JSS MAHAVIDYAPEETHA
JSS Science and Technology University

"Impact Of Different Embeddings On Text Classification"

A technical project report submitted in partial fulfillment of the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE & ENGINEERING

BY

Sharath H N	01JST18CS125
Anusha S G	01JST18CS015
Milind M	01JST18CS060
Manju Charan M B	01JST18CS056

Under the Guidance of

Dr. ANIL KUMAR K M

Associate Professor
Department of Computer Science & Engineering
JSS STU Mysore

2021-22

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

JSS MAHAVIDYAPEETHA

JSS Science and Technology University

"Impact Of Different Embeddings On Text Classification"

A technical project report submitted in partial fulfillment of the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

BY

Sharath H N	01JST18CS125
Anusha S G	01JST18CS015
Milind M	01JST18CS060
Manju Charan M B	01JST18CS056

Under the Guidance of

Dr. ANIL KUMAR K M

Associate Professor
Department of Computer Science & Engineering
JSS STU Mysore

2021-22

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

JSS MAHAVIDYAPEETHA

JSS Science and Technology University

CERTIFICATE

This is to certify that the work entitled "**Impact Of Different Embeddings On Text Classification**" is a bonafide work carried out by **SHARATH H N (01JST18CS125), ANUSHA S G (01JST18CS015), MILIND M (01JST18CS060), MANJU CHARAN M B (01JST18CS056)** in partial fulfillment of the award of the degree of Bachelor of Engineering in Computer Science and Engineering for the award of Bachelor of Engineering by JSS Science and Technology University, Mysuru, during the year 2021-2022. The project report has been approved as it satisfies the academic requirements with respect to project work prescribed for the Bachelor of Engineering degree in Computer Science and Engineering.

Under the guidance of

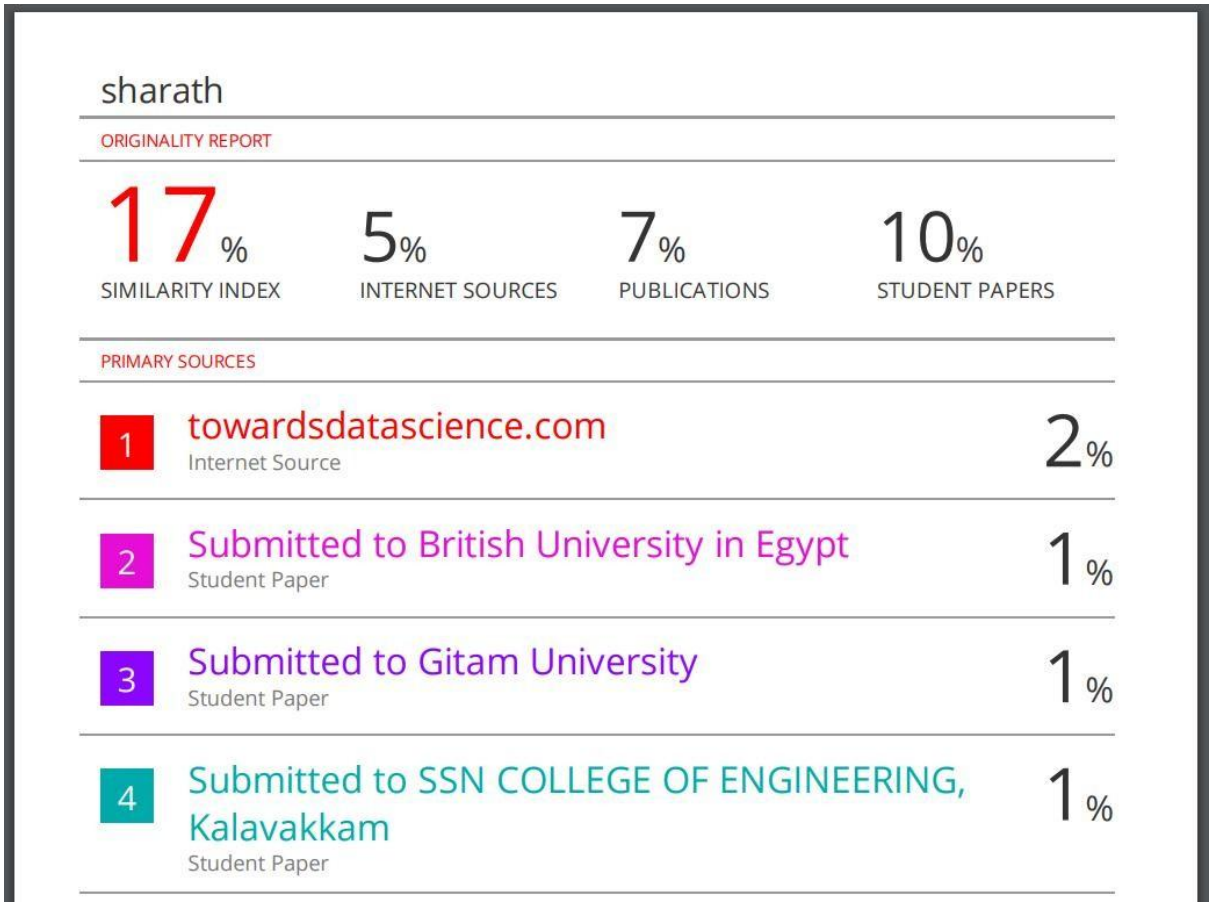
Dr. Anil Kumar K M.
Associate Professor
Dept. of CS &E
JSS STU Mysore - 06

Head of the Department

Dr. Srinath S.
Assoc. Prof. and HOD
Dept. of CS &E
JSS STU Mysore - 06

Name of Examiner	Signature with Date
1.	.....
2.	.....
3.	.....

Certificate of Plagiarism Check



sharath

ORIGINALITY REPORT

17%	5%	7%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	towardsdatascience.com Internet Source	2%
2	Submitted to British University in Egypt Student Paper	1%
3	Submitted to Gitam University Student Paper	1%
4	Submitted to SSN COLLEGE OF ENGINEERING, Kalavakkam Student Paper	1%

DECLARATION

We do hereby declare that the project titled **"Impact Of Different Embeddings On Text Classification"** is carried out by **SHARATH H N (01JST18CS125), ANUSHA S G (01JST18CS015), MILIND M (01JST18CS060), MANJU CHARAN M B (01JST18CS056)**, under the guidance of **DR. Anil Kumar K M**, **Associate Professor**, Department of Computer Science and Engineering, JSS Science and Technology University, Mysuru, in partial fulfillment of the requirement for the award of Bachelor of Engineering by JSS Science and Technology University, Mysore, during the year 2021-2022.

We also declare that we have not submitted this dissertation to any other university for the award of any degree or diploma courses.

Date:
Place: Mysore

Team Members	**Signature**
Sharath H N (01JST18CS125)	_____
Anusha S G (01JST18CS015)	_____
Milind M (01JST18CS060)	_____
Manju Charan M B (01JST18CS056)	_____

ACKNOWLEDGEMENT

It gives us immense pleasure to write an acknowledgment to this project, a contribution of all the people who helped to realize it. We extend our deep regards to Dr. S.B. Kivade, Honorable Principal of JSS Science and Technology University, for providing an excellent environment for our education and his encouragement throughout our stay in college. We would like to convey our heartfelt thanks to our HOD, Dr. Srinath S, for giving us the opportunity to embark upon this topic. We would like to thank our project guide, Dr. Anil Kumar K M for his invaluable guidance and enthusiastic assistance and for providing us support and constructive suggestions for the betterment of the project, without which this project would not have been possible. We appreciate the timely help and kind cooperation of our lecturers, other staff members of the department, and our seniors, with whom we have come up all the way during our project work without whose support this project would not have been successful. Finally, we would like to thank our friends for providing numerous insightful suggestions. We also convey our sincere thanks to all those who have contributed to this learning opportunity at every step of this project.

ABSTRACT

Text representation has always been a major problem in natural language processing. Features such as frequency, position in the text, and co-occurrence of words are used extensively to design various text representation techniques. There are different techniques currently used for text representation. Some of them are a bag of words, TF-IDF, word2vec, GloVe, and, Fast text. As a part of this project, we have performed a comparative analysis of how different embeddings behave when subjected to different classification algorithms. Our experimental results conclude that bag of words and TF-IDF have higher accuracy in ML classifiers and word2vec, GloVe, and, Fast text has performed extremely well for neural network-based classifiers.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1. INTRODUCTION

1.1 Natural Language Processing

The discipline of "artificial intelligence" (AI) known as "natural language processing" (NLP) in computer science is more specifically focused with giving Computers can now comprehend human language in the form of text or audio data. NLP powers a variety of computer systems, including those that translate text across languages, respond to spoken requests, and sum up voluminous volumes of text quickly—even in real-time.(Yuwono, 2015.)

1.2 Tasks Associated With NLP

Since human language is so ambiguous, it is very difficult to create algorithms that can accurately determine the intended meaning of text or audio data. For natural language-driven applications to be useful, programmers must teach them to recognize and understand correctly from the outset the linguistic quirks that take humans years to learn.(*What Is Natural Language Processing? | IBM*, 2020.)

A number of NLP tasks dissect human text and speech input in ways that assist the computer understand what it is absorbing. Several of these duties consist of the following:

1.2.1 Speech recognition

Its goal is to accurately translate speech input into written data. Any program that responds to voice commands or inquiries must use speech recognition. Speech recognition is particularly difficult due to how rapidly individuals speak, how their words are slurred together, how their tone and emphasis fluctuate, how they speak in a variety of dialects, and how rarely they use proper syntax.

1.2.2 Named entity recognition

Words or phrases are recognized by NEM as helpful entities. NEM identifies "Kentucky" as a place or "Fred" as the name of a guy. There are special words that represent particular entities that are more informative and have a specific context that may be found in any text source. Named entities are phrases that reflect real-world items like people, locations, organizations, and so forth, which are frequently identified by proper names(*Entity Types – Quick and Easy Documentation | Community*, 2021.). Finding them

by looking at the noun phrases in text documents could be a foolish strategy. In information extraction, named entity recognition (NER), also known as entity chunking or extraction, is a well-liked method for locating and classifying named entities into a variety of predetermined categories.

1.2.3 Coreference resolution

Finding out if and when two words refer to the same thing is the issue at hand. The most typical example is figuring out who or what a certain pronoun refers to (e.g., "she" = "Mary"). Still, it may also require figuring out a metaphor or idiom that is used in the text (e.g., when "bear" refers to a big, hairy person rather than an animal). Finding all linguistic statements (also known as mentions) that relate to the same real-world object in a given text is the problem of coreference resolution (CR)(*Nlp Coreference Resolution*, 2018.).

1.2.4 Natural language generation

It is often defined as the other of speech reputation or speech-to-textual content. It is the challenge of placing dependent facts into human language. The natural language era is the system of growing significant terms and sentences with inside the shape of herbal language, consistent with Artificial Intelligence Natural Language Processing Fundamentals. In essence, it generates narratives at a fee of lots of pages in step with 2d that appropriately describe, encapsulate, or give an explanation for incoming dependent records. NLG software program can write, however, it can't read. Natural Language Understanding (NLU) is the thing of NLP that translates human language and converts its unstructured records into dependent records that computer systems can comprehend.

1.2.5 Sentiment analysis

It makes an effort to draw out of the text's actual substance subjective traits, attitudes, emotions, sarcasm, and mistrust. A natural language processing (NLP) technique called sentiment analysis is used to determine whether or not records are positive, negative, or neutral. Sentiment analysis is frequently carried out on text records to help organizations exhibit logo and product sentiment in customer feedback and identify customer demands.

1.3 Aim

To perform a comparative analysis on the performance of different classification algorithms when different word embeddings are used.

1.4 Objectives

The objectives of the proposed project work are to

1. Create different word embeddings
2. Build a sentiment analysis model
3. Perform a comparative analysis of different models
4. Compare the performance of different embeddings
5. Build a web application to test our work

1.5 Introduction to the Problem Domain

Artificial intelligence has been improved tremendously while not having to vary the underlying hardware infrastructure. Users can run a man-made intelligence program in an old automatic data processing system. On the opposite hand, the beneficial effect of machine learning is unlimited. Language Processing is one of all the branches of AI that provides machines the flexibility to read, understand, and deliver meaning.

Texts and speeches make up the majority of unstructured data types. Although there is a lot of it, it might be challenging to extract relevant information. If not, mining the knowledge may take a long time. Both speech and writing carry a wealth of information. It's because speech and writing are our two main forms of communication as intelligent creatures. NLP can do jobs like sentiment analysis, cognitive assistant, span filtering, spotting false news, and real-time language translation on our behalf and evaluate this data.

Understanding messages, comparing them, classifying them, and creating current meaningful content are all aspects of natural language processing. The depiction of text has always been difficult. It's crucial to transform text that can be read by humans into a machine-readable format. Numerous text representation algorithms make heavy use of features including frequency, location within the text, and word co-occurrence. One of them is word embedding.

Word embedding may be a technique for representing words using continuous real-number vectors of fixed length. It converts a word from a vocabulary into a latent vector space containing terms with comparable contexts. A word is changed into a vector that

sums up both the syntactic and semantic information of the word through word embedding. Bengio et al. created the term word embeddings(Bengio et al., 2003) in 2003, after training a neural language model together with the model's parameters. Later in 2013, Mikolov et al.(Mikolov et al., 2013) were answerable for bringing word embedding to the forefront by developing word2vec, a toolbox that permits the training and application of pre-trained embeddings which has contributed to the progress of NLP. Facebook's AI Research (FAIR)(Bojanowski et al., 2016) team developed fastText in 2020 for learning word embeddings and text classification.

1.6 Existing Solution Methods

Existing research involves comparing either only the machine learning algorithms or only the deep learning networks.

1.7 Proposed Solution Methods

The proposed solution is to perform a comparative analysis on both machine learning-based classifiers and neural network classifiers.

1.8 Time Schedule

Figure 1 and Figure 2 describe the time schedule followed by use in completion of the project.

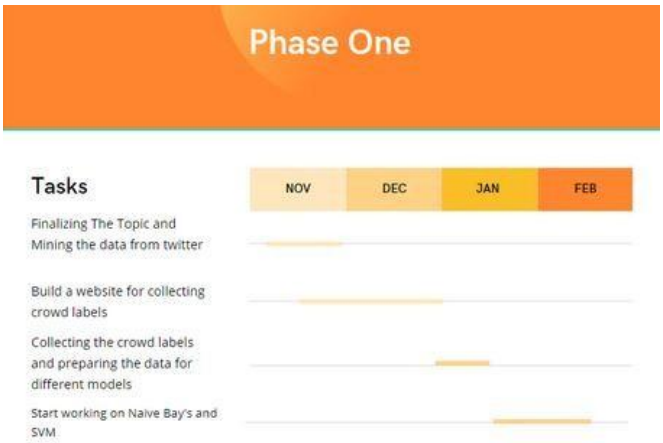


Figure 1. Phase one project timeline

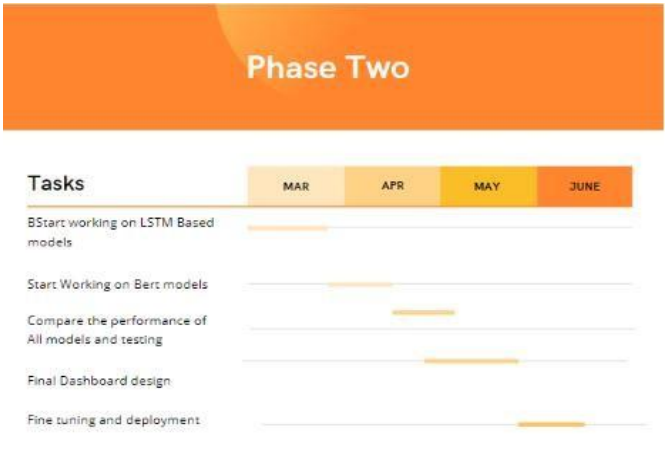


Figure 2. Phase two project timeline

Chapter 2. LITERATURE SURVEY

This section provides a detailed summary of the papers that were reviewed before the project implementation began. The papers mentioned in this section provide insight and simple changes that can lead to great performance improvements. This section also summarizes comparative analysis performed on different datasets and we have adopted some in our project as well.

In the paper titled 'Twitter Sentiment Analysis Using Supervised Machine Learning'(*Twitter Sentiment Analysis Using Supervised Machine Learning | SpringerLink*, n.d.), Nikhil Yadav, Omkar Kudale, Srishti Gupta, and Aditi Rao performed a comparative analysis of the IMDB dataset. They have applied Naive Bayes, Logistic Regression, and Support Vector Machine and have compared the results. Table 1 has the compiled results of the accuracy achieved by the authors. They have concluded that linear SVC gives the highest accuracy of 83.71 and has outperformed all other models like logistic regression having an accuracy of 82.47 and Naive Bayes with an accuracy of 80.61.

Table 1. Accuracy results for IMDB dataset

Method	IMDB dataset (accuracy)
Multinomial Naive Bayes	0.8061
Logistic Regression	0.8247
Support Vector Machine	0.8371

Wen Zhang, Taketoshi Yoshida, and Xijin Tang in the paper(Zhang et al., 2011), have presented a comparative study of TF/IDF, LSI, and multi-words as indexing methods for text representations on two datasets: the Chinese corpus, TanCorpV1.0, and for the English corpus, Reuters-21578 distribution 1.0. To evaluate the performances of the said indexing methods. A support vector machine was used to categorize the documents. In both the document sets, experimental findings showed that LSI performs better than other approaches in document classification. LSI offers the best retrieval performance for English documents and has both good semantic and statistical quality, as opposed to the argument that it lacks the discriminative capacity for indexing.

In the paper(Park et al., 2020), they have applied various machine learning-based classification techniques to IMDb dataset. They used the BagOfWords, TF-IDF, and Word2Vec and compared the impact on the performance. Bar graph in Figure 3 shows a comparison of different classification models and different word embeddings.

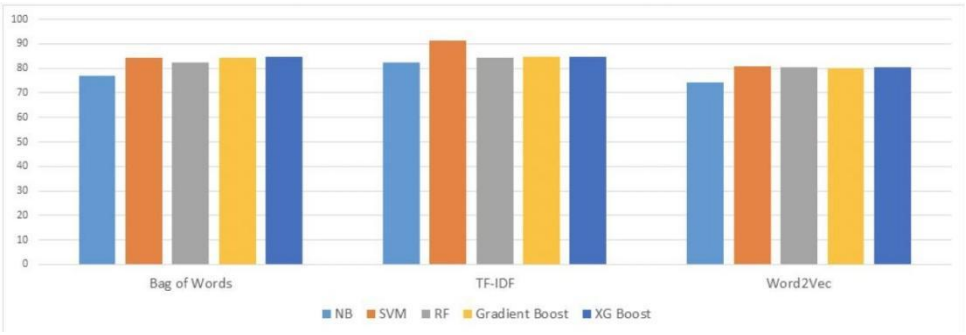


Figure 3. The bar chart of the performance for each model

With the Reuters-21578 dataset, Mukund N. Helaskar and Sheetal S. Sonawane performed a comparative analysis of text embedding techniques such as Bag Of Words and Word2Vec in their paper(Helaskar & Sonawane, 2019). With Support Vector Machine as the training model, Word2Vec outperformed BagOfWords. BoW model improves with the number of features but Word2Vec performed well(89.27%) even with only 100 features, but the improvement with the increase in the number of features is insignificant.

R. Janani and S. Vijayarani analyzed the performance of three important word embedding algorithms: Word2Vec, GloVe, and WordRank algorithm on 3 datasets: Reuters dataset, 20Newsgroup dataset, and 5AbstractsGroup in their paper (Janani & Vijayarani, 2019)Word2Vec+skipgram and GloVe algorithms for the Reuter dataset showed better accuracy than the rest. For the 20Newsgroup Dataset, Word2Vec+CBoW showed better accuracy. The accuracies in 20Newsgroup Dataset are slightly lower than that of the Reuter dataset as the Word2Vec algorithm performs better when the size of the corpus is big. For the 5AbstractsGroup dataset, GloVe performed better. The accuracies are higher than the other two datasets. Figure 4 shows the comparison between the accuracy of different models on 3 different datasets used in the research.

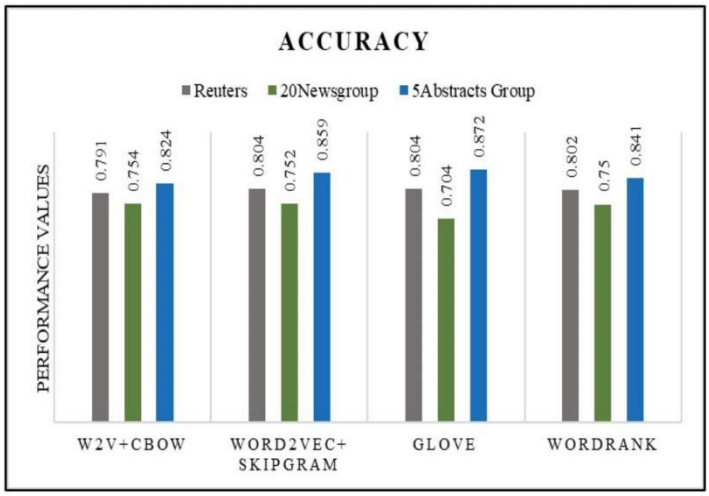


Figure 4. Accuracy values of three datasets

In 2020, Rafly Indra Kurnia, Yoshua Daniel Tangkuman, and Abba Suganda Girsang performed a classification of play store comments into 5 different classes(ratings from 1 to 5) using word2Vec and SVM classifier and presented this in their paper(Kurnia, 2020). They obtained the best F1 score of 0.795 when the reviews were preprocessed. They also performed the same experiment by undersampling but the results weren't significantly different from before.

In the paper(Wang et al., 2020), the authors examined context-based embeddings: both ELMo and BERT with 2 encoders: CNN and BiLSTM. Figure 5 shows the accuracy of CNN and BiLSTM on 4 different datasets. The experiments showed that CNN outperformed BiLSTM in most situations, especially for document context-insensitive datasets and BERT outperformed ELMo for long document datasets. In this type of downstream classification job, it is preferable to use pre-trained weights to initialize the embedding layer, but the amount of performance improvement is primarily controlled by the dataset features. Concatenating sub-word pre-trained embeddings for performance increase is preferable to focusing on downstream model selection based on dataset features.

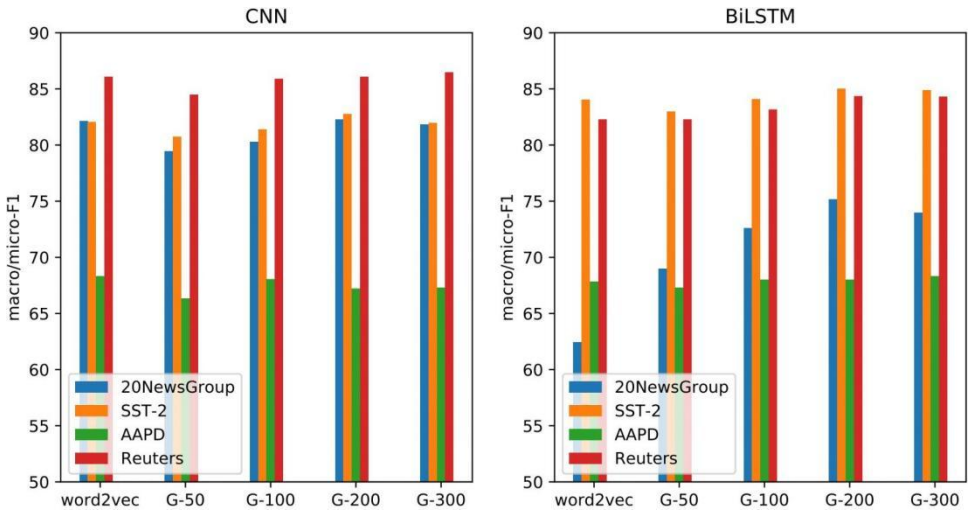


Figure 5. Different embeddings (word2vec and GloVe) versus macro (20NewsGroup and SST-2) and micro (AAPD and Reuters) -F1 with CNN and BiLSTM.

Different word embedding methods such as word2vec, doc2vec, tfidf, and embedding layers were trained by different Deep learning models such as MLP, CNN, LSTM, GRU, 2 layer GRU, TextCNN, CNNGRU, and CNNGRU_Merge for automatic text classification of Sohu news dataset from Sougo Lab in the paper(Li & Gong, 2021) and the Experimental findings have shown that 2 layer GRU with pre-trained Word2Vec embeddings has the highest accuracy of 93%. CNNGRU and CNNGRU_Merge have 92.7% and 92.6% accuracy, respectively, with pre-trained data. LSTM has the least accuracy at 87.2%.

In the paper(*Multi-Class Review Rating Classification Using Deep Recurrent Neural Network | SpringerLink*, 2014.), Junaid Hassan and Umar Shoaib have performed a comparative analysis of the IBM 50k and Hotel Review dataset Table 2 gives an insight into the datasets used by the authors. They have applied Recurrent neural network-based architecture like Vanilla RNN, Long Short Term Memory commonly known as LSTM, and Gated Neural Units (GRU) and have compared the results. Figure 6 describes the architecture used by the authors.. They have concluded that GRU has outperformed all other models and is suited for multi-class classification of short texts. They have used custom word embeddings as compared to traditional Google news embeddings and the results are in their favor.From Table 3 they concluded that their DSWE-GRNN performed slightly better than some of the common baseline models.

Table 2. Statistical information of IMDB and Hotel Reviews datasets

Data Set	Doc	Train	Test	W/Doc	Vocabulary
IMDB	50000	25000	25000	129	119022
HotelReview	14985	11915	2979	52	20310

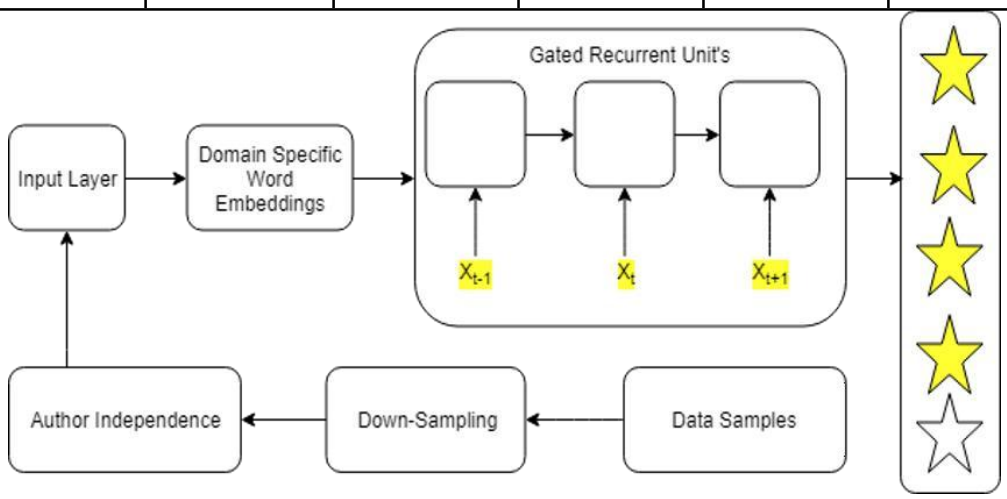


Figure 6. The graphical representation of the proposed model (DSWE-GRNN)

Table 3. Comparison with Baseline Methods

Method	IMDB dataset (accuracy)	Hotel Reviews(accuracy)
WE-SimpleNN	0.8675	0.8024
CNN	0.8645	0.7877
LSTM	0.8360	0.8092
CNN-LSTM	0.8268	0.7843
DSWE-GRNN	**0.8780**	**0.8132**

Chapter 3. SYSTEM REQUIREMENTS AND ANALYSIS

This section lays out the many requirements that had to be met before the System design and implementation process could begin. The functional and non-functional criteria are mentioned in this section. System Requirements, which include both hardware and software requirements, are stated.

3.1 Software Requirements

3.1.1 Python

Python is a powerful, interactive, object-oriented, interpreted scripting language. Python has been created to be very readable. It has fewer syntactic structures than other languages and usually employs English terms rather than punctuation. Python has an autonomous memory management system and a dynamic type system. It includes a sizable and thorough standard library and supports a variety of programming paradigms, including imperative, functional, procedural, and object-oriented.

3.1.2 Google Colab

A free Jupyter notebook environment called Colaboratory(*Google Colaboratory*,2017.) (often referred to as Colab) operates on the cloud and keeps its notebooks on Google Drive. Colab was once a Google internal project. Attempts were made to open-source all the code and work more directly upstream, which resulted in the creation of the "Open in Colab" Chrome extension, but these efforts finally failed, and internal Colab development continued. The Julia language may also be used on Colab (together with, for example, Python and GPUs; Julia can also be used on Colab with Google's tensor processing units).

3.2 Hardware Requirements

The training of the models used in this project was done on the Google Colab platform and utilized a remote system that is a part of the Google Cloud Platform. The hardware requirements for this project are mainly the system on which the code was written and used to access the remote system using a browser. The hardware requirements are as follows:

1. Processor 64 bit, 4 core, 2.40 GHz

2. RAM: 8 GB
3. GPU: 8GB
4. Input Device: Standard keyboard and mouse
5. Output Device: Monitors with decent resolution
6. Internet: A stable connection to the internet for Google Colab

3.3 Functional Requirements

- Classification with significant accuracy
- Classification results should be displayed in a tolerable amount of time
- The classification process should not consume a large number of computing resources

3.4 Non-Functional Requirements

- Classification results should be conveyed in a manner interpretable by average humans
- The results can be conveyed in charts, graphic meters, etc.

Chapter 4. TOOLS AND TECHNOLOGIES

This chapter goes over the tools and technologies we used in our project in detail. This covers the Python libraries and frameworks we utilized, as well as a full list of the datasets we used to complete our project. This part also goes through the library requirements that led to specific libraries being selected for the implementation of our project.

4.1 Python Libraries

Following Python libraries are used in our research project

4.1.1 Tweepy

You may access the Twitter API with Python quite conveniently using Tweepy, an open-source Python package(*Tweepy*,2017.). In addition to handling numerous implementation concerns, such as Data encoding and decoding, transparently, Tweepy contains a collection of classes and methods that represent Twitter's models and API endpoints.

4.1.2 Sklearn

Scikit-learn is a free machine learning package for Python that was originally known as scikits.learn (*Scikit-Learn: Machine Learning in Python — Scikit-Learn 1.1.1 Documentation*, 2022.).

The majority of Scikit-code learn's is written in Python, and it makes considerable use of NumPy for high-performance array and linear algebra operations. To enhance performance, several fundamental algorithms are also written in Cython. Support vector machines, logistic regression, and linear support vector machines are implemented using wrappers built-in Cython for LIBSVM and LIBLINEAR, respectively. It might not be viable to expand these functions with Python in such circumstances. Scikit-learn works nicely with many other Python libraries, including SciPy, Pandas data frames, NumPy for array motorization, Matplotlib and Plotly for graphing, and many more.

4.1.3. TensorFlow

TensorFlow is an open-source toolkit for data flow graph-based numerical

computing. The graph's nodes stand in for mathematical processes, while its edges stand-in for the multidimensional data arrays (tensors) that are sent between them. With a single API, you may use the flexible architecture to distribute compute to one or more CPUs or GPUs in a desktop, server, or mobile device. It is employed in machine learning programs like neural networks. At Google, it is frequently used in place of DistBelief, a closed-source predecessor, in both research and production.

Stateful dataflow graphs are used to represent TensorFlow calculations. The actions that these neural networks carry out on multidimensional data arrays are where Tensor-Flow gets its name. Tensors are the name given to these arrays. TensorFlow further separates the description and execution of computations by having them occur in different locations: a graph describes the operations, but the operations only take place during a session. Sessions and Graphs are made separately. A session is like a building site, and a graph is like a plan.

In order to do expensive operations like matrix multiplication outside of Python, utilizing extremely efficient code built in another language, we often utilize libraries like NumPy. Unfortunately, returning to Python for each operation can still result in significant costs. If you wish to conduct calculations on GPUs or in a distributed fashion, where data transport might be expensive, this penalty is especially problematic. TensorFlow goes one step farther to eliminate this cost while still doing its labor-intensive tasks outside of Python. TensorFlow enables us to design a network of interconnected operations that execute totally outside of Python rather than conducting a single costly action independently of Python.

4.1.4 Numerical Python (NumPy)

The core Python module for scientific computing is called NumPy. It is a Python library that offers a multidimensional array object, various derived objects (like masked arrays and matrices), and a variety of routines for quick operations on arrays, such as discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and much more. The n-darray object is at the center of the NumPy package. This contains homogenous n-dimensional arrays of data kinds, with many operations carried out in compiled code for speed. The conventional Python sequences and NumPy arrays differ in a number of significant ways.

- In contrast to Python lists, NumPy arrays have a predetermined size at construction (which can grow dynamically). An n-size array's change will result in the creation of a new array and the deletion of the old one.
- A NumPy array's elements must all be the same data type in order to share the same amount of memory. The exception: arrays of (Python, including NumPy) objects are possible, enabling arrays with various-sized members.
- NumPy arrays make it easier to do complex mathematical and other operations on enormous amounts of data. As opposed to utilizing Python's built-in sequences, such operations are often carried out more quickly and with less code.
- NumPy arrays are used by a rising number of Python-based scientific and mathematical tools, however, they generally support Python sequence.

In other words, simply understanding how to utilize Python's built-in sequence types is insufficient; one also has to understand how to use NumPy arrays in order to effectively use a large portion (perhaps even the majority) of today's scientific/mathematical Python-based software.

4.1.5 Matplotlib

For the Python programming language and its NumPy extension for numerical mathematics, there is a graphing library called Matplotlib. For integrating charts into programs utilizing all-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+, it offers an object-oriented API. It is not recommended to utilize the procedural "pylab" interface, which is built on a state machine (similar to OpenGL) and was created to closely mimic the MATLAB interface. Python programs are used to generate 2D graphs and charts. It provides a module called pyplot that simplifies the charting process by offering tools to adjust line styles, font attributes, axis formatting, etc. Histograms, bar charts, power spectra, error charts, etc. are just a few of the many graphs and plots it offers. It is used with NumPy to provide an environment that serves as a strong open-source replacement for MatLab. Also compatible with wxPython and PyQt, two graphical toolkits are this one. The pyplot package offers a MATLAB-like interface for basic graphing, especially when used with IPython. For the power user, there are two options for controlling line styles, font settings, axis properties, etc.: an object-oriented interface or a collection of MATLAB-friendly methods.

4.1.6 Keras

Python-based open-source Keras is a neural network library. It may be used with TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML as a foundation. Its user-friendliness, modularity, and extensibility are its main design goals as it aims to facilitate quick experimentation with deep neural networks. It was created as a component of the ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) research initiative.

In order to make dealing with picture and text data easier and to reduce the amount of coding required to create deep neural network code, Keras includes multiple implementations of widely used neural network building blocks such as layers, goals, activation functions, and optimizers. The Slack channel and GitHub bugs page serve as community support channels for the code, which is stored on GitHub. Keras supports convolutional and recurrent neural networks in addition to conventional neural networks. Additionally, it permits the use of distributed deep learning model training on clusters of GPUs and TPUs, primarily in combination with CUDA.

4.2 Dataset

We use the following datasets for our experiment. Table 4 gives an overall summary of the datasets we have used.

4.2.1.Sentiment140

As mentioned in the paper by Go et al.(Go et al., 2009.), Sentiment140 is a collection of 1.6 million positive and negative tweets, scraped from Twitter. They have used emoticons to label the tweets positive and negative instead of manually reading them. This is a well-balanced dataset with an equal number of positive and negative tweets. As this dataset is not restricted to any specific topic, this was our first choice for comparing embeddings. We have taken a subset of 100000 tweets with 50132 positive and 49868 negative tweets. This was done mainly because of a shortage of training resources.

4.2.2 IMDb50K

Maas et al.(Maas et al., 2011.) have collected 50000 movie reviews from IMDB and the reviews are restricted to not more than 30 per movie. This is also a class-balanced dataset with 25000 positive and negative reviews respectively. Compared to the Sentiment140 dataset, this restricts the content of the text to only movies.

4.2.3 CAA-NRC-Tweets

We have also scrapped a very small topic-specific dataset containing 3101 tweets. As the name suggests, this dataset contains tweets related to 'Citizenship Amend ACT 2019'. Unlike the other two datasets we have used, this is a class-imbalanced dataset with a bias towards negative and neutral tweets. The tweets were labeled with the help of an annotator, who was not part of the experimenting team. The dataset is publicly available to anyone interested in working on it.

Table 4: Dataset Summary

Dataset	No. of Positive texts	No. of Negative texts	No. of Neutral texts
Sentiment140	50312	49868	NA
IMDB50K	25000	25000	NA
CAA-NRC-Tweets	625	1394	1082

Chapter 5. System Design

This chapter provides an in-depth analysis of the process of System Design in the context of our project. This section briefly describes the mathematical equations that are the basis for these techniques. This section also provides a detailed overview of the architecture of each technique including data flow, layer orientation, and layer order.

5.1 Models

The following models have been used for classification in our research.

5.1.1 Logistic Regression

Differently from Naive Bayes, the Logistic Regression classifier is a discriminative model, i.e., we are interested in P(y= k|x) and not in the joint distribution p(x,y).

Logistic regression predicts a binary outcome, i.e., (Y/N) or (1/0) or (True/False). It is a special case of linear regression. It produces an S-shaped curve called a sigmoid. Figure 7 describes the sigmoid function. It takes real values between 0 and 1. The model of logistic regression is given by:

$$\text{Hypothesis} = Wx + b \quad (1)$$

$$h_\theta = \text{sigmoid}(\text{Hypothesis}) \quad (2)$$

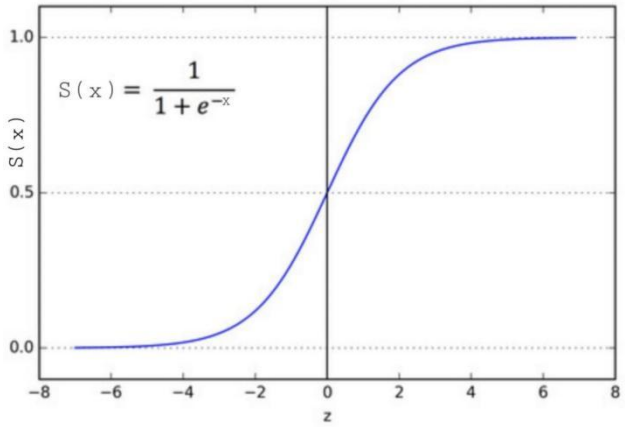


Figure 7. Sigmoid function

Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.

The accuracy and recall levels have a significant impact on the choice of the threshold value. In a perfect world, accuracy and recall should both equal 1, but this is rarely the case.

We utilize the following justifications to choose the threshold in the event of a Precision-Recall tradeoff:

1. Low Precision/High Recall: We pick a decision value that has a low value of Precision or a high value of Recall in situations where we wish to cut down on the number of negative examples without necessarily reducing the number of errors. For instance, in a cancer diagnostic application, we don't want any impacted patients to be labeled as unaffected without paying close attention to whether the patient is receiving a false cancer diagnosis. This is due to the fact that additional medical conditions can identify the absence of cancer but cannot detect its existence in a candidate who has previously been rejected.
2. High Precision/Low Recall: We select a decision value that has a maximum level of Precision or a low value of Recall in situations where we want to cut down on false positives without necessarily cutting down on false negatives. For example, if we are classifying customers whether they will react positively or negatively to a personalized advertisement, we want to be absolutely sure that the customer will react positively to the advertisement because otherwise, a negative reaction can cause a loss of potential sales from the customer.

5.1.2 Support Vector Machine

SVM is another technique for classifying texts based on characteristics created by hand. The bag-of-words and bag-of-n-grams features for SVMs are the most fundamental ones.

SVMs with linear kernels outperform Naive Bayes with these basic characteristics. SVM picks the extreme vectors and points that assist in the construction of the hyperplane. Support vectors are the name given to these extreme circumstances, and as a result, the technique is known as a Support Vector Machine.

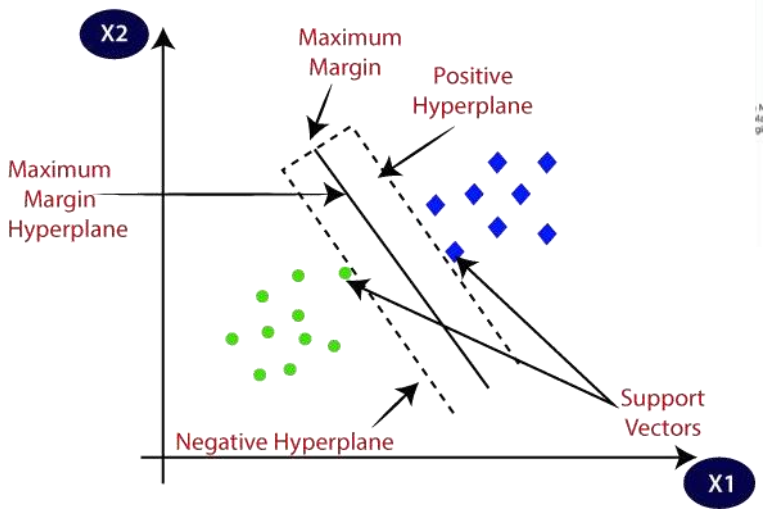


Figure 8. Two different categories are classified using a decision boundary or hyperplane

Finding a hyperplane in N-dimensional space (N is the number of features) that clearly classifies the data points is the goal of the support vector machine method.

There are a variety of different hyper planes that might be used to split the 2 classes of data points. Figure 8 shows how a hyperplane is used to classify the dataset into 2 different classes. Finding a plane with the greatest margin—that is, the greatest separation between data points from both classes—is our goal. Increasing the margin distance adds some support, increasing the confidence with which future data points may be categorized.

5.1.3 Decision tree Classifier

Although it may be used to address classification and regression problems, the decision tree method is most frequently utilized to address classification difficulties. In this tree-structured classifier, internal nodes hold dataset properties, and branches stand in for decision rules, but each leaf node offers the classification. The Decision tree's two nodes are the Decision Node and the Leaf Node. Choice nodes are used to make decisions and have many branches, whereas Leaf nodes represent the outcome of those choices and do not have any more branches. The tests or judgments are developed on the basis of the given dataset. It is a graphical representation for finding every viable solution to an issue or choice based on particular criteria.

It's termed a decision tree because, like a tree, it starts with a root node and grows into a tree-like structure with additional branches. We utilize the CART algorithm, which

stands for Classification and Regression Tree algorithm, to form a tree. A decision tree simply asks a question and divides the tree into subtrees based on the answer (Yes/No).

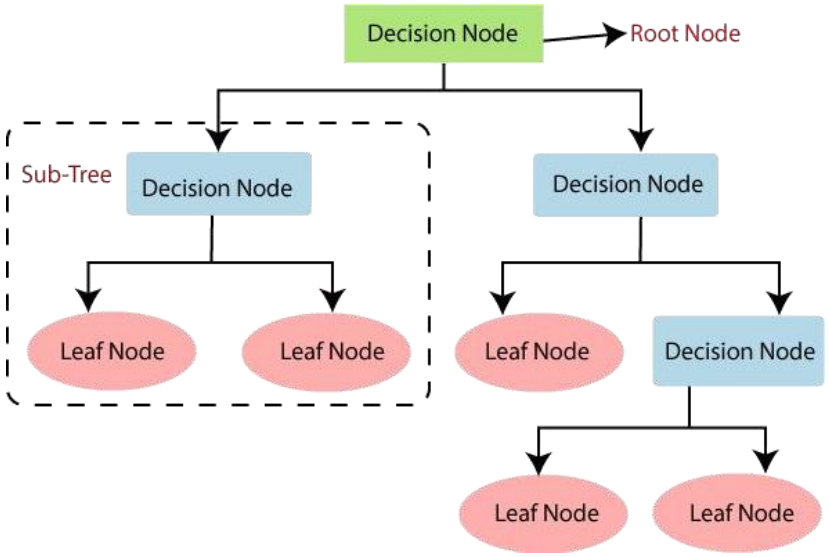


Figure 9. Structure of general decision tree

As shown in Figure 9, at the root node of a decision tree, the classification of a provided dataset begins. This technique compares the values of the record (actual dataset) attribute to the values of the root property, then follows the branch and jumps to the next node. After confirming the parameter value with the other sub-nodes, the algorithm goes on to the next node. It keeps doing this until it approaches the tree's leaf node.

This is how the algorithm works:

Step 1: The root node, which contains the whole dataset, should be the first step, according to S.

Step 2: Select the dataset's top attribute that uses the Attribute Selection Measure (ASM).

Step 3: Separate the S into groups that contain the possible values for the best attribute.

Step 4: Create the node in the tree that will have the optimal parameters.

Step 5: Employing the subsets of the dataset obtained from step 3 to iteratively build more decision trees. Follow this procedure up until the final node, which is referred to as a leaf node, cannot be further categorized.

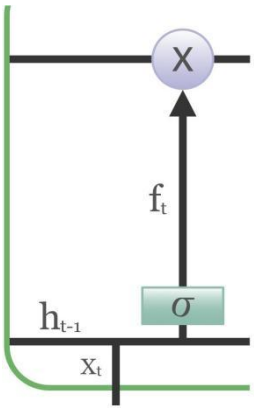
5.1.4 LSTM

Recurrent neural networks are used in the Long Short Term Memory(Hochreiter & Schmidhuber, 1997). The output from the previous phase is sent into the current step of an RNN as input. The LSTM was created by Hochreiter & Schmidhuber. It deals with the

problem of RNN long-term reliance, in which the RNN can predict words from current input but cannot predict words from long-term memory. As the gap length increases, RNN performance is inefficient. By design, LSTM may keep data on hand for a protracted amount of time. It is employed in the processing, prediction, and categorization of time-series data.

Structure Of LSTM:

The LSTM is made up of a chain-like structure with four neural networks and several memory cells. The gates control memory, whereas the cells store information. Three gates are present:



1. **Forget Gate:** The forget gate purges the cell state of information that is no longer relevant. Preceding biasing the gate, two inputs—x t (input at a certain time) and h t-1 (output from a prior cell)—are multiplied using weight matrices. Figure 10 shows the structure of Forget gate. The output of an activation function, which receives the outcome, is a binary value. Information is lost if the output for a certain cell state is 0, but is retained for later use if the output is 1.

Figure 10. Structure of Forget gate

2. **Input gate:** The input gate is responsible for adding important information to the cell state.Figure 11 describes the structure of Input gate. The information is first controlled using the sigmoid function, which filters the values to be remembered using the inputs h t-1 and x t, similar to the forget gate. Then, using the tanh function, which returns a value between -1 and +1, a vector is generated that contains all of the possible values from h t-1 and x t. Finally, the vector's values are multiplied by the controlled values to produce relevant information.

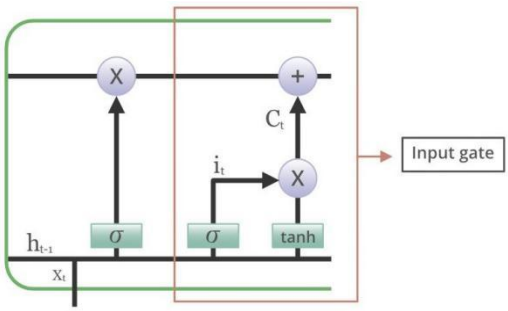


Figure 11. Structure of Input gate

3. **Output gate:** The output gate's responsibility is to gather pertinent information from the current cell state and output it. The tanh function is initially used by the cell to create a vector. The sigmoid function and inputs h t-1 and x t are then used to filter the information by the values to be remembered. Output gate structure is described in Figure 12. The values of the vector are then multiplied by the regulated values and sent as an input and output to the following cell.

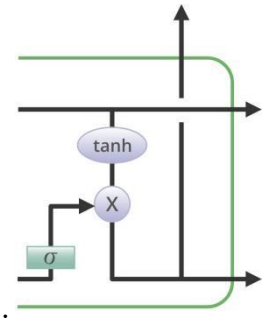


Figure 12. Structure of Output gate

The three inputs are E, short-term memory, and long-term memory. The three inputs are sent to the forgetting or learn gates as soon as they enter the LSTM. The forget gate houses long-term data. The learn gate is given "E" and the short-term data. The remember gate and the utilize gate, respectively, will receive information that passes through the forget gate and the learn gate. The Remember Gate uses data from the Forget Gate and other data to calculate the new long-term memory from the learn gate. The use gate combines the LTM from the forget gate and the STM + E from the learn gate to produce a new short-term memory or an output.

5.1.5 DNN

A neural network with a specific level of complexity, often at least two layers, is known as a deep neural network (DNN), or deep net. Deep networks evaluate data in complex ways by using advanced mathematical modeling. A rough structure of DNN can be seen in figure 13.

On the other hand, it is ideal to think of deep neural networks as an evolution. A few things are required to be created before deep nets. First, machine learning must be developed. In order to increase prediction accuracy, machine learning (ML) provides a framework for automating statistical models like linear regression models (via algorithms). A single model that predicts anything is referred to be a model. Those predictions come

close to being correct. A learning model (machine learning) takes into account all of its inaccurate predictions and modifies the model's internal weights to create a model with fewer mistakes.

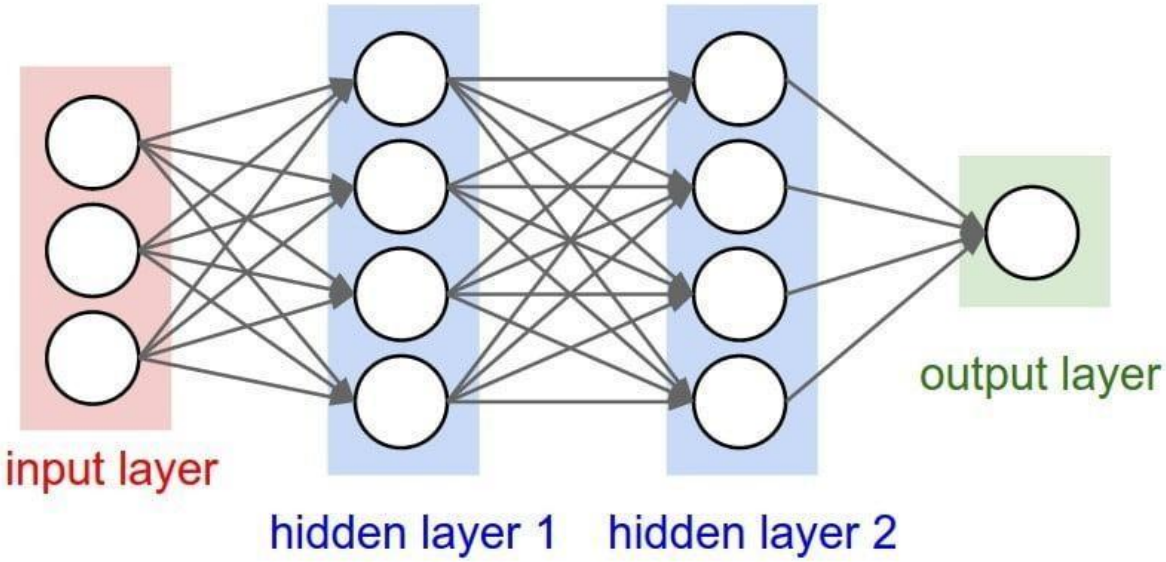


Figure 13. Deep learning model

5.1.6 GRU

The conventional RNN is improved by the Gated Recurrent Unit (GRU)(Cho et al., 2014) (recurrent neural network). Kyunghyun Cho and associates first proposed it in 2014. Long short term memory and GRUs have a number of commonalities (LSTM). GRU controls how information flows via gates, much as LSTM. Compared to LSTM, they are quite new. Because of this, they outperform LSTM and Haven and have a simpler architecture. Figure 14 compares LSTM and GRU cells.

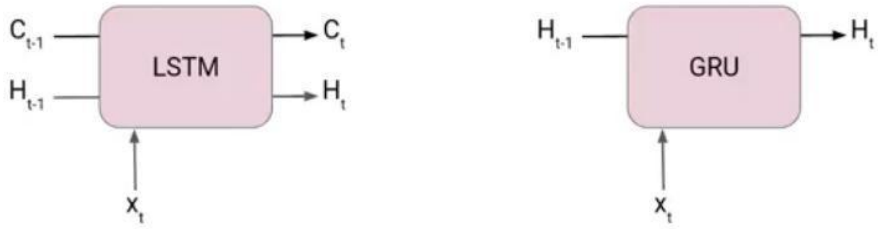


Figure 14. Structure of LSTM and GRU cells

5.2 Embeddings

Humans have always been good at understanding language. Computers may struggle to comprehend the link between words, but humans find it easy. For instance, a

machine cannot understand the link between terms like king and queen, man and woman, or tiger and tigress that we humans can.

Word representation techniques called word embeddings fill the gap between human and machine language understanding. In other words, two similar words are represented in a vector space by nearly identical vectors that are placed next to one another. The bulk of problems with natural language processing must be resolved using these.

Individual words are represented as real-valued vectors in a given vector space when employing word embeddings. Each word is represented by a single vector, with the values of the vectors being learned in a manner like a neural network.

5.2.1 Bag of Words

A text modeling method for Natural Language Processing is called "bag of words". We might refer to it as a feature extraction strategy for text data in technical terms. It is both easy and versatile. A textual illustration that pinpoints the placement of words in a document is known as a "bag of words." Grammar and word order concerns are not taken into account; just word counts are tracked. It is referred to as a "bag" of words since no information on the order or structure of the words is retained in the document. The model just cares about whether identified phrases are in the document, not where they are.

For instance,

Sentence 1: "Welcome to Great Learning, now start learning,"

Sentence 2: "Learning is a valuable habit,"

Because the case of the word carries no information, take into account these phrases and change the capitalization of the previous ones. The text is then cleaned up by removing stopwords and special characters. Stopwords are words like "is," "a," and several others that do not provide much information about the content. After completing the aforementioned instructions, the sentences are modified.

Sentence 1: “Welcome to fantastic learning; begin learning now.”

Sentence 2: "Learning is good practice”.

Then, by looking through the terms in the text before, make a list of every word in our model vocabulary. With one location in the fixed-length array, we can now represent the seven words.

Then, Make a list of all the words in our model vocabulary by going over the words in the preceding text. Now we can represent the 7 words using a fixed-length array, with one position in the vector to score each word.

For sentence 1, The count is as follows:

Writing the above frequencies in the vector form: Sentence 1 → **[1,1,2,1,1,0,0]**

For sentence 2, The count is as follows:

Writing the above frequencies in the vector form: Sentence 2 → **[0,0,1,0,0,1,1]**

The dataset is preprocessed and is widely used in the Bag-of-Words approach since machine learning datasets are so large and can contain a vocabulary of thousands or even millions of words. Therefore, it's important to prepare the text before employing a bag of words.

Although the Bag-of-Words tactic is incredibly effective and simple to implement, it does have certain drawbacks, which are listed below:

- The model disregards the geographical information of the word. Information about the text's location is very important. Both "today is off" and "Is today off" will have identical vector representations in the BoW model.
- The meaning of the term is disregarded by the bag of word models. For instance, the terms "football" and "soccer" are widely used interchangeably. The vectors corresponding to these words, however, are quite different in the bag of words model. The dilemma becomes worse as you mimic sentences. The concepts "buy used cars" and "buy old cars" are represented by entirely different vectors in the Bag-of-words model.
- The vocabulary range of the Bag-of-Words paradigm is a big concern. If the model encounters a new term it is unfamiliar with, we may use the strange yet instructive word "Biblioklept" (means one who steals books). This phrase will probably be disregarded since the BoW model has not yet encountered it.

5.2.2 TfIdf

A text modeling method used in natural language processing is called Tfidf. Word frequency is rescaled according to how frequently it appears in all papers, punishing words like "the" that are used frequently throughout all texts. The Tf-Idf

technique for scoring is also known as the term frequency-inverse document frequency approach. The TF-IDF is intended to demonstrate how significant a phrase is inside a certain document.

The TF-IDF is calculated for each word in a document by multiplying two metrics:

1. term frequency (TF) of a word in a text. The simplest method for determining this frequency is to count the instances of each word in a document. There are further ways to modify the frequency. For instance, split the raw frequency of the word that appears the most frequently in the manuscript by its length. The formula is used to determine the term-frequency.

   $TF(i,j) = n(i,j) / \Sigma\, n(i,j)$ (3)

   Where,

   n (i,j)= number of times nth word occurred in a document

   Σn(i,j) = total number of words in a document.

2. Over a group of documents, the word's inverse document frequency (IDF). This shows how frequently or seldom a term appears in the corpus of documents. The term becomes increasingly prevalent the nearer it gets to 0. By dividing the total number of documents by the number of documents that include a phrase, you may get the logarithm. Therefore, this number will be near to 0 if the word is commonly used and appears in several books. If not, it will be very near to 1.

   $IDF = 1 + log(N/dN)$ (4)

Where

N=Total number of documents in the dataset

dN=total number of documents in which nth word occurs

The TF-IDF score of a word in a document is calculated by multiplying these two numbers. The greater the score, the more important the word in that paper is.

To put it in mathematical terms, the TF-IDF score is calculated as follows:

$TF\text{-}IDF=TF*IDF$ (5)

5.2.5 Glove

An alternative approach to creating word embeddings is using GloVe(Pennington et al., n.d.)(Global Vectors for Word Representation). It is based on methods for matrix

factorization and word-context matrices. When working with a large corpus, you may count the number of times each "word" (represented by rows) appears in a given "context" (columns). We normally scan our corpus by looking for context terms inside a window size before the phrase and a window size after the term for each individual keyword. Furthermore, we put less importance to claims that are made from a distance.

It is essentially combinatorial in scale, therefore there are a huge variety of "contexts." In order to obtain a lower-dimensional matrix with each row corresponding to a vector representation of a single word, we must factorize this matrix. Most often, this is done via minimizing "reconstruction loss." This loss looks for lower-dimensional models that can mostly explain the variation in high-dimensional data.

For instance, if a 10-word frame in the document corpus contains the terms "cat" and "dog" 20 times, then:

$$\text{Vector(cat)}.\ \text{Vector(dog)} = \log(10) \tag{6}$$

The model is thus compelled to incorporate the frequency distribution of surrounding words in a broader context.

In fact, both GloVe and Word2Vec yield outcomes that are equivalent when we incorporate our text. With a window size of around 5–10, we train our model on Wikipedia material for use in practical applications. The corpus has over 13 million words, thus it takes a lot of time and resources to create these embeddings. Pre-trained word vectors, which have already been trained and are straightforward to use, can be used to get around this.

5.2.4 Word2Vec

A common method for learning word embeddings using a shallow neural network is Word2Vec. Tomas Mikolov created it in 2013 while working at Google. A two-layer neural network-based method called Word2vec is used to effectively create word embeddings. It was developed by Tomas Mikolov and colleagues at Google in 2013 to boost the effectiveness of neural-network-based embedding training, and it has subsequently taken over as the industry norm for creating pre-trained word embeddings. A set of feature vectors that represent the words in the input text corpus are produced by Word2vec. Word2vec transforms text into a numerical representation that deep neural networks can understand, despite the fact that it is not a deep neural network.

The objective function of Word2Vec allows us to compare embeddings of words in contexts with similarity. Therefore, in this vector space, these words are quite close together. In other words, the angle should be close to 0. The cosine of the angle (Q) in between vectors must be close to 1. Word2vec combines the Skip-gram model and the Continuous Bag of Words (CBOW) approach. Figure 15 gives a insight into how both of these methods work. Both of these are word-to-word word-to-word mappings using shallow neural networks (s). These two methods both include the learning of weights that serve as word vector representations.

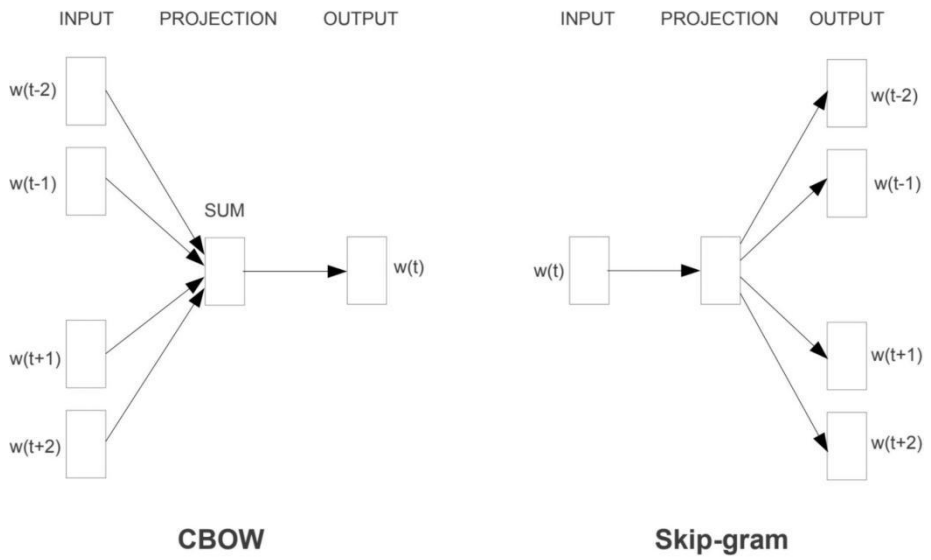


Figure 15. Word2Vec Training Models taken from "Efficient Estimation of Word Representations in Vector Space", 2013

Continuous Bag-of-Words model (CBOW): Based on the words around a word, CBOW determines how likely it is to appear. A word or group of words may be taken into consideration. To anticipate a single target word, however, we just use one context word.

The model of Skip-gram: In contrast to the CBOW model, the Skip-gram model design aims to achieve the reverse. It makes an effort to anticipate the source context terms given a target word. For large corpora with more dimensions, skip-gram performs better, but it takes longer to train. On the other hand, CBOW is simpler to learn and perfect for small groups.

5.2.5 FastText

The word2vec model is extended by the fastText word embedding technique. Instead of learning word vectors directly, FastText represents each word as an n-gram of letters.

Take the word "artificial" with n=3, for instance. This term's text representation is ar, art, rti, tif, ifi, fic, ici, ial, al>, with the beginning and end of the word denoted by angle brackets. As a result, the embeddings can comprehend the meaning of shorter words and distinguish suffixes and prefixes. The embeddings are learned by a skip-gram model that has been trained to represent words using character n-grams. This model is known as a bag of words model with a sliding window over a word because it ignores the underlying structure of the word. As long as the n-grams fit within this frame, it doesn't matter in what order they appear.

FastText performs effectively with rare words. Even if it wasn't visible during training, breaking a word down into its n-grams can expose its embeddings. For terms not contained in the model dictionary, neither Word2vec nor GloVe succeed in producing a vector representation. This is a crucial advantage of this tactic.

Chapter 6. SYSTEM IMPLEMENTATION

This chapter covers the project's System Implementation phase. This section also includes a detailed description of each step. Data pre-processing, model initialization, and the training procedure are the steps that have been covered in this section. This section also includes information on the web interface implementation process.

6.1 Data Pre-processing

After labeling the data, The raw tweets are prepared to make them suitable for the machine learning models by reducing noise, missing values, and unusable data.

- The raw tweets are first turned to lower case as the case of a word is irrelevant as it doesn't hold any information.
- If the tweets have any stop words, URLs, or non-ASCII characters in them, They are removed from the tweets.
- If the tweet has integers, they are replaced by their textual representation.
- The words in the tweets are tokenized and then lemmatized and are reduced to their Lemmas.

6.2 Text Encoding

Word embedding is a language modeling technique that converts words into real-number vectors.

- **Bag Of Words**

  A list of all the words in our model vocabulary is made by going over the words in the preceding text. The frequencies of the words are represented in the vector form.

- **Tf-Idf**

  For each word in a document, the TF-IDF is calculated by multiplying two metrics.

  1. Term-Frequency is calculated using the formula $TF(i,j) = n(i,j) / \Sigma\, n(i,j)$ (4)

  Where, n (i,j)= number of times nth word occurred in a document and

  $\Sigma n(i,j)$ = total number of words in a document.

2. Inverse document frequency (IDF) is calculated by

$$IDF = 1 + log(N/dN) \quad (5)$$

Where N=Total number of documents in the dataset and dN=total number of documents in which nth word occurs

- **Glove**

In a huge corpus, A massive matrix of co-occurrence information and counting each "word" (rows) and how frequently we see this word in some "context" (columns) is created.

- **Word2Vec**

Word2vec takes a text corpus as input and returns a set of feature vectors representing the corpus's words.

- **FastText**

FastText depicts each word as an n-gram of characters rather than learning vectors for words directly.

6.3 Model Implementation

- Necessary libraries such as sklearn, pandas, and time are imported.
- Necessary Modules such as sklearn.model_selection.train_test_split, sklearn.linear_model.LogisticRegression, sklearn.linear_model.svm, sklearn.tree, tensorflow.keras.Sequential, tensorflow.keras.layers.Dense, tensorflow.keras.layers.Bidirectional, tensorflow.keras.layers.LSTM, tensorflow.keras.layers.GRU are imported.
- The models are provided with the encoded text document along with the preprocessed document

6.3.1 Implementation of Logistic Regression

Following steps are followed for implementing Logistic Regression

- Import sklearn.linear_model.LogisticRegression
- The text embeddings of 3100 tweets are randomly split allocating 25% to the testing set and 75% training set.
- Fit the model according to the given training data and set the maximum iteration to 10000.
- Predict class labels for samples in the test set.
- Calculate and display Accuracy, Precision, recall, and F1-score.

6.3.2 Implementation of Support Vector Machine

Following steps are followed for implementing SVM

- Import sklearn.linear_model.svm
- The text-embedded dataset of 3100 embedded tweets is randomly split allocating 25% to the testing set and 75% training set.
- Fit the model according to the given training data and set the maximum iteration to 10000, the type of kernel to 'Linear' and verbose to 1.
- Predict class labels for samples in the test set.
- Calculate and Display Accuracy, Precision, Recall, and F1 score.

6.3.3 Implementation of Decision tree Classifier

Following steps are followed for implementing Deciscion tree classifier

- Import sklearn.linear_model.tree
- The text embeddings of 3100 tweets are randomly split allocating 25% to the testing set and 75% training set.
- Fit the model according to the given training data.
- Predict class labels for samples in the test set.
- Calculate and Display Accuracy, Precision, Recall, and F1 score.

6.3.4 Implementation of LSTM

Following steps are followed for implementing LSTM

- Import NumPy, pandas, TensorFlow.
- Import dependencies such as tensorflow.keras import Sequential, tensorflow.keras.initializers.Constant, tensorflow.keras.layers.ReLU, tensorflow.keras.layers.Dropout, tensorflow.keras.layers.Embedding, tensorflow.keras.layers.LSTM, tensorflow.keras.layers.Dense.
- The text embeddings of 3100 tweets are randomly split allocating 25% to the testing set and 75% training set.
- LSTM also had an embedding layer with an output dimension of 300 followed by two LSTM layers with 128 hidden units.
- This was fed to 2 hidden layers with 32 and 16 neurons respectively.
- A dropout of 0.6 was added to each hidden layer.
- An output layer with a Softmax function for CAA-NRC was used and the relu was the activation function for the hidden layers.

- They were trained for five epochs for all three datasets.
- Predict class labels for samples in the test set.
- Calculate and Display Accuracy, Precision, Recall, and F1 score.

6.3.5 Implementation of DNN

Following steps are followed for implementing DNN

- Import NumPy, pandas, TensorFlow.
- Import dependencies such as tensorflow.keras import Sequential, tensorflow.keras.initializers.Constant, tensorflow.keras.layers.Dropout, tensorflow.keras.layers.Embedding, tensorflow.keras.layers.Dense.
- The text embeddings of 3100 tweets are randomly split allocating 25% to the testing set and 75% training set.
- DNN consisted of an embedding layer with an output dimension of 300.
- This was fed to 2 hidden layers with 32 and 16 neurons respectively.
- A dropout of 0.6 was added to each hidden layer.
- An output layer with a Softmax function for CAA-NRC was used and the relu was the activation function for the hidden layers.
- Predict class labels for samples in the test set.
- Calculate and Display Accuracy, Precision, recall, and F1-score.

6.3.6 Implementation of GRU

Following steps are followed for implementing GRU

- Import NumPy, pandas, TensorFlow.
- Import dependencies such as tensorflow.keras import Sequential, tensorflow.keras.initializers.Constant, tensorflow.keras.layers.Dropout, tensorflow.keras.layers.Embedding, tensorflow.keras.layers.Dense.
- The text embeddings of 3100 tweets are randomly split allocating 25% to the testing set and 75% training set.
- GRU also had an embedding layer with an output dimension of 300 followed by two GRU layers with 128 hidden units.
- This was fed to 2 hidden layers with 32 and 16 neurons respectively.
- A dropout of 0.6 was added to each hidden layer.
- An output layer with a Softmax function for CAA-NRC was used and the relu was the activation function for the hidden layers.

- They were trained for five epochs for all three datasets.
- Predict class labels for samples in the test set.
- Calculate and Display Accuracy, Precision, Recall, and F1 score.

6.5 User Interface

Figure 16 gives an insight into the UI we have designed to showcase the accuracy of different models and embeddings.

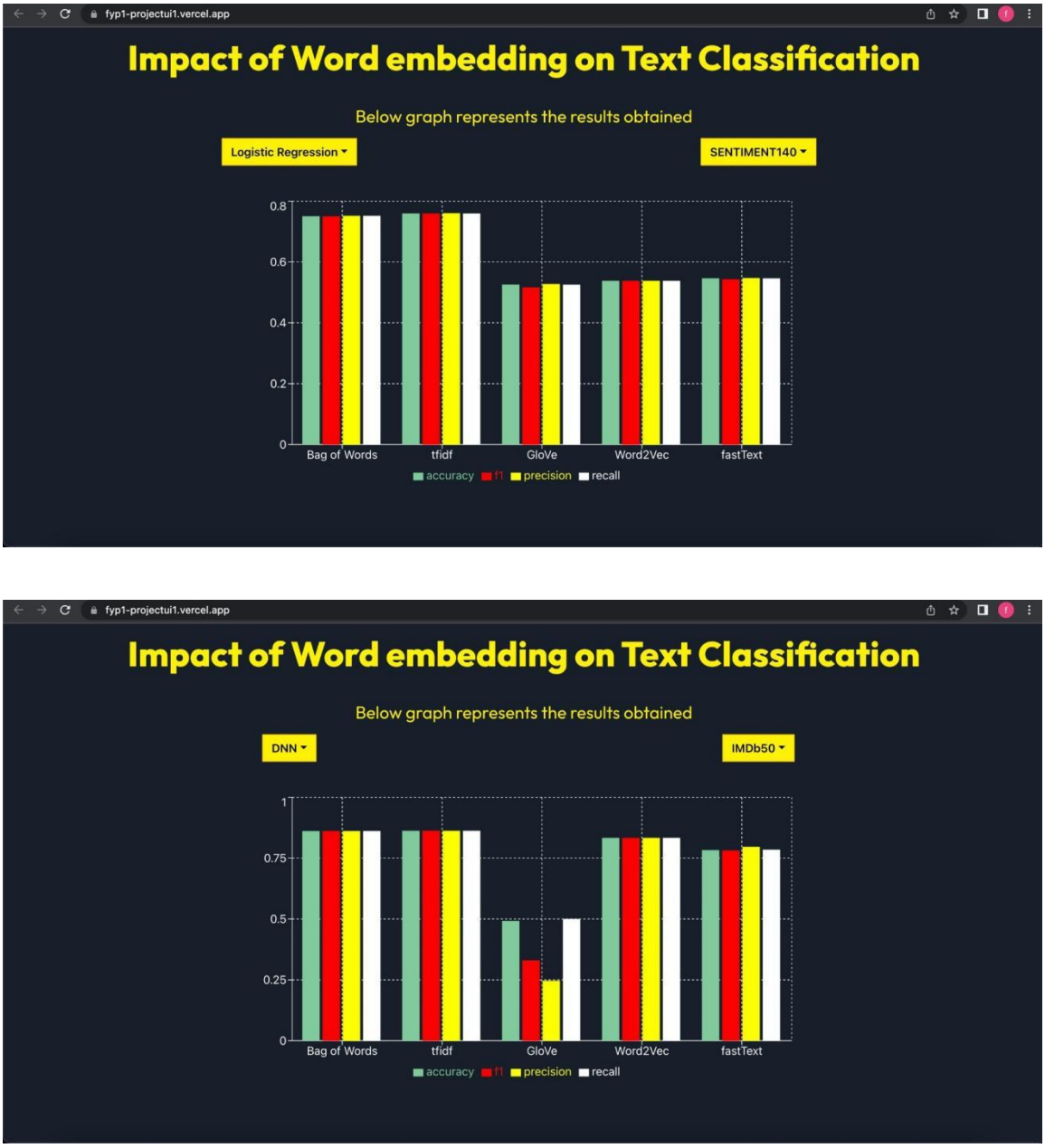


Figure 16. UI Interface

Chapter 7. SYSTEM TESTING AND RESULT ANALYSIS

7.1 System Testing

Testing is the process of validating and verifying the artifacts and behavior of the software that is being tested. Program testing may also give a firm an unbiased, independent perspective of the software so that it can see and comprehend the risks associated with software installation.

7.1.1 Unit testing

Unit testing is a software testing technique used in computer programming to evaluate the suitability of single units of source code—sets from one or more computer program modules along with related control data, use processes, and operating procedures.

7.1.2 Integration Testing

Software testing's integration testing stage involves combining and evaluating many software modules simultaneously. Integration testing is done to determine whether a system or component complies with a list of functional requirements. Before system testing and following unit testing, it happens. Unit-tested modules are used as the input for integration testing, which collects them into bigger aggregates and applies the tests specified in an integration test plan to those aggregates to produce an integrated system suitable for system testing.

7.1.3 Test cases documented

Both the unit test cases as well as the inter modular integration test cases are recorded in Table 5

Table 5. Test cases for documenting behavior.

Module Name	Expected Input	Expected Output	Input Provided	Behavior
Input receiver	Input of alpha numeric character	Nothing	Anything from keyboard	Same as expected
Text preprocessor	Input from text receiver	Preprocessed Text	only text	preprocessed text
Text preprocessor	Input from text receiver	Preprocessed Text	Only numbers	Does Not modify anything

Text preprocessor	Input from text receiver	Preprocessed Text	Special characters	Removes everything
TFIDF	Preprocessed Text	Weight for each number	"hello"	0.3451
Word2Vec	Preprocessed Text	Vector representation of each word	"hello"	[0,0,1,2,3.45,0.12...]
Classifier 2 class	Either vector or weight representation	A number between 0 and 1	Same as expected Input	0.88
Classifier 3 Class	Either vector or weight representation	An array of 3 number between 0 and 1	Same as expected Input	[0.1,0.1,0.79]

7.2 Result Analysis

Performance of Logistic Regression, SVM, Decision tree, LSTM, DNN, and GRU for datasets Sentiment140, IMDb, and CAA-NRC with BagOfWords, TF-IDF, Word2Vec, GloVe, and FastText embeddings are given below.

In Table 6, the Performance of Classifiers with BagOfWords is shown. Here, Logistic Regression performs the best for all the three datasets with Bag of Words as the embedding, Closely followed by DNN.

Table 6. Comparison of Performance of classifiers with BagOfWords embedding

	Sentiment140			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	**0.7504**	**0.7501**	**0.7517**	**0.7517**
SVM	0.5575	0.5362	0.5694	0.5568
decisionTree	0.6831	0.6830	0.6832	0.6831
LSTM	0.5015	0.3340	0.2508	0.5000
DNN	0. 7390	0.7383	0.7416	0.7391
GRU	0.4985	0.3327	0.2492	0.5000
	IMDb50			

	Sentiment140			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	**0.8750**	**0.8750**	**0.8751**	**0.8749**
SVM	0.6602	0.6528	0.6715	0.6582
decisionTree	0.7242	0.7242	0.7243	0.7243
LSTM	0.4926	0.3300	0.2463	0.5000
DNN	0.8614	0.8614	0.8614	0.8615
GRU	0.5841	0.5839	0.5839	0.5839
	CAA-NRC			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	**0.7113**	**0.7020**	**0.7154**	**0.6927**
SVM	0.6830	0.6721	0.6704	0.6753
decisionTree	0.6443	0.6365	0.6421	0.6325
LSTM	0.4549	0.2084	0.1516	0.3333
DNN	0.7113	0.7000	0.7049	0.6963
GRU	0.4549	0.2084	0.1516	0.3333

In Table 7, The performance of classifiers for TF-IDF is shown. Logistic regression has the highest accuracy for Sentiment140 and IMDb50 compared to the rest of the classifiers while for the CAA-NRC dataset, DNN has the highest accuracy.

Table 7. Comparison of Performance of classifiers with TF-IDF embedding

	Sentiment140			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	**0.7596**	**0.7595**	**0.7602**	**0.7596**
SVM				

decisionTree	0.6866	0.6866	0.6866	0.6866
LSTM	0.4985	0.3327	0.2492	0.5000
DNN	0.7128	0.7050	0.7370	0.7124
GRU	0.5015	0.3340	0.2508	0.5000
	IMDb50			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	**0.8914**	**0.8913**	**0.8916**	**0.8912**
SVM	0.8862	0.8862	0.8864	0.8861
decisionTree	0.7207	0.7207	0.7208	0.7208
LSTM	0.5074	0.3366	0.2537	0.5000
DNN	0.8622	0.8622	0.8622	0.8623
GRU	0.5286	0.5178	0.5343	0.5309
	CAA-NRC			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.7191	0.7026	0.7495	0.6833
SVM	0.7229	0.7116	0.7485	0.6939
decisionTree	0.6108	0.6002	0.6036	0.5977
LSTM	0.4549	0.2084	0.1516	0.3333
DNN	**0.7268**	**0.7115**	**0.7333**	**0.6998**
GRU	0.4549	0.2084	0.1516	0.3333

Table 8 shows the performance of classifiers with Word2Vec as the embedding for all the datasets. GRU performs the best for the Sentiment140 dataset, closely followed by LSTM and GRU. For IMDb50 and CAA-NRC datasets, LSTM works best, closely followed by DNN and GRU. The performance of Logistic regression, SVM and decision tree is not up to par.

Table 8. Comparison of Performance of classifiers with Word2Vec embedding

	Sentiment140			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.5381	0.5378	0.5381	0.5380

SVM	0.5053	0.3749	0.5228	0.5039
decisionTree	0.5541	0.5541	0.5541	0.5541
LSTM	0.7481	0.7480	0.7483	0.7481
DNN	0.7320	0.7319	0.7320	0.7320
GRU	**0.7598**	**0.7598**	**0.7598**	**0.7598**
	IMDb50			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.5527	0.5527	0.5529	0.5529
SVM	0.5312	0.5230	0.5313	0.5294
decisionTree	0.5201	0.5201	0.5201	0.5201
LSTM	**0.8551**	**0.8550**	**0.8556**	**0.8548**
DNN	0.8332	0.8332	0.8333	0.8333
GRU	0.8464	0.8463	0.8465	0.8462
	CAA-NRC			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.4691	0.4282	0.4375	0.4328
SVM	0.3930	0.3648	0.3783	0.3644
decisionTree	0.4936	0.4892	0.4943	0.4861
LSTM	**0.6714**	**0.6473**	**0.7067**	**0.6348**
DNN	0.6572	0.6293	0.6692	0.6168
GRU	0.6701	0.6509	0.6710	0.6400

In Table 9, the Performance of classifiers for the three datasets with Glove embedding is shown. For Sentiment140, LSTM has the better performance, followed by GRU and DNN. For IMDB50 and the biased CAA-NRC dataset, LSTM worked best followed by GRU. DNN, Logistic Regression, SVM, and decision

Table 9. Comparison of Performance of classifiers with GloVe embedding

	Sentiment140			
	ACCURACY	F1 SCORE	PRECISION	RECALL

Logistic Regression	0.5259	0.5162	0.5276	0.5255
SVM	0.5057	0.4216	0.5107	0.5045
decisionTree	0.5399	0.5398	0.5399	0.5399
LSTM	**0.7506**	**0.7493**	**0.7562**	**0.7508**
DNN	0.7219	0.7216	0.7231	0.7220
GRU	0.7378	0.7362	0.7429	0.7375
	IMDb50			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.5140	0.5139	0.5143	0.5143
SVM	0.5098	0.4914	0.5082	0.5071
decisionTree	0.5171	0.5171	0.5171	0.5171
LSTM	0.8158	0.8158	0.8158	0.8158
DNN	0.4926	0.3300	0.2463	0.5000
GRU	**0.8251**	**0.8251**	**0.8251**	**0.8251**
	CAA-NRC			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.4240	0.3120	0.3719	0.3409
SVM	0.4253	0.2604	0.2472	0.3240
decisionTree	0.4497	0.4233	0.4229	0.4237
LSTM	**0.6727**	**0.6588**	**0.6738**	**0.6493**
DNN	0.4549	0.2084	0.1516	0.3333
GRU	0.6456	0.6341	0.6390	0.6322

Table 10 shows the performance of classifiers with FastText as the embedding. LSTM and GRU work the best for Sentiment140 and IMDb50 while only LSTM works best for CAA-NRC dataset.

Table 10. Comparison of Performance of classifiers with FastText embedding

	Sentiment140			
	ACCURACY	F1 SCORE	PRECISION	RECALL

Logistic Regression	0.5465	0.5431	0.5476	0.5462
SVM	0.5094	0.4065	0.5263	0.5082
decisionTree	0.5658	0.5658	0.5658	0.5658
LSTM	**0.7948**	**0.7945**	**0.7966**	**0.7949**
DNN	0.5015	0.3340	0.2508	0.5000
GRU	0.7936	0.7933	0.7951	0.7937
	IMDb50			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.5535	0.5535	0.5538	0.5538
SVM	0.5062	0.4884	0.5040	0.5035
decisionTree	0.5338	0.5338	0.5338	0.5338
LSTM	0.8514	0.8504	0.8587	0.8504
DNN	0.7831	0.7813	0.7964	0.7847
GRU	**0.8562**	**0.8561**	**0.8587**	**0.8569**
	CAA-NRC			
	ACCURACY	F1 SCORE	PRECISION	RECALL
Logistic Regression	0.4317	0.3254	0.3989	0.3507
SVM	0.4240	0.2984	0.3914	0.3356
decisionTree	0.4768	0.4543	0.4567	0.4526
LSTM	**0.6701**	**0.6623**	**0.6849**	**0.6506**
DNN	0.6572	0.6434	0.6618	0.6329
GRU	0.5760	0.5557	0.5738	0.5471

From Table 6 and Table 7, considering all the datasets, it is clear that Machine learning algorithms work better compared to deep learning classifiers with frequency-based embeddings: BagOfWords and TF-IDF embeddings, and from Tables 8, 9, and, 10 Deep learning classifiers work better with context-based embeddings: Word2Vec, GloVe, and FastText. This variation in the performance of models is because of the size of features extracted by each embedding.

Features extracted from BagOfWords and TF-IDF are sufficient for Machine Learning Classifiers but are not enough for high data-dependent RNNs. Features extracted from Word2Vec, GloVe and FastText are too high for Machine Learning classifiers as they are in 2-dimensional form. When dimensionality is reduced, there is a loss of information and hence, a significant loss in accuracy can be noticed in Machine Learning Classifiers.

When we look at the results of DNN, its performance is comparable irrespective of the embeddings used. But the training of DNN is highly dependent on the embedding used.

Chapter 8. CONCLUSION AND FUTURE WORK

From the above-tabulated results, it can be concluded that the performance of each type of classifier is highly dependent on the embedding used. The results suggest that for any particular dataset, though the highest accuracy achieved by ML-based algorithms is higher than that of RNNs, their training time is also comparatively longer. Continued to this work we can also compare the performance of embeddings generated by various context-based methods based on the nature of the dataset, i,e whether the text present in the dataset is general or extremely topic-specific. We can also compare the accuracy between pre-trained embeddings and dataset-specific embeddings.

Chapter 9. APPENDIX A: TEAM DETAILS

Project Title	Impact of different embeddings on text classification		
USN	Team Members	Email ID	Mobile number
01JST18CS125	Sharath H N	sharathhn.vasistha@gmail.com	8073084955
01JST18CS015	Anusha S G	anusha.sgorawar@gmail.com	9945790707
01JST18CS060	Milind M	milindmgowda77@gmail.com	6362635358
01JST18CS056	Manju Charan M B	manjucharan1818@gmail.com	6362635358

Name: Sharath H N
USN: 01JST18CS125

Name: Anusha S G
USN: 01JST18CS015

Name: Milind M
USN: 01JST18CS060

Name: ManjuCharan M B
USN: 01JST18CS056

Chapter 10. APPENDIX B - COs, POs, and PSOs Mapping for the project work (CS83P)

10.1 Course Outcomes:

CO1: Formulate the problem definition, conduct a literature review and apply requirements analysis.

CO2: Develop and implement algorithms for solving the problem formulated.

CO3: Comprehend, present, and defend the results of exhaustive testing and explain the major findings.

10.2 Program Outcomes:

PO1: Apply knowledge of computing, mathematics, science, and foundational engineering concepts to solve computer engineering problems.

PO2: Identity, formulate and analyze complex engineering problems.

PO3: Plan, implement and evaluate a computer-based system to meet desired societal needs such as economic, environmental, political, healthcare and safety within realistic constraints.

PO4: Incorporate research methods to design and conduct experiments to investigate real-time problems, to analyze, interpret and provide a feasible conclusion.

PO5: Propose innovative ideas and solutions using modern tools.

PO6: Apply computing knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.

PO7: Analyze the local and global impact of computing on individuals and organizations for sustainable development.

PO8: Adopt ethical principles and uphold the responsibilities and norms of computer engineering practice.

PO9: Work effectively as an individual and as a member or leader in diverse teams and in multidisciplinary domains.

PO10: Effectively communicate and comprehend.

PO11: Demonstrate and apply engineering knowledge and management principles to manage projects in multidisciplinary environments.

PO12: Recognize contemporary issues and adapt to technological changes for lifelong learning.

10.3 Program Specific Outcomes:

PSO1: Problem Solving Skills: Ability to apply standard practices and mathematical methodologies to solve computational tasks, and model real-world problems in the areas of database systems, system software, web technologies, and Networking solutions with appropriate knowledge of Data structures and Algorithms.

PSO2: Knowledge of Computer Systems: An understanding of the structure and working of the computer systems with performance study of various computing architectures.

PSO3: Successful Career and Entrepreneurship: The ability to get acquainted with state-of-the-art software technologies leading to entrepreneurship and higher studies.

PSO4: Computing and Research Ability: Ability to use knowledge in various domains to identify research gaps and to provide solutions to new ideas leading to innovations.

Table of Mapping of CO, PO and PSO:

SUBJECT	CODE	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
Project Work	CS83P	CO1	2	2	0	3	1	0	0	0	3	3	2	1	2	2	1	3
		CO2	2	2	0	3	1	0	0	0	3	3	2	1	2	2	1	3
		CO3	2	2	0	3	1	0	0	0	3	3	2	1	2	2	1	3

Note:

Scale

0 - Not Applicable

1 - Low relevance Scale

2 - Medium relevance Scale

3 - High relevance Scale

10.4 Justification for the mapping

The first CO is related to problem definition, literature survey and requirement analysis. Planning the

project such that it meets the needs of society, by considering all the constraints is very relevant for this. Department of Computer Science and Engineering plays a crucial role in deciding the requirements and using latest technology to make our implementation better is of high relevance.

The second CO, design and implementation highly depends on the way we apply already acquired knowledge about computing, mathematics, etc., the innovation we bring in to our implementation, make our implementation adaptable to technological changes that might happen in future and the way we look at the problem and apply our knowledge. The ability to analyze global and local impact of the system, ability to uphold the ethical principles of engineering practices, the way in which we communicate and comprehend the concepts, the way in which the project is handled in multidisciplinary environments hold great relevance in defending our work and explaining major findings during our project. Subjects helped us in developing our project:

•To formulate the problem definition, conduct literature review and apply requirements analysis that we gained the knowledge from Software Engineering and System Software.

•To Develop and implement algorithms for solving the problem formulated , we gained the knowledge from subjects such as Neural Networks, Object Oriented Programming, Machine Learning, Data Warehousing and Data Mining.

•To Comprehend, present and defend the results of exhaustive testing and explain the major findings, we gained the knowledge from Engineering Mathematics, Web Technologies and Operating Systems.

Chapter 11. APPENDIX C - PUBLICATION DETAILS

Selected journal : International Journal of Modern Education and Computer Science(IJMECS)

Paper number: PAPER007107

File number: PAPER007107-2 Manuscript Download Cover Letter Download

Paper title: Impact of different embeddings on text classification

Paper authors: Sharath H N, Anusha S G, Milind M, Manju Charan M B, Dr Anilkumar K M

Research area of the paper: Machine learning

Paper keywords: Natural Language processing, Word embeddings, Text classification, Comparative analysis.

Paper abstract: Text representation has always been a major problem in natural language processing. Features such as frequency, position in the text, and co-occurrence of words are used extensively to design various text representation techniques. There are different techniques currently used for text representation. Some of them are the bag of words, TF-IDF, word2vec, GloVe, and, Fast text. Normally frequency-based text representations which convert each word to a single number are paired with machine learning algorithms and context-based word embeddings that convert each word to n-dimensional vectors are used with neural network classifiers. So in this paper, we have used both frequency and context based on multiple machine learning and neural network-based classifiers and have compared the results. Our experimental results conclude that bag of words and TF-IDF(frequency-based) have higher accuracy in ML classifiers and word2vec, GloVe, and, Fast text(Context-based) has performed extremely well for neural network-based classifiers.

Chapter 12. REFERENCES

Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C., Ca, J. U., Kandola, J., Hofmann, T., Poggio, T., & Shawe-Taylor, J. (2003). A Neural Probabilistic Language Model. In *Journal of Machine Learning Research* (Vol. 3, pp. 1137–1155).

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). *Enriching Word Vectors with Subword Information*. http://arxiv.org/abs/1607.04606

Cho, K., Merrienboer, B. van, Bahdanau, D., & Bengio, Y. (2014). *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. http://arxiv.org/abs/1409.1259

Entity Types – Quick and easy documentation | Community. (2021). Retrieved July 3, 2022, from https://community.expert.ai/articles-showcase-48/entity-types-quick-and-easy-documentation-206

Go, A., Bhayani, R., & Huang, L. (2009). *Twitter Sentiment Classification using Distant Supervision*. http://tinyurl.com/cvvg9a

Google Colaboratory. (2017). Retrieved July 1, 2022, from https://colab.research.google.com/?utm_source=scs-index#scrollTo=Nma_JWh-W-IF

Helaskar, M. N., & Sonawane, S. S. (2019). Text Classification Using Word Embeddings. *2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, 1–4. https://doi.org/10.1109/ICCUBEA47591.2019.9129565

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*,*9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Janani, R., & Vijayarani, S. (2019). Text Classification A Comparative Analysis of Word

Embedding Algorithms. *International Journal of Computer Sciences and Engineering*, *7*(4), 818–822. https://doi.org/10.26438/ijcse/v7i4.818822

Kurnia, R. I. (2020). Classification of User Comment Using Word2vec and SVM Classifier. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(1), 643–648. https://doi.org/10.30534/ijatcse/2020/90912020

Li, S., & Gong, B. (2021). Word embedding and text classification based on deep learning methods. *MATEC Web of Conferences*, *336*, 06022. https://doi.org/10.1051/matecconf/202133606022

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (n.d.). *Learning Word Vectors for Sentiment Analysis* (pp. 142–150).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. http://arxiv.org/abs/1301.3781

Multi-class Review Rating Classification using Deep Recurrent Neural Network | SpringerLink. (2014). Retrieved July 4, 2022, from https://link.springer.com/article/10.1007/s11063-019-10125-6

Nlp Coreference Resolution. (2018). Retrieved July 3, 2022, from https://www.adoclib.com/blog/nlp-coreference-resolution.html

Park,) H. Oyeon, Kim, K.-J., Candidate, P. D., First, •, & Park, H. (2020). Impact of Word Embedding Methods on Performance of Sentiment Analysis with Machine Learning Techniques. 한국컴퓨터정보학회논문지|*Journal of The Korea Society of Computer and Information*, *25*(8), 181–188. https://doi.org/10.9708/jksci.2020.25.08.181

Scikit-learn: Machine learning in Python—Scikit-learn 1.1.1 documentation. (2022). Retrieved July 4, 2022, from https://scikit-learn.org/stable/

Tweepy. (n.d.). Retrieved July 4, 2022, from https://www.tweepy.org/

Twitter Sentiment Analysis Using Supervised Machine Learning | SpringerLink. (2017). Retrieved July 4, 2022, from https://link.springer.com/chapter/10.1007/978-981-15-9509-7_51

Wang, C., Nulty, P., & Lillis, D. (2020). A Comparative Study on Word Embeddings in Deep Learning for Text Classification. *ACM International Conference Proceeding Series*, 37–46. https://doi.org/10.1145/3443279.3443304

What is Natural Language Processing? | IBM. (2020). Retrieved July 3, 2022, from https://www.ibm.com/cloud/learn/natural-language-processing

Yuwono, J. (2015). *Natural Language Processing | Artificial Intelligence in Palm Oil Mill and Plantation*. Plantation.Ai. Retrieved July 3, 2022, from https://www.plantation.ai/features/natural-language-processing.html

Zhang, W., Yoshida, T., & Tang, X. (2011). A comparative study of TF*IDF, LSI and multi-words for text classification. *Expert Systems with Applications*, *38*(3), 2758–2765. https://doi.org/10.1016/j.eswa.2010.08.066

Annexure: Details of Plagiarism Report

sharath

by Plaigarism check

Submission date: 10-Jul-2022 12:49PM (UTC-0400)
Submission ID: 1868679097
File name: plagerism_test_1_-_42_B_Sharath_H_N.docx.pdf (202.78K)
Word count: 8823
Character count: 46471

sharath

ORIGINALITY REPORT

17%	5%	7%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	towardsdatascience.com Internet Source	2%
2	Submitted to British University in Egypt Student Paper	1%
3	Submitted to Gitam University Student Paper	1%
4	Submitted to SSN COLLEGE OF ENGINEERING, Kalavakkam Student Paper	1%
5	Congcong Wang, Paul Nulty, David Lillis. "A Comparative Study on Word Embeddings in Deep Learning for Text Classification", Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval, 2020 Publication	1%
6	Zhang, W.. "A comparative study of TF*IDF, LSI and multi-words for text classification", Expert Systems With Applications, 201103 Publication	1%

7	Submitted to PEC University of Technology Student Paper	1%
8	Submitted to Jawaharlal Nehru University (JNU) Student Paper	1%
9	"Intelligent Data Communication Technologies and Internet of Things", Springer Science and Business Media LLC, 2021 Publication	1%
10	Submitted to SHAPE (VTC college) Student Paper	1%
11	doctorpenguin.com Internet Source	1%
12	Submitted to Coventry University Student Paper	1%
13	Submitted to University of Wales Institute, Cardiff Student Paper	1%
14	www.iscmi.us Internet Source	1%
15	ethesis.nitrkl.ac.in Internet Source	<1%
16	www.woorank.com Internet Source	<1%
	Submitted to South Bank University	

17	Student Paper	<1%
18	download.entrust.com Internet Source	<1%
19	ebin.pub Internet Source	<1%
20	Nikhil Oswal. "Predicting Rainfall using Machine Learning Techniques", Institute of Electrical and Electronics Engineers (IEEE), 2021 Publication	<1%
21	repositories.lib.utexas.edu Internet Source	<1%
22	Long, Jun, Luda Wang, Zude Li, Zuping Zhang, Huiling Li, and Guihu Zhao. "Lexical-semantic SLVM for XML Document Classification", Journal of Software, 2014. Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		