A Project Report

On

**Title**

BY

**Name**

**Roll No**

Under the supervision of

**Faculty Name**

**(Month Year**)

**ACKNOWLEDGMENTS**

**ABSTRACT**

**CONTENTS**

Title page……………………………………………………..………….….……..1

Acknowledgements………………………………………………….…….….…...2

Abstract……………………………….……………………………………..…….

1. Introduction…..………………………….………………...……….….…....
2. Data Collection..……………………………….……..…………….…...….
3. Models………………………………………………………………….…
   1. . Logistic Regression…………………………………………………...
   2. . Support Vector Machines…………………………………….……….
   3. . Random Forests……………………………………………….………
   4. . Feed-Forward Neural Network……………………………….……….
   5. . LSTM Network……………………………………………….……….
4. Conclusion……………………………………………....………….………

References…………………………………………….…………………….……..

1. **INTRODUCTION**

The challenge of identifying real news from fake news can be tackled by different approaches. Getting the information about the article, like the source of the news, date of publishing, whether it was reported online or in print etc, is one of the approaches where these features will help classify the news as real or fake. But getting source information on an article is not always possible.

So, from a Natural Language Processing perspective, this challenge posses a possibility of an underlying pattern that can be identified by training a binary classifier. This implies, we just need to focus on the body of the article and extract relevant features that might form a pattern which can be recognised by our classifier.

1. **Data Collection**

Fake News for the Classifier was acquired from an open Kaggle dataset, a collection of 12000 articles that span various genres [1].

The Real News was procured from ‘The Gaurdian’ using one of their APIs collect Real World news from their website. The Real News must belong to similar subjects as the Fake News in order to get consistent results from the data. Hence, with this aim in mind I decided to create a WordCloud of the Fake News articles from the Kaggle dataset.



Figure 1 Fake News WordCloud

From the above WordCloud, it was clear that the Real News is to be collected from what subjects. So, the news articles published from January, 2016 to September, 2018 belonging to the genres of Politics, Business, World-News, UK-News, Government and US-News were scrapped using the GaurdianAPI. In total I was ables to collect 68000 articles of real news.

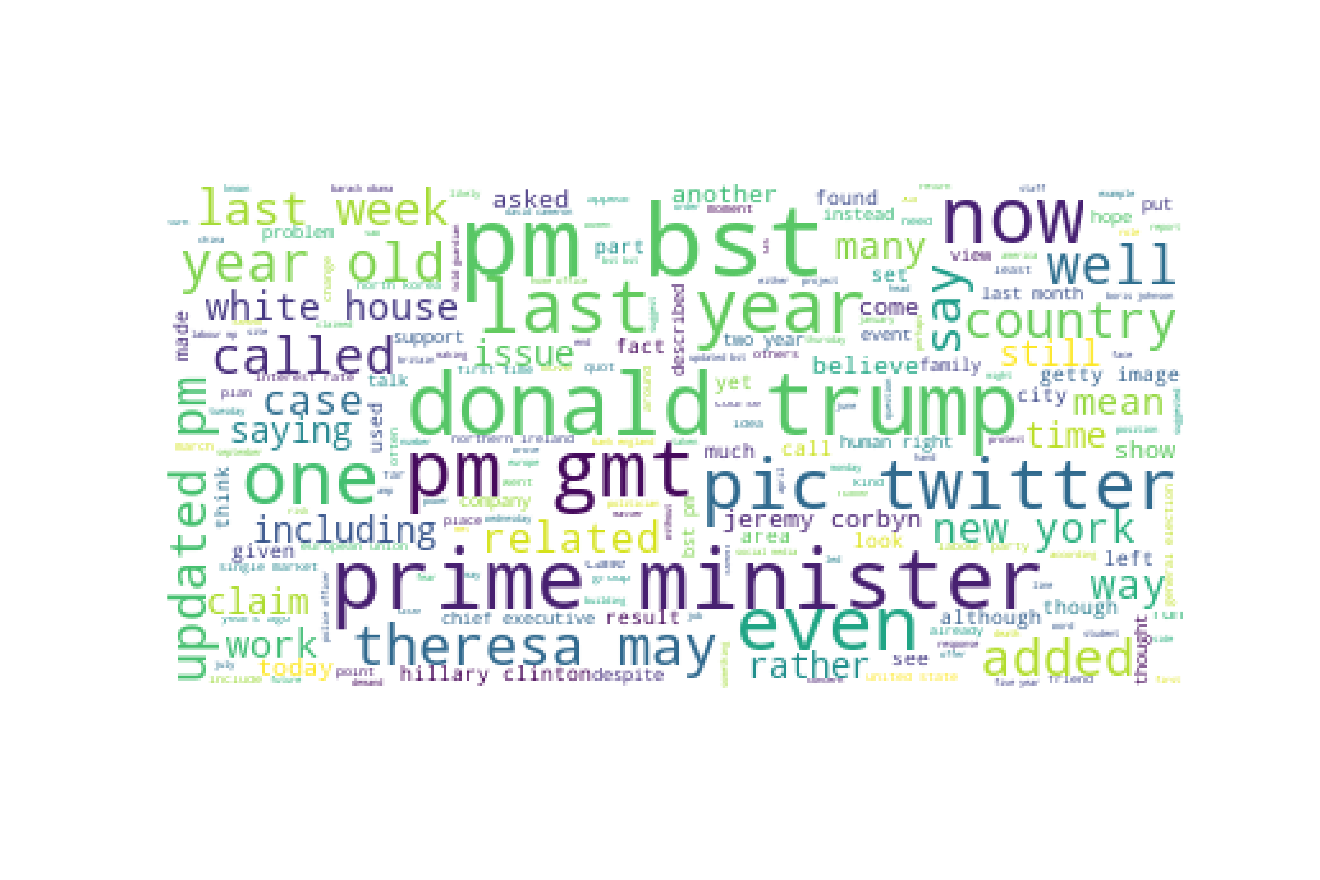


Figure 2 Real News WordCloud

The real news data is now clubbed with the fake news data. The ratio of the fake to non-fake news was estimated on the basis of empirical evidence with an attempt to replicated the state of published news in the real world. I used 12000 fake news articles with 45000 real news articles for a total of around 57000 news articles as my complete dataset.

The dataset of news articles is first filtered by removing any article’s body with less than 100 characters. Then using Regular Expressions, non textual characters like punctuation marks, numbers or even html tags are removed from the articles. All the characters are lower cased, and then **Stemming** is done to avoid creating extra features for word with same root word while tokenising the dataset.

After filtering, the fake news and the real news articles are clubbed together into a dataset. This dataset is shuffled and divided into ‘train-dataset’ and ‘test\_dataset’ in the ratio 80%:20%.

1. **DATA REPRESENTATION**
   1. **TF-IDF Vectorization**

The training dataset is now tokenized and vectorised using term frequency-inverse document frequency (tf-idf) numerical statistics. This gives importance to a word in a document based on how many times it occurs in that document (term frequency) and penalises the importance based on how many documents that particular word show up in (inverse document frequency).

The vectorization is done using **TfidfVectorizer** function of sklearn library. The **Stopword** from the English dictionary are excluded from the vectorized set of words from the documents because they do not contribute enough to the content of the article. Next, words that occur in more than 90% of the documents (max\_df=0.9) and at most 1 documents (min\_df=2) are excluded from the vectorized output as they are too common or too rare to account for any pattern in them. The training dataset is now created by **fitting** and **tranforming** the train news articles through the TfidfVectorization object.

This creates a vast matrix that with the shape of [number of documents, number of tokens]. The number of documents in the training dataset are in the order of 45000 and the tokens are of the order 73000. This matrix is given output by the TfidfVectorizer function as a **sparse matrix** as most of the elements of any tf-idf vector are zeros. The sparse matrix are convenient to handle and can be used for computation without occupying a lot of computer memory.

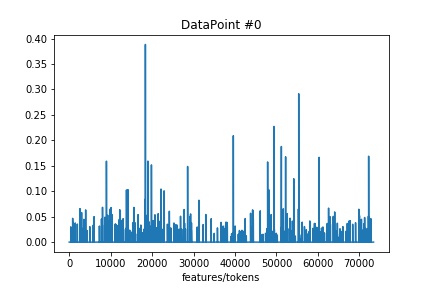


Figure 3 tf-idf Vectorization of Article #0

The ‘n’ number of features of the tf-idf vector are very vast to work with and optimization of such enormous featureset can occupy lots of computer memory very quickly, as they generally require solving inverse of the n dimensional matrix, leading to the crashing of the system. Hence, we need to find a way to decrease the number of featureset keeping only the tokens/features that highly contribute to the variance of the classification of these datapoints.

We will use the technique of **Latent Semantic Analysis (LSA)** to reduce the dimensionality of our featureset by calculating the **singular value decomposition (SVD)** of our tf-idf vector. The LSA method is based on the **distributional hyposthesis** that words that are closer in meaning will occur in similar pieces of texts, or in our case news articles. The computation of SVD will help us reduce the number of tokens in our vector while still preserving the similarity structure among columns.

Contrary to **Principle Component Analysis (PCA),** this estimator does not center the data before computing the SVD. This means it can work with scipy.sparse matrices efficiently, as centering the data will give non zero values to zero elements in our tf-idf vector.

I used TrucatedSVD function from the sklearn library. The TruncatedSVD is used to calculate the SVD rather than simple SVD because the truncated version just computes the SVD corresponding to the largest ‘n’ singular values, and the rest of the matrix is discarded. This can be much quicker and economical than the traditional SVD when n<<number of total features, which is generally in the case of tfidf vectors. The input arguments are the algorithm which can be ‘arpack’ or ‘randomized’, n\_iter corresponds to the number of iterations for the randomized svd solver and n\_components is the number of components to be selected from the diagonal singular value matrix after computation of SVD. The diagonal positive definite matrix has values along the diagonal in descending order and the top n\_components of this matrix are used to reconstruct our reduced featureset. Hence, n\_components will be the number of features which our whole featureset is reduced to. The reduced training dataset is now created by **fitting** and **tranforming** the training dataset’s tfidf vector through this TruncatedSVD object.

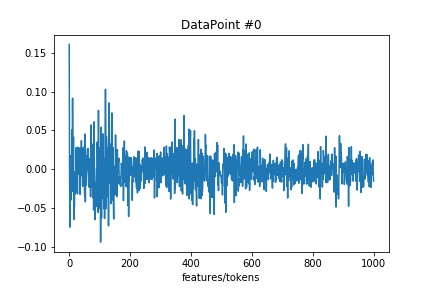


Figure 4 LSA reduced Vectorization of Article #0

We used n\_components = 1000 which means the tokens have been reduced from 73000 to 1000 while preserving the structure of the important class defining tokens. We can also represent these selected tokens as the percentage of variance they contribute to the total variance of the dataset.

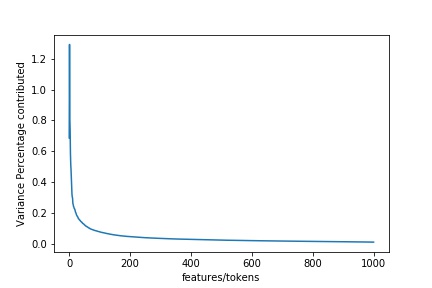


Figure 5 Feature Variance Contribution in %

This plot of variance ratio shows the declining contribution of the tokens to the total variance of our dataset. The first 50 tokens of our reduced featureset contribute maximum to the class defining properties and therefore will be the most important in classifying our data.

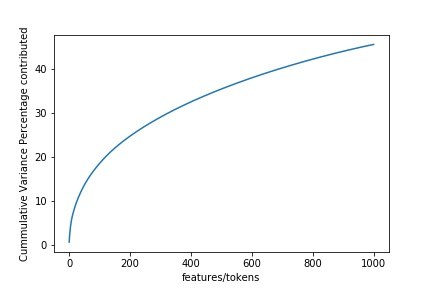


Figure 6 Cummulative Feature Variance Contribution in %

We see from the above plot that our total reduced featureset of 1000 tokens contributes to around 45% to our total variance of the initial tf-idf vectorization.

We can also use the first two tokens (two most important in class defining) to visually see how they form any pattern in identifying a fake news from a non-fake news article. Because our tf-idf vector is not centered, the first feature after LSA reduction just contains information on the frequency of the words in the documents. Therefore, to really see any difference in features of the two types of articles, we will use second and third most variant feature’s plot to see any distinct pattern.

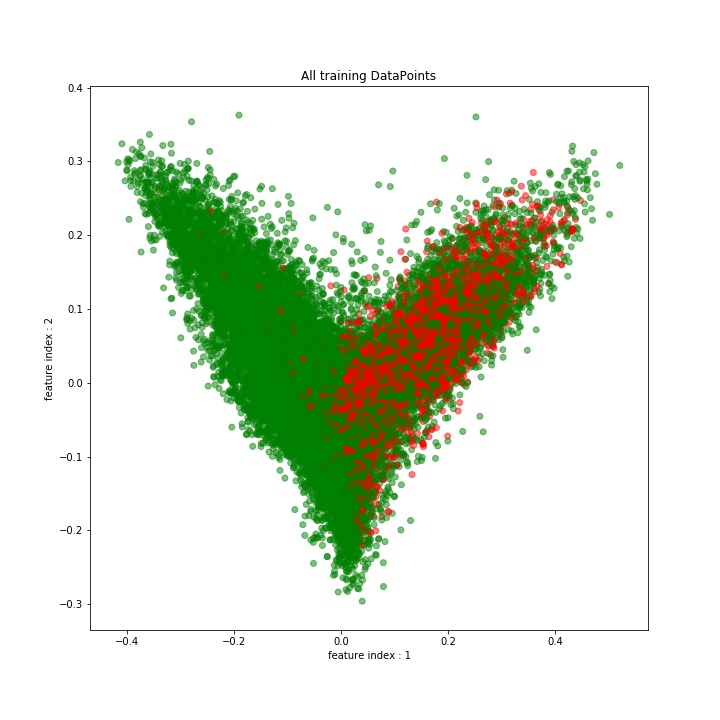


Figure 7 Visual Representation of train Dataset using top 2 features

Here, the green dots represent a non-fake news datapoint, whereas a red dot represent a fake news datapoint. We see there indeed exist a pattern among the the news articles, corresponding to the most two important tokens, in defining which class a datapoint belong to. This mean, the data can be used create a reasonable classifier model using machine learning algorithm.

The test dataset comprising of around 11000 news articles is now created by **tranforming** the test news articles through the TfidfVectorization object and then the TruncatedSVD object creating the same featureset for our test news articles.

* 1. **Word Embeddings**

Word **embedding** is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers.

We took the original news articles, and stripped the articles of any type of html tags or image urls. We removed any kind of numerical or punctuation characters and lower cased the whole dataset. Also the articles having less than 100 words are removed from our dataset. Then, the dataset was trimmed to just the first 1000 words, the assumption that the model will be able to predict if the that article is fake or not using the first 1000 words.

Generation of vectors of real numbers corresponding to each English word requires training on a relevant dataset. In our case we did not train an embedding model to create a word representation, instead we used a standard word representation dictionary known as GloVe (**Global Vectors for Word Representation**).

* **GloVe** is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.
* The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics. For large corpora, this pass can be computationally expensive, but it is a one-time up-front cost.

These vector representation of words possesses information like:

1. **Nearest neighbors**

* The **Euclidean distance** (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. For example, here are the closest words to the target word *frog*:
* *frog :* frogs, toad, litoria, leptodactylidae, rana etc.

1. **Linear substructures**

* The **similarity metrics** used for nearest neighbor evaluations produce a single scalar that quantifies the relatedness of two words. This simplicity can be problematic since two given words almost always exhibit more intricate relationships than can be captured by a single number.

Here, for this news article dataset we used the ‘glove.6B.100d.txt’ file which is essentially an embedding matrix trained on 6 billion tokens from sources Wikipedia (2014) and English Gigawords. It is a dictionary of 400,000 vocabulary size with a 100 dimensional vector representation of each word.

The order of the phrases in the article is maintained and hence this type of representation of the news articles can be of use for Sequential Training of data. So, we took first 1000 words from each article for tokenization. Hence, first 1000 words from each article are combined to from a vocabulary dictionary of length 187,000. For each article, the words are mapped to their integer counterpart based on their position in this dictionary. For articles, with word length shorter than 1000 words, the sequence of integers is padded with 0s to make the size 1000. So, the **training set** has a dimension of 45000x1000.

The vocabulary dictionary of unique words formed from the first 1000 words of each article are mapped to their corresponding vectors from the GloVe embeddings. The words in this dictionary but not in the GloVe embeddings are mapped to a vector of 0s. Hence, an **embedding matrix** of size 187,000x100 is formed.

1. **Models**

Several different Classifying Models were implemented to accurately predict if a News article is fake or non-fake. The training dataset and test dataset from above data preprocessing is used to train and check the reliability of our model. The Accuracy and F1 Scores on the test set are reported for each model. Here, the F1 Score is a more reliable testing parameter as the ratio of the fake to non-fake news is not 0.5, rather it was estimated on the basis of empirical evidence to replicate the state of published news in the real world.

1. **Logistic Regression**

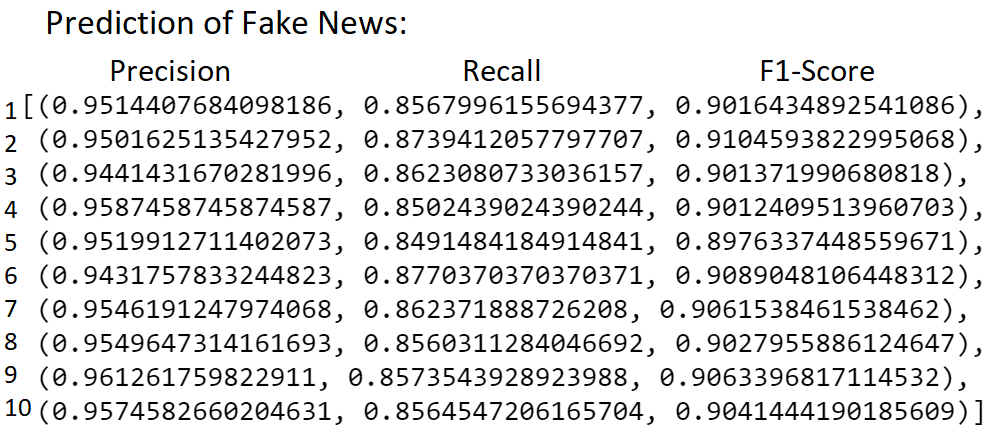
A Logistic Regression algorithm is used to create a binary classifier that is optimised on our training dataset. The LogisticRegression function from sklearn library is used to crete and train our classifier. The parameters of the function used :

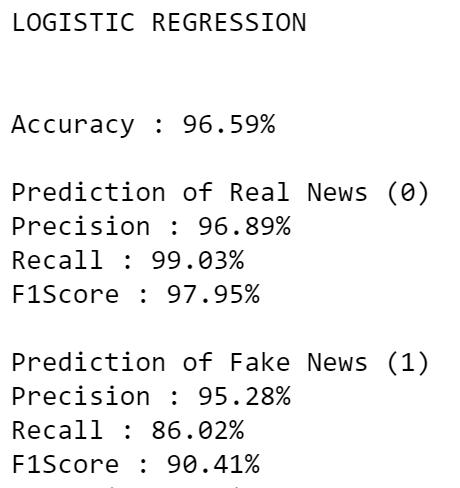
* **penalty** : The kind of norm used for regularization, I used ‘L2’ normalization
* **C** : The inverse of the regularization parameter value, left at default of 1.0
* **solver** : algorithm to be used in the optimization problem, I used ‘lbfgs’. The BFGS method is one of the most popular member of the Quasi-Newton methods of hill-climbing optimization techniques that seek a stationary point of a function. The L-BFGS is a limited memory version of BFGS that is particularly suited to problems with large number of variables (1000 in our case).
* **multi\_class** : the multi\_class can be ‘ovr’ (one-vs-rest) or ‘multinomial’, I used the ‘multinomial’ as the one-vs-rest will create a classifier for every class and use the highest probability of them to predict the class. It is unnecessary here as there are just two classes.

The model was trained using the tfidf vector after dimensionality reduction. Then the model was tested for accuracy on the test dataset and a Confusion Matrix was plotted along with test Accuracy, Precision, Recall and F1Score was repored.

**Results** :

The training and then prediction for the test set was performed 10 times using the same total dataset. For every iteration the dataset was split randomly into Training and Test set then the tf-idf and dimensionality reduction was carried out on the training and the test set. The model performance in terms of the Accuracy, Precision, Recall and F1 Score was averaged across all ten iterations and the final results are as follows.

****



1. **Support Vector Machine**

The SVM training algorithm builds a model that assigns an datapoint to one class or the other. The SVM model is the representation of sample points in n dimensional space, mapped so that examples of different class are divided by a clear boundary or gap that is as wide as possible. The test or unseen examples are then mapped to the same space belonging to one category or the other based on which side of the boundary they fall. This boundary solve both linear or non linear classification problems based on the kernel methods used for training. A Support Vector Classifier is created using the SVC function of sklearn library. The parameters of the function used :

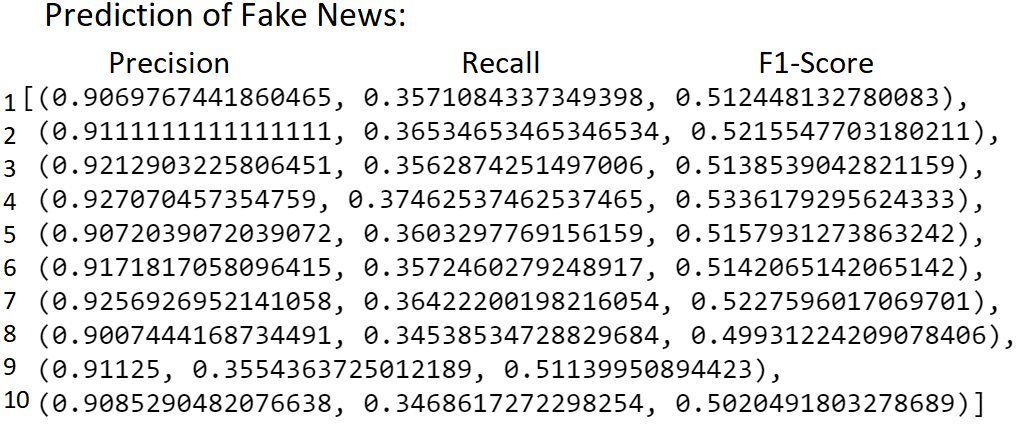
* **kernel** : Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. I used the ‘rbf’ (radial basis function) kernel.

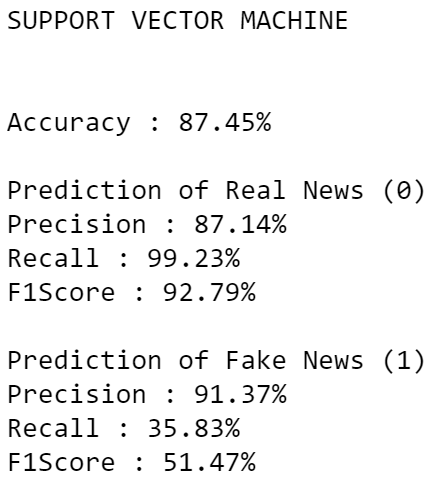
The model was trained using the tfidf vector after dimensionality reduction. Here the number of features I used are 50 rather than the featureset of 1000 features. This is because the results are better on the top 50 features compared to using the whole 1000 featureset. This observation is partly because the formation of a clear boundary is more defined for important features that contribute more to the class defining. Hence the introduction of relatively non important features for training the classifier results in blurring of the boundary reducing the F1 Score and the Accuracy of our classifier.

Then the model was tested for accuracy on the test dataset and a Confusion Matrix was plotted along with test Accuracy, Precision, Recall and F1Score was repored.

**Results** :

The training and then prediction for the test set was performed 10 times using the same total dataset. For every iteration the dataset was split randomly into Training and Test set then the tf-idf and dimensionality reduction was carried out on the training and the test set. The model performance in terms of the Accuracy, Precision, Recall and F1 Score was averaged across all ten iterations and the final results are as follows.





1. **Random Forest**

The random forest algorithm creats a forest with a number of Decision Trees. It is a type of Ensemble machine learning algorithm, which use a divide-and-conquer approach. The main principle behind ensemble algorithms is **boosting**, that is a group of weak learners (single estimator or a decision tree) can work together to form a strong learner (group of estimators or a forest) to classify the data. The random decision forests can correct for the decision trees’ habit of overfitting to the training dataset. Hence, random forest algorithm comprises of **bagging** (Bootstrap aggregating), which is the approach to reduce overfitting by combining the classifications of randomly generated training sets, together with the random selection of features to construct a collection of decision forests.

The Random Forest Classifier is created using the RandomForestClassifier function of sklearn library. The parameters of the function used :

* **n\_estimators** : The number of decision trees in the forest, I selected 100.
* **Criterion** : I have used ‘gini’ importance criterion or the Mean Decrease in Impurity (MDI), which calculates each feature importance as the sum over the number of splits (across all tress) that include the feature, proportionally to the number of samples it splits.

The model was trained using the tfidf vector after dimensionality reduction. The model can also show the feature importance of all the tokens based on the gini importance criterion.

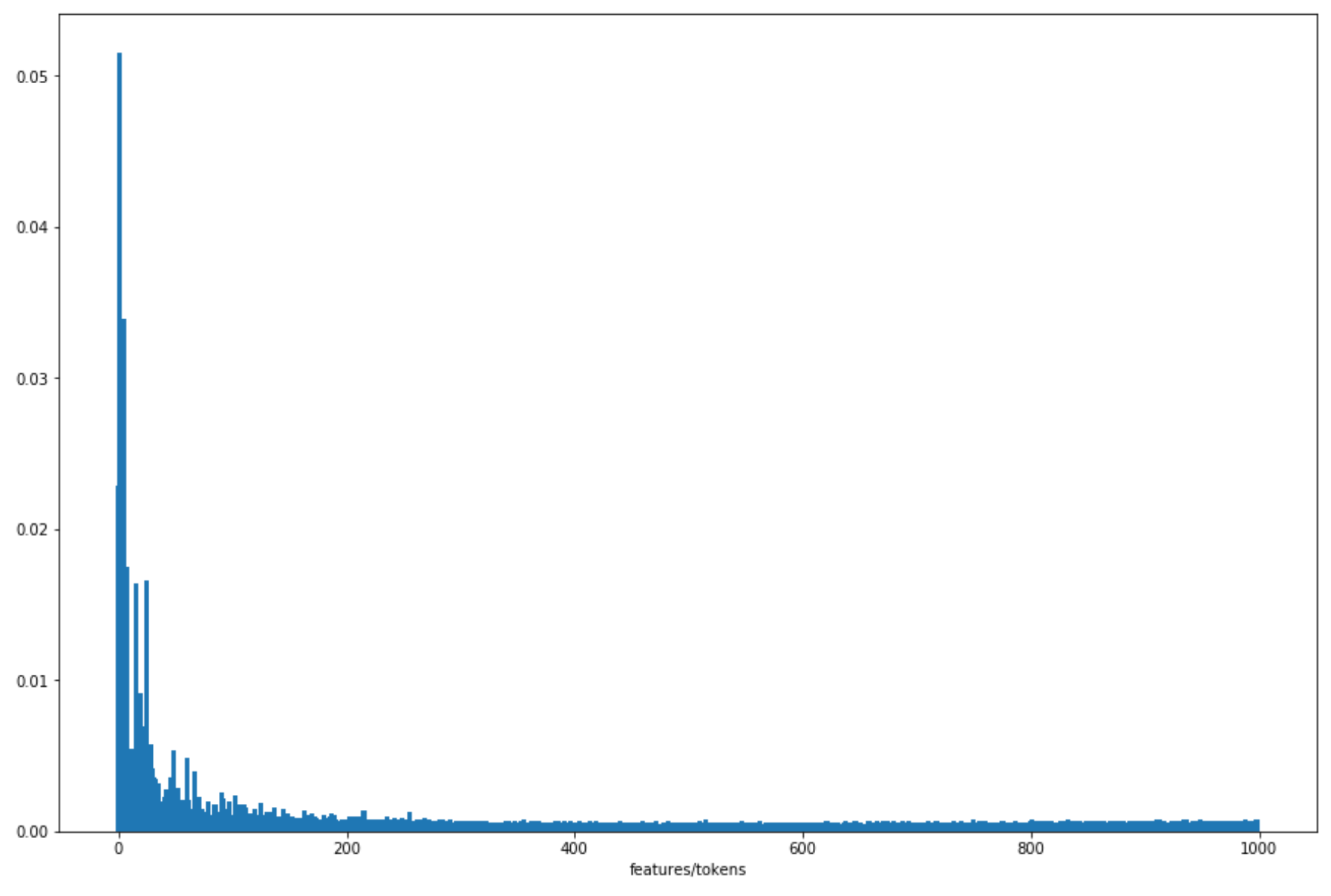


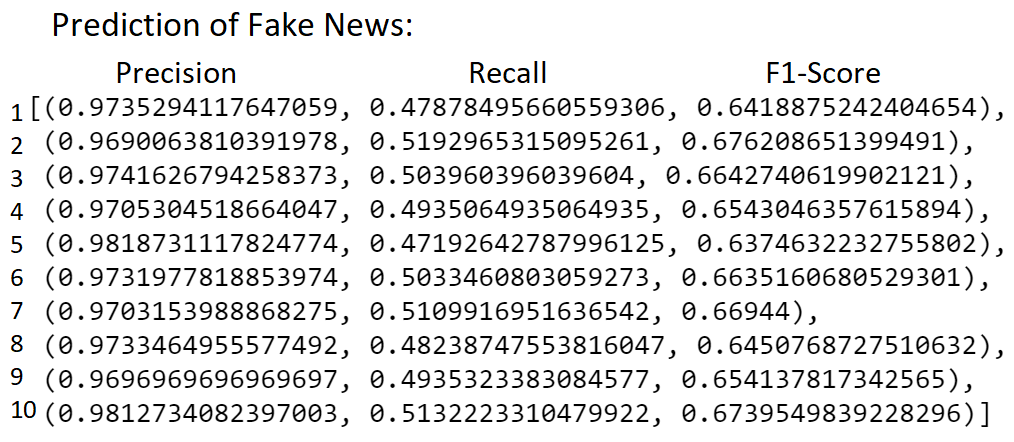
Figure 8 Feature Importance

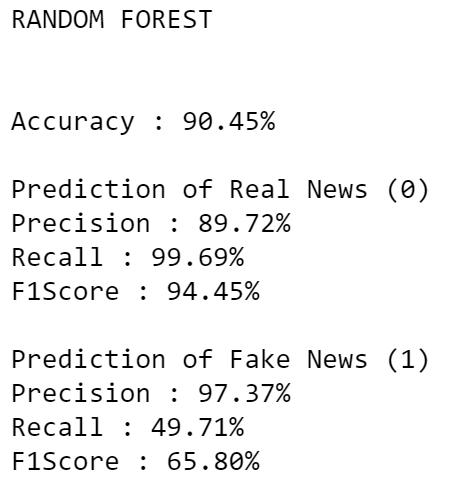
The feature importance graph is similar to the feature variance plot after the dimensionality reduction.

Then the model was tested for accuracy on the test dataset and a Confusion Matrix was plotted along with test Accuracy, Precision, Recall and F1Score was repored.

**Results** :

The training and then prediction for the test set was performed 10 times using the same total dataset. For every iteration the dataset was split randomly into Training and Test set then the tf-idf and dimensionality reduction was carried out on the training and the test set. The model performance in terms of the Accuracy, Precision, Recall and F1 Score was averaged across all ten iterations and the final results are as follows.





1. **Feed-Forward Neural Network**

A feed forward artificial neural network or a Multilayer Perceptron approach can be used to solve the non-linearly separable datapoints. It consists of atleast three layers of nodes: an input layer, a hidden layer and an output layer. Our model will be created using the Keras API of the tensorflow library. The model will be made of two hidden layers with 600 nodes in the first layer and 300 nodes in the second layer, an input layer of 1000 nodes from our trainind dataset features and an output layer of 1 node giving a binary prediction of 1 or 0. The activation used in the first two hidden layers is Rectified Linear Unit and for the output layer it is Sigmoid function giving a probability of what the binary output will be. The optimization technique used for the backpropagation will be the Adam optimizer, which is precisely a complicated and adaptive version of Stochastic Graddient Descent.

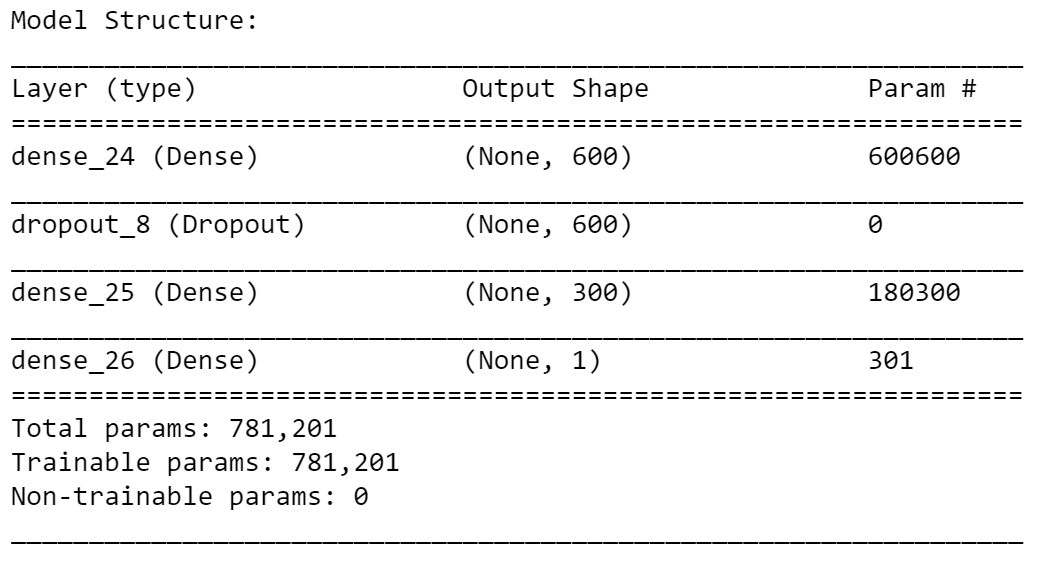


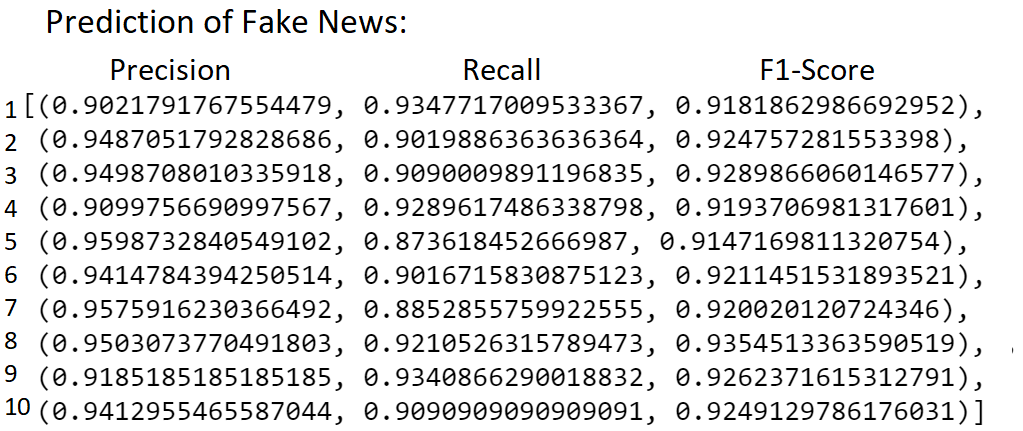
Figure 9 Multilayered Perceptron Model

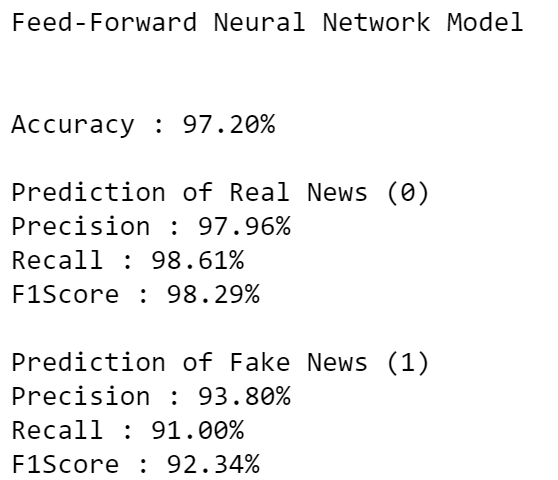
We can see the model layers and the total number of trainable parameters in our model.

This model was trained using the tfidf vector after dimensionality reduction. Then the model was tested for accuracy on the test dataset and a Confusion Matrix was plotted along with test Accuracy, Precision, Recall and F1Score was repored.

**Result** :

The training and then prediction for the test set was performed 10 times using the same total dataset. For every iteration the dataset was split randomly into Training and Test set then the tf-idf and dimensionality reduction was carried out on the training and the test set. The model performance in terms of the Accuracy, Precision, Recall and F1 Score was averaged across all ten iterations and the final results are as follows.





1. **LSTM Network**

For training of this model we will use the Embedding Representation of our news articles in the Section 3.2.

Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks and belong to the most promising algorithms out there at the moment because they are the only ones with an internal memory. Because of their internal memory, RNN’s are able to remember important things about the input they received, which enables them to be very precise in predicting what’s coming next.

This is the reason why they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more because they can form a much deeper understanding of a sequence and its context, compared to other algorithms.

Traditional neural networks can’t do this, and it seems like a major shortcoming. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

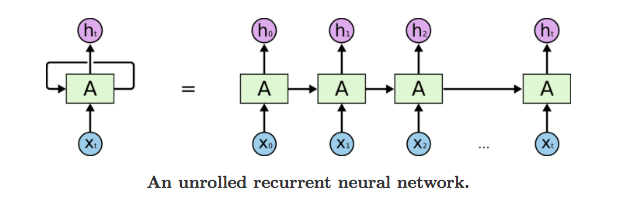


Figure 10 An unrolled Recurrent Neural Network

In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

A glaring limitation of Vanilla Neural Networks (and also Convolutional Networks) is that their API is too constrained: they accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes).

The core reason that recurrent nets are more felxible is that they allow us to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both.

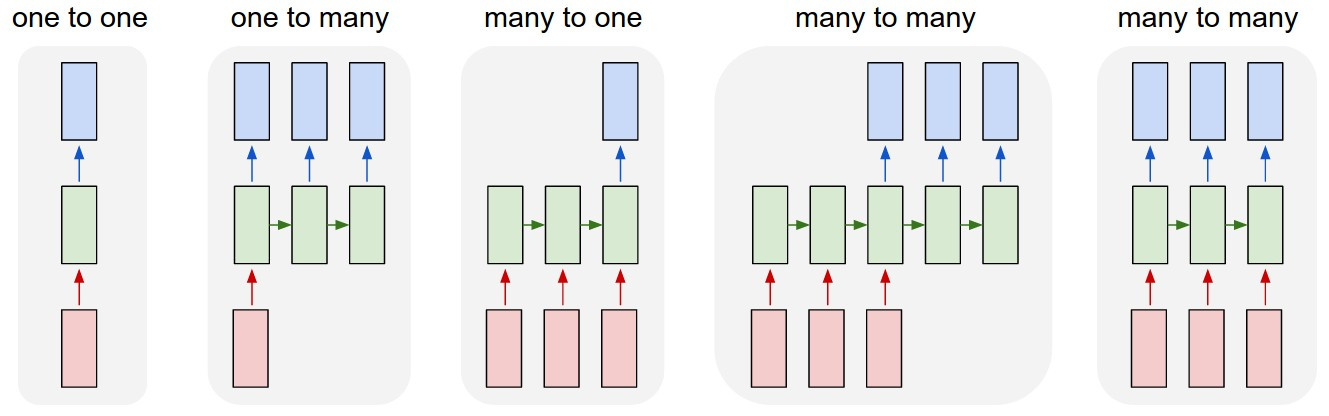


Figure 11 Types of RNN architectures

Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN’s state.

For our application, which is to predict if a news article is fake or real, the model we should use is **many-to-one.** This is because for each article there is a sequence of input data (ie: first 1000 words of the article) and the output is just a **binary classification** if that particular article is fake or not.

The **LSTM** is a particular type of recurrent network that works slightly better in practice, owing to its more powerful update equation and some appealing backpropagation dynamics. An RNN composed of LSTM units is often called an **LSTM network**.

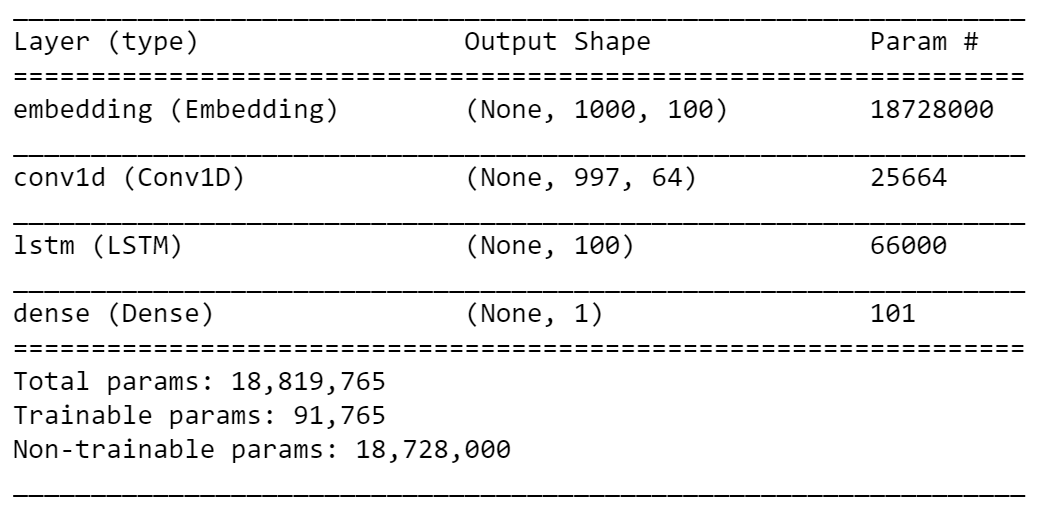


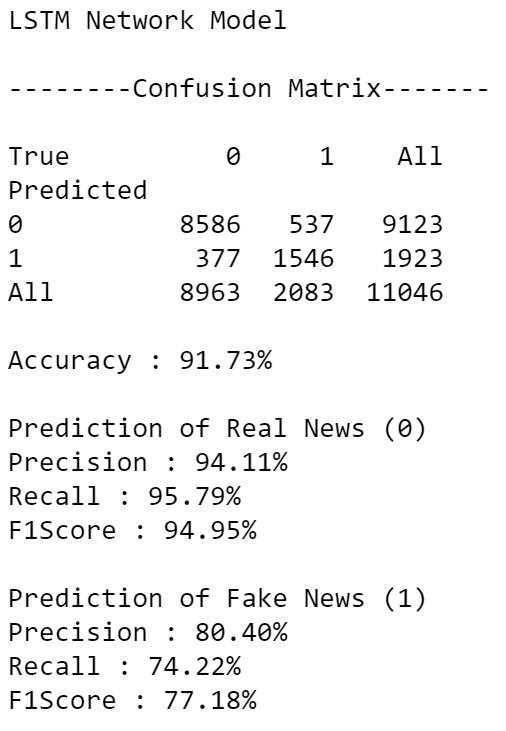
Figure 12 LSTM Network model summary

The model for LSTM network was created using ‘keras’ library. It consists of the four layers:

* **Embeddings Layer** : Maps every integer in the input sequence (1x1000) to its embedding vector using the embedding matrix. Therefore, this layer was given a precalculated embedding dictionary which was the embedding matrix created using GloVe. Since, this layer does not require its parameters to be changed we will set its trainability to false.
* **1D Convolutional layer** : The above layer creates an input matrix of dimensions 100x1000 to be fed into this layer with 64 filters and a kernel size of 4 which reduces the dimesions of the input.
* **LSTM layer** : The output matrix from above layer passes through this LSTM layer consisting of 100 units. This creates 100 outputs (one from every many-to-one units).
* **Output Layer** : The above 100 outputs are fully connected to one unit which gives the binary output for classification.

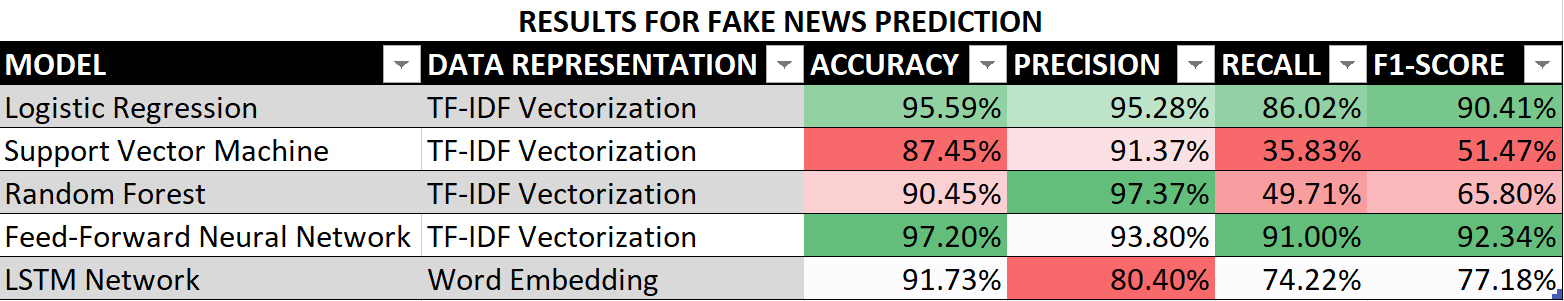
This model was trained using the news articles consisting of sequence of integer. Then the model was tested for its performance on the test dataset and a Confusion Matrix was plotted along with test Accuracy, Precision, Recall and F1Score was repored.

**Result** :



1. **Conclusion**

The aim of this project was to test different data representation methods and train various models using this data. We collected our real news articles from sources like The Gaurdian and the fake news dataset was aquired from Kaggle. Then two type of data representation techniques were used namely, TF-IDF Vectorization and Word Embeddings. The resulting transformed data was trained on the models mentioned in Section 4, and the following results were extrapolated from this experiment.



The ratio of real news to fake news in the dataset were biased to resemble the real world scenario. Hence, in situation like these the model **accuracy** is really not the best metric to base our results. We generally test the performace of a model trained a biased dataset using **F1-Score**, which is nothing but the harmonic mean of **precision** and **recall**. The above scores are given for prediction of the fake news article.

We observe that the **Feed-Forward Neural Network** performed the best in accurately predicting the fakeness of the news article using a TF-IDF data representation giving a F1-Score of 92.34%. A close second best performer was the Logistic Regression model with an F1-Score of 90.41%.

The LSTM Network model took the most time to train and was very computationally expensive, and yet it performed poorly. This might be partly because as this LSTM network analyses the sequence of words from an article. Whereas, the TF-IDF representation mostly focuses on the frequency of words occurring in the article and peroforms.

This implies that a fake news article is much better categorized by the vocabulary used and its frequency rather than the semantic analysis of the article using the sequence of words.

**References**

1. *Bajaj, “The Pope Has a New Baby!” Fake News Detection Using Deep Learning, Stanford University CS 224N - Winter 2017*
2. *Kaggle, Getting Real About Fake News, https://www.kaggle.com/mrisdal/fake-news, URL obtained on August 28, 2018.*
3. *Banerjee, Suvro. “An Introduction to Recurrent Neural Networks – Explore Artificial Intelligence – Medium.” Medium.com, Medium, 23 May 2018, medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912.*
4. *“Keras: The Python Deep Learning Library.” Keras Documentation, keras.io/.*
5. *Pennington, Jeffrey. Single-Link and Complete-Link Clustering, nlp.stanford.edu/projects/glove/.*
6. *“TensorFlow.” TensorFlow, www.tensorflow.org/.*