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 summarisation 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] normalisation 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.**

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

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

4.2 Mathematical Background

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.

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

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**.

Part 2: Function to prepare the Bag of Words

```
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

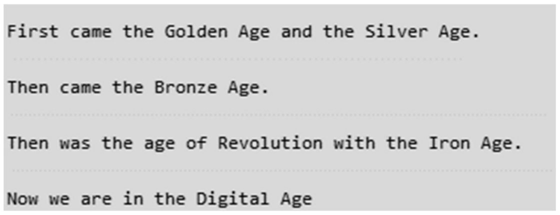
Part 1: Text Pre-processing

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.

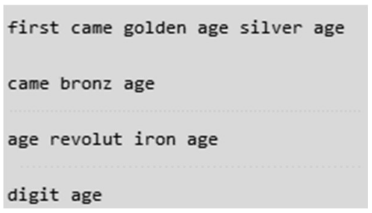
b. It uses a set of hard and defined rules to reduce a word to its base form. These rules are based on the suffixes of the words. The rules for the Snowball Stemmer can be found at here[9].

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



```
First came the Golden Age and the Silver Age.
Then came the Bronze Age.
Then was the age of Revolution with the Iron Age.
Now we are in the Digital Age
```

Before pre-processing



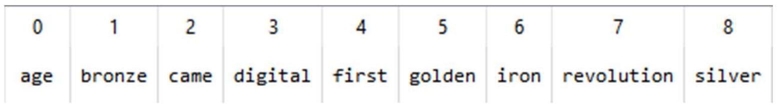
```
first came golden age silver age
came bronz age
age revolut iron age
digit age
```

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.

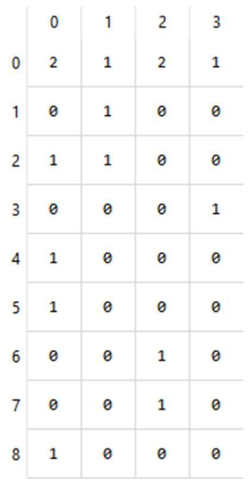
Our vocabulary in the previous example is



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] normalises 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

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

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

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

$$\boldsymbol{n} = total\ number\ of\ documents$$

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

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

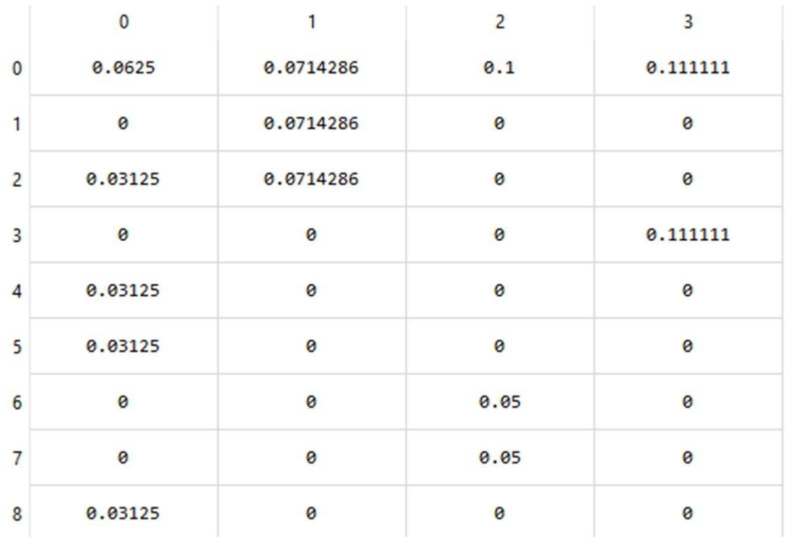
Part 4: Finding the TF-IDF matrix

```
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.

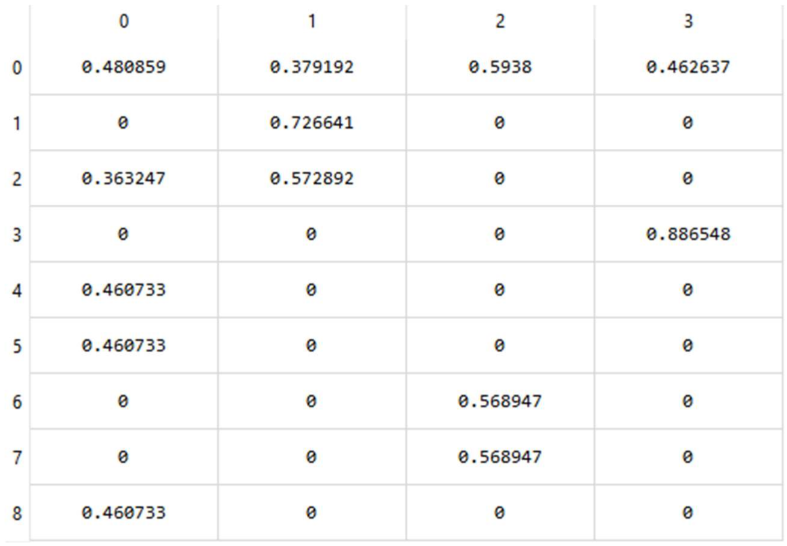


	0	1	2	3
0	0.0625	0.0714286	0.1	0.111111
1	0	0.0714286	0	0
2	0.03125	0.0714286	0	0
3	0	0	0	0.111111
4	0.03125	0	0	0
5	0.03125	0	0	0
6	0	0	0.05	0
7	0	0	0.05	0
8	0.03125	0	0	0

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

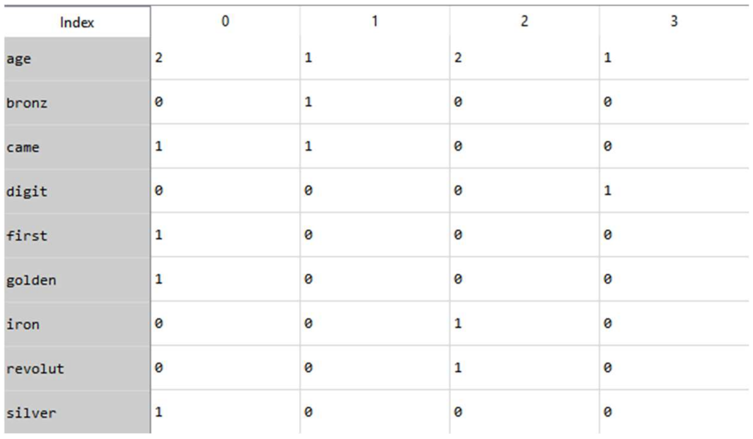


	0	1	2	3
0	0.480859	0.379192	0.5938	0.462637
1	0	0.726641	0	0
2	0.363247	0.572892	0	0
3	0	0	0	0.886548
4	0.460733	0	0	0
5	0.460733	0	0	0
6	0	0	0.568947	0
7	0	0	0.568947	0
8	0.460733	0	0	0

Final TF-IDF matrix

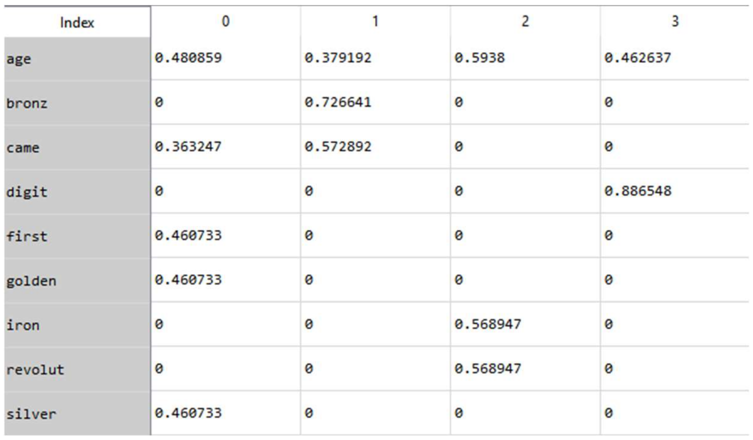
5.5 Conclusion

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.

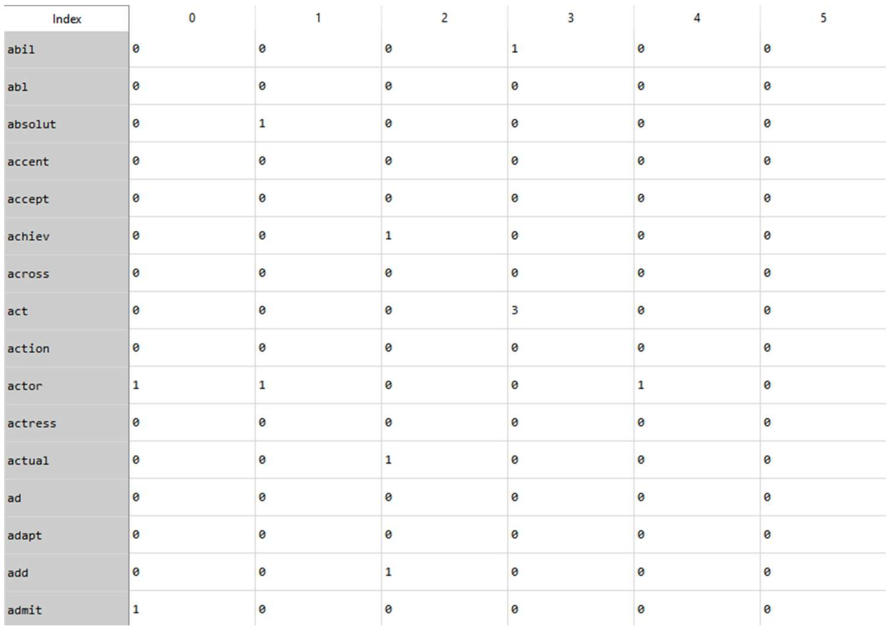
While implementing the following algorithm, we ensure to perform

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.

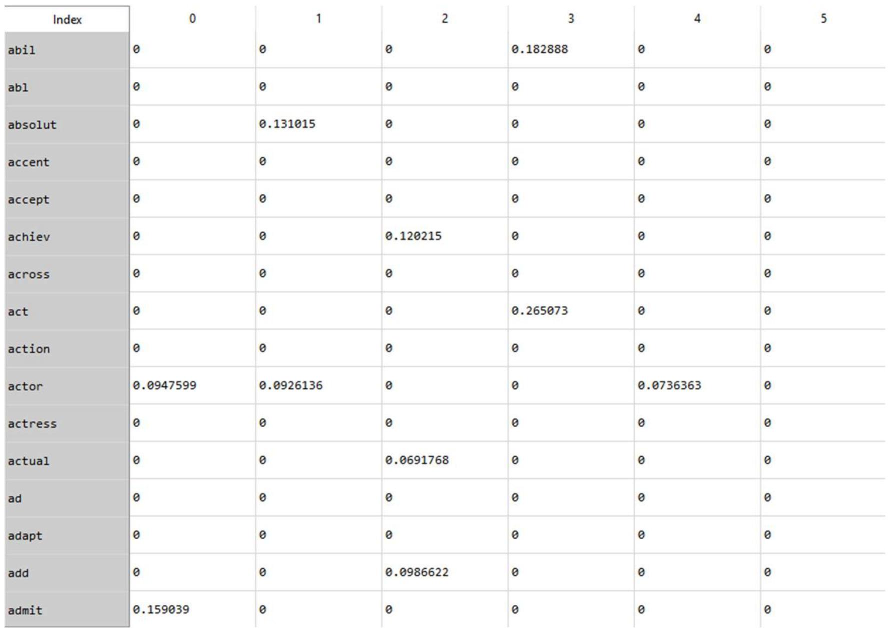
7. Conclusion

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