DOCUMENT CLASSIFICATION USING VARIOUS SIMILARITY TECHNIQUES

Mrs. Y. Swapna Assistant Professor Dept of Computer Science & Engineering from Vignan institute of technology and science. Email ID: yenugulaswapna@gmail.com

Mr. D. Sai Srujan student Dept of Computer Science & Engineering from Vignan institute of technology and science. Email ID: dsaisrujan@gmail.com

Mr. V. Mahaveera Sai Teja student Dept of Computer Science & Engineering from Vignan institute of technology and science. Email ID: mahaveerasaiteja@gmail.com

Mr. P. Sandeep Kumar student Dept of Computer Science & Engineering from Vignan institute of technology and science. Email ID: sandeeppsk5@gmail.com

ABSTRACT:

Over the past few years, the number of digital text documents has grown rapidly. So, great importance has been put on the classification of documents into groups that describe the content of the documents. And online document classification is a problem in information and computer science. The proposed document classifier automates the manual classification of the process by using advanced data mining algorithms such as K-Nearest Neighbor, in this system we are finding out the similarity between two documents and grouping them together using describes data mining algorithm which is KNN and the similarity between the documents is determined using some of the several methods which are Jaccard index, cosine similarity, LSA, LDA, word embeddings. As there are some other existing algorithms for classification of documents which needs training data for training the model, also using vector space model (VSM). Which are supervised learning and time-consuming process. The proposed system is mainly based up on the textual content documents with special emphasis on text classification and working principle for text classification is based on similarity algorithms. In the classification of documents Python programming language are used.

Keywords: Natural language processing, Machine learning, LDA, LSA, Jaccard similarity, cosine similarity, word embeddings, soft cosine similarity.

I. INTRODUCTION

Text similarity has been one among the foremost important applications of tongue Processing. Two texts are said to be similar if they infer an equivalent meaning and have similar words or have surface closeness. Semantic similarity measures the degree of semantic equivalence between two linguistic items, be they concepts, sentences, or documents .A proper method that computes semantic similarities between documents will have an excellent impact upon different NLP applications like document classification, document clustering, information retrieval, MT , and automatic text summarization. There are different approaches to compute similarities between documents that use lexical matching, linguistic analysis, or semantic features. within the context of short texts, methods for lexical matching might work for trivial cases. But it is arguably not an efficient method because this method considers whether the words of short texts look-alike eg. in terms of distances, lexical overlap or largest common substring. Generally, linguistic tools like parsers and syntactic trees are used for brief text similarity, however, all the texts like tweets won't be parsable. A linguistically valid interpretation could be extremely far away from the intended meaning. So, semantic features are quite important in text mining. For semantic features, external sources of structured semantic knowledge like Wikipedia, WordNet, or word embeddings are used.

The main objective of this paper is to match the semantic similarities between short texts to maximise human interpretability. the essential idea to compute text similarities is by determining feature vectors of the documents then calculating the space between these features. A little distance between these features means a high degree of similarity, whereas an outsized distance means a coffee degree of similarity. Euclidean distance, Cosine distance, Word Mover, Jaccard distance are several the space metrics utilized in the computation of text similarity. This paper presents two different methods to get features from the documents or corpus: (1) using Tf-idf vectors and (2) using Word Embeddings and presents six different methods to compute semantic similarities between short texts: (1) Cosine, Jaccard's, Euclidean similarities with tf-idf vectors, (2) Cosine similarity with doc2vec vectors, (3) Soft cosine similarity with glove vectors, (4) word movers distance with doc2vec vectors, (5) cosine similarity with LSA vectors, (6) cosine similarity with LDA vectors. And developed a GUI model in

which, with pre-loaded folders of the text classification given a text file by calculating all the similarity between the uploaded text file or document to all the text documents in the existing folders and returns a optimal classified category.

II. LITERATURE SURVEY

Computation of similarity between the texts has been an important method of data analysis which can be further used in different NLP applications like information retrieval, sentiment analysis. So, there has been wide research for this computation using different features of texts and implementing different algorithms and metrics. Different case studies have also been carried out showing various applications of semantic similarity. Some of the researchers performed a comparative study to measure the semantic similarity between academic papers and patents, using three methods like the Jaccard coefficient, cosine similarity of tf-idf vector, and cosine similarity of log-tf-idf vectors. All these methods are corpus-based methods and they also performed a case-study for further analysis. explored the different applications of semantic similarity related to a semantic similarity calculation algorithm that used the large corpus to compare and analyse several words. This method used the Word2Vec model to calculate semantic information of the corpus and used Li similarity measure to compute similarity. And discussed three text similarity approaches: string-based, corpus-based and knowledge-based similarities and proposed different algorithms based on it.

The above works showed that different methods and metrics have already been carried out for this research. In this experiment, we have also introduced three different semantic similarity measures based on both corpus-based and knowledge-based methods. The basic concept was that the small features from text words are not enough, and the inclusion of word semantic information in the process can improve the method. The tf-idf vectors were used to gain information about the corpus or document in case of a corpus-based method i.e. cosine similarity using tf-idf vectors. The knowledge-based method included computation of word embedding for each text, and cosine and soft-cosine similarity metrics were used for similarity calculation between these word embedding vectors.

VECTORIZATION

tf-idf:

Term Frequency:

This measures how frequently a word occurs in a document. This highly depends on the length of the document and the generality of word, for example a very common word such as “was” can appear multiple times in a document and with the length of the document the count increases, so to normalise the value, we divide the the frequency with the total number of words in the document.

So, in worst case if the term does not exist in the document, then the TF value will be zero and in other extreme case, if all the words in the document are same, then it will be one. The final value of the normalised TF value will be in the range of [0 to 1]. 0, 1 inclusive.

Hence, we can formulate TF as follows.

- t—term
- d—document
- N—count of corpus
- corpus—the total document set

tf(t,d) = count of t in d / number of words in d

We use the TF score to know the important of a word in a document, then if we already calculate the TF value, why not use just TF to find the relevance between documents? why do we need IDF? Let me explain, though we calculated the TF value, still there are few problems, for example, stop words which are the most common words like “is, are” will have very high giving those general words a very high importance, that is the reason we also consider IDF alongside TF.

Document Frequency:

This measures the importance of document in whole set of corpora, this is very similar to TF. The only difference is that TF is frequency counter for a term t in document d, whereas DF id the count of **occurrences** of term t in the document set N. we consider one occurrence if the term consists in the document at least once, we do not need to know the number of times the term is present.

df(t) = occurrence of t in documents

To keep this also in a range, we normalise by dividing with the total number of documents. Our main goal is to know the informativeness of a term, and DF is the exact inverse of it. that is why we inverse the DF

Inverse Document Frequency:

IDF is the inverse of the document frequency which measures the informativeness of term t. When we calculate IDF, it will be very low for the most occurring words such as stop words. This finally gives what we want, a relative weightage.

idf(t) = N/df

Now there are few other problems with the IDF, in case of a large corpus, say 10,000, the IDF value explodes. So to dampen the effect we take log of IDF. In worst case, there could be no document which has 0 occurrence, and we cannot divide by 0. so to smoothen the effect we generally add 1 to the denominator.

idf(t) = log (N/(df + 1))

Finally, by taking a multiplicative value of TF and IDF, we get the TF-IDF score, there are many different variations of TF-IDF but for now let us concentrate on the this basic version.

tf-idf(t, d) = tf(t, d) * log(N/(df + 1))

TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)

Word Embeddings:

Word embeddings are vector representations of a word obtained by training a neural network on a large corpus. It has been widely used in text classification using semantic similarity. Word2vec is one of the most widely used forms of word embeddings. The word2vec takes text corpus as input and produces word vectors as output, which can be further used to train any other word to obtain its corresponding vector value. This word2vec model uses a continuous skip-gram model, based on the distributional hypothesis. An open-source library, Gensim provides different pre-trained word2vec models trained on different kinds of datasets like GloVe, google news, ConceptNet, Wikipedia, twitter. A pre-trained word embedding named Glove and Doc2vec were used as a model to create feature vectors for the dataset in this experiment.

LDA: Latent Dirichlet Allocation

It is an unsupervised generative model that assigns topic distributions to documents. At a high level, the model assumes that each document will contain several topics, so that there is topic overlap within a document. The words in each document contribute to these topics. The topics may not be known a priori, and need not even be specified, but the number of topics must be specified a priori. Finally, there can be words overlap between topics, so several topics may share the same words.

The model generates to latent (hidden) variables

1) A distribution over topics for each document

2) A distribution over words for each topic

After training, each document will have a discrete distribution over all topics, and each topic will have a discrete distribution over all words.

LSA: Latent Semantic Analysis

The Latent Semantic Analysis model is a theory for how meaning representations might be learned from encountering large samples of language without explicit directions as to how it is structured. The problem at the hand is not supervised, that is we do not have fixed labels or categories assigned to the corpus.

We form a document-term matrix, *A* using this transformation method (TF-IDF) to vectorize our corpus. And then we perform a Low-Rank Approximation using a Dimensionality reduction technique using a Truncated Singular Value Decomposition (SVD). Singular value decomposition is a technique in linear algebra that factorizes any matrix M into the product of 3 separate matrices: M=U*S*V, where S is a diagonal matrix of the singular values of M. Truncated SVD reduces dimensionality by selecting only the t largest singular values, and only keeping the first t columns of U and V. In this case, t is a hyperparameter we can select and adjust to reflect the number of topics we want to find. With these document vectors and term vectors, we can now easily apply measures such as cosine similarity to evaluate.

SIMILARITY

Cosine similarity:

In our academics, we have come across dot product and cross product of two vectors. Dot product of two vectors is calculated as multiplication of magnitudes of each vector and the cosine of angle between the vectors i.e.

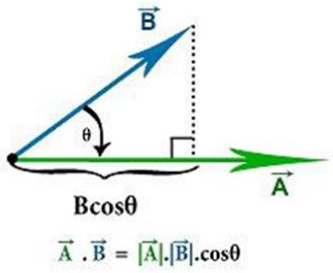

Fig.No.1. cosine similarity between two vectors

It is a similarity measure which measures the cosine of the angle between two vectors projected in a multi-dimensional plane. It is the judgment based on orientation rather than magnitude. Given two vectors of attributes, A and B, the cosine similarity is presented using a dot product and magnitude as:

similarity = cos(θ) = (A.B)/(|A||B|)

Since the ratings are positive our vectors will always lie in the first quadrant. So, we will get cosine similarity in the range [0,1], 1 being highly similar.

Soft cosine similarity:

A soft cosine or soft similarity between two vectors generalizes the concept of cosine similarity and considers similarities between pairs of features. The traditional cosine similarity considers the Vector Space Model (VSM) features as independent or completely different, while soft cosine measure considers the similarity of features in VSM as well. The similarity between a pair of features can be calculated by computing Levenshtein distance, WordNet similarity or other similarity measures. For example, words like "cook" and "food" are different words and are mapped to different points in VSM. But semantically they are related to each other. Given two N-dimensional vectors a and b, the soft cosine

similarity can be calculated as follows:

$$\text{soft\_cosine}_1(a, b) = \frac{\sum_{i,j}^{N} s_{ij} a_i b_j}{\sqrt{\sum_{i,j}^{N} s_{ij} a_i a_j} \sqrt{\sum_{i,j}^{N} s_{ij} b_i b_j}},$$

where s_{ij} = similarity *(*feature_i, feature_j*)*.

If there is no similarity between features ($s_{ii} = 1$, $s_{ij} = 0$ for $i \neq j$), the given equation is equivalent to the conventional cosine similarity formula.

Word Movers distance:

Word Mover's Distance (WMD) is based on recent results in word embeddings that learn semantically meaningful representations for words from local co-occurrences in sentences. WMD leverages the results of advanced embedding techniques like word2vec and Glove, which generates word embeddings of unprecedented quality and scales naturally to very large data sets. These embedding techniques demonstrate that semantic relationships are often preserved in vector operations on word vectors.

Word Mover's Distance (WMD), suggests that distances and between embedded word vectors are to some degree semantically meaningful. It utilizes this property of word vector embeddings and treats text documents as a weighted point cloud of embedded words. The distance between two text documents A and B is calculated by the minimum cumulative distance that words from the text document A needs to travel to match exactly the point cloud of text

document B. Refer image below.

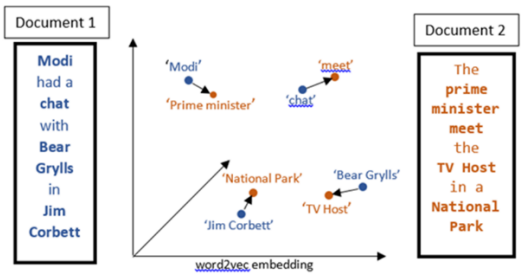

Fig. No.2 Word movers distance on two sentences

The WMD distance measures the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to "travel" to reach the embedded words of another document. WMD shows that this distance metric can be cast as an instance of the Earth Mover's Distance

III. SYSTEM ANALYSIS

EXISTING SYSTEM

In the existing text classifications algorithms will not be using the similarity techniques and for training the model they may use large amount of data, which consumes a lot of time and needs a lot of training data which may or may not be available in the real world environment, some of the classification algorithms are Bayesian classification techniques, regression models, neural networks (without word embeddings), decision tree models like random forest algorithm, As there are some other existing algorithms for classification of documents which needs training data for training the model, also using vector space model (VSM). which is not suggestable for large documents consumes more memory.

PROPOSED SYSTEM

Based on methods to compute feature vector and similarity measures, this paper presents three methods to compute similarities between short text data. As previously there will be majorly two parts for finding similarity between two texts the first one is to convert text to number form which are known as vectors this is to be done after preprocessing and then comparing the vectors of each document with each other and determining the most similar document for the uploaded document.

We are using cosine similarity, soft cosine similarity, Jaccard's similarity and Euclidean distance for finding out similarity between the vectors for converting text to vectors we are using TF-IDF weights, Count Vectors, Doc2vec, Glove word embeddings, LDA. And we are using LSA as principle component analysis with tfidf and word embeddings. After calculating all the similarities between the test document and existing documents the most similar

document is selected by majority classified text document from all the similarity measures. And thorough testing is done on all the proposed combinations of similarity models

We created a GUI for the given text similarities by which the test document selected from the GUI will be directly moved to the most similar document folder. As a sample we created four folder each containing text files of different topics and on selecting the test document from the project interface it calculates the most similarity between the uploaded document and all the documents in each folder and moves the uploaded file to most similar folder. i.e., in which the most similar documents found.

IV. IMPLEMENTATION

The project is done in python 3.6.8 language for obtaining faster results and for better readability "Scikit-learn", "Gensim" packages are used.

Preprocessing

we used the following methods for preprocessing the text data:

1. convert all the text to lower case
2. convert all the numbers into words
3. remove punctuation from the text data
4. remove white spaces from the data
5. remove nltk stopwords from the data
6. perform stemming on the remaining data

Now we will implement similarity models

(1). Cosine similarity, Jaccard Similarity, Euclidean distance with TF-IDF:

we directly use scikit-learn tfidf vectorizer for fast execution time the code is as follows:

"tfidf = TfidfVectorizer()

train = tfidf.fit_transform("X_train")

query = tfidf.transform("X_test")"

The above code returns the tfidf weights for each document.

Here we need to compare each document with test document and return the most similar document based on the similarity measure we use this process is done by "NearestNeighbors" method in scikit learn which is also known as unsupervised KNN.

"clf = NearestNeighbors(n_neighbors=2, metric = "cosine | jaccard | euclidean")

clf.fit(train, data["label"])

ep = clf.kneighbors(query)"

The above code returns the top 2 similar indices of the documents

(2). Cosine similarity with LSA:

As we have already discussed that LSA needs tfidf vectors from which LSA performs dimensionality reduction so the code is as follows:

"svd = TruncatedSVD(100)

lsa = make_pipeline(svd, Normalizer(copy= False))

X_train_lsa = lsa.fit_transform(train)

X_test_lsa = lsa.transform(query)"

This returns vectors on which dimensionality reduction is done and then we can proceed to find the cosine similarity between these obtained vectors as discussed above in (1).

(3). Cosine similarity with LDA

LDA uses only term frequency or count vectors as input and generates the LDA vectors the code is shown below:

"lda.fit(train2,data["label"])

train_lda = lda.transform(train2)

qu_lda = lda.transform(query2)"

where the train2, query 2 are the count vectors of the text document. And now we can compare these LDA vectors with help of cosine similarity as shown in (1)

(4). Cosine with Word Embeddings (Doc2Vev)

Here we are using Doc2vec word embeddings which is an extension to famous Word2vec embedding this uses "gensim" package first train the model and then save the trained model and reuse the saved model from next time, the code is as follows:

"model_dbow = Doc2Vec(dm=1, vector_size=300, negative=5, hs=0, min_count=2, sample = 0, alpha=0.025, min_alpha=0.001)

model_dbow.build_vocab(train_document)"

model_dbow.train(train_documents,total_examples=len(train_documents), epochs=30)"

and after converting the text into vectors the below code helps us to find most similar document by returning the index of the most similar document and calculated distance

"ep6 = model_dbow.docvecs.most_similar(positive=[model_dbow.infer_vector(test_doc)],topn=1)"

By default the similarity is calculated with cosine similariy.

(5). Cosine similarity with LSA + Word Embeddings

In the previous section we calculated cosine similarity directly with word embeddings but now we are going to extend this by applying dimensionality reduction with obtained word

embeddings which means applying LSA to the word embeddings this might results up to the improvement in accuracy of finding similarity between the texts , the code is obtained by just combining the word embeddings in (3) followed by LSA described in (2) and after find out the similarity between the documents with cosine similarity which is described in (1).

(6). Word movers distance

Word movers algorithm takes text as input directly in the form of word tokens. We are using Gensim for this approach, the code is given below:

"instance = WmdSimilarity(train_tokens, model_dbow,num_best=1)

ep7 = instance[tokenize_text(user)]"

this returns the dissimilarity between the documents. And internally this algorithm calculates the cosine distances between two vectors.

(7). Word Embeddings with Soft cosine similarity

we here are using a pretrained word embedding model which is directly available to download from gensim package

"w2v_model = api.load("glove-wiki-gigaword-50")"

We have used the GloVe data as pretrained model with 50 features in it. And now we can save the model and load it from local directory to fasten up the process as we have done in (4) and after use the soft cosine similarity which is also available in the same gensim package:

"bow_corpus = [dictionary.doc2bow(document) for document in train_tokens]

te_corpus = [dictionary.doc2bow(tokenize_text(user))]

docsim_index = SoftCosineSimilarity(bow_corpus, similarity_matrix, num_best=1)

ep8 = docsim_index[dictionary.doc2bow(tokenize_text(user))]"

as seen first we need to construct a similarity matrix for which the code is given below:

"similarity_index = WordEmbeddingSimilarityIndex(w2v_model)

similarity_matrix = SparseTermSimilarityMatrix(similarity_index, dictionary)"

now we have included all these similarity techniques in our proposed model and find out the most classified label from the results of above prescribed techniques and the uploaded text document will be automatically moved to the folder

V. RESULT AND DISCUSSION

As part of testing our model is tested among all the text document similarity techniques which are listed above. The testing is done on news 20 data set which is available from scikit-learn package for this testing we have considered only 4 class labels respectively and performed testing the results are as follows:

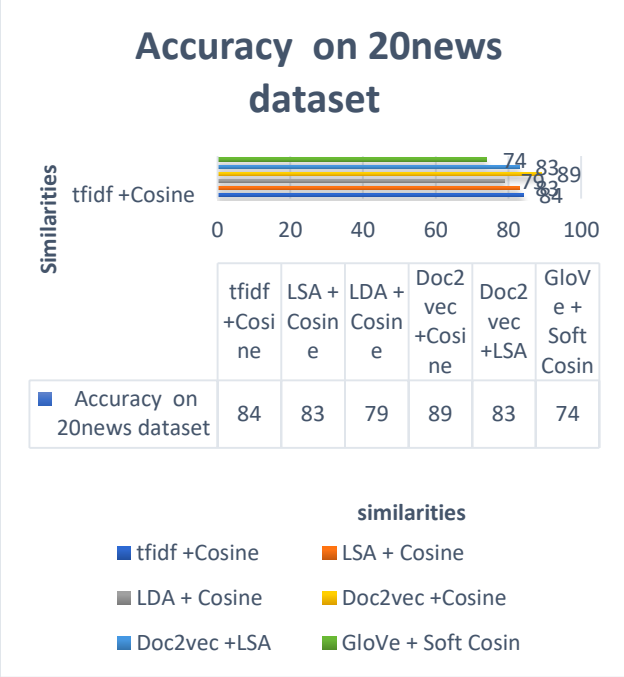

Fig.No.3 accuracy of text similarity models on news 20 dataset with 4 class labels

As observed the word embeddings has got top accuracy among all the text similarity models which are prescribed above as GUI concerns it will look like this:

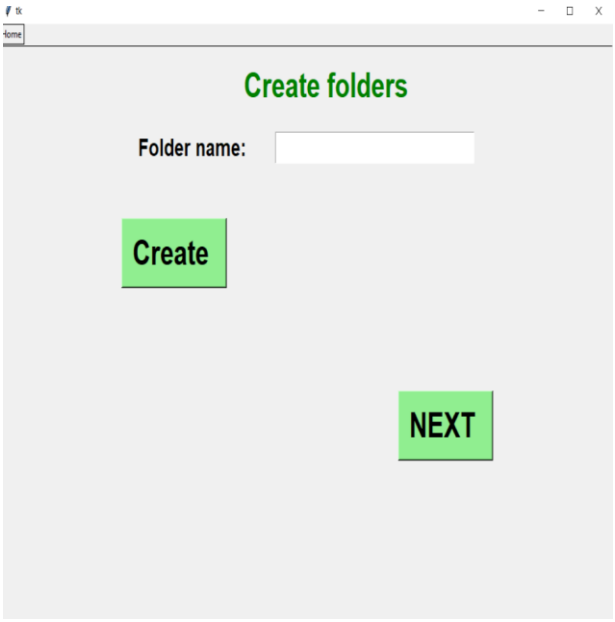

Fig No. 4 UI for creating folders

(i).Which we can create folders to for which you test document should be moved after classification

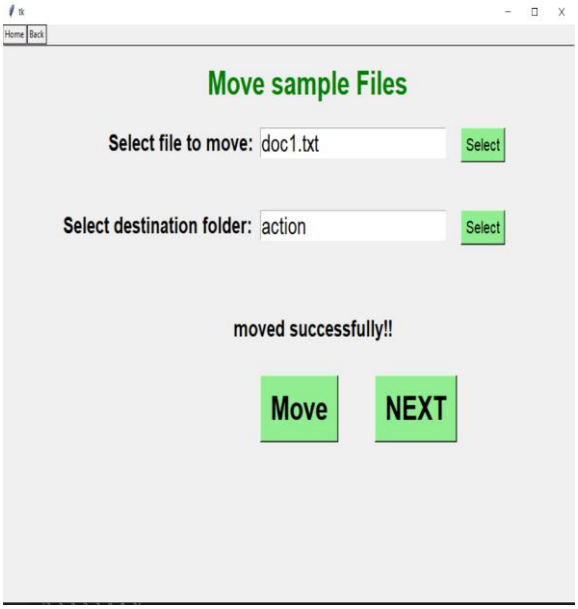

Fig. No. 5. UI for moving your sample files into your created folders

(ii).In this step you will now move the sample text files into your class labels or designated folders.

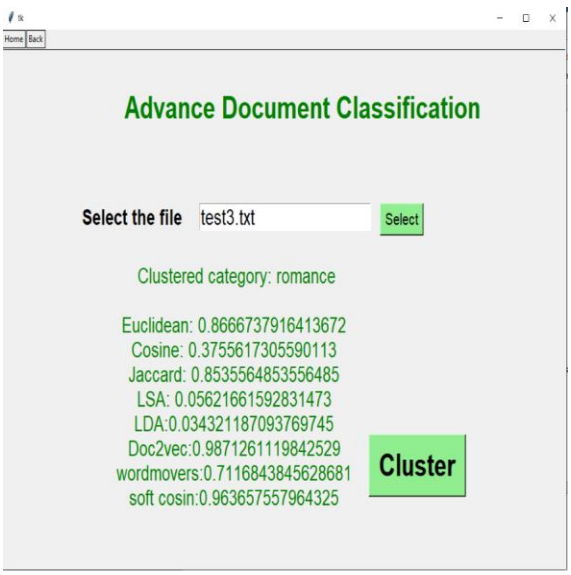

Fig.No.6. classified test document into "romance" folder with their respective similarity indices or distances.

(iii). As you can see now this user interface allows you to select a test (text) document for classification, as an example we have selected test3.txt file as an test document now the model calculates and compares the given text file with all the text files which are present in the folders and according to the similarities after which it will find the most similar documents folder and moves the file to the respective classified folder or most similar folder.

VI. CONCLUSION AND FUTURE SCOPE

We can conclude that all the text similarity model are classifying the text documents almost to the class labels defined to it. As our testing results also are quite satisfying. Most of the similarity models does not need any huge amount of training data but the more the samples the more the accuracy. We can also use this project as to find the similarity between two given sentence with slight modifications. In this research paper, we performed a comparison between six different approaches for measuring the semantic similarity between two short text news articles. 20news data sets were wont to verify the experiment as far it goes only up to 89% accuracy. The six approaches are Cosine similarity with tf-idf vectors, cosine similarity with doc2vec vectors, soft cosine similarity with GloVe pretrained vectors, LSA with tf-idf vectors, LDA with Count Vectors, Word Mover's distance, applying LSA to doc2vec vectors. All of those three methods had shown promising results. Among these six vectors, cosine with doc2vec had the very best accuracy when the results were cross-validated and therefore the newsgroup of a news story and its corresponding most similar article were compared. The most similar documents given by the tactic are easy to interpret, which makes it easier to use in several sorts

of information retrieval methods. The accuracy of the opposite two methods are often increased by using the Doc2Vec model rather than the Word2Vec model, which represents a document as a vector and doesn't average the word vectors of a document. This model are often implemented for further improvement of the methods.

VII. REFERENCES

1. Sitikhu, Pinky, et al. "A Comparison of Semantic Similarity Methods for Maximum Human Interpretability." *2019 Artificial Intelligence for Transforming Business and Society (AITB)*. Vol. 1. IEEE, 2019.
2. Trstenjak, Bruno, Sasa Mikac, and Dzenana Donko. "KNN with TF-IDF based framework for text categorization." *Procedia Engineering* 69 (2014): 1356-1364.
3. Gurusamy, Vairaprakash & Kannan, Subbu. (2014). Preprocessing Techniques for Text Mining.
4. Shehata, Shady & Karray, Fakhri & Kamel, Mohamed S.. (2007). Enhancing Text Clustering Using Concept-based Mining Model. 1043 - 1048. 10.1109/ICDM.2006.64.
5. https://medium.com/@adriensieg/text-similarities-da019229c894
6. https://nlp.town/blog/sentence-similarity/
7. Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global Vectors for Word Representation." Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2014): n. pag. Crossref. Web.
8. https://radimrehurek.com/gensim/auto_examples/core/run_similarity_queries.html#sphx-glr-auto-examples-core-run-similarity-queries-py
9. https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html#sphx-glr-auto-examples-tutorials-run-doc2vec-lee-py
10. https://markroxor.github.io/gensim/static/notebooks/WMD_tutorial.html
11. http://mccormickml.com/2016/11/04/interpreting-lsi-document-similarity/
12. https://medium.com/mlreview/implementing-malstm-on-kaggles-quora-question-pairs-competition-8b31b0b16a07
13. https://www.thinkinfi.com/2019/10/doc2vec-implementation-in-python-gensim.html
14. http://www1.se.cuhk.edu.hk/~seem5680/lecture/LSI-Eg.pdf

15. https://markroxor.github.io/gensim/static/notebooks/WMD_tutorial.html
16. https://www.machinelearningplus.com/nlp/cosine-similarity/
17. https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e
18. http://poloclub.gatech.edu/cse6242/2018spring/slides/CSE6242-820-TextAlgorithms.pdf
19. https://github.com/makcedward/nlp/blob/master/sample/nlp-word_embedding.ipynb
20. http://stefansavev.com/blog/beyond-cosine-similarity/
21. https://towardsdatascience.com/word-movers-distance-for-text-similarity-7492aeca71b0
22. https://www.renom.jp/index.html?c=tutorial
23. https://iq.opengenus.org/document-similarity-tf-idf/
24. https://www.kaggle.com/ktattan/lda-and-document-similarity
25. GitHub - kk7nc/Text_Classification: Text Classification Algorithms: A Survey
26. https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/

AUTHORS

Mrs. Y. Swapna Assistant Professor Dept of Computer Science & Engineering from Vignan institute of technology and science, Telangana, INDIA.
Email ID: yenugulaswapna@gmail.com

Mr. D. Sai Srujan student Dept of Computer Science & Engineering from Vignan institute of technology and science, Telangana, INDIA.
Email ID: dsaisrujan@gmail.com

Mr. V. Mahaveerasaiteja student Dept of Computer Science & Engineering from Vignan institute of technology and science,Telangana, INDIA.
Email ID: mahaveerasaiteja@gmail.com

Mr. P. Sandeep kumar student Dept of Computer Science & Engineering from Vignan institute of technology and science, Telangana, INDIA.
Email ID: sandeeppsk5@gmail.com