Classifying the News: A Study on Identifying Fake News with Machine Learning Algorithms

Submitted by: Sameeksha Chiguru
IST 529 Text Analysis
University at Albany, SUNY
Albany, NY 12222, United States

May 2, 2023

Abstract

With the rise of online news consumption, fake news has become increasingly prevalent, posing a significant threat to journalism's integrity and causing political disruption. Fake news refers to deliberately misleading and factually incorrect information published on various online platforms, including social media and news websites. Detecting fake news is a challenging task because it is purposely crafted to deceive readers. Therefore, it is crucial to develop methods to identify and address this problem. The report presents a comprehensive analysis of the existing research on fake news detection and proposes an approach that uses traditional machine learning models, such as Python's scikit-learn and NLP, to classify news articles as either true or false based on their textual content. The proposed method involves using feature extraction, vectorization, and feature selection techniques to achieve the highest precision in identifying fake news, ultimately contributing to the ongoing efforts to combat the spread of false information.
Keywords: Fake news, API, Supervised Learning, NLP

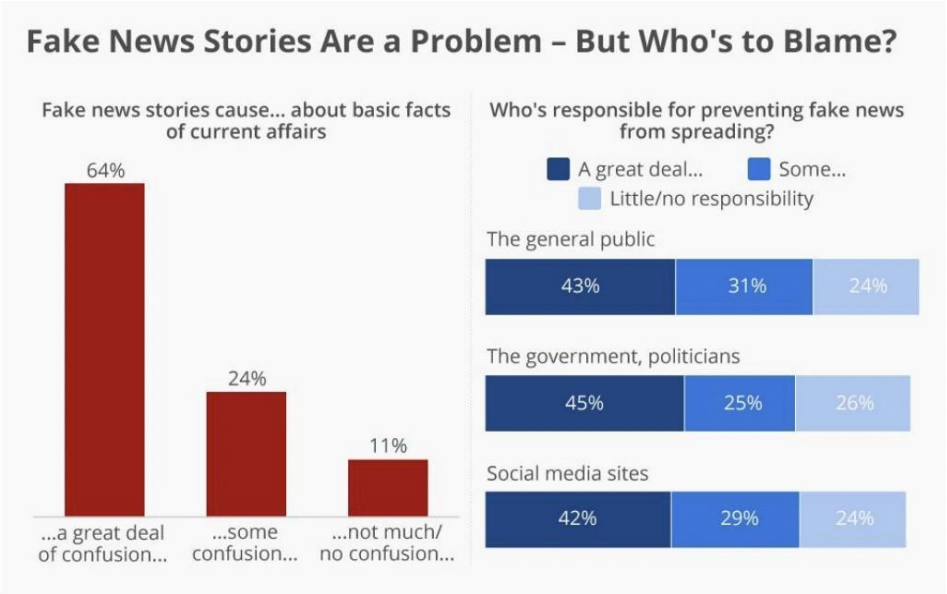
Contents

1 INTRODUCTION

In recent years, the widespread use of online social networks has led to a fundamental shift in the way people consume news and information.According to a report by the Pew Research Center, 62 percent of people in the United States gained news from social media in 2016, compared to 49 percent who watched news through social media in 2012. However, the convenience of social networks has also led to the dissemination of fake news, which poses a significant threat to individuals and society due to the lack of control, supervision, and automatic fact-checking. This has led to low-quality and fake content generation, making the general public vulnerable to countless disinformation and misinformation on social networks, including fake news. Therefore, there is a pressing need to develop algorithms for the automated identification of fake news to minimize its impacts in the political, economic, or social contexts.

In recent years, numerous studies have been conducted to detect fake news using algorithms, achieving relevant accomplishments in terms of accuracy. However, there is a need for more successful algorithms and datasets in identifying fake news to better support researchers' understanding of the state-of-the-art.

Fake news analysis



This research paper aims to analyze the accuracy obtained and the datasets used in fake news identification algorithms to consolidate recommendations for future studies. In this context, tech companies such as Google, Facebook, and Twitter have attempted to address this particular concern, but their efforts have hardly contributed towards solving the problem. The organizations have resorted to denying the individuals associated with such sites the revenue that they would have realized from the increased traffic. Users, on the other hand, continue to deal with sites containing false information, and their involvement tends to affect the reader's ability to engage with actual news. In this research paper, we present a machine learning-based project that uses various classification algorithms and natural language processing (NLP) to classify news articles into fake or real based on their title and body content. The data has been

collected from two different sources. Classification is allocating objects into classes and groups [1]. Classification or supervised learning supposes a relationship *R* between elements of a set *E*. The objective of this research is to analyze the accuracy obtained and the datasets used in fake news identification algorithms. The study intends to contribute by highlighting more successful algorithms and datasets in identifying fake news, to better support researchers' understanding of the state-of-the-art, given the variety of options available. Besides, it consolidates recommendations for future studies.

1.1 Research Problem

This research project aims to investigate and evaluate the efficacy of various machine learning algorithms and NLP methods for precisely categorizing news articles as either real or fake.

1.2 Proposed Solution

The aim of this research is to address the problem of identifying fake news by creating a model that performs binary classification to predict the credibility of news articles as either fake or real. This involves assessing the effectiveness of different machine learning algorithms, such as Random Forest and SVM, in accurately classifying news based on their attributes. The performance of these algorithms will be evaluated using metrics such as accuracy, precision, recall, and F1-score.

2 LITERATURE REVIEW

The problem of fake news detection is of utmost importance in today's information age. Many studies have been conducted from various perspectives to address this problem, utilizing different techniques to combat misinformation. However, traditional approaches based on verification by humans and expert journalists are not scalable to the volume of news content generated online [2]. Text classification, a fundamental task in natural language processing (NLP), has been extensively researched in this regard [3]. For instance, one study proposed a model that could check the real-time credibility of news within 35 seconds by combining user-based, propagation-based, and content-based text [4].

Detecting fake news on social media platforms is challenging because such content can be easily shared. However, Facebook examines machine learning classifiers' effectiveness in identifying fake news by using information sources such as visual and cognitive cues and social judgment, including cognitive, behavioral, and affective factors [5]. In detecting fake news, the Support Vector Machine algorithm has been preferred due to its extensive research and development in recent times [6]. However, the selection of classifiers depends on organizational requirements. Additionally, stance detection of the headline for binary classification through n-gram matching can also be assessed after comparing ”related” vs. ”unrelated” pairs, which can be useful in detecting fake news, especially clickbait detection.

2.1 Supervised Learning

In supervised learning, a model learns from labeled datasets and then applies that learning to label a new data point. Supervised learning [7] is divided into two types i.e. Classification and Regression. The difference between classification and regression is the type of output. Classification predicts a class label of an unlabeled observation while regression predicts a numerical label. The objective is to come up with a mapping from x to y given a training set that is made up of (x_i,y_i). y_i is the label of x_i. The function that is created using the training set can be evaluated through prediction of the same function on a test set.

2.2 Classification

Unsupervised learning helps in identification of unknown structures

2.2.1 Random Forest Classifier

Random Forests are built on the concept of building many decision tree algorithms, after which the decision trees get a separate result. The results, which are predicted by large number of decision trees, are taken up by the random forest. To ensure a variation of the decision trees, the random forest randomly selects a subcategory of properties from each group [8][9] The applicability of Random forest is best when used on uncorrelated decision trees. If applied on similar trees, the overall result will be more or less similar to a single decision tree. Uncorrelated decision trees can be obtained by bootstrapping and feature randomness.

Figure 1: Random forest classifier pseudo-code

Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - sub-sampling size

Output: a set of t *iTrees*

1: **Initialize** $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: **for** $i = 1$ to t **do**
4: $X' \leftarrow sample(X, \psi)$
5: $Forest \leftarrow Forest \cup iTree(X', 0, l)$
6: **end for**
7: **return** $Forest$

2.2.2 Support Vector Machine(SVM)

The SVM algorithm is based on the layout of each data item in the form of a point in a range of dimensions (the number of available properties), and the value of a given property is the number of specified coordinates [10]. Given a set of n features, SVM algorithm uses n dimensional

space to plot the data item with the coordinates representing the value of each feature. The hyper-plane obtained to separate the two classes is used for classifying the data.

Figure 2: SVM pseudo-code

Require: $\boldsymbol{X}$ and $\boldsymbol{y}$ loaded with training labeled data, $\boldsymbol{\alpha} \Leftarrow 0$ or $\boldsymbol{\alpha} \Leftarrow$ partially trained SVM

$C \Leftarrow$ some value (10 for example)
repeat
  for all $\{\boldsymbol{x_i}, y_i\}, \{\boldsymbol{x_j}, y_j\}$ **do**
    Optimize α_i and α_j
  end for
until no changes in $\boldsymbol{\alpha}$ or other resource constraint criteria met

Ensure: Retain only the support vectors ($\boldsymbol{\alpha_i} > 0$)

2.2.3 Logistic regression

Logistic regression is a statistical method for binary classification problems, where the goal is to predict the probability of a binary outcome based on one or more independent variables. The logistic function, also known as the sigmoid function, is used to model the relationship between the dependent variable and the independent variables. The logistic function maps real-valued inputs to a value between 0 and 1, allowing the output of the model to be interpreted as the probability of the dependent variable taking a particular value.

Figure 3: Logistic regression pseudo-code

Input: Training data
Begin
For i = 1 to k
For each training data instance d_i.
Set the target value for the regression to $z_i = \frac{y_i - P(1|d_j)}{[P(1|d_j)(1-P(1|d_j))]}$
Initialize the weight of instance d_j to $[P(1|d_j)(1 - P(1|d_j))]$
Finalize a $f(j)$ to the data with class value (Z_j) and weight (w_j)
Classical label decision
Assign (class label: 1) if $P_{id} > 0.5$, otherwise (class label: 2)
End

2.3 Combining Classifiers

The main objective in designing paradigm detection systems is to achieve optimal taxonomic performance. To this end, various classification strategies for action detection models can be developed. Even if one model exhibits the best performance, it is not necessary for the style sets correctly categorized by different classifiers to overlap. Different classification strategies can provide additional insights into the models, which can improve the performance of individual models [11].

2.4 Related Work on Fake News Detection

[12] pointed out various sources of media and made suitable studies on whether the submitted article is reliable or fake. The paper utilizes models based on speech characteristics and predictive models that do not fit with the other current models.

[13] estimated various ML algorithms and made the research on the percentage of the prediction. The accuracy of various predictive patterns including bounded decision trees, gradient enhancement, and support vector machine were assorted. The patterns are estimated based on an unreliable probability threshold with 85-91% accuracy.

[14] utilized the Naive Bayes classifier, discussing how to implement fake news discovery to different social media sites. They used Facebook, Twitter and other social media applications as data sources for news. Accuracy is very low because the information on this site is not 100% credible.

[15][16][17] discuss misleading and discovering rumors in real time. It utilizes a novelty-based characteristic and derives its data source from Kaggle. The accuracy average of this pattern is 74.5%. Clickbait and sources are not considered unreliable, resulting in a lower resolution.

[18] presented a counterfeit detection model using N-gram analysis by the lenses of various characteristic extraction techniques. In addition, we examined the extraction techniques of various features and six different methods of machine learning. The proposed model achieves the highest accuracy in use Contains a unigram and a linear SVM workbook. The highest accuracy is 92%.

[19] used naïve Bayes classifier to detect fake news by Naive Bayes. This method was performed as a software framework and experimented with various records from Facebook, etc., resulting in an accuracy of 74%. The paper neglected the punctuation errors, resulting in poor accuracy.

3 METHODOLOGY

3.1 Data

In this project, we used data from two different sources, namely The Guardian News articles, and Kaggle. For the Guardian dataset, we accessed their articles through an API key and obtained the data in json format. To clean the dataset, we extracted the headlines and body, removed unwanted news sections, removed empty spaces, and stored it in a csv file. This process resulted in a dataset with 13,000 rows.

The fake news dataset was obtained from Kaggle and cleaned by filtering out non-English articles, removing unwanted and empty columns, and assigning a binary value of 1 to fake news and 0 to real news. We merged these datasets into a single csv file for

implementation. We then used the TF-IDF scores to determine the frequency of each term in the entire dataset.

Dataframe of Data collected from Guardian API

```
df_guard.head()
```

	id	published	body	headline	fakeness
0	world/live/2022/jan/01/covid-live-limits-on-fr...	2022-01-01T23:55:23Z	Here's a roundup of this evening's Covid-19 ne...	Today's Covid news, as it happened: US Omicron...	0
1	world/2022/jan/01/boris-johnson-asks-ministers...	2022-01-01T22:30:00Z	Boris Johnson has instructed ministers to come...	Boris Johnson asks ministers to plan for Covid...	0
2	world/2022/jan/01/schools-in-england-told-wear...	2022-01-01T22:30:00Z	All pupils in secondary schools in England sho...	Schools in England told to wear masks in class...	0
3	politics/2022/jan/01/priti-patel-vows-to-curb-...	2022-01-01T22:04:28Z	The home secretary has said she intends to cra...	Priti Patel vows to curb eco protests and asyl...	0
4	us-news/2022/jan/01/us-omicron-covid-cases-tee...	2022-01-01T21:07:03Z	As the US is seeing record numbers of daily co...	Teens and young adults driving record Covid ca...	0

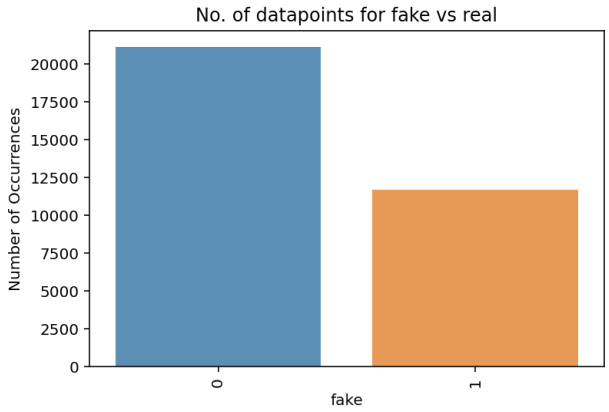
Dataframe of Data collected from Kaggle

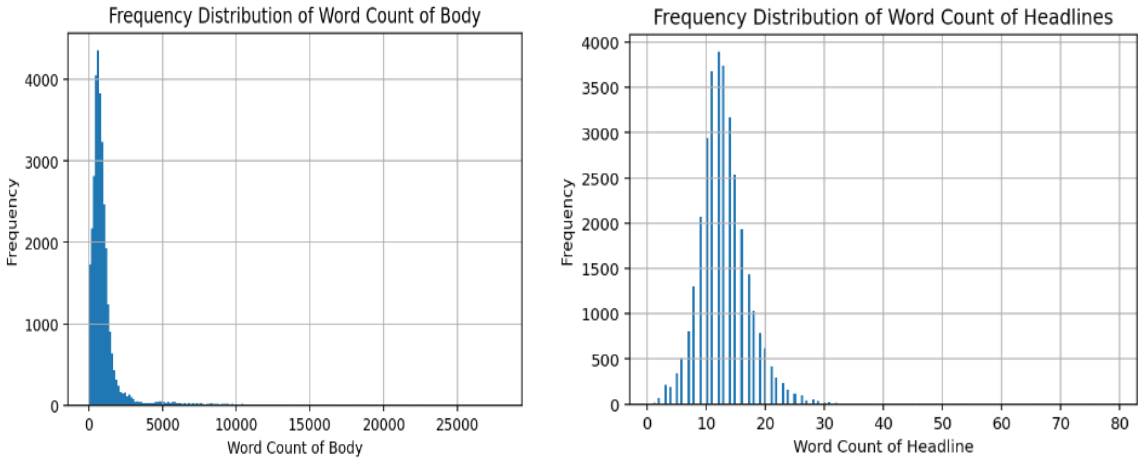
```
df_kaggle.head()
```

	headline	body	published	id	fakeness
0	Muslims BUSTED: They Stole Millions In Gov't B...	Print They should pay all the back all the mon...	2016-10-26T21:41:00.000+03:00	6a175f46bcd24d39b3e962ad0f29936721db70db	1
1	Re: Why Did Attorney General Loretta Lynch Ple...	Why Did Attorney General Loretta Lynch Plead T...	2016-10-29T08:47:11.259+03:00	2bdc29d12605ef9cf3f09f9875040a7113be5d5b	1
2	BREAKING: Weiner Cooperating With FBI On Hilla...	Red State : \nFox News Sunday reported this mo...	2016-10-31T01:41:49.479+02:00	c70e149fdd53de5e61c29281100b9de0ed268bc3	1
3	PIN DROP SPEECH BY FATHER OF DAUGHTER Kidnappe...	Email Kayla Mueller was a prisoner and torture...	2016-11-01T05:22:00.000+02:00	7cf7c15731ac2a116dd7f629bd57ea468ed70284	1
4	FANTASTIC! TRUMP'S 7 POINT PLAN To Reform Heal...	Email HEALTHCARE REFORM TO MAKE AMERICA GREAT ...	2016-11-01T21:56:00.000+02:00	0206b54719c7e241ffe0ad4315b808290dbe6c0f	1

The attributes of the final dataset are:

Id: A unique identifier for the article
Publication date: Article's publication date
Headline: Headline of the article
Body: The textual content of the article
Fakeness: binary 0/1 for real and fake news



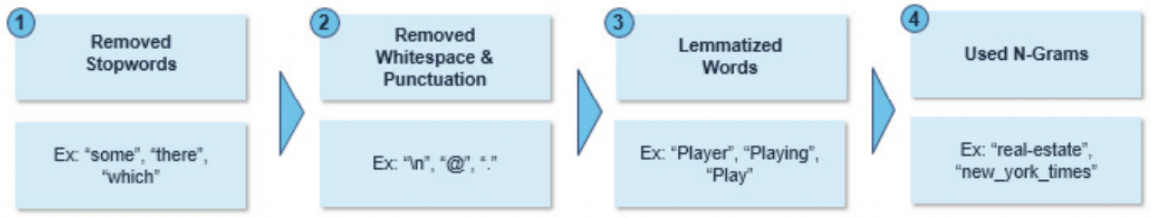


3.1.1 Data cleaning

The dataset used in this project underwent cleaning and filtering to remove unwanted columns and NULL values, as well as to retain only the English language.

For the Kaggle dataset, the initial dataset contained 12,999 rows and 8 columns, but was further cleaned by dropping various columns and rows with null values, resulting in 11,677 rows. A new column named "fakeness" was added and defined as 1 to indicate fake news.

The Guardian dataset was collected by scraping articles from their website in json format, which was then combined and converted into a csv file. The dataset initially had 59,151 rows and 14 columns, but was further cleaned by dropping unnecessary columns and rows with null values. Only news articles from the "business", "world news", "politics", and "US news" sections were retained, resulting in 13,071 rows.



To extract and clean Guardian dataset we perform text preprocessing and analysis on news articles from The Guardian. It reads in multiple JSON files, combines them into a single dataframe, and filters the articles based on topic categories. The body text and headline columns are extracted and cleaned of empty values. The cleaned data frame is then saved to a CSV file. Next, the TfidfVectorizer is applied to the body text and headline columns separately to obtain a

weighted score for each term in the text. These scores are then plotted using Seaborn to visualize the importance of the top terms. This code can be used as a starting point for analyzing text data and gaining insights from it.

The kaggle dataset is then cleaned by removing articles that are not in English and those that contain missing values in their text or title fields. The cleaned dataset is then processed using the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm, which calculates the importance of each word in the text of the articles, and generates a feature vector representation for each article. The importance of each word is visualized using a bar chart generated by matplotlib and seaborn libraries. Finally, the cleaned dataset is saved to a CSV file for future us

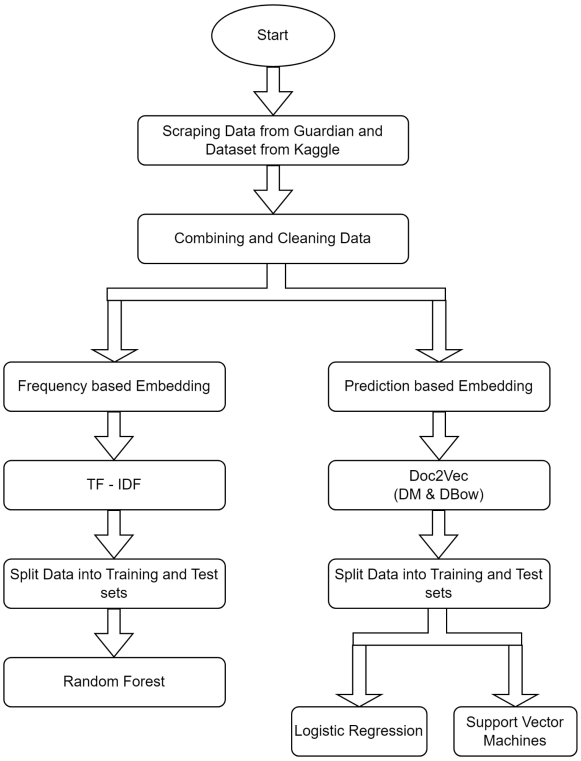
3.2 Model

Word Embedding: Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text. A Word Embedding format generally tries to map a word using a dictionary to a vector, in more simpler terms it finds out the relation between different text written in context.

The different types of word embeddings can be broadly classified into two categories

1. Frequency based Embedding
2. Prediction based Embedding

Flow Diagram of the model

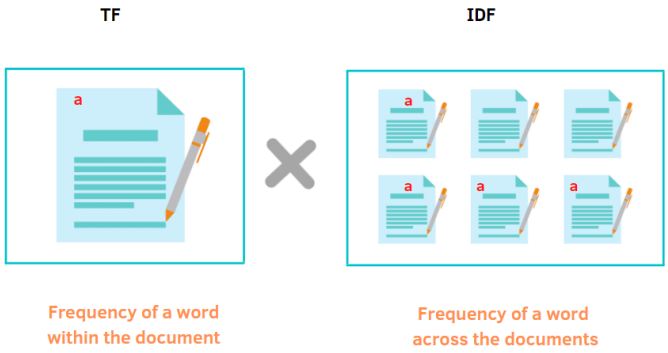


The first approach involves using a frequency-based embedding method, specifically TF-IDF, to convert the text data into numerical vectors. After applying TF-IDF, the data is split into a training set and a test set, with 75% of the data being used for training and 25% for testing. Then, a random forest classifier is trained on the training data, which is a type of machine learning algorithm that constructs a multitude of decision trees and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

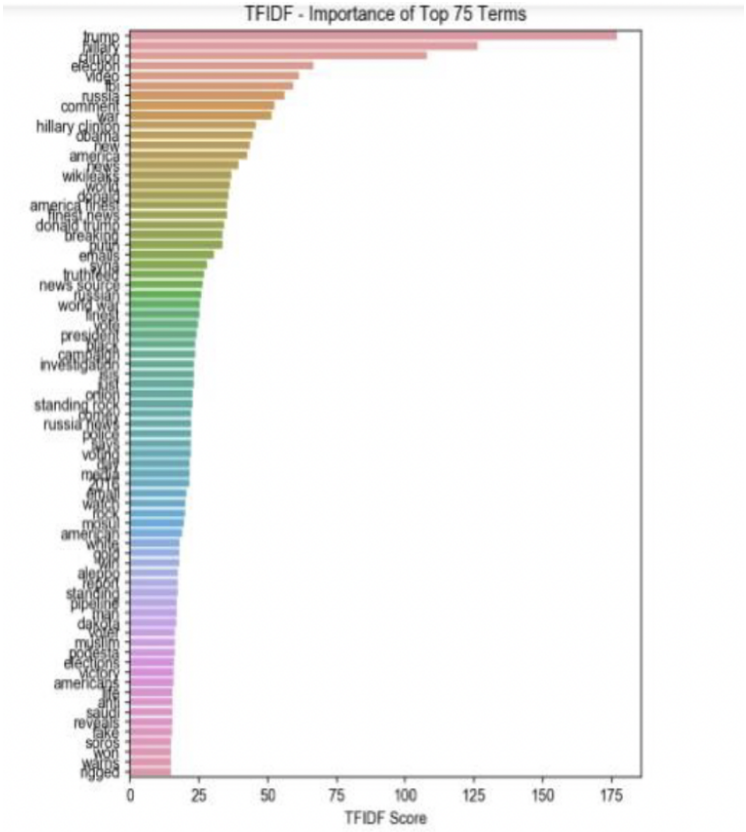
The second approach involves using a prediction-based embedding method, specifically Doc2Vec, to convert the text data into numerical vectors. Doc2Vec is an extension of the popular Word2Vec algorithm that learns vector representations of words in a corpus. After applying Doc2Vec, the data is split into a training set and a test set, with 75% of the data being used for training and 25% for testing. Then, two different classifiers, Support Vector Machine (SVM) and logistic regression are used.

3.2.1 Frequency based Embedding

There are different types of vectors used for text analysis, including count vector, TF-IDF vector, and co-occurrence vector. For this project, we utilized the TF-IDF vectorization technique to calculate the weight of each word in the dataset. We applied TF-IDF to individual datasets to identify the most important words and then to the final dataset. We used an n-gram window range of (1,2) to generate additional features along with each word. TF-IDF tokenized our string dataset into corresponding vectors by assigning each word a score.



The resulting dataset was then ready for training using any algorithm. To train and test our model, we split the dataset into training and testing data using the train_test_split function from the sklearn.cross_validation module with a 0.75 and 0.25 split respectively. We then applied the Random Forest Classifier algorithm for training the dataset. Random Forest is an improvement over the decision tree algorithm, as it trains the dataset on various features that were formed by the TF-IDF vectorizer. We predicted the values of the testing data and evaluated the accuracy using the F1 score and accuracy matrix. The accuracy of our model on the body of the news dataset was 93.9%.



The datasets are concatenated, and a new column called "fakeness" is added to the combined dataset to distinguish between fake and real news articles. The combined dataset is split into a training and a testing set, and then vectorized using TfidfVectorizer. The vectorization is performed on two different features of the dataset, the headline and the body text. The Random Forest Classifier is applied on both vectorized headline and body text data to train the machine learning model.

3.2.2 Prediction based Embedding

These methods were prediction based in the sense that they provided probabilities to the words and proved to be state of the art for tasks like word analogies and word similarities.These methods are able to perform tasks like "King-man + woman = Queen," which was once considered almost incredible. The approach for learning paragraph vectors is inspired by the methods for learning word vectors.

The idea is that word vectors contribute to a prediction task about the next word in the sentence. Word vectors can capture semantics as an indirect result of the prediction task, despite being randomly initialized. The same idea is used for paragraph vectors, where they are asked to contribute to the prediction task of the next word given many contexts sampled from the paragraph.

Word2Vec is a combination of two techniques, CBOW and Skip-gram, which convert every word into a vector. Both of these are shallow neural networks that map words to the target variable, which is also a word(s). These techniques learn weights that act as word vector representations. Doc2Vec modifies the Word2Vec algorithm to unsupervised learning of continuous representations for larger blocks of text, such as sentences, paragraphs, or entire documents. The Doc2Vec architecture has two corresponding algorithms, "distributed memory" (DM) and "distributed bag of words" (DBOW). The input to Doc2Vec is an iterator of labelled sentence objects, where each object represents a single sentence and consists of two simple lists: a list of words and a list of labels.

The algorithm runs through the sentence iterator twice, once to build the vocab and once to train the model on the input data, learning a vector representation for each word and for each label in the dataset. The DM defines the training algorithm. If DM=1, it means "distributed memory" (PV-DM), and if DM=0, it means "distributed bag of words" (PV-DBOW). The Distributed Memory model preserves the word order in a document, whereas the Distributed Bag of Words just uses the bag-of-words approach, which doesn't preserve any word order.

We split the data into training and testing sets, and create Doc2Vec models for Distributed Memory (DM) and Distributed Bag of Words (DBOW) approaches. These models learn to generate document embeddings, which are vectors that capture the meaning of the entire document. The embeddings are then used as features in a logistic regression model, which is trained and tested on the respective training and testing sets. Finally, the script concatenates the two Doc2Vec models into one model and evaluates its performance using the same logistic regression model.

4 RESULTS

4.1 Evaluation metrics

Accuracy (A): Accuracy is a measure of the classifier's ability to correctly classify a piece of information as either fake or real. The accuracy can be estimated using Equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision (P): Precision is a measure for the classifier's exactness such that a low value indicates a large number of false positives. The precision represents the number of positive predictions divided by the total number of positive class values predicted and is calculated using Equation

$$Precision = \frac{TP}{TP + FP}$$

Recall (R): Recall is considered a measure of a classifier's completeness (e.g., a low value of recall indicates many false negatives), where the number of true positives is divided by the number of true positives and the number of false negatives, as can be clearly seen in Equation

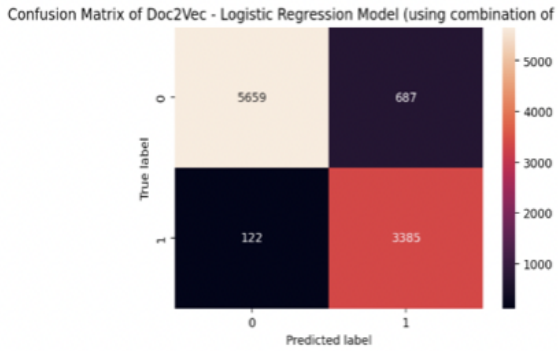
$$Recall = \frac{TP}{TP + FN}$$

F1 score (F1): The F1 score is calculated as the weighted harmonic mean of the precision and recall measures of the classifier using Equation

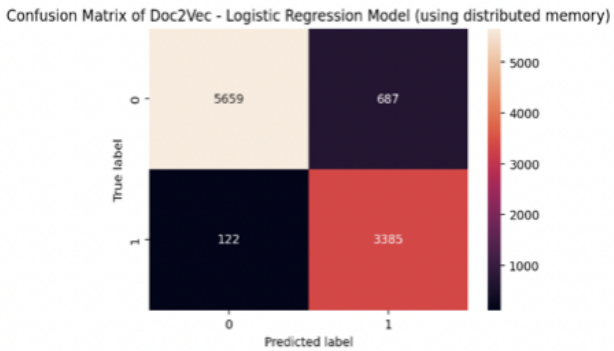
$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

4.2 Visualizations

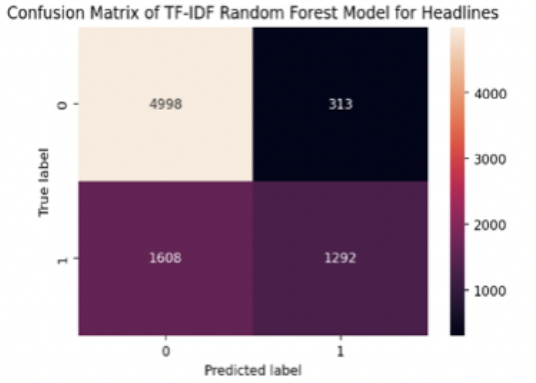
A confusion matrix is a table that is commonly used to evaluate the performance of a classification model. It allows the visualization of the accuracy of predictions made by a model on a particular dataset.The confusion matrix helps to calculate important metrics such as accuracy, precision, recall, and F1 score, which are used to evaluate the performance of the classification model.



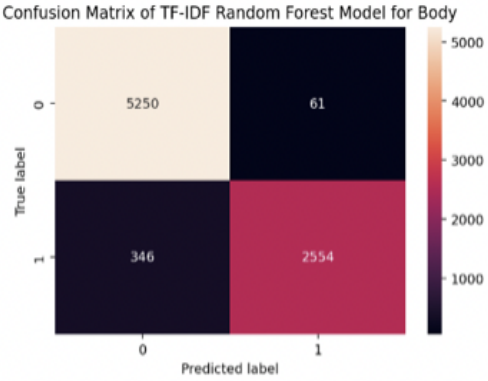
	precision	recall	f1-score	support
0	0.98	0.89	0.93	6346
1	0.83	0.97	0.89	3507
accuracy			0.92	9853
macro avg	0.91	0.93	0.91	9853
weighted avg	0.93	0.92	0.92	9853



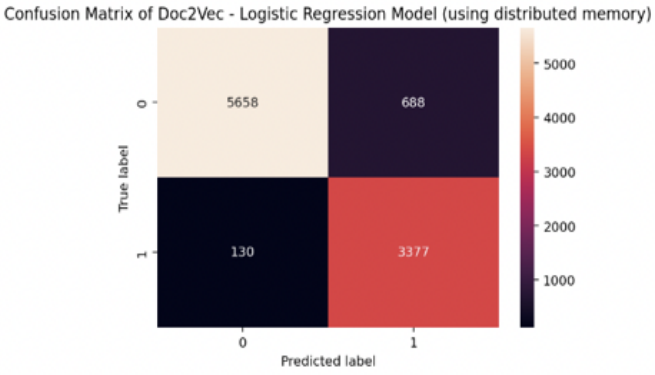
	precision	recall	f1-score	support
0	0.98	0.89	0.93	6346
1	0.83	0.97	0.89	3507
accuracy			0.92	9853
macro avg	0.91	0.93	0.91	9853
weighted avg	0.93	0.92	0.92	9853



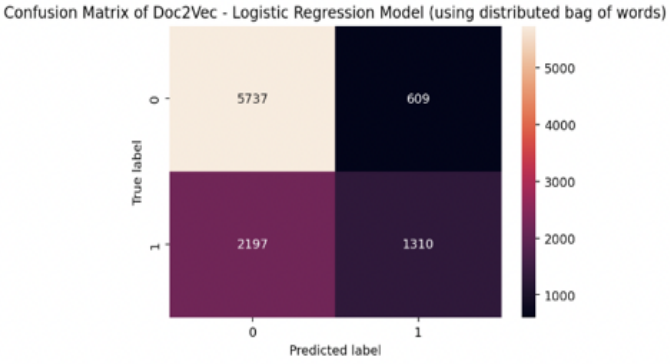
	precision	recall	f1-score	support
0	0.76	0.94	0.84	5311
1	0.80	0.45	0.57	2900
accuracy			0.77	8211
macro avg	0.78	0.69	0.71	8211
weighted avg	0.77	0.77	0.75	8211



	precision	recall	f1-score	support
0	0.94	0.99	0.96	5311
1	0.98	0.88	0.93	2900
accuracy			0.95	8211
macro avg	0.96	0.93	0.94	8211
weighted avg	0.95	0.95	0.95	8211



	precision	recall	f1-score	support
0	0.98	0.89	0.93	6346
1	0.83	0.96	0.89	3507
accuracy			0.92	9853
macro avg	0.90	0.93	0.91	9853
weighted avg	0.93	0.92	0.92	9853



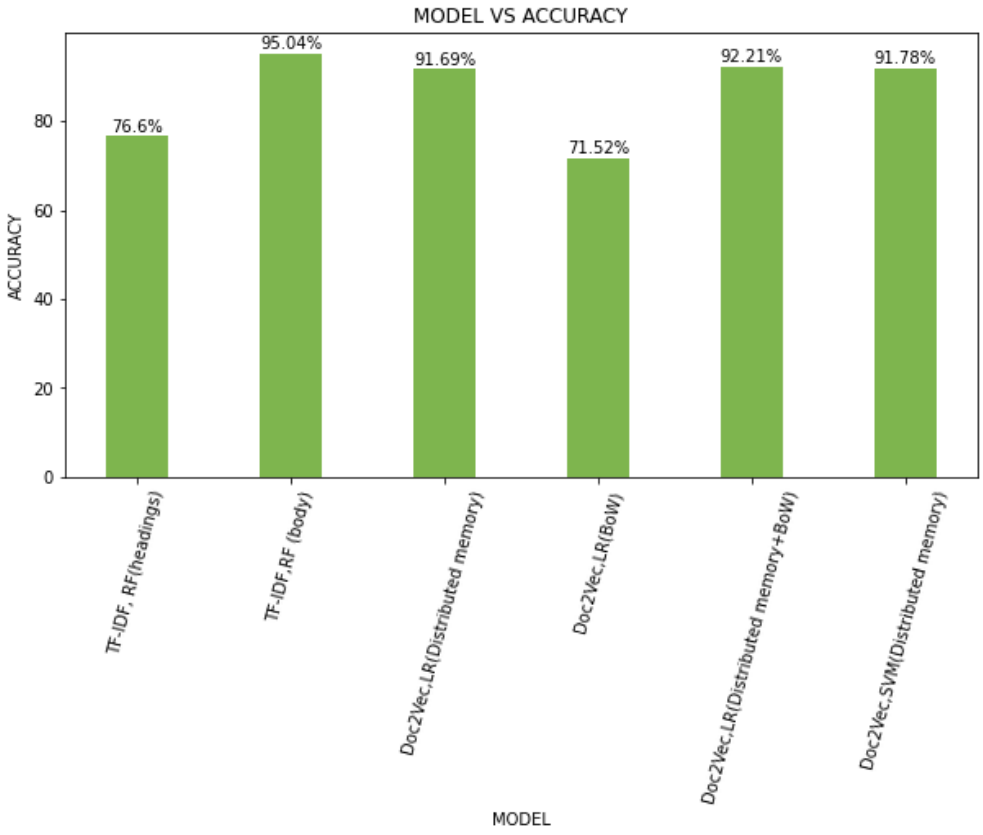
	precision	recall	f1-score	support
0	0.72	0.90	0.80	6346
1	0.68	0.37	0.48	3507
accuracy			0.72	9853
macro avg	0.70	0.64	0.64	9853
weighted avg	0.71	0.72	0.69	9853

4.3 Conclusion

A table is drawn between Models and Evaluation metrics for comparison.

Model		Accuracy	Precision	Recall	F1 Score
TF-IDF and Random Forest	with headings	76.6%	78%	69%	71%
	with body	95.04%	96%	93%	94%
Doc2Vec and Logistic Regression	using Distributed Memory	91.69%	90%	93%	91%
	using Bag of words	71.52%	70%	64%	64%
	combination of distributed memory and bag of words	92.21%	91%	93%	91%
Doc2Vec and SVM	using Distributed Memory	91.78%	91%	93%	91%

The TF-IDF (Term Frequency-Inverse Document Frequency) technique is a popular approach used in natural language processing and information retrieval to identify the importance of a word or phrase within a text document. In the case of the news articles, the TF-IDF technique would have identified the words or phrases that are commonly used across many news articles (i.e., high term frequency), but at the same time, the technique would have also identified the words or phrases that are unique to a specific news article (i.e., low document frequency). By using the TF-IDF technique, the Random Forest model was able to learn and identify the most significant features (words or phrases) that are most relevant to the classification task. These features would have been given higher importance scores, allowing the model to distinguish between different categories of news articles more accurately. In summary, the TF-IDF technique helped the Random Forest model to identify the most relevant features and highlight the significant words or phrases within the text documents that are most useful in predicting the category of the news articles. As a result, this model achieved high accuracy and F1 scores, making it one of the best models for this particular task.



The above graph is a bar graph between all the models and their accuracies. We can see that TF-IDF with Random forest has highest accuracy followed by Doc2Vec using Logistic regression with distributed memory.

5 DISCUSSION AND FUTURE WORK

Considering the changing landscape of the modern business world, the issue of fake news has become more than just a marketing problem as it warrants serious efforts from security researchers. It is imperative that any attempts to manipulate or troll the Internet through fake news or Clickbaits are countered with absolute effectiveness. We proposed a simple but effective approach to allow users to install a simple tool into their personal browser and use it to detect and filter out potential fake news.The work in this project proposes a stacked model which fine tunes the informational insight gained from the data at each step and then tries to make a prediction. Although many attempts have been made to solve the problem of fake news, any significant success is yet to be seen. With huge amounts of data collected from social media

websites like Facebook, Twitter, etc., the best models improve every day. With the use of deep neural networks, the future work in this field seems a lot more promising.

5.1 Limitations

Difficulty in detecting context-specific fake news: The model may struggle to detect fake news that is specific to a certain context or location, as it may not be able to generalize well to other contexts.

5.2 Future work

Future work includes building an automated fact-checking system that combines data and knowledge to help non-experts and checks the content of the news thoroughly after comparing it with known facts. We want to look into the issue of fake news from different angles like known facts, source, topic, associated URLs, geographical location, year of publication, and credibility of the source for a better understanding of the problem.

News can contain a vast range of topics. Just the classification based on labels is not enough if realistic results are desired. For this reason, an advanced technique called topic modelling can come in handy. Topic modelling categories each piece of text into topics and using this one can make more accurate predictions. The most popular topic modelling technique used in NLP is “Latent Dirichlet Allocation‟, also known as LDA.Use of LDA can add another layer of depth to the fake news classification task.

6 REFERENCES

[1] Aldwairi, Monther & Alwahedi, Ali. (2018). Detecting Fake News in Social Media Networks. Procedia Computer Science. 141. 215-222. 10.1016/j.procs.2018.10.171.

[2] de Beer D, Matthee M. Approaches to Identify Fake News: A Systematic Literature Review. Integrated Science in Digital Age 2020. 2020 May 5;136:13–22. doi: 10.1007/978-3-030-49264-9_2. PMCID: PMC7250114.

[3] Reis, J. C., Correia, A., Murai, F., Veloso, A., Benevenuto, F., & Cambria, E. (2019). Supervised Learning for Fake News Detection. IEEE Intelligent Systems, 34(2), 76-81.

[4] Shao, C., Ciampaglia, G. L., Varol, O., Flammini, A., &Menczer, F. (2017). The spread of fake news by social bots. arXiv preprint arXiv:1707.07592, 96-104

[5] Wang, W.Y., 2017. ”liar, liar pants on fire”: A new benchmark dataset for fake news detection. CoRR abs/1705.00648. URL: http: //arxiv.org/abs/1705.00648, arXiv:1705.00648.

[6] Brewer, P.R., Young, D.G., Morreale, M., 2013. The impact of real news about fake news”: Intertextual processes and political satire. International Journal of Public Opinion Research 25, 323–343. URL: http://dx.doi.org/10.1093/ijpor/edt015, doi:10.1093/ijpor/ edt015.

[7] Lorent, S. (2019). Master thesis: Fake news detection using machine learning.

[8] E. C. Tandoc Jr et al. “Defining fake news a typology of scholarly definitions”. Digital Journalism , 1–17. 2017.

[9] Jain, A., & Kasbe, A. (2018, February). Fake news detection. In 2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS) (pp. 1-5). IEEE.

[10] Pal, S., Kumar, T. S., & Pal, S. (2019). Applying Machine Learning to Detect Fake News. Indian Journal of Computer Science, 4(1), 7- 12.

[11] N. J. Conroy, V. L. Rubin, and Y. Chen, “Automatic deception detection: Methods for finding fake news,” Proceedings of the Association for Information Science and Technology, vol. 52, no. 1, pp. 1–4, 2015.

[12] MykhailoGranik and Volodymyr Mesyuarat. “Fake news detection using naive Bayes classifier.” First Ukraine Conference on Electrical and Computer Engineering (UKRCON). Ukraine : IEEE. 2017.

[13] Gilda, S. “Evaluating machine learning algorithms for fake news detection.” 15th Student Conference on Research and Development (SCOReD) (pp. 110-115). IEEE. 2017.

[14] Akshay Jain and AmeyKasbe. “Fake News Detection.” 2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS). Bhopal, India: IEEE. 2018.

[15] Yumeng Qin et al. “Predicting Future Rumours.” Chinese Journal of Electronics (Volume: 27 , Issue: 3 , 5 2018, 514 - 520. [24]. ArushiGupta and RishabhKaushal. “Improving spam

detection in Online Social Networks.” International Conference on Cognitive Computing and Information Processing (CCIP). semanticscholar.org.2015.

[16] Khanam, Z., Ahsan, M.N.”Evaluating the effectiveness of test driven development: advantages and pitfalls.”International. J. Appl. Eng. Res. 12, 7705–7716, 2017

[17] Khanam, Z. “Analyzing refactoring trends and practices in the software industry.” Int. J. Adv. Res. Comput. Sci. 10, 0976–5697, 2018.

[18] Veronica Perez-Rosas et al. Available at: https://www.researchgate.net/publication/319255985_Automatic_Detection_of_Fake_News August, 2017.

[19] Supanya Aphiwongsophon et al. “ Detecting Fake News with Machine Learning Method.” 2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON). Chiang Rai, Thailand, Thailand: IEEE . 2018.

A Appendix - Source code

Guardian_API.py

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
import re
import seaborn as sb
get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('config',
"InlineBackend.figure_format = 'retina'")


# First file to start with
df = pd.read_json(r"D:\projects\sameeksha\Text Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\art
icles\2022-01-01.json" , encoding='utf-8')


# Combining all json files to a one data frame
count = 0
for filename in os.listdir(r"D:\projects\sameeksha\Text
Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\art
icles"):
    count+=1
    if count>1:
        file_path = os.path.join(r'D:\projects\sameeksha\Text
Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\art
icles', filename)
        #print(file_path)
        df_ = pd.read_json(file_path , encoding='utf-8')
```

```
        df = pd.concat(objs= [df,df_], axis=0,ignore_index=True)
print(df.shape)

# Getting bodytext from the dataframe
lst = []
for i in range(df.shape[0]):
    lst.append(df.fields[i]["bodyText"])

# Creating a column with bodytext
df["bodyText"] = lst

# Getting headlines from the dataframe
lst_head = []
for i in range(df.shape[0]):
        lst_head.append(df.fields[i]["headline"])

# Creating a column with headline
df["headline"] = lst_head

# Getting rid of the embty bodies
df = df[df.bodyText != ""]

# Total Number of Articles
df.shape

# Filtering articles based on topics
df = df[(df.sectionName == 'US news') | (df.sectionName ==
'Business') | (df.sectionName == 'Politics') | (df.sectionName ==
'World news')]

# Getting to see the dataframe info (data type, non-null value
etc.)
df = df.reset_index(drop=True)
```

```
df.info()

# To see how many articles in each category
from collections import Counter
Counter(df.sectionName)

# Overview for the dataframe
df.dropna(subset=['bodyText','headline'],inplace=True)
print(df.shape)

# Publication data range
df.webPublicationDate.min() ,df.webPublicationDate.max()

# Saving the cleaned data into csv file
df.to_csv(r"D:\projects\sameeksha\Text Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\Cle
aning Data\guardian cleaned.csv")

tdf = TfidfVectorizer(stop_words='english',ngram_range=(1,2) )
vectorizer = tdf.fit(df.bodyText)
transformed_text = vectorizer.transform(df.bodyText)
transformed_title = vectorizer.transform(df.headline)

def get_tfidf_term_scores(feature_names):
    '''Returns dictionary with term names and total tfidf scores
for all terms in corpus'''
    term_corpus_dict = {}
    # iterate through term index and term
    for term_ind, term in enumerate(feature_names):
        term_name = feature_names[term_ind]
        term_corpus_dict[term_name] =
np.sum(transformed_title.T[term_ind].toarray())
    return term_corpus_dict
```

```
# list of features created by tfidf
feature_names = tdf.get_feature_names()
term_corpus_dict = get_tfidf_term_scores(feature_names)

def get_sorted_tfidf_scores(term_corpus_dict):
     #Returns sort words from highest score to lowest score
    # sort indices from words wit highest score to lowest score
    sortedIndices = np.argsort(
list(term_corpus_dict.values()))[::-1]
    # move words and score out of dicts and into arrays
    termNames = np.array(list(term_corpus_dict.keys()))
    scores = np.array(list(term_corpus_dict.values()))
    # sort words and scores
    termNames = termNames[sortedIndices]
    scores = scores[sortedIndices]
    return termNames, scores

termNames, scores = get_sorted_tfidf_scores(term_corpus_dict)

def plot_tfidf_scores(scores,termNames, n_words = 18):
    '''Returns one plot for Importance of Top Terms'''

    # Create a figure instance, and the two subplots
    fig = plt.figure(figsize = (14, 18))
    override = {'fontsize': 'large'}
    fig.add_subplot(221)   #top left
    n_words = 75
    sb.set()
    sb.barplot(x = scores[:n_words], y = termNames[:n_words]);
    plt.title("TFIDF score ".format(n_words));
    plt.xlabel("TFIDF Score");
```

```
plot_tfidf_scores(scores, termNames,  n_words = 18)
```

Kaggle_data.py

```
# First file to start with
df = pd.read_json(r"D:\projects\sameeksha\Text Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\art
icles\2022-01-01.json" , encoding='utf-8')


# Combining all json files to a one data frame
count = 0
for filename in os.listdir(r"D:\projects\sameeksha\Text
Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\art
icles"):
    count+=1
    if count>1:
        file_path = os.path.join(r'D:\projects\sameeksha\Text
Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\art
icles', filename)
        #print(file_path)
        df_ = pd.read_json(file_path , encoding='utf-8')
        df = pd.concat(objs= [df,df_], axis=0,ignore_index=True)


print(df.shape)


# Getting bodytext from the dataframe
lst = []
for i in range(df.shape[0]):
    lst.append(df.fields[i]["bodyText"])


# Creating a column with bodytext
```

```
df["bodyText"] = lst

# Getting headlines from the dataframe
lst_head = []
for i in range(df.shape[0]):
        lst_head.append(df.fields[i]["headline"])

# Creating a column with headline
df["headline"] = lst_head

# Getting rid of the embty bodies
df = df[df.bodyText != ""]

# Total Number of Articles
df.shape

# Filtering articles based on topics
df = df[(df.sectionName == 'US news') | (df.sectionName ==
'Business') | (df.sectionName == 'Politics') | (df.sectionName ==
'World news')]

# Getting to see the dataframe info (data type, non-null value
etc.)
df = df.reset_index(drop=True)
df.info()

# To see how many articles in each category
from collections import Counter
Counter(df.sectionName)

# Overview for the dataframe
df.dropna(subset=['bodyText','headline'],inplace=True)
print(df.shape)
```

```
# Publication data range
df.webPublicationDate.min() ,df.webPublicationDate.max()

# Saving the cleaned data into csv file
df.to_csv(r"D:\projects\sameeksha\Text Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\Cle
aning Data\guardian cleaned.csv")

tdf = TfidfVectorizer(stop_words='english',ngram_range=(1,2) )
vectorizer = tdf.fit(df.bodyText)
transformed_text = vectorizer.transform(df.bodyText)
transformed_title = vectorizer.transform(df.headline)

def get_tfidf_term_scores(feature_names):
    '''Returns dictionary with term names and total tfidf scores
for all terms in corpus'''
    term_corpus_dict = {}
    # iterate through term index and term
    for term_ind, term in enumerate(feature_names):
        term_name = feature_names[term_ind]
        term_corpus_dict[term_name] =
np.sum(transformed_title.T[term_ind].toarray())

    return term_corpus_dict

# list of features created by tfidf
feature_names = tdf.get_feature_names()
term_corpus_dict = get_tfidf_term_scores(feature_names)

def get_sorted_tfidf_scores(term_corpus_dict):
     #Returns sort words from highest score to lowest score
    # sort indices from words wit highest score to lowest score
```

```
    sortedIndices = np.argsort(
list(term_corpus_dict.values()))[::-1]
    # move words and score out of dicts and into arrays
    termNames = np.array(list(term_corpus_dict.keys()))
    scores = np.array(list(term_corpus_dict.values()))
    # sort words and scores
    termNames = termNames[sortedIndices]
    scores = scores[sortedIndices]
    return termNames, scores

termNames, scores = get_sorted_tfidf_scores(term_corpus_dict)

def plot_tfidf_scores(scores,termNames, n_words = 18):
    '''Returns one plot for Importance of Top Terms'''
    # Create a figure instance, and the two subplots
    fig = plt.figure(figsize = (14, 18))
    override = {'fontsize': 'large'}
    fig.add_subplot(221)   #top left
    n_words = 75
    sb.set()
    sb.barplot(x = scores[:n_words], y = termNames[:n_words]);
    plt.title("TFIDF score ".format(n_words));
    plt.xlabel("TFIDF Score");

plot_tfidf_scores(scores, termNames,  n_words = 18)
```

Combining.py

```
# Reading The Guardian dataset
df_guard = pd.read_csv(r"D:\projects\sameeksha\Text
Analysis\final
```

```
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\Cle
aning Data\guardian cleaned.csv")


# Adding column of fakeness to the dataset and showing the column
names
df_guard["fakeness"] = 0


df_kaggle = pd.read_csv(r"D:\projects\sameeksha\Text
Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\Cle
aning Data\kaggle cleaned.csv")
# showing the column names
df_kaggle['fakeness'] = 1


df_guard.drop(['Unnamed: 0', 'apiUrl', 'fields',
        'isHosted', 'sectionId', 'sectionName', 'type',
         'webTitle',
'webUrl','pillarId','pillarName'],inplace=True,axis=1)


# Dropping unnecesary columns
df_kaggle.drop(['language', 'site_url'
,'type','author'],inplace=True,axis=1)


# Changing the name of the column for concating later
df_guard = df_guard.rename(columns={'bodyText' :
'body','webPublicationDate':'published'})
df_kaggle =
df_kaggle.rename(columns={'text':'body','title':'headline','uuid'
:'id'})
print("The number of non genuine articles in kaggle dataset are
"+str(df_kaggle.shape[0]))
print("The number of articles in Gaurdian dataset are
"+str(df_guard.shape[0]))


# Concat the datasents
```

```
df = df_kaggle.append(df_guard, ignore_index=True)

#Dropping the Nan values and info
df.dropna(inplace=True)
print(df.shape)

df.to_csv(r"D:\projects\sameeksha\Text Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\Cle
aning Data\final.csv")

#modeling
tdf =
TfidfVectorizer(stop_words='english',ngram_range=(1,2),max_df=0.8
5, min_df=0.01 )
X_body=tdf.fit_transform(df.body)
X_headline=tdf.fit_transform(df.headline)
Y=df.fakeness

from sklearn.model_selection import train_test_split
X_headline_train, X_headline_test, y_headline_train,
y_headline_test = train_test_split(X_headline,Y, test_size =
0.25, random_state=42)
X_body_train, X_body_test, y_body_train, y_body_test =
train_test_split(X_body,Y, test_size = 0.25, random_state=42)

#Random Forest
#Using headlines from the dataset
from sklearn.ensemble import RandomForestClassifier
rcf_headline = RandomForestClassifier(n_estimators=10,n_jobs=3)
rcf_headline.fit(X_headline_train, y_headline_train)
y_rc_headline_pred = rcf_headline.predict(X_headline_test)

# print metrics
print ("Random Forest F1 and Accuracy Scores : \n")
```

```
print ( "F1 score {:.4}%".format( f1_score(y_headline_test,
y_rc_headline_pred, average='macro')*100 ) )
print ( "Accuracy score
{:.4}%".format(accuracy_score(y_headline_test,
y_rc_headline_pred)*100) )

import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
y_true = y_headline_test
y_pred = rcf_headline.predict(X_headline_test)
print(confusion_matrix(y_true, y_pred))
sns.heatmap(confusion_matrix(y_true, y_pred), annot=True,
fmt='d')
plt.title('Confusion Matrix of TF-IDF Random Forest Model for
Headlines')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

import scikitplot.plotters as skplt
def plot_cmat(y_test, y_pred):
    skplt.plot_confusion_matrix(y_test,y_pred)
    plt.show()
print(classification_report(y_headline_test, y_rc_headline_pred))

#Random Forest
#Using body from the dataset
rcf_body = RandomForestClassifier(n_estimators=10,n_jobs=3)
rcf_body.fit(X_body_train, y_body_train)
y_rc_body_pred = rcf_body.predict(X_body_test)

# print metrics
```

```
print ("Random Forest F1 and Accuracy Scores : \n")
print ( "F1 score {:.4}%".format( f1_score(y_body_test,
y_rc_body_pred, average='macro')*100 ) )
print ( "Accuracy score
{:.4}%".format(accuracy_score(y_body_test, y_rc_body_pred)*100) )

import scikitplot.plotters as skplt
def plot_cmat(y_test, y_pred):
    skplt.plot_confusion_matrix(y_test,y_pred)
    plt.show()

plot_cmat(y_body_test, y_rc_body_pred)

sns.heatmap(confusion_matrix(y_body_test, y_rc_body_pred),
annot=True, fmt='d')
plt.title('Confusion Matrix of TF-IDF Random Forest Model for
Body')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
print(classification_report(y_body_test, y_rc_body_pred))
```

Algorithms.py

```
df=pd.read_csv(r"D:\projects\sameeksha\Text Analysis\final
project\FAKE-NEWS-DETECTION-master\FAKE-NEWS-DETECTION-master\Cle
aning Data\final.csv")

cnt_pro = df['fakeness'].value_counts()
plt.figure(figsize=(4,5))
sns.barplot(cnt_pro.index, cnt_pro.values, alpha=0.8)
plt.ylabel('Number of Occurrences', fontsize=10)
```

```
plt.xlabel('fake', fontsize=10)
plt.title('No. of datapoints for fake vs real')
plt.xticks(rotation=90)
plt.show();

from bs4 import BeautifulSoup
def cleanText(text):
    text = BeautifulSoup(text, "html").text
    text = re.sub(r'\|\|\|', r' ', text)
    text = re.sub(r'http\S+', r'<URL>', text)
    text = text.lower()
    text = text.replace('x', '')
    return text
df['body'] = df['body'].apply(cleanText)

train, test = train_test_split(df, test_size=0.3,
random_state=42)
import nltk
from nltk.corpus import stopwords
def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            if len(word) < 2:
                continue
            tokens.append(word.lower())
    return tokens
train_tagged = train.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['body']),
tags=[r.fakeness]), axis=1)
test_tagged = test.apply(
    lambda r: TaggedDocument(words=tokenize_text(r['body']),
tags=[r.fakeness]), axis=1)
```

```
import multiprocessing
cores = multiprocessing.cpu_count()

model_dbow = Doc2Vec(dm=0, vector_size=300, negative=5, hs=0,
min_count=2, sample = 0, workers=cores)
model_dbow.build_vocab([x for x in tqdm(train_tagged.values)])

#time
for epoch in range(45):
    model_dbow.train(utils.shuffle([x for x in
tqdm(train_tagged.values)]),
total_examples=len(train_tagged.values), epochs=1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha

def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors

def vec_for_learning(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors

y_train_dbow, X_train_dbow = vec_for_learning(model_dbow,
train_tagged)
y_test_dbow, X_test_dbow = vec_for_learning(model_dbow,
test_tagged)
logreg = LogisticRegression(n_jobs=1, C=1e5)
```

```
logreg.fit(X_train_dbow, y_train_dbow)
y_pred_dbow = logreg.predict(X_test_dbow)
from sklearn.metrics import accuracy_score, f1_score
print('Testing accuracy %s' % accuracy_score(y_test_dbow,
y_pred_dbow))
print('Testing F1 score: {}'.format(f1_score(y_test_dbow,
y_pred_dbow, average='weighted')))


model_dmm = Doc2Vec(dm=1, dm_mean=1, vector_size=300, window=10,
negative=5, min_count=1, workers=5, alpha=0.065, min_alpha=0.065)
model_dmm.build_vocab([x for x in tqdm(train_tagged.values)])


for epoch in range(15):
    model_dmm.train(utils.shuffle([x for x in
tqdm(train_tagged.values)]),
total_examples=len(train_tagged.values), epochs=1)
    model_dmm.alpha -= 0.002
    model_dmm.min_alpha = model_dmm.alpha


y_train_dm, X_train_dm = vec_for_learning(model_dmm,
train_tagged)
y_test_dm, X_test_dm = vec_for_learning(model_dmm, test_tagged)
logreg.fit(X_train_dm, y_train_dm)
y_pred_dm = logreg.predict(X_test_dm)
print('Testing accuracy %s' % accuracy_score(y_test_dm,
y_pred_dm))
print('Testing F1 score: {}'.format(f1_score(y_test_dm,
y_pred_dm, average='weighted')))


model_dbow.reset_training(keep_raw_vectors=True)
model_dmm.reset_training(keep_raw_vectors=True)


from gensim.test.test_doc2vec import ConcatenatedDoc2Vec
```

```
new_model = ConcatenatedDoc2Vec([model_dbow, model_dmm])

def get_vectors(model, tagged_docs):
    sents = tagged_docs.values
    targets, regressors = zip(*[(doc.tags[0],
model.infer_vector(doc.words)) for doc in sents])
    return targets, regressors

y_train, X_train = get_vectors(new_model, train_tagged)
y_test, X_test = get_vectors(new_model, test_tagged)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print('Testing accuracy %s' % accuracy_score(y_test, y_pred))
print('Testing F1 score: {}'.format(f1_score(y_test, y_pred,
average='weighted')))

import scikitplot.plotters as skplt
from sklearn.metrics import confusion_matrix,
classification_report

def plot_cmat(y_test, y_pred, title):
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True,
fmt='d')
    plt.title('Confusion Matrix of Doc2Vec - Logistic Regression
Model ({0})'.format(title))
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()

    print(classification_report(y_test, y_pred))

plot_cmat(y_test_dm, y_pred_dm, 'using distributed memory')
```

```
plot_cmat(y_test_dbow, y_pred_dbow, 'using distributed bag of
words')
plot_cmat(y_test, y_pred, 'using combination of both models')

C = 1.0
from sklearn import svm
svc = svm.SVC(kernel='linear', C=C).fit(X_train_dm, y_train_dm)
y_pred= svc.predict(X_test_dm)
print('Testing accuracy %s' % accuracy_score(y_test_dm, y_pred))
print('Testing F1 score: {}'.format(f1_score(y_test_dm, y_pred,
average='weighted')))

plot_cmat(y_test_dm, y_pred, 'using distributed memory')
```