Fake News Detection

Rahul Wadhwa
rwadh004@ucr.edu

Varun Sapre
vsapr002@ucr.edu

Karthik Harpanahalli
kharp009@ucr.edu

Ameya Padole
apado003@ucr.edu

Rohit Kumashi
rkuma069@ucr.edu

1. INTRODUCTION

Due to the advent of technology, we consume much of our daily news online. There is an abundance of information which is available online. Distinguishing true from false is becoming difficult as the technology improves and incorporates itself more into our daily lives. To tackle this surge of fake news articles we incorporate the help of machine learning. We present our fake new detection machine learning model, which uses machine learning methods such as Logistic Regression, k-Nearest Neighbors, Support Vector Machine, Naive Bayes, and Convolution Neural Network. We aim to analyze the features from the given dataset to determine with high accuracy whether the given article holds any veracity or not.

Fake news has long plagued our society. But with the usher of the technological era, malicious actors have found a new outlet to spread false information whilst masquerading it to sway unsuspecting consumers. Extensive research has been conducted on fake news and its detrimental effects. Few of the detrimental effects are as follows:

- Distrust in Media
- Undermining of Democracy
- Instigation of violence and thereby disrupting communal harmony
- Spread of false science, which could lead to bodily harm and death.

2. RELATED WORK

We have surveyed the following papers. Each paper outlines the theory and methodology behind classifying data according to fake and real news. The first paper gives review of all the models and their effectiveness. The second paper provides experiments over Naive Bayes, SVM and Neural Network. The third paper focuses on creating ensemble methods for collecting and analyzing using Decision Tree methods. In the fourth paper, the authors focus on analyzing eight different datasets itself and then choose machine learning methods to find the most valuable dataset for training.

1. S. I. Manzoor, J. Singla and Nikita, "Fake News Detection Using Machine Learning approaches: A systematic Review," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019, pp. 230-234, doi: 10.1109/ICOEI.2019.8862770.

2. S. Aphiwongsophon and P. Chongstitvatana, "Detecting Fake News with Machine Learning Method," 2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2018, pp. 528-531, doi: 10.1109/ECTICon.2018.8620051.

3. Saqib Hakak, Mamoun Alazab, Suleman Khan, Thippa Reddy Gadekallu, Praveen Kumar Reddy Maddikunta, Wazir Zada Khan, An ensemble machine learning approach through effective feature extraction to classify fake news, Future Generation Computer Systems

4. R. R. Mandical, N. Mamatha, N. Shivakumar, R. Monica and A. N. Krishna, "Identification of Fake News Using Machine Learning," 2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2020, pp. 1-6, doi: 10.1109/CONECCT50063.2020.9198610.

3. DATASET

The dataset used contains 6335 rows and 4 columns (number, title, text, label). Below is a distribution of the count of the real vs the fake articles in the dataset.

The dataset also contains a few duplicate entries (about 80) which were removed before we begin training our models as this could affect the tf-idf scores for the documents..

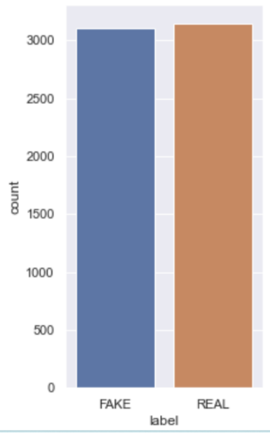


Figure 01: Comparison of of Fake and Real Article Count

On removal of the duplicate entries in the dataset,

```
count     6256.000000
mean       496.829444
std        529.336529
min          4.000000
25%        194.750000
50%        388.000000
75%        640.000000
max      12142.000000
```

we obtain approximately 6256 entries. The features of the final dataset considered are as shown.

A density distribution for the document length of the different documents in the dataset can be observed below.

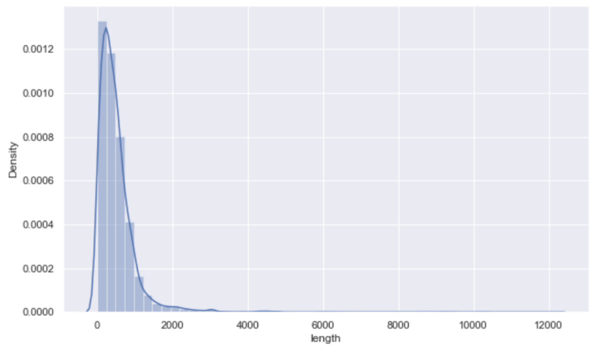


Figure 02: Density Distribution

4. DATA PREPROCESSING

For preprocessing the data, we made use of the Python library nltk. Preprocessing involved the following steps-

1. Removal of punctuation - Punctuation was eliminated using regular expressions from the different forms of punctuation in string.punctuation.
2. Stop word removal - Stop words are any word in a stop list which are filtered out before or after processing of natural language data. These include the most common words in the English language (such as the articles, prepositions, etc.). nltk's stopwords package was used to obtain a list of commonly found stopwords in the English language.
3. Stemming - Stemming is the process of reducing a word to its basic form. This makes it simpler to perform any sort of normalization on the data. Porter Stemmer of nltk was used to perform the task of stemming.
4. Label preprocessing- The labels were also converted to a 0,1 format to make it easier for comparison later on.

5. PROPOSED METHODS

We have implemented 5 classification algorithms to predict whether each news is 'REAL' or 'FAKE'. These are Naive Bayes, Logistic

Regression, k-Nearest Neighbors, Support Vector Machine, and Convolution Neural Network. In this section we discuss each of these algorithms and explain how we implement each of these algorithms used from scratch.

5.1 Naive Bayes Algorithm

The Naive Bayes (also called simple Bayes or independence Bayes) classifier is based on the concept of Bayes theorem and it assumes conditional independence between features. This algorithm assigns class labels to problem instances, which are depicted as vectors of feature values, where the class labels are drawn from some finite set. In the problem that we are solving this finite set is either ‘REAL’ or ‘FAKE’. The type of algorithm used to implement Naive Bayes here is the multinomial Naive Bayes algorithm. The reason why this was chosen was because this multinomial Naive Bayes is suitable for classification for problems with discrete features (like word counts for text classification; our problem is based on text classification).

5.1.1 Implementation

The multinomial Naive Bayes algorithm will need to know the probability values of the two equations below to be able to classify new messages:

$$P(FAKE|w_1, w_2, ..., w_n) \propto P(FAKE) \cdot \prod_{i=1}^{n} P(w_i|FAKE)$$

and

$$P(REAL|w_1, w_2, ..., w_n) \propto P(REAL) \cdot \prod_{i=1}^{n} P(w_i|REAL)$$

These 2 equations come from using the Bayes theorem and using the assumption of conditional independence amongst features(here, it's the words in the vocabulary). Note that P(FAKE) and P(REAL) are the probabilities of a news article to be deemed as ‘FAKE’ or ‘REAL’ considering the training set(prior probability).

Also, to calculate P(wi|FAKE) and P(wi|REAL) inside the formulas above, we need to use these equations:

$$P(w_i|FAKE) = \frac{N_{w_i|FAKE} + \alpha}{N_{FAKE} + \alpha \cdot N_{Vocabulary}}$$

and

$$P(w_i|REAL) = \frac{N_{w_i|REAL} + \alpha}{N_{REAL} + \alpha \cdot N_{Vocabulary}}$$

where,

$N_{w_i|FAKE}$- the number of times the word wi occurs in news classified as ‘FAKE’,

$N_{w_i|REAL}$- the number of times the word wi occurs in news classified as ‘REAL’ ,

N_{FAKE} - total number of words in news classified as ‘FAKE’ ,

N_{REAL}- total number of words in news classified as ‘REAL’ ,

$N_{Vocabulary}$- total number of words in the vocabulary.

To deal with words that are not part of the vocabulary, we have added a smoothing parameter α. To keep the probability values proportional across all words, use the additive smoothing for every word (We have used α=1, called as Laplace smoothing)

Now that we've calculated all the constants and parameters we need, we then implement fake news filter which could be understood as a function that:

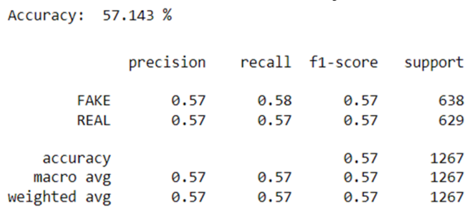
1. Takes in as input a new news (w1, w2, ..., wn)
2. Calculates P(FAKE|w1, w2, ..., wn) and P(REAL|w1, w2, ..., wn)
3. Compares the values of P(FAKE|w1, w2, ..., wn) and P(REAL|w1, w2, ..., wn), and:
    - If P(REAL|w1, w2, ..., wn) > P(FAKE|w1, w2, ..., wn), then the message is classified as ‘REAL’.

- If P(REAL|w1, w2, ..., wn) < P(FAKE|w1, w2, ..., wn), then the message is classified as ‘FAKE’.
- If P(REAL|w1, w2, ..., wn) = P(FAKE|w1, w2, ..., wn), then the algorithm may randomly assign labels based on distribution of labels in the training set

5.1.3 Results

The final implementation is detailed in the code. For fair comparison between the accuracy of the sklearn model and our own implementation of the Naive Bayes, we use the same preprocessed data for both.

The accuracy obtained using the sklearn model for the Multinomial Naive Bayes was ~83%. For fair comparison, after setting the alpha value, the final accuracy achieved with our own implementation was ~57%. More details can be found in the “Implementation Correction Report” under the section 1.1 Naive Bayes.



```
Accuracy:  57.143 %

              precision    recall  f1-score   support

        FAKE       0.57      0.58      0.57       638
        REAL       0.57      0.57      0.57       629

    accuracy                           0.57      1267
   macro avg       0.57      0.57      0.57      1267
weighted avg       0.57      0.57      0.57      1267
```

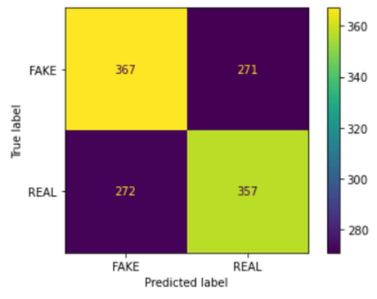


Figure 03 : **Final Metrics of the predicted values**

5.2 Logistic regression

Logistic Regression models the probability of the dependent variable given the weighted inputs. Typically Logistic Regression models binary classification i.e Yes / No, True / False etc. In other words, the dependent variable is categorical.

Note: Some of the following figures and formulas are taken by the article published in “Towards Data Science” by Animesh Agarwal titled “Building a Logistic Regression in Python” [1]

5.2.1 Sigmoid Function

Compared to linear regression, logistic regression is very similar except the final output is classified into 0 and 1 (or such binary values). For this, all the predicted values are passed through an activation function that maps the values categorically.

The activation function is known as sigmoid function. In the figure, 0.5 is the probability threshold and thus classifies any value either above and below the line. This threshold can be tuned.

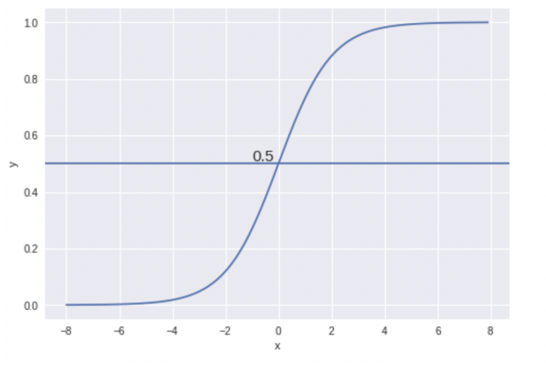


Figure 04: **Sigmoid function curve** [1]

5.2.2 Hypothesis

The Logistic Regression is represented by the equation:

$$h(x) = \sigma(\theta^T x)$$

where, the sigmoid function is represented by:

$$\sigma(t) = \frac{1}{1+e^{-t}}$$

Thus, the final hypothesis equation is:

$$h(x) = \frac{1}{1+e^{-\theta^T x}}$$

$$h(x) = \begin{cases} > 0.5, & \text{if } \theta^T x > 0 \\ < 0.5, & \text{if } \theta^T x < 0 \end{cases}$$

5.2.3 Cost Function

We define the cost function as follows:

$$cost = \begin{cases} -log(h(x), & \text{if } y = 1 \\ -log(1 - h(x)), & \text{if } y = 0 \end{cases}$$

The objective is to minimize the cost function. That is if the actual category is 0 and the model outputs the value 1, it should be heavily penalized and the same for the other case.

To achieve this we use Gradient descent which is an iterative optimization algorithm.

5.2.4 Results

The final implementation is detailed in the code. For fair comparison between the accuracy of the sklearn model and our own implementation of the Logistic Regression, we use the same preprocessed data for both.

The accuracy obtained using the sklearn models was ~92%. After tweaking the alpha and iteration values, the final accuracy achieved with our own implementation was ~82%. The final alpha value and iteration values were 0.1 and 300. More details regarding the tuning of the parameters and varying accuracy is in the "Implementation Correction Report" under the section 1.2 Logistic Regression.

```
Number of iterations: 300
Alpha Value: 0.1
Accuracy: 82.481%


              precision    recall  f1-score   support

    negative       0.80      0.86      0.83       638
    positive       0.85      0.79      0.82       629

    accuracy                           0.82      1267
   macro avg       0.83      0.82      0.82      1267
weighted avg       0.83      0.82      0.82      1267
```

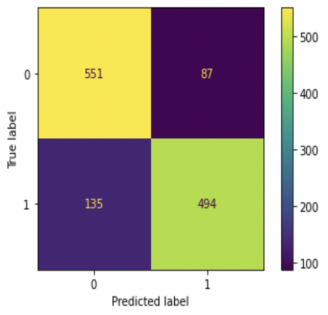


Figure 05: Final metrics of the predicted values

5.3 Support Vector Machine

Support Vector Machine, or support vector network, is a supervised learning, classification and regression analysis algorithm which is capable of efficiently performing non-linear classification. SVM uses Maximum-Margin classifier, whose objective is to draw a hyperplane that can help distinguish between data points.

5.3.1 Linear SVM Kernel

Linear Support Vector Machine Kernel uses two margins viz, (i) Soft-Margin and (ii) Hard-Margin. These two margins help the model find the Maximum-Margin hyperplane(which is the maximum distance between two classes in the data-set) which is the foundation of Maximum-Margin classification. Hard-Margin can be defined when the SVM is presented with data that is linearly separable from each other. Two hyperplanes are used to segregate the data and the maximum distance between the two drawn hyperplanes is called the Maximum-Margin hyperplane.

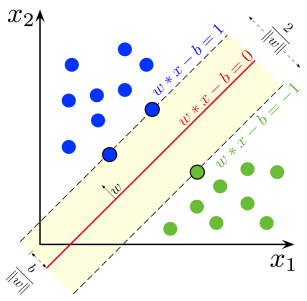


Figure 06 : Maximum-Margin Hyperplane.

Soft-Margins extend SVM when the data points are not linearly separable. To achieve this Soft-Margin uses a hinge loss function. A hinge loss function is, as the name states, a loss function, which can help SVM extend for multiclass classification. The hinge loss function is as follows:

$$\max\left(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b)\right).$$

Here y_i represents the i^{th} target and the rest of the equation is the i^{th} output.

5.3.2 Implementation

The goal is to obtain parameters of the maximum-margin hyperplane by solving the optimization problem. That is the minimization of the following equation:

$$\begin{aligned} \text{maximize}_\alpha \quad & \textstyle\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ & K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \\ & \textstyle\sum_i \alpha_i y_i = 0 \\ & C \geq \alpha_i \geq 0 \end{aligned}$$

5.3.3 Results

For the final implementation, we used the same pre-processed data from the previous implementation of Support Vector Machine(which was implemented using built-in functions from sklearn), to test if the actual implementation was at par with the previous implementation. Following are the results of our implementation:

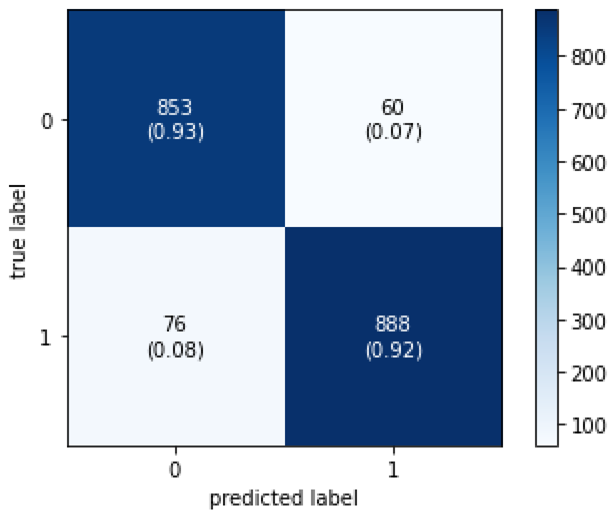
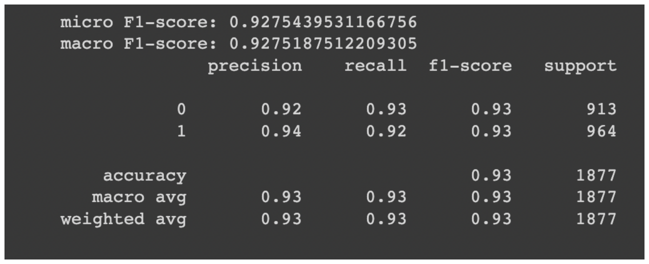


Figure 07 : Confusion matrix



```
micro F1-score: 0.9275439531166756
macro F1-score: 0.9275187512209305
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       913
           1       0.94      0.92      0.93       964

    accuracy                           0.93      1877
   macro avg       0.93      0.93      0.93      1877
weighted avg       0.93      0.93      0.93      1877
```

Figure 08 : Classification Report

5.4 Convolutional Neural Network (CNN)

Artificial neural networks, usually simply called neural networks are based on a collection of connected units or nodes called neurons, which loosely model the neurons in a biological brain. The neurons will be connected to each other using edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection.[8] A Recurrent Convolutional Neural Network is the combination of two of the most prominent neural networks - Recurrent Neural Networks and Convolutional Neural Networks. The reason for making use of this is that Recurrent convolutional neural networks capture contextual information with the recurrent structure and construct the representation of text using a convolutional neural network.[7]

5.4.1 Implementation

The model used consists of several different types of sequential operations and layers including-

1. **A tokenizer** - to split each of the articles into a sequence of words.
2. **A word embedding layer** - that learns an embedding vector with m dimensions for each unique word and applies this embedding for the first n words in each news article, generating a m x n matrix.
3. **1D convolutional and max-pooling layers**- for tasks in NLP we make use of a 1D convolutional layer that has a series of kernels, which are low-dimensional vectors that incrementally slide across the input vector as dot products are computed to produce the output vector. In addition to 1D convolutional layers we make use of 1D max pooling layer as well to reduce the size of the input by selecting a max value from a particular local region.
4. **LSTM layers followed by dropout layers-** The main reason for using an LSTM layer is that it allows us to remember selective information and keep track of the context of the sentence over a longer duration. This helps since there may be some useful information at the beginning of the article that is relevant to the latter parts of the document as well. The LSTM also helps us to decide which part of the information is relevant and should be retained and which is irrelevant and can be discarded.
5. **A final fully-connected layer**- The final part of the neural network is very similar to a "Vanilla" Neural Network that computes the weighted sum of the incoming values and performs sigmoid activation on it to produce a final result that is a value between 0 and 1. This value is then rounded to the nearest integer (i.e. 0 or 1) in order to make the final prediction of whether the article is "FAKE" or "REAL".

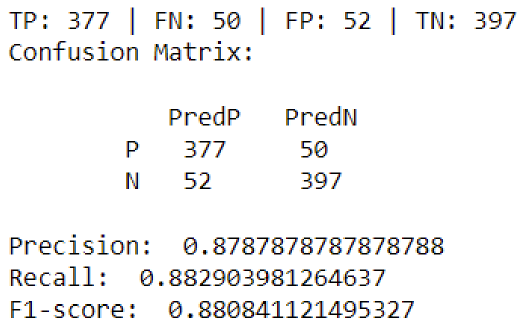
This model was run for **5 epochs** with the best model chosen at the end of the 5 epochs by making use of the validation accuracy.

5.4.2 Results

The neural net consisted was tested with the following parameters-

- Larger context retention (i.e. double the LSTM layers) - 3 layers, kernel size=5 with a pool size of 2 and 256 LSTM layers- An accuracy of around 96.2% was obtained by using the above parameters.

Validation accuracy metrics-

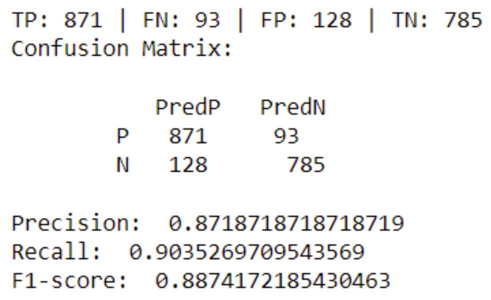


```
TP: 377 | FN: 50 | FP: 52 | TN: 397
Confusion Matrix:

        PredP   PredN
    P    377     50
    N    52      397

Precision:  0.8787878787878788
Recall:  0.882903981264637
F1-score:  0.880841121495327
```

Testing accuracy metrics-



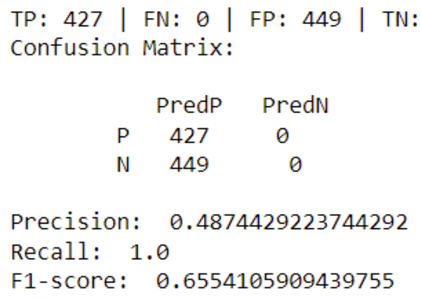
```
TP: 871 | FN: 93 | FP: 128 | TN: 785
Confusion Matrix:

        PredP   PredN
P    871     93
N    128      785

Precision:  0.8718718718718719
Recall:  0.9035269709543569
F1-score:  0.8874172185430463
```

- Lesser number of hidden neurons (i.e. smaller kernel size) - 3 layers, kernel size=3 with a pool size of 2 and 128 LSTM layers- An accuracy of around 95.1% was obtained by using the above parameters.

Validation accuracy metrics-

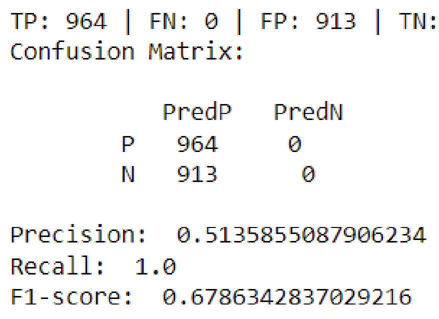


```
TP: 427 | FN: 0 | FP: 449 | TN: 0
Confusion Matrix:

        PredP   PredN
P    427     0
N    449      0

Precision:  0.4874429223744292
Recall:  1.0
F1-score:  0.6554105909439755
```

Testing accuracy metrics-



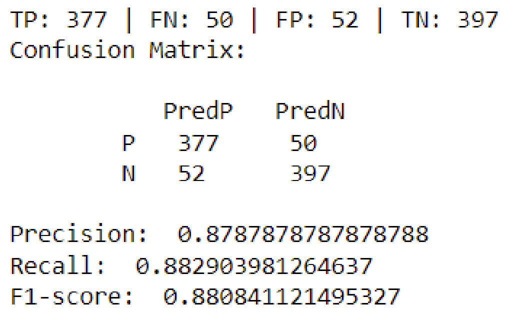
```
TP: 964 | FN: 0 | FP: 913 | TN: 0
Confusion Matrix:

        PredP   PredN
P    964     0
N    913      0

Precision:  0.5135855087906234
Recall:  1.0
F1-score:  0.6786342837029216
```

- Increased model depth (adding a hidden layer) - 4 layers, kernel size=5 with a pool size of 2 and 128 LSTM layers- An accuracy of around 95.6% was obtained by using the above parameters.

Validation accuracy metrics-

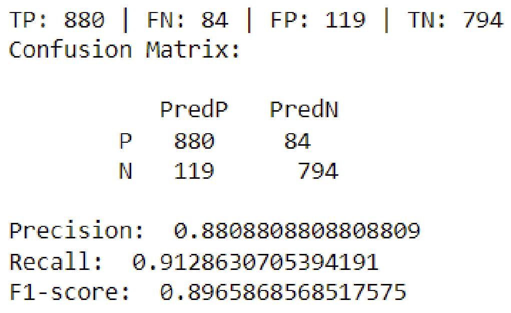


```
TP: 377 | FN: 50 | FP: 52 | TN: 397
Confusion Matrix:

        PredP   PredN
P    377     50
N    52      397

Precision:  0.8787878787878788
Recall:  0.882903981264637
F1-score:  0.880841121495327
```

Testing accuracy metrics-



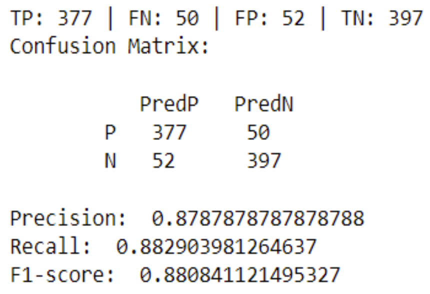
```
TP: 880 | FN: 84 | FP: 119 | TN: 794
Confusion Matrix:

        PredP   PredN
P    880     84
N    119      794

Precision:  0.8808808808808809
Recall:  0.9128630705394191
F1-score:  0.8965868568517575
```

- Hyperparameter tuned model configuration - 3 layers, kernel size=5 with a pool size of 2 and 128 LSTM layers- An accuracy of around 97.2% was obtained by using the above parameters.

Validation accuracy metrics-

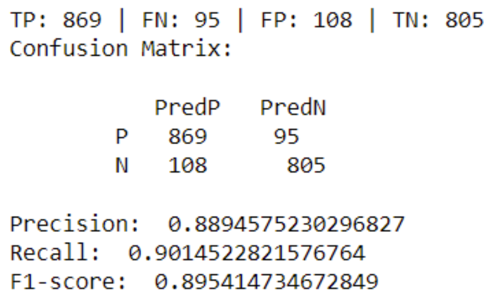


```
TP: 377 | FN: 50 | FP: 52 | TN: 397
Confusion Matrix:

        PredP   PredN
P    377     50
N    52      397

Precision:  0.8787878787878788
Recall:  0.882903981264637
F1-score:  0.880841121495327
```

Testing accuracy metrics-



```
TP: 869 | FN: 95 | FP: 108 | TN: 805
Confusion Matrix:

        PredP   PredN
P    869     95
N    108      805

Precision:  0.8894575230296827
Recall:  0.9014522821576764
F1-score:  0.895414734672849
```

5.5 k-Nearest Neighbors (kNN)

k-Nearest Neighbors is a supervised learning algorithm that is used either as a classifier or a regressor. In this experiment we have used kNN as a classifier to classify if an input document (news article) is fake or real. kNN is known to be a “lazy” algorithm since for every document it goes through the entire dataset looking for the ‘k’ nearest neighbors to the corresponding document. To compare the documents, we have used TF-IDF on the dataset which converts each document to a word vector and compares vectors using a similarity measure. We have used Euclidean Distance, Cosine Similarity and Manhattan Distance as the similarity measures.

5.5.1 Distance Measures

Euclidean Distance (ED) is the direct length of the line segment between the two points (vectors in our case). A lower euclidean distance means the vectors (data points) are closer to each other.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Cosine Similarity (CS) measures the similarity between the vectors by finding the cosine angle between them. It is calculated using the following formula:

$$1 - \frac{u \cdot v}{\|u\|_2 \|v\|_2}.$$

Manhattan Distance (MD) goes by many names, but it essentially is the distance between the two points according to the cartesian coordinates of the points. This makes it a slightly less accurate measure. It is calculated using the formula:

$$\sum_{i=1}^{n} |p_i - q_i|$$

5.5.2 Implementation

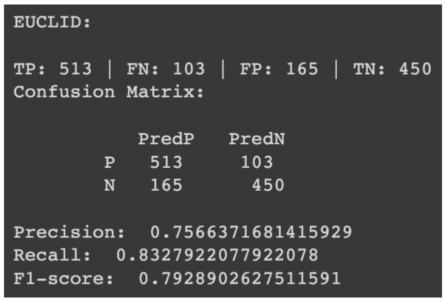
kNN being a computationally intensive and lazy algorithm we have implemented multiprocessing to increase the speed of execution and maximize utilization of CPU resources. The entire algorithm has multiple steps which are described below:

1. First we read the entire dataset as a dataframe and perform pre-processing operations that clean up the news article text.
2. Next, we perform TF-IDF on the dataset to create the word vectors of the news articles and then split the data set in a 80:20 split between training and testing datasets.
3. In the implementation using sklearn, we also performed Latent Semantic Indexing on the TF-IDF sparse matrix to reduce the dimensions.
4. After this, we move on to training the kNN model. For each data point in the test set, we find the 3 nearest neighbors.
5. To find the nearest neighbors we use the following distance measures: euclidean distance, cosine similarity and manhattan distance.
6. For each distance measure, we find that the resultant nearest neighbors are different, so keeping track of the different neighbors, we classify the test data point differently for each measure.
7. Classification of the data point is done by counting the majority votes from each of those neighbors. A majority vote will result in that test data point to be classified as either Fake or Real.
8. We now have the predictions made for each test data point using the different distance measures and find the accuracy of each measure.

5.5.3 Results

The kNN algorithm was tested for each distance measure’s prediction for each data point. The results show that the scratch implementation performs poorly when using cosine similarity and manhattan distance measures. We feel this is because of the TF-IDF implementation which for performance reasons uses only the most significant

1000 words from the vocabulary. This significantly reduces the capability of the algorithm to find accurate neighbors.



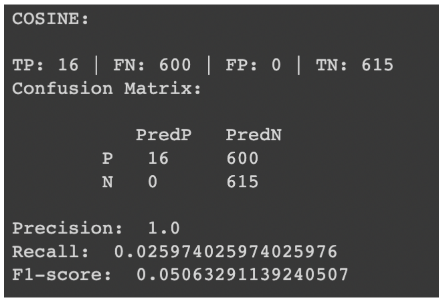
```
EUCLID:

TP: 513 | FN: 103 | FP: 165 | TN: 450
Confusion Matrix:

           PredP   PredN
        P   513     103
        N   165      450

Precision:  0.7566371681415929
Recall:  0.8327922077922078
F1-score:  0.7928902627511591
```

Figure 09 : Confusion matrix for Euclid Distance predictions



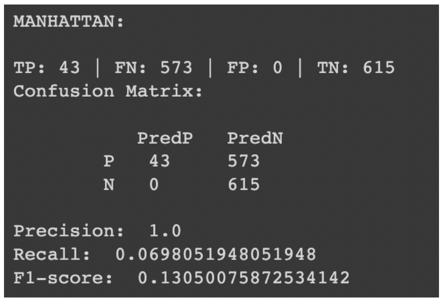
```
COSINE:

TP: 16 | FN: 600 | FP: 0 | TN: 615
Confusion Matrix:

           PredP   PredN
        P   16      600
        N   0       615

Precision:  1.0
Recall:  0.025974025974025976
F1-score:  0.05063291139240507
```

Figure 10 : Confusion matrix for Cosine Similarity predictions



```
MANHATTAN:

TP: 43 | FN: 573 | FP: 0 | TN: 615
Confusion Matrix:

           PredP   PredN
        P   43      573
        N   0       615

Precision:  1.0
Recall:  0.0698051948051948
F1-score:  0.13050075872534142
```

Figure 11 : Confusion matrix for Manhattan Distance predictions

6. EXPERIMENTAL EVALUATION

The final implementation is detailed in the code. For fair comparison between the accuracy of the sklearn model and our own implementation of the models, we use the same preprocessed data for both.

The accuracy obtained using the sklearn model for the **Multinomial Naive Bayes** was ~83% for the. For fair comparison, after setting the alpha value (to 1), the final accuracy achieved with our own implementation was ~57%.

The accuracy obtained using the sklearn models for **Logistic Regression** was ~92%. After tweaking the alpha and iteration values, the final accuracy achieved with our own implementation was ~82%. The final alpha value and iteration values were 0.1 and 300.

The accuracy obtained using the sklearn models for **Support Vector Machine** was ~93%. After tweaking the kernels, the final accuracy achieved in the implementation was ~x%.

The accuracy obtained using the sklearn models for **k-Nearest Neighbors** was around 67% for ED, 52.6% for MD and 82% for CS. For the scratch implementation, we got around 78% for ED, 54% for MD and 52% for CS.

The accuracy obtained using the **Recurrent Convolutional Neural Network** approach was approximately 97% after tuning all the hyperparameters.

7. CONCLUSION

We implemented the five chosen supervised machine learning algorithms both using sklearn and after referring to the original research work, we encountered that the original implementation required a lot of tuning to obtain accuracy that is on par with the sklearn implementation. We could achieve acceptable accuracy and could build successful models that could classify real news from fake news.

8. REFERENCES

[1] "Building a Logistic Regression in Python" Animesh Agarwal, published in Towards Data Science
[2] "SVM From Scratch in Python,,Qandeel Abbassi published in Towards Data Science
[3] https://en.wikipedia.org/wiki/Support-vector_machine#Implementation
[4] Hinge Loss - https://en.wikipedia.org/wiki/Hinge_loss
[5] https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[6] Multinomial Naive Bayes Explained by Gautam Solanki
[7] Siwei Lai, Liheng Xu, Kang Liu, Jun Zhao, Recurrent Convolutional Neural Networks for Text Classification.
[8] https://en.wikipedia.org/wiki/Artificial_neural_network
[9] Towards data science - Amol Mavuduru, https://towardsdatascience.com/fake-news-classification-with-recurrent-convolutional-neural-networks-4a081ff69f1a