} \end{bmatrix}
$$

Now the dot product $\mathbf{t}_i^T \mathbf{t_p}$ *between two term vectors gives the correlation between the terms over the set of documents. The matrix product* $\mathbf{XX^T}$ *contains all these dot products. Element (i,p)(which is equal to element (p,i)) contains the dot product* $\mathbf{t}_i^T \mathbf{t_p}$*. Likewise, the matrix* $\mathbf{X^T X}$ *contains the dot products between all the document vectors, giving their correlation over the terms* $\mathbf{d}_j^T \mathbf{d_q} = \mathbf{d}_q^T \mathbf{d^T}$*.*

Now, from **Theorem 1**, *the singular value decomposition of an **m x n** real matrix* **X** *is a factorization of the form* **USV^T**, *where* **U** *and* **V** *are orthogonal matrices and* **S** *is a diagonal matrix with non-negative real numbers on the diagonal.*

The matrices giving us the term and document correlations become **XX^T = USS^{T}U^T** *and* **X^{T}X = VS^{T}SV^T**. **U** *contains the eigenvalues of* **XX^T** *and* **B** *contains the eigen values of* **X^{T}X**.

The decomposition of the matrices looks like this

$$
(\mathbf{t}_i^T) \rightarrow
\begin{bmatrix}
x_{1,1} & \cdots & x_{1,j} & \cdots & x_{1,n} \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
x_{i,1} & \cdots & x_{i,j} & \cdots & x_{i,n} \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
x_{m,1} & \cdots & x_{m,j} & \cdots & x_{m,n}
\end{bmatrix}
=
(\mathbf{\hat{t}}_i^T) \rightarrow
\begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} \cdots \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix}
\cdot
\begin{bmatrix}
\sigma_1 & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & \sigma_l
\end{bmatrix}
\cdot
\begin{bmatrix}
[& \mathbf{v}_1 &] \\
 & \vdots & \\
[& \mathbf{v}_l &]
\end{bmatrix}
$$

with column labels: X over $(\mathbf{d}_j)\downarrow$, U, Σ, V^T over $(\mathbf{\hat{d}}_j)\downarrow$.

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.

4.3 Implementation

We implemented LSA by first performing text pre-processing to remove unwanted lexicons and stop words to reduce noise and dimensions of our features. We then transformed our

documents into a vector space model by converting them into *n*-dimensional TF-IDF vectors. We finally applied Singular Value Decomposition (SVD) on these vectors and then truncated them by picking the **k** largest singular values and then returned the components to get our term-topic and document-topic matrices. The weights of the topics with respect to the terms and the documents can be obtained from the results of the decomposition.
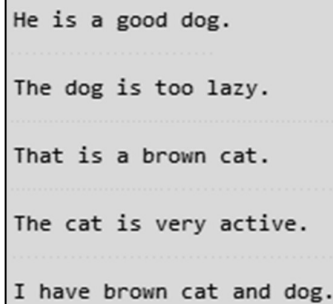
4.4 Algorithm

```
def LSA(k, X):

    X_clean = list(map(clean,X))

    vectorizer = TfidfVectorizer(smooth_idf=True)

    X_transformed = vectorizer.fit_transform(X_clean)

    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,index=X)

    dic = vectorizer.get_feature_names()

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

    return document_topic_matrix, term_topic_matrix
```

4.5 Final Implementation

We implemented Latent Semantic Analysis from scratch using Singular Value Decomposition on the vector space models. We initially started by performing pre-processing of our corpus.

Let us assume we have the following five sentences about cats and dogs.

```
He is a good dog.

The dog is too lazy.

That is a brown cat.

The cat is very active.

I have brown cat and dog.
```

Documents 1, 2, 3 and 4 are about cats and dogs respectively while document 5 is about both

These sentences represent two topics, namely cats and dogs. We would like to know the underlying topic distribution across these documents and which terms in each of these documents help us assign a topic to them.

After pre-processing and converting them into a vector space model, we performed truncated SVD with the number of components as 2 (since we have two topics) and obtained the low rank approximation matrix **X**.

	0	1
0	0.341383	0.719978
1	0.341383	0.719978
2	0.860949	-0.365984
3	0.516666	-0.385005
4	0.949412	0.0236303

Low rank approximation matrix obtained

on performing SVD

To analyze these scores in a more meaningful way, we reconstruct the document-topic and term-topic matrices using the components of the SVD. After constructing these, we obtain the following

Index	Topic 1	Topic 2
He is a good dog.	0.341383	0.719978
The dog is too lazy.	0.341383	0.719978
That is a brown cat.	0.860949	-0.365984
The cat is very active.	0.516666	-0.385005
I have brown cat and dog.	0.949412	0.0236303

Document-Topic Matrix

We can see a clear topic distinction between the documents. We can also conclude that Topic 1 is about cats and Topic 2 is about dogs and the first four documents have been

appropriately scored. The Document-Topic matrix returns a score of how strongly a document aligns itself with the given topic and a higher score implies a strong correlation to the topic.

Index	Topic 1	Topic 2
activ	0.200354	-0.242441
brown	0.596512	-0.20181
cat	0.629338	-0.329886
dog	0.415831	0.616903
good	0.132383	0.453377
lazi	0.132383	0.453377

Term-Topic Matrix

We can see how each term in our corpus affects the topic distribution. As expected, terms such as "dog" and "lazy" are strongly related to topic 2, which is about dogs and terms such as "cat" and "brown" are strongly related to topic 1 which is about cats.

4.5 Conclusion

We can thus extract the underlying topics from any given set of documents by converting it into a vector space and then applying truncated SVD. The low rank approximation resultant matrix from the decomposition holds information regarding the topic-term scores and the topic-document scores. We can extract these components from the SVD to obtain our Term-Topic and Document-Topic matrices. The number of extracted topics is a hyperparameter (one which is provided to the algorithm and not learnt on its own) and this decides the truncating size and the dimensions of the two Topic matrices. It is often difficult to find the right number of topics to extract from a corpus, and clustering methods have proved successful in approximating the right number of topics for information extraction.

4.6 Improvements

LSA is a highly optimized algorithm to perform topic modelling, but it does have a few drawbacks[4] with respect to information retrieval applications. The results of LSA are difficult to interpret, and it might not be feasible to construct Topic matrices for every decomposition. LSA also does not completely capture polysemy (multiple meanings of the same word) since each occurrence is assumed to have the same semantics. LSA can further be improved upon by using more advanced vector space models like Word2Vec or Doc2Vec which retain semantics as well as context. Modifying TF-IDF to use *n*-grams instead of individual words is also a popular method, along with the recently developed Probabilistic LSA[5] and LDA.

Currently, Non-Negative Matrix Factorization or NMF is preferred to perform topic modelling and segmentation since NMF allows for high readability and interpretation of data. It is also faster and easier to compute compared to LSA, since it factorizes the Term-Document matrix into only two sub-matrices.

5. Non-Negative Matrix Factorization - NMF

5.1 Linear Algebra Background

Theorem 1. *Non-negative matrix factorization (NMF or NNMF), also non-negative matrix approximation is a group of algorithms in multivariate analysis and linear algebra where a matrix* **V** *is factorized into two matrices* **W** *and* **H***, with the property that all three matrices have no negative elements.*

Theorem 2. *When* **V = WH***, the multiplicative factor of* **W** *and* **H** *is* **I***.*

5.1 Mathematical Background

Non-negative Matrix Factorization is a Linear-algebraic model that factors high-dimensional vectors into a low-dimensionality representation. Like Principal component analysis (PCA), NMF takes advantage of the fact that the vectors are non-negative. By factoring them into the lower-dimensional form, NMF forces the coefficients to also be non-negative.

Let **V** *be a matrix where* **V$_{ij}$** *describes the occurrence of term* **i** *in document* **j** *(this can be, for example, the frequency).* **V** *will look like this.*

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

t$_i$ *stands for term* **i** *and* **d$_j$** *stands for document* **j**

From **Theorem 1***, we know that this matrix can be factorized into two matrices* **W** *and* **H** *that is,* **V** = **WH***. There are several ways to obtain these matrices, the most conventional being Lee and Seung's multiplicative update rule which uses an iterative approach to find* **W** *and* **H** *until convergence.*

Initialize **W** *and* **H** *as non-negative matrices. We then update the values in* **W** *and* **H** *by computing the following, with* **n** *as an index of the iteration until* **W** *and* **H** *are stable.*

$$\mathbf{H}_{[i,j]}^{n+1} \leftarrow \mathbf{H}_{[i,j]}^{n} \frac{((\mathbf{W}^n)^T \mathbf{V})_{[i,j]}}{((\mathbf{W}^n)^T \mathbf{W}^n \mathbf{H}^n)_{[i,j]}}$$

$$\mathbf{W}_{[i,j]}^{n+1} \leftarrow \mathbf{W}_{[i,j]}^{n} \frac{(\mathbf{V}(\mathbf{H}^{n+1})^T)_{[i,j]}}{(\mathbf{W}^n \mathbf{H}^{n+1}(\mathbf{H}^{n+1})^T)_{[i,j]}}$$

When multiplying matrices **W** *and* **H***, the dimensions of the factor matrices may be significantly lower than those of the product matrix and it is this property that forms the basis of NMF. NMF generates factors with significantly reduced dimensions compared to the original matrix. For example, if* **V** *is an* **m** × **n** *matrix,* **W** *is an* **m** × **p** *matrix, and* **H** *is a* **p** × **n** *matrix then* **p** *can be significantly less than both* **m** *and* **n***.*

The matrices **W** *and* **H** *hold information about the documents and its features, along with the underlying topics.* **W** *translates to a Term-Topic matrix which stores the topics associated with each term in our corpus and* **H** *translates to a Topic-Document matrix which tells us about the underlying topics in each document of our corpus. Since both* **W** *and* **H** *are non-negative matrices, the minimum score within these matrices is zero with the highest being one.*
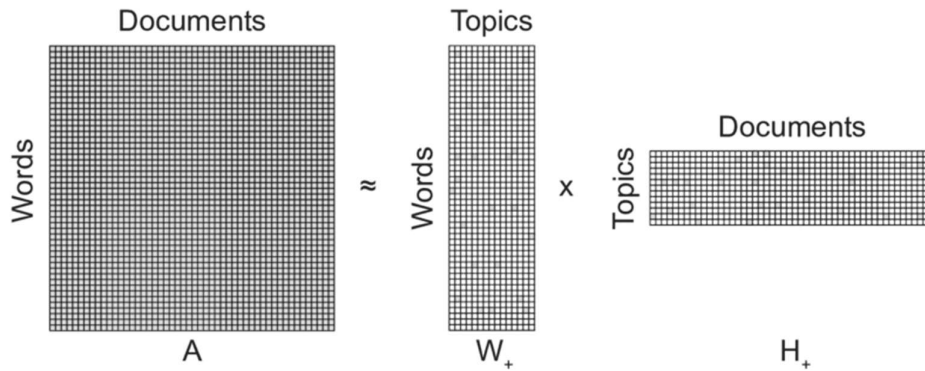
5.3 Implementation

We implemented NMF by again performing text pre-processing to remove unwanted lexicons and stop words to reduce noise and dimensions of our features. We then transformed our documents into a vector space model by converting them into *n*-dimensional TF-IDF vectors with +1 IDF smoothening factor. We finally applied Non-Negative Matrix Factorization (NMF) on these vectors and obtained our term-topic and document-topic matrices which were returned. The weights of the topics with respect to the terms and the documents can be obtained from the results of the decomposition.

5.4 Algorithm to perform NMF

Performing NMF on the Term-Document matrix almost immediately gives us the two components. We finally put the components together along with the documents and terms and return the Term-Topic and Document-Topic matrices.

```
def NMFFunction(k, X):

    X_clean = list(map(clean,X))

    vectorizer = TfidfVectorizer(smooth_idf=True)

    X_transformed = vectorizer.fit_transform(X_clean)

    model = NMF(n_components=k, random_state=122, init='nndsvd')

    topicmodel = model.fit_transform(X_transformed)

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

    document_topic_matrix =
pd.DataFrame(topicmodel,columns=column_names, index = X)

    dic = vectorizer.get_feature_names()

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

    return topicmodel, document_topic_matrix, term_topic_matrix
```

5.5 Final Implementation

We implemented NMF on the vector space models. We initially started by performing pre-processing of our corpus.

Let us assume we are again working with the same example about about cats and dogs.

```
He is a good dog.

The dog is too lazy.

That is a brown cat.

The cat is very active.

I have brown cat and dog.
```

Documents 1, 2, 3 and 4 are about cats and dogs respectively

while document 5 is about both

After pre-processing and converting them into a vector space model, we performed NMF with the number of components as 2 (since we have two topics) and obtained the product matrix **V**.

	0	1
0	0	0.815757
1	0	0.815757
2	0.77254	0
3	0.496321	0
4	0.677987	0.325285

*Matrix **V** obtained from **W x H***

To analyze these scores in a more meaningful way, we reconstruct the document-topic and term-topic matrices as done before.

Index	Topic 1	Topic 2
He is a good dog.	0	0.815757
The dog is too lazy.	0	0.815757
That is a brown cat.	0.77254	0
The cat is very active.	0.496321	0
I have brown cat and dog.	0.677987	0.325285

Document-Topic Matrix

We can view the topic distribution between the documents. Like before we can also conclude that Topic 1 is about cats and Topic 2 is about dogs and the first four documents have been appropriately scored. However, the advantage here is that unlike LSA, NMF only returns non-negative values, so if a document does not align itself with a topic, its topic strength for that topic is zero. This allows for clearer interpretation and readability and can be used for topic classification purposes if we apply a suitable activation function.

Index	Topic 1	Topic 2
activ	0.316532	0
brown	0.789383	0.0256452
cat	0.870847	0
dog	0.156597	0.729724
good	0	0.471764
lazi	0	0.471764

Term-Topic Matrix

We can again similarly see how each term in our corpus affects the topic distribution. Just like before, since the minimum score returned is zero, it is easier to interpret these values and given any term in our corpus, we can easily classify it into one of the segmented topics.

5.6 Conclusion

We can thus extract the underlying topics from any given set of documents by converting it into a vector space and then applying Non-Negative matrix Factorization. The advantage with NMF is that it not only retains the semantic space, but it also provides better interpretation of the factorized matrix. However, unlike LSA which returns the singular values in matrix **S** which tells us about the importance of each topic, NMF does not give us this interpretation.

Since NMF returns only non-negative values, each value can be the strength or the confidence of a document belonging to a topic, with higher values indicating it is strongly related to that topic. Just like LSA, NMF also uses a hyperparameter **k** to segment the corpus into topics and choosing the right value of **k** is crucial to the algorithm's accuracy. NMF tends to return a better segmentation of topics compared to LSA and is used in multiple applications not limited to natural language processing such as bioinformatics and search engine recommendations.

5.7 Improvements

Although NMF is a powerful linear algebraic technique to perform matrix factorization, it does have its own downsides. NMF works better when the matrix is sparse since it the

algorithm assumes missing values by default. LSA also follows a more deterministic approach instead. Modern developments have been made to improve the performance of NMF by evaluating and ranking the bases in the parallel multiplicative generation algorithm. NMF can further be improved upon by finding the right number of topics by performing a simple cluster analysis and then performing topic modelling to generate the exact segmentation of topics in our corpus.

6. Implementation

The input to our problem is the 20 News Group dataset, which contains about 20,000 news articles across 20 topics. We will be using both the linear algebraic approaches of topic modelling – LSA and NMF with the value of the hyperparameter **k** as 20 (since there are 20 categories). We will generate the Term-Topic and Document-Topic matrices as done before and compare them. We will then proceed to perform Uniform Manifold Approximation and Projection (UMAP), which will reduce our dimensions so we can view the document cluster on a 2D plot which will show us the distribution of topics in a two dimensional space.

7. Conclusion

After loading the dataset and performing topic modelling, we obtain the following Term-Topic and Document-Topic matrices. Listed below are the top few entries in each of the Document-Topic matrices

Index	alt.atheism	comp.graphics	mp.os.ms-windows.m	mp.sys.ibm.pc.hardwa
wonder anyon could enlighten…	0.195646	0.0509242	0.0920605	0.112394
fair number brave soul upgr…	0.143378	0.121813	0.0216391	0.0520389
well folk mac plus final gave…	0.355203	0.0985431	0.0651727	0.0632765
weitek addressphon num…	0.217907	0.0459492	0.0444813	0.011621
articl c owcbn pworldstdcom to…	0.156041	0.0249135	-0.043921	-0.0501711
cours term must rigid defin bil…	0.166449	-0.0336527	-0.0492141	-0.0739376
peopl respond request info tr…	0.208286	0.133073	-0.0201041	0.278505
show know much scsi scsi scsi …	0.104093	0.107465	-0.00308423	-0.110728
win download sever icon bmps…	0.16753	0.0918116	0.0259329	0.0253661
board year work diskdoubl autod…	0.16697	0.123614	-0.0273144	-0.0652995
line ducati gts model k clock r…	0.231978	0.0926173	0.0820895	0.130546
yep pretti much jew understand …	0.277909	-0.127147	-0.147867	0.0266521
	-1.06479e-30	9.0535e-31	4.78485e-31	8.15203e-31
descript extern tank option ssf…	0.229461	0.0366356	0.0255892	-0.104136
reduc price list thing forsal be…	0.208682	0.053691	0.0717666	0.0231271

Document-Topic Matrix from Latent Semantic Analysis

Index	alt.atheism	comp.graphics	mp.os.ms-windows.m	mp.sys.ibm.pc.hardwa	omp.sys.mac.hardwar
wonder anyon could enlighten…	0	0	0	0.0366359	0
fair number brave soul upgr…	0.00406746	0.00182183	0.00553203	0.0574255	0
well folk mac plus final gave…	0.0185118	0.0227253	0.0163129	0.0473744	0.00409532
weitek addressphon num…	0	0	0	0.00403714	0
articl c owcbn pworldstdcom to…	0.0199533	0.00393471	0	0.00011983	0.00566076
cours term must rigid defin bil…	0.0323607	0.0393349	0.00248761	0	0
peopl respond request info tr…	0.0124809	0	0.00904719	0.121035	0
show know much scsi scsi scsi …	0.00031073	0.0151207	0	0.00125558	0
win download sever icon bmps…	0	0.0485054	0.0179806	0.0133384	0.00514315
board year work diskdoubl autod…	0.005702	0.0108695	0.00300793	0.00819633	0
line ducati gts model k clock r…	0	0	0.000388301	0.0693099	0.00503329
yep pretti much jew understand …	0.021365	0.00468138	0	0	0.0030094
	0	0	0	0	0
descript extern tank option ssf…	0.0142314	0.0362906	0.0205059	0	0.00299523
reduc price list thing forsal be…	0.00534023	0.00858136	0.0129741	0.0224905	0

Document-Topic Matrix from Non-Negative Matrix Factorization

We can hence see that the sparse matrix obtained by performing NMF is easier to interpret and understand compared to the matrix obtained by LSA since topics that are not related to a document are given the minimum score of zero.

We can further analyze our results by viewing both the Term-Topic matrices.

Topic	Words
alt.atheism	would use one get like know think 34eople
comp.graphics	window use thank file card drive email program
comp.os.ms-windows.misc	window file game team program run win play
misc.forsale	would like window get car think know problem
sci.space	file use like get pleas one 34eople look
talk.religion.misc	one get team window player would anyon good
rec.sport.hockey	would game card sale include offer new price

*Term-Topic Matrix from Latent Semantic Analysis*

Topic	Words
alt.atheism	peopl us right say make law gun go
comp.graphics	use port help set program also printer imag
comp.os.ms-windows.misc	game team play player year win season last
misc.forsale	new sale price offer includ ship sell condit
sci.space	like look sound bike someth thing good realli
talk.religion.misc	armenian israel isra jew kill arab turkish attack
rec.sport.hockey	know anyon heard want game anyth wonder let

*Term-Topic Matrix from Non-Negative Matrix Factorization*

We observe from the above concept that the topics have a lot of overlapping words, causing a term to get segmented into multiple topics. However, we can also conclude that NMF performs better than LSA as shown from the relevant terms in each topic. We proceeded to plot our data points to find the reason behind our overlapping terms and to find out which algorithm performed better on the entire dataset.

7. Visualizations

We performed dimension reduction using Uniform Manifold Approximation and Projection (UMAP) on our low rank approximation matrix after performing LSA and our approximate product matrix after performing NMF. After reducing the dimensions of the Document-Topic matrix to a two-dimensional space, we plotted our results to find the topic segmentation across the documents.

LSA Topic Modelling
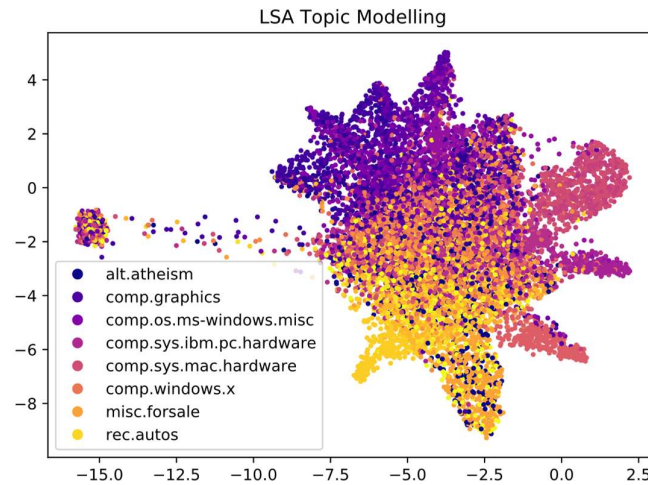
We observe that there are a lot of overlaps between the topics and the terms, resulting in ineffective clustering of topics. Each colour represents a different topic and each point on the scatter plot is a document. We can also see that completely unrelated topics such as atheism and automobiles have zero overlaps but related topics such as Windows, hardware and IBM have a large overlapping section in the middle of the cluster. However, we can also observe that there are many data points which couldn't be classified into one topic alone and have terms which might contribute to multiple topics, as shown by the stray point in the left region of the plot.

NMF Topic Modelling
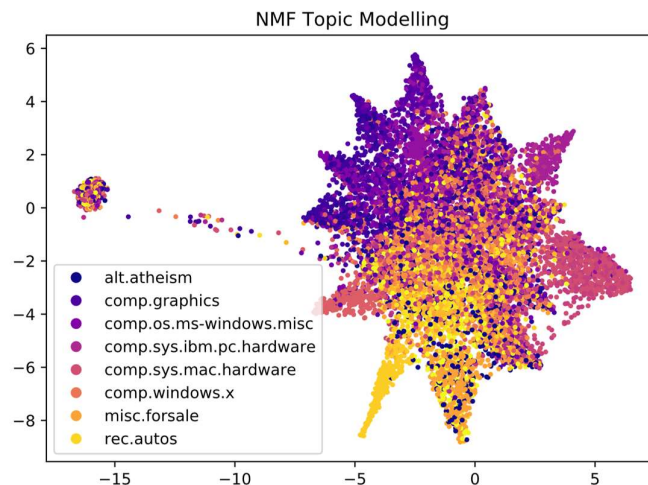
We can observe from the NMF plot that the topics are better segregated and unrelated topics are quite well apart from each other. We can also observe that the boundaries are more well defined and all documents under the same topic are in close vicinity of each other. We can also conclude that NMF has done a better job at identifying the topics and the terms

associated with each topic since there are lesser number of stray points which represent documents with multiple underlying topics.

8. Conclusion

We can thus find the underlying the topics in any document using simple yet powerful linear algebraic methods for topic modelling. Although NMF is faster and less computationally expensive to perform, it does not give us insights about how important the topics are and the weightage they carry in each document. However, LSA apart from being slower to compute, is also less interpretable compared to NMF and the matrices obtained after performing LSA are harder to read compared to the ones obtained after performing NMF since unlike NMF, LSA returns not only the terms related to a topic, but also the unrelated terms adding to clutter and disorganization. However, it is preferred over NMF since knowing the weightage each topic carries in a document is extremely significant and gives us more insights which portions of a document could hold valuable information. LSA is also used in other natural language processing applications such as text summarization and information extraction and retrieval.

9. Scope of Future Work

There have been a few improvements to the LSA algorithm over the years, the biggest being the introduction of p-LSA or Probabilistic LSA. Topic modelling has also been implemented using other nonlinear algebraic methods using LDA and Deep Learning techniques, which have provided state-of-the-art results in recent years. We can also perform simple EDA to find the optimum number of topics and feed that value into our LSA or NMF algorithm to further increase the accuracy of our topic segmentation.

10. Summary

- We used the 20 News Group Dataset to find the underlying topics and their relation to the documents and terms by removing stop words and unnecessary lexicons
- We converted these pre-processed documents into word vector models by converting them into TF-IDF vectors
- We then applied Latent Semantic Analysis and Non-Negative Matrix Factorization to extract the Term-Topic and Document-Topic matrices and compared them
- We finally plotted the topic segmentation in a two-dimensional space and analyzed the topics

References

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

[2] https://en.wikipedia.org/wiki/Topic_model

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 summarize 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)**.

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

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.*

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

*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 $\mathbf{s_j}$ of the elements for every column vector $\mathbf{d_j}$ and store the pair **(j, s_j)** in $\mathbf{S_j}$.*

*We then rank these sentences based on their aggregate scores $\mathbf{s_j}$ 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 summarized version of the document.

4.4 Algorithm

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

```python
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

```python
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 summarized 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 prioritizes keywords rather than topical sentences and 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* $\mathbf{USV^T}$, *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* $\mathbf{V^T}$ *are real orthonormal matrices.*

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

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* $\mathbf{V^T}$ *corresponding to the **t** largest singular values in* **S** *and discarding the rest of the matrices.*


43
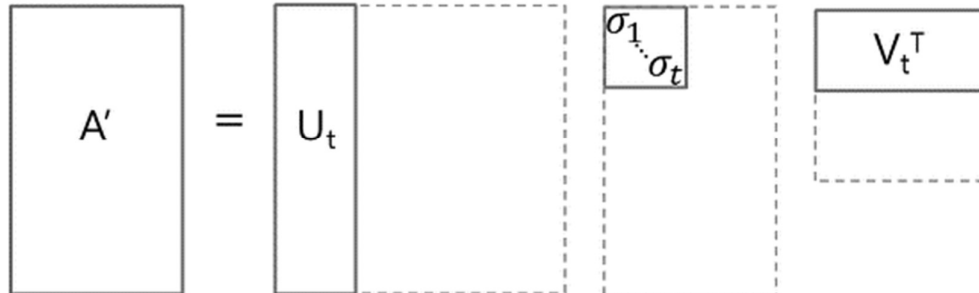
5.2 Mathematical Background

*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.*

$$X_k = U_k \, S_k \, V_k^T$$

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.
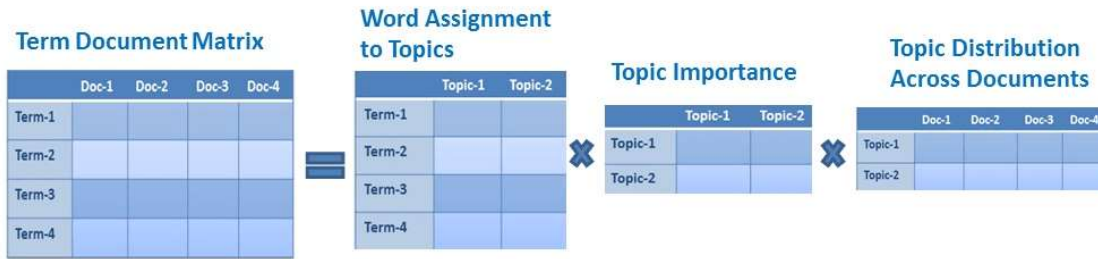
5.3 Implementation

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 V^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

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 hyperparameter **k** for number of topics. We returned the decomposed matrices along with the term-topic document-topic matrices.

Part 1: Performing LSA to identify topics

```python
def LSA(k, X):

    X_clean = list(map(clean,X))

    vectorizer = TfidfVectorizer(smooth_idf=True, max_features=1000)

    X_transformed = vectorizer.fit_transform(X_clean)

    feature_count = len(vectorizer.get_feature_names())

    if k>feature_count:

        k = feature_count-1

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

    lsa = svd.fit_transform(X_transformed)

    Sigma = svd.singular_values_

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

    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 k,Sigma, 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

```python
def percent_sigma(Sigma, X):

    agg = sum(Sigma)

    return (Sigma/agg)*len(X)
```

We finally generated the summary from the underlying topics

```python
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 250, we obtain the summarized 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 might 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. 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 analyze 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.

```python
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())


    k = 250

    new_k, Sigma, lsa, document_topic_matrix, term_topic_matrix =
SummarizationLSA.LSA(k,X)

    percentage_topic = SummarizationLSA.percent_sigma(Sigma,X).astype(int)

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

    r.extract_keywords_from_text(summary2)

    rake_count_lsa = len(r.get_ranked_phrases())


    t = (rake_count_tfidf/rake_count_total,rake_count_lsa/rake_count_total)

    print(t)

    keyvalues.append(t)

    ctr+=1
```

We observe that TF-IDF consistently does better than LSA when it comes to large scale documents. However, the difference between them decreases as the documents get smaller.

TF-IDF	LSA
0.817423	0.394248
0.830996	0.464743
0.709882	0.584459
0.749057	0.536156
0.824207	0.399309
0.845638	0.482705
0.703937	0.572606
0.843424	0.447808
0.799104	0.430397

Comparison of keyword retention factors over 9 books

7. Conclusion

After implementation of both methodologies we observe that TF-IDF performs better than LSA when the size of the document is large, but only slightly better when dealing with medium scale documents. 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 and it performs poorly. The time taken to factorize a matrix into three decompositions and then summarize 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.

TF-IDF performs summarization based on the features and selects sentences with the highest scoring features. Hence, it ensures to cover most of the keywords but does not cover all the underlying topics. On the other hand, LSA ensures that every topic (as provided by the hyperparameter **k**) or concept has been justifiably represented based on their weights, but due to this prioritization it skips out on keywords crucial to the document. TF-IDF score-based summarizations leads to larger summaries, since there is no limit on the number of sentences being selected. Hence these summaries tend to contain at least 75% of the original contents (and therefore, more keywords). LSA based summaries get limited to the only top **k** topics, hence choosing a right value for the hyperparameter is crucial for topic selection as well as sentences chosen.

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 analyze 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.