CM3070 Computer Science Final Project Final Report

Project Idea: Fake News Detection

Project Theme: Natural Language Processing (CM3060)

Student Name: Tan Zhao Fei

Table of Contents

Introduction

The premise of this project is on fake news detection, which is a text classification task that falls under Natural Language Processing (NLP). The project will be researching on different text classifiers, implementing them and evaluating them to determine the most suitable solution for fake news detection.

Fake news are news reports that do not contain verified facts or sources and are hence, false. Fake news is the dissemination and, in some cases, the creation of misinformation and disinformation. Misinformation is the less harmful of the two, as the intention is not to deceive. The readers are misinformed and truly believe that the information is verified and true. This could be due to poor fact checking or false labelling. Disinformation is the more harmful of the two, as the intention is to deceive. Disinformation is deliberately fabricated and spread to cover truths and/or influence the opinion of the public. (Shevon Desai, 2022)

Fake news has existed since the ancient times, ever since human beings started to learn how to communicate. However, with the invention of the internet, fake news has become prevalent due to the widespread reach and sometimes anonymity the internet provides. Fake news often tends to draw in and mislead their readers using flashy buzz words and largely exaggerated clickbait headlines for many reasons, such as: economic incentives, social incentives, and political incentives.

Some examples of economic incentive are referral links, more traffic flow to new sites and investment opportunities. Some examples of social incentives are improving one's social image, improving one's credibility and popularity. (Anubhav Mishra, 2021) Some examples of political incentives are influencing votes during elections, spreading misinformation about opposing parties and propaganda. (Robert B.Michael, 2021)

Most fake news is spread via social media platforms such as Facebook, Twitter, and Instagram where users are able to be bold and anonymous, often without facing any repercussions. Millions of posts per day on these platforms make it impossible fully to curb and control. The impact of fake news varies greatly. There are two recent events that fake news had a huge impact on: the COVID-19 pandemic and the invasion of Ukraine. Fake news during COVID-19 led to the public being distrustful towards healthcare professionals and scientific research. An example would be the issue regarding the safety of vaccinations. Fake news currently has led to many Russians being distrustful towards their own government due to propaganda controlling the narrative of the invasion.

This project aims to tackle this issue by proposing a deep machine learning model that is capable of filtering out fake news. The model will be using recurrent neural networks (RNN) for fake news classification. The model must be able to accurately differentiate real and fake news in a reasonable amount of time and will be evaluated and be compared to other baseline text classifiers.

Literature Review

The objective of fake news detection is to classify news as real or fake. As such, this is a binary text classification task and there are several machine learning models and classifiers capable of this task. Research has been done on several of these models and classifiers.

Techniques and Methods

Non-Neural Network

Naïve Bayes

Naïve Bayes is modified Bayesian classifier based on Bayes' theorem, as seen below in Figure 1.

$$P(L \mid \text{features}) = \frac{P(\text{features} \mid L)P(L)}{P(\text{features})}$$

Figure 1. Bayes' Theorem, where P is probability, L is label

Bayes' theorem is an equation that describes the probability of an event using existing knowledge of features related to said event. In order to compute P(features|L) P (L), a generative model is used as it will specify the hypothetical random process which generates data. This specification by the generative model is the main bulk of training for a Bayesian classifier. Naïve Bayes eliminates the training needed for this generative model by making naïve assumptions for the labels. As a result, naïve Bayes classifiers are known to be quick and easy to implement and train and are often used as baselines for classification problems. (VanderPlas, 2016)There are three types of naïve Bayes: Multinomial Naïve Bayes, Gaussian Naïve Bayes, and Bernoulli Naïve Bayes. (Kaliyar, 2018)

Multinomial naïve Bayes assumes that the features of a dataset follows a multinomial distribution. The data distribution will be modelled with a best-fit multinomial distribution.

Gaussian naïve Bayes assumes that label data follows a Gaussian distribution. This is the simplest as a Gaussian distribution only requires the mean and standard deviation of the data points of each label. The data distribution will be modelled with a best-fit Gaussian distribution.

Bernoulli naïve Bayes is used when label data follows a Bernoulli distribution. As such, it is only used when the label data contains binary values.

As mentioned above, the naïve bayes classifiers are both quick and simple, due to the simple assumptions made. This gives them the a few advantages; namely that they are quick and straightforward to implement and train to use for prediction. Another advantage would be that because of their simplicity, they have little to no parameters that can be tuned. This also causes them to be easily interpreted. However, having almost no parameters that can be tuned is also a disadvantage. If the classifier does not perform well after evaluation, not much can be done except to use a more complicated model for better results. (VanderPlas, 2016)

Random Forest

Random forests are made up of decision trees. Decision trees classify or label objects by asking binary questions that narrow down the classification or labelling in a tree-like structure as shown below in Figure 2. (VanderPlas, 2016)

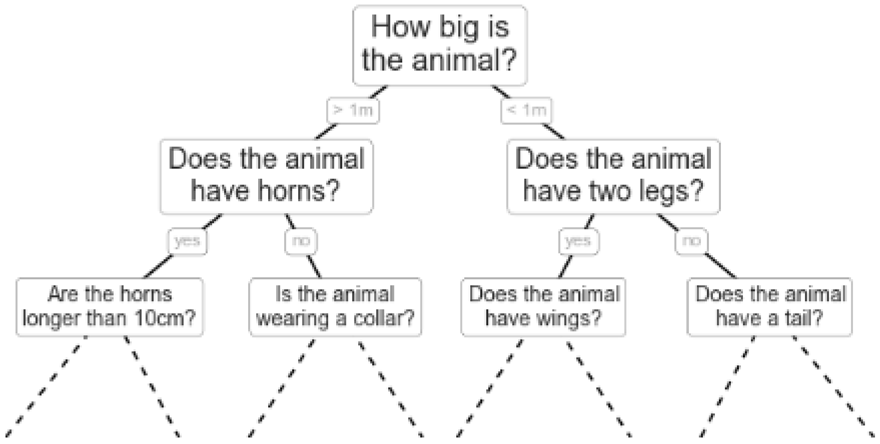

How big is the animal?
> 1m
< 1m
Does the animal have horns?
Does the animal have two legs?
yes
no
yes
no
Are the horns longer than 10cm?
Is the animal wearing a collar?
Does the animal have wings?
Does the animal have a tail?

Figure 2. Decision Trees

Decision trees are very efficient due to its binary nature; each question will narrow down the possible options by half. Well-constructed trees can be extremely quick for classification or labelling. In a machine learning use-case, the binary questions are axis-aligned splits in the data using a reference value from one of the features of a dataset. An illustration is shown below in Figures 3 and 4. (VanderPlas, 2016)

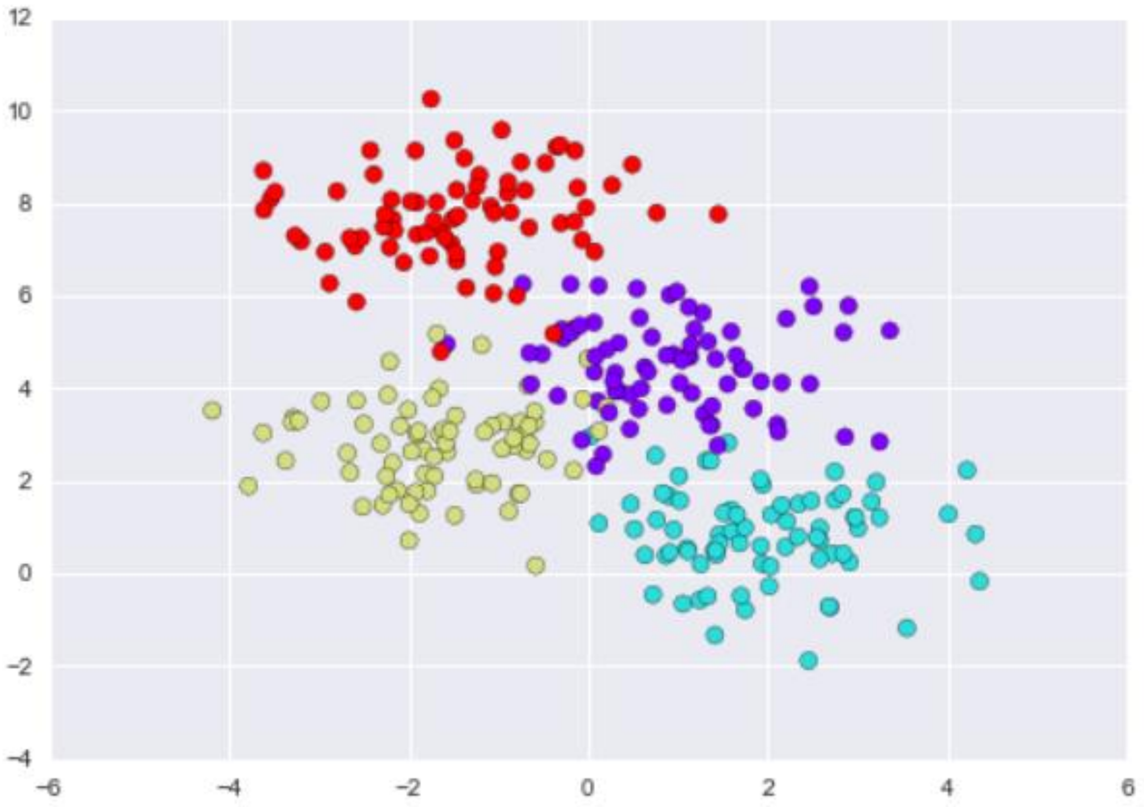


Figure 3. Initial Data

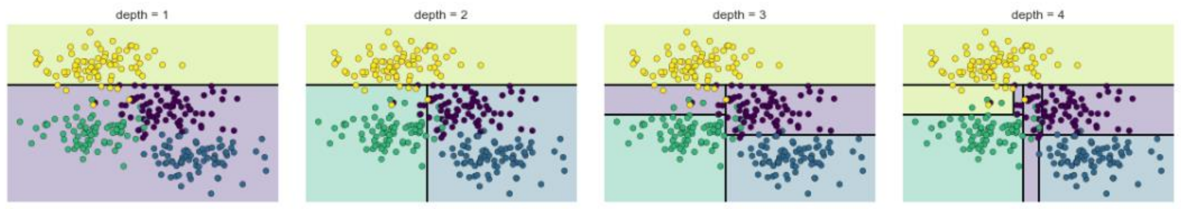


Figure 4. Visualization of data splits

After each split, only regions that contain different coloured data points are further split. This can be seen above as after the first split, the yellow region only contained yellow coloured data points and will no longer be split going deeper down the tree. (VanderPlas, 2016)

Random forests are quick to implement and train to use for prediction due to its aforementioned simple binary nature. Each decision tree is also considered an independent entity, which allows parallelizing to be straightforward. However, a disadvantage of random forests is that going too deep down the trees can easily cause overfitting and with a high number of splits, interpretation could prove to be difficult. (VanderPlas, 2016)

Linear Regression

Linear regression is a good baseline due to its simplicity to implement and ease of interpretation. The most common and well known linear regression is to straight line fit to data. The straight line equation is shown below in Figure 5.

$$y = ax + b$$

Figure 5. Straight Line Fit, where a is gradient, and b is intercept

Linear regression can also be done in three dimension to fit a hyperplane to data points in higher dimensions. This will sacrifice some interpretability due to the addition of dimensions. The multidimensional linear model equation is shown below in Figure 6.

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots$$

Figure 6. Multidimensional Linear Model Equation

Linear regression is therefore very flexible as it is able to fit lines (1D), planes (2D) and even hyperplanes (3D) to data..

Neural Network

Neural networks take in numerical vectors as input to perform parameter tuning calculations. Neural networks have many parameters that can be tuned: weight, bias, optimizer, and the activation and loss functions. (Jamal Abdul Nasir, 2021)

Weights are initialized randomly and will be trained. The optimizer's purpose is to reduce the error between the desired and actual result.

At each layer, the activation function takes in an input and, along with the initialized weights and biases, output an input for the next layer. This will repeat until the final layer's output which is considered the result. The loss function is responsible for computing the error between the desired and actual result for each input. (Jamal Abdul Nasir, 2021)

Convolutional Neural Network

An example of a neural network is a Convolutional Neural Network (CNN). CNNs process input data by using kernels to extract features from overlapping regions of the input data. Convolution is then applied to these features to represent the input data in a higher level. Convolution is the multiplication of arrays or matrices that will provide outputs for further training. This results in the CNN learning the temporal and spatial features of the dataset. In an NLP use case, the words in a sentence are represented as word vectors.

The word vectors will then be used to train a CNN. Training a CNN involves specifying a kernel size as well as the number of filters as shown below. A filter of that specified size will then iterate though training data. Each iteration multiplies the input with the weights of the filter, which will then produce an output that is to be stored in an output array as shown below. (Jamal Abdul Nasir, 2021)

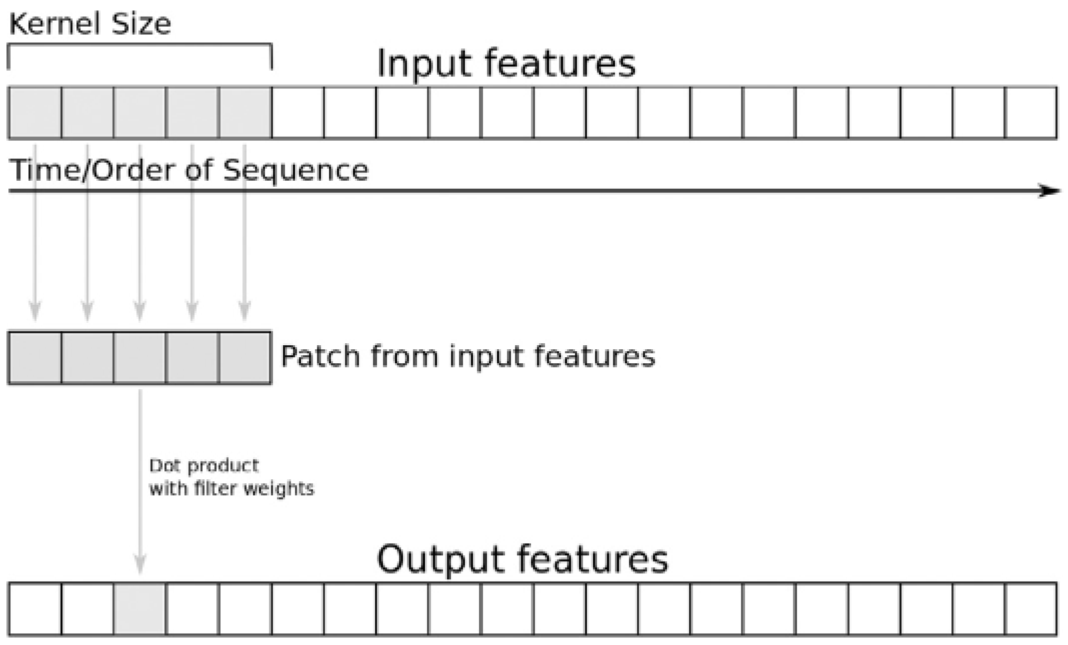


Figure 7. Convolution

The output array is considered the output filter of the data, which makes it a feature map of the data. As such, just from training the CNN, a feature is detected from the input training data.

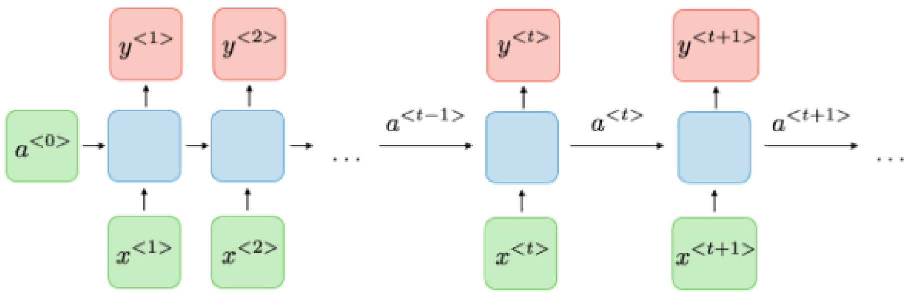
Although CNNs can be multi-dimensional, for the use-case of NLP, a one-dimensional CNN is preferred. An example of a one-dimensional CNN is Conv1D. As mentioned above for NLP, the words in a sentence are represented as word vectors and Conv1D handles one dimensional arrays representing word vectors. (Jamal Abdul Nasir, 2021)

CNNs have huge learning capabilities and data capacity since it is able to run multiple process at a time. However, this results in CNNs being slow, requiring more computational power and a longer training period. CNNs are mainly used to process structure grid data such as images and videos. They are more suited for object detection or image classification problems and as such, will not be very suitable for this project which is a text classification problem.

Recurrent Neural Network

Another example of a neural network is a Recurrent Neural Network (RNN). It is recurrent as the output of each step is passed on as input for the next step. As such, RNN processes training data sequentially during training and is able to “remember” the output of the previous step. This allows RNN to learn long-term dependencies in the training data. In an NLP use case, it is possible to use multiple inputs so that training can be done in relation to each input, which increases efficient as compared to just using a single input. (Jamal Abdul Nasir, 2021)

RNNs are able to utilize different types of memory cells. Some example of these memory cells are Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM). LSTM comprises of a carry, a cell state, weights, three different gates for learning and the current word vector at the time. The carry ensures that no information is lost during sequential processing. The three different gates are an input gate for the current input, an output gate for predict values and a forget gate for discarding irrelevant information. (Jamal Abdul Nasir, 2021) The architecture of an RNN as well as the anatomy of GRU and LSTM is shown below in Figure 7 and 8 respectively.



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

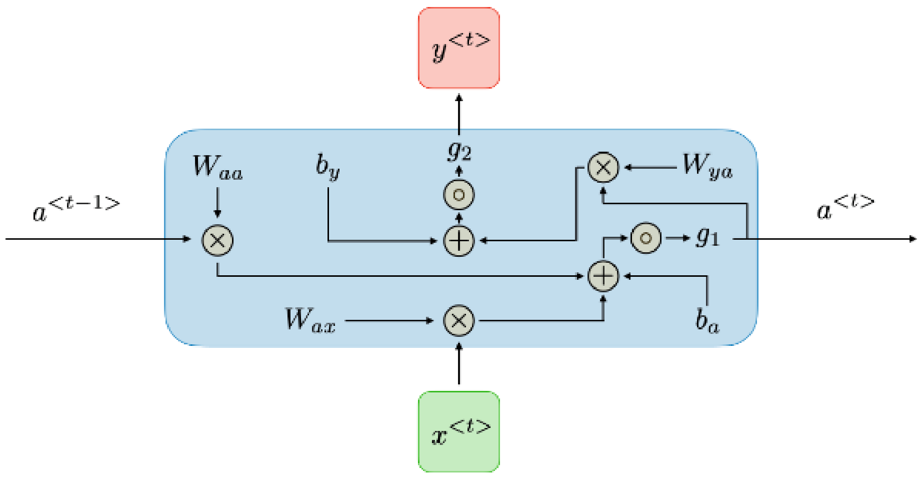


Figure 8. RNN Architecture

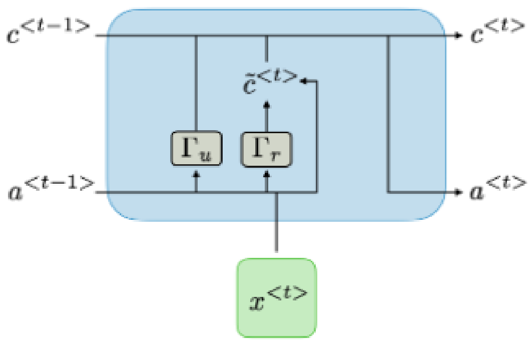
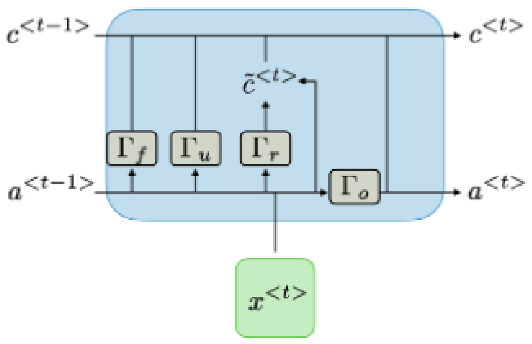
Gated Recurrent Unit (GRU)	**Long Short-Term Memory** (LSTM)
$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$c^{<t>}$	$\Gamma_o \star c^{<t>}$
 $c^{<t-1>}$ $c^{<t>}$ $\tilde{c}^{<t>}$ Γ_u Γ_r $a^{<t-1>}$ $a^{<t>}$ $x^{<t>}$	 $c^{<t-1>}$ $c^{<t>}$ $\tilde{c}^{<t>}$ Γ_f Γ_u Γ_r Γ_o $a^{<t-1>}$ $a^{<t>}$ $x^{<t>}$

Figure 9. Anatomy of GRU and LSTM

RNNs are able to process input of any length without affecting the size of the model. As mentioned above, RNNs are also able to consider previous inputs during processing and share weights across time. However, RNN requires more computational power and may face difficulty when trying to access historical information if a long time has passed. Another possible issue that RNNs face is a exploding gradient due to multiplicative gradients which scale exponentially with the number of layers. This causes difficulty in learning and capturing long term dependencies, although the aforementioned GRU and LSTM do help to deal with this issue. (Afshine Amidi, n.d.)

Project Design

Pipeline

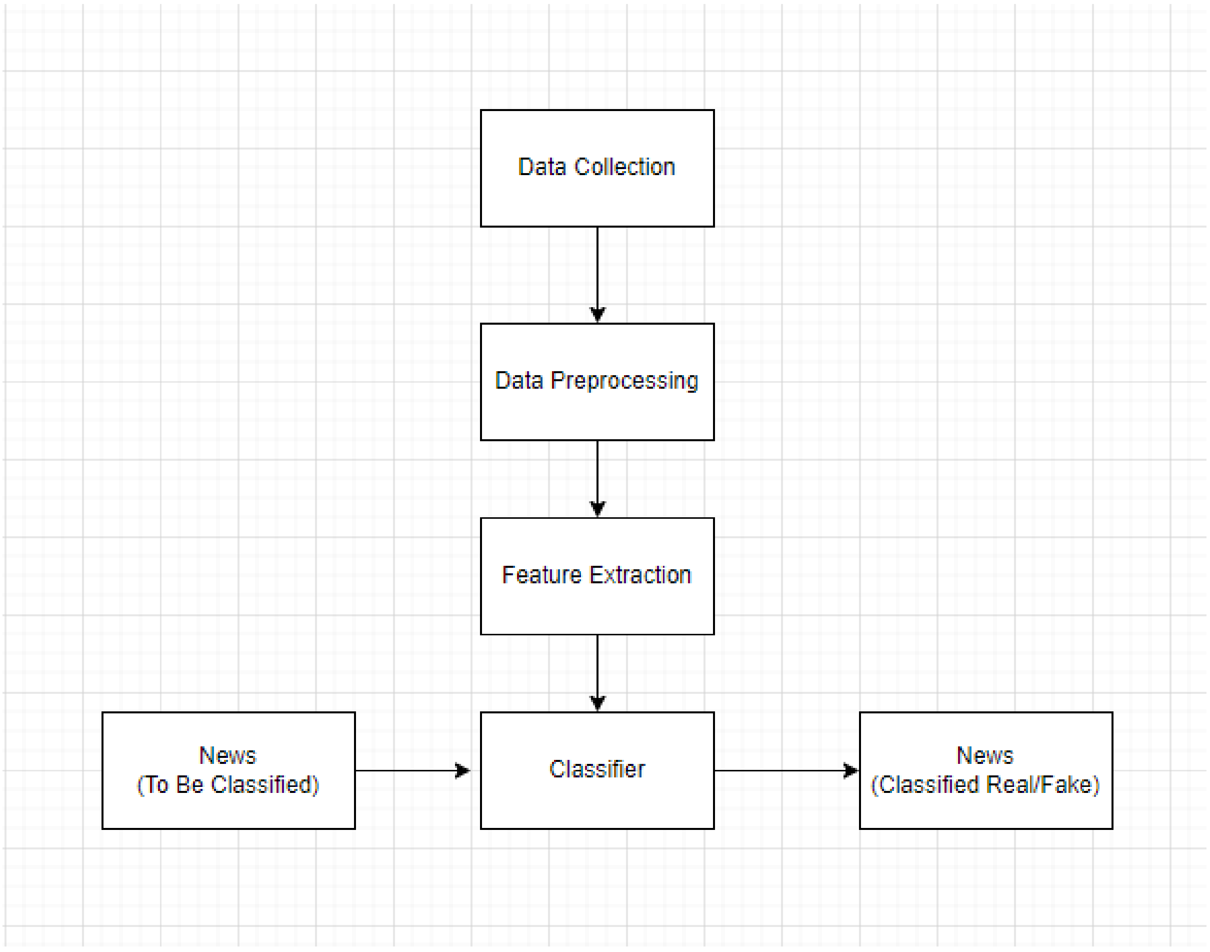


Figure 10. Basic Pipeline

Data Collection

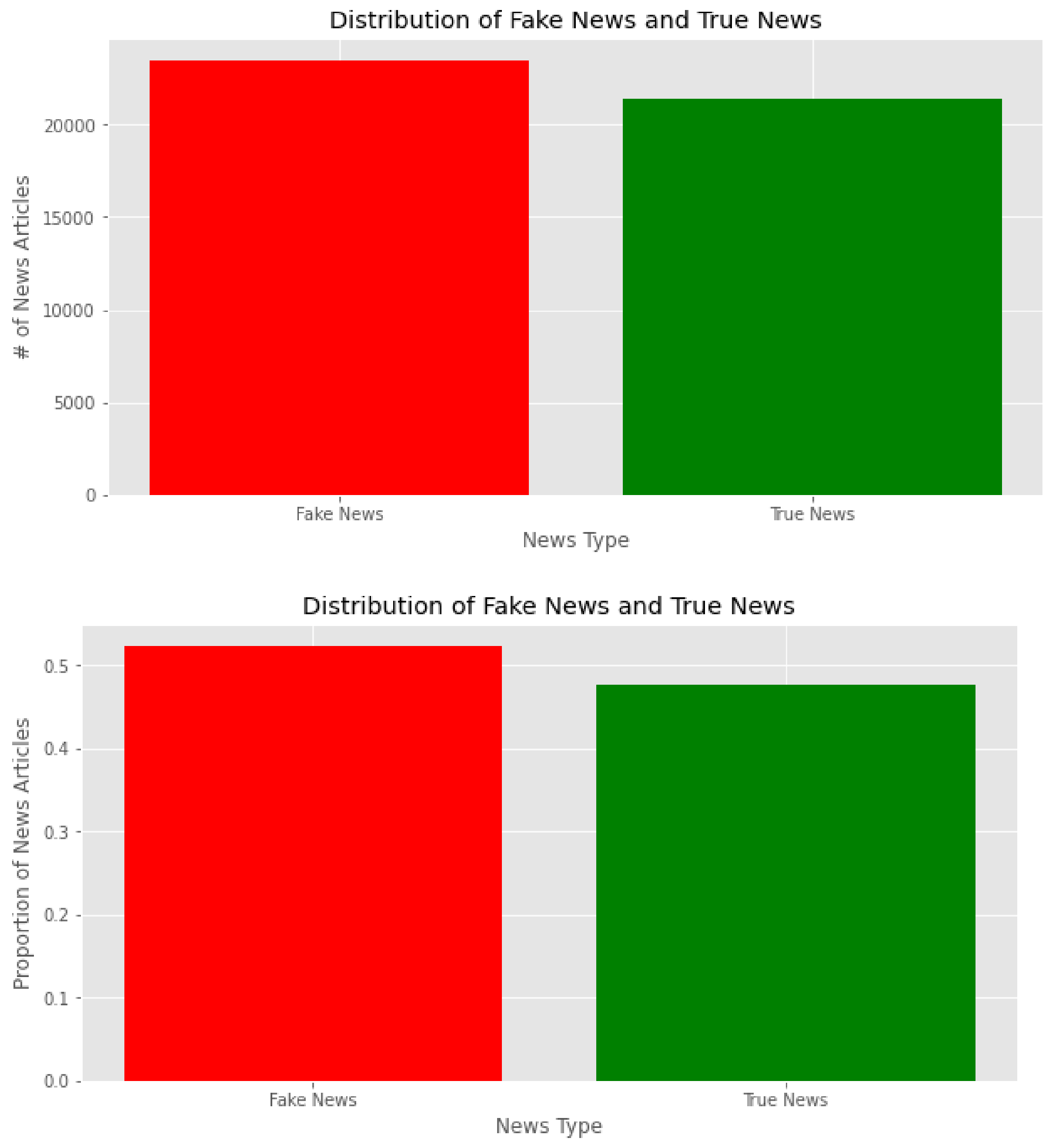


Figure 11. Fake and True News Dataset Distribution

Data collection is done through Kaggle, an online data science and machine learning community. The dataset is split into two csv.files: True.csv and Fake.csv. True.csv has 21418 rows and 4 columns while Fake.csv has 23503 rows and 4 columns. (Bisaillon, 2020)

Data Pre-processing

After data collection, the data is processed to prepare for future steps. This includes:

- Removal of white spaces
- Removal of stop words
- Removal of URL links

- Checking of null values
- Dropping irrelevant column, date
- Convert text to lowercase
- Splitting training and testing data
- Converting text in training and testing data to vectors for neural network input

Feature Extraction

The texts_to_sequences method from the Tokenizer class in keras is used to perform feature extraction. Below is a brief summary of what is performed.

Tokenization

Tokenization is used to split the text into their constituent words.

Indexing

A dictionary is then created to map each of these words to a unique integer value.

Vectorization

This dictionary is then used to convert sentences to a sequence of integers. Each unique word in the sentence will be represented by an integer.

Padding

Padding is added to the integer sequences to ensure that all sentences are of the same length for further processing.

Classifier

After the literature review section, the use of RNN for classification is decided. The other classifiers, naïve Bayes, random forest, logistic regression and CNN, will also be implemented and used to compare and evaluate the performance of RNN.

Planning

Gantt Chart

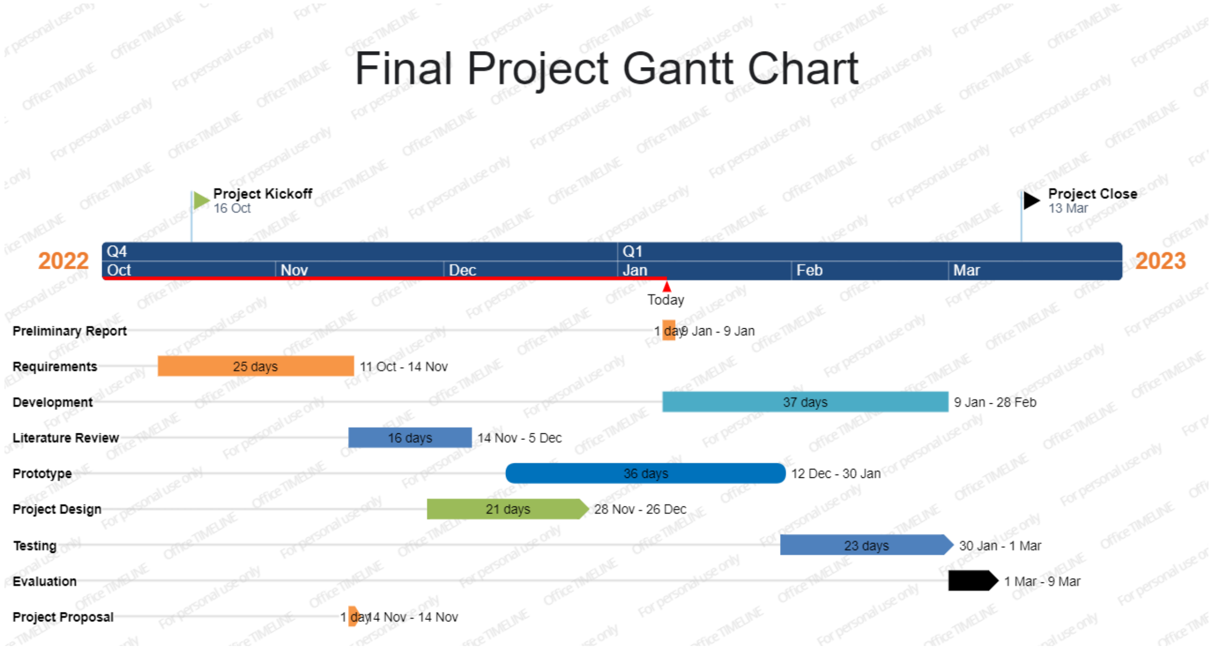


Figure 12. Gantt Chart

Testing and Evaluation

Metrics

The classifier will be evaluated using the following metrics: accuracy, precision and f-score. Accuracy will measure how well the classifier is able to correctly classify real and fake news. (scikit-learn, n.d.) Precision will measure how well the classifier is able to not classify real news as fake, false positives. Recall will measure how well the classifier is able to classify real news as real. The f-score is an overmeasure of the classifier's accuracy, precision, and recall. The formulas for all 4 metrics are shown below in Figure 12. (scikit-learn, n.d.)

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \quad Recall = \frac{truepositives}{truepositives + falsenegatives}$$

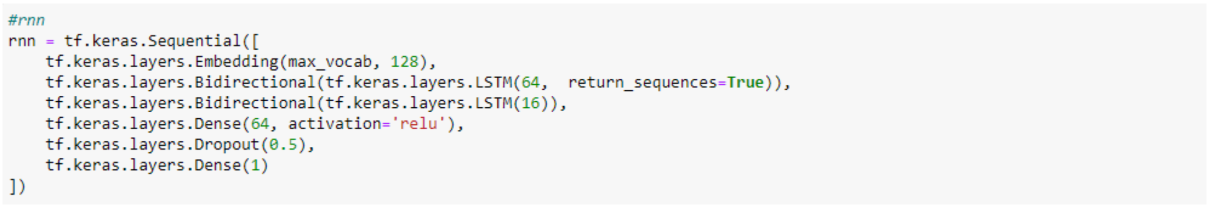
$$Precision = \frac{truepositives}{truepositives + falsepositives}$$

Figure 13. Accuracy, Precision, Recall, F-Score Formulas

Visualizations techniques such as a confusion matrix, heatmap and graphs will also be implemented to easier interpretation.

Implementation

RNN



```
#rnn
rnn = tf.keras.Sequential([
    tf.keras.layers.Embedding(max_vocab, 128),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1)
])
```

Figure 14. Building RNN Model Code

The RNN model is built primarily using the keras and TensorFlow libraries. A sequential model is created, and the following neural network layers are added: an embedding layer, two bidirectional LTSM layers, two dense layers and a dropout layer.

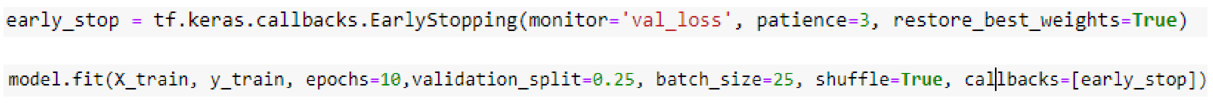
Embedding Layer

The purpose of the embedding layer is to map a vector of real numbers to each word in a vocabulary. This allows the neural network to learn the syntactic and semantic relationship between words and how to represent them, which is extremely useful in text classification, the focus of this project. This will then be used as input to the neural network. (Keras, n.d.)

Bidirectional/LSTM Layer
The bidirectional layer works by processing the input sequence forward and backwards, which is useful as it allows the neural network to take into consideration the context around the word. The LSTM layer, as previously discussed, is able to use gates to selectively remember or forget information while capturing any long-term dependencies over time. These two layers are added to improve the performance of the neural network. (Keras, n.d.)

Dense Layer
The dense layer works by applying a linear transform on the input data and output a higher-level representation of the data. This allows the neural network to learn of any intricate relationships between the input and output data. (Keras, n.d.)

Dropout Layer
The dropout layer works by randomly setting a portion of the input to zero during each iteration. This forces the neural network to learn about the robust features of the data which do not depend on any single input. This improves the generalization and prevents overfitting of the neural network. (Keras, n.d.)

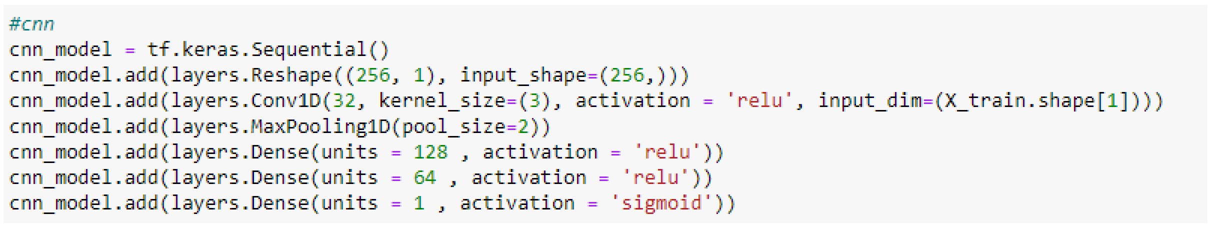
Early Stop Callback


```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

model.fit(X_train, y_train, epochs=10,validation_split=0.25, batch_size=25, shuffle=True, callbacks=[early_stop])
```

Figure 15. Early Stop Initialization

The early stop callback parameter is an EarlyStopping object that is set to monitor validation loss (val_loss). The training process will be stopped if the validation loss is not improving. The patience parameter determines how many epochs before the callback kicks in and stops the training process. This increases training efficiency as the training process automatically stops if no improvement is seen and the parameters can be changed quicker. (TensorFlow, n.d.)

CNN


```
#cnn
cnn_model = tf.keras.Sequential()
cnn_model.add(layers.Reshape((256, 1), input_shape=(256,)))
cnn_model.add(layers.Conv1D(32, kernel_size=(3), activation = 'relu', input_dim=(X_train.shape[1])))
cnn_model.add(layers.MaxPooling1D(pool_size=2))
cnn_model.add(layers.Dense(units = 128 , activation = 'relu'))
cnn_model.add(layers.Dense(units = 64 , activation = 'relu'))
cnn_model.add(layers.Dense(units = 1 , activation = 'sigmoid'))
```

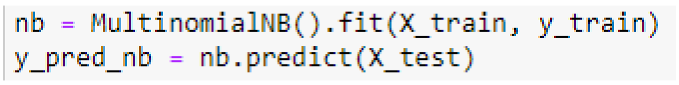
Figure 16. Building CNN Model Code

The CNN model is also built primarily using the keras and TensorFlow libraries. The process is very similar to the RNN model, the main differences being the layers used. A sequential model is created, and the following neural network layers are added: a reshape layer, a one-dimensional convolutional layer, a one dimensional MaxPooling layer, and 3 dense layers.

Reshape Layer
The reshape layer is used to reshape the input data to have the suitable number of dimensions for the next neural network layer. (Keras, n.d.)

Convolutional Layer
The convolutional layer works by sliding kernels to the input data. It also applies a dot product between the kernel weights in each sliding window, producing a feature map of the output. This allows the neural network to extract features or patterns from data sequences to learn how to hierarchically represent the data. (Keras, n.d.)

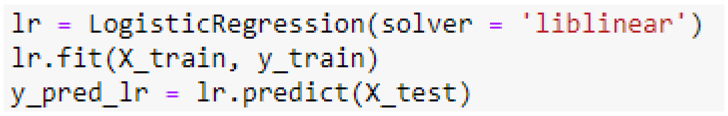
MaxPooling Layer
The MaxPooling layer works by also sliding over the output of the convolutional layer and returns a maximum value. This essentially down samples the output of the convolutional layer and allows the reduction of parameters and chance of overfitting of the model. The MaxPooling layer also reduces the spatial size of data, increasing data processing efficiency. (Keras, n.d.)

Naïve Bayes


```
nb = MultinomialNB().fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
```

Figure 17. Naive Bayes Implementation

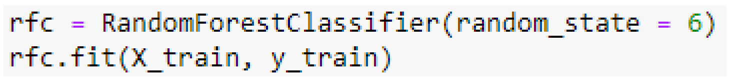
The naïve Bayes model is built using the scikit-learn library. The model is already available as a module which makes implementation simple.

Logistic Regression


```
lr = LogisticRegression(solver = 'liblinear')
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

Figure 18. Logistic Regression Implementation

The logistic regression model is built using the scikit-learn library. The model is already available as a module which makes implementation simple.

Random Forest


```
rfc = RandomForestClassifier(random_state = 6)
rfc.fit(X_train, y_train)
```

Figure 19. Random Forest Implementation

The random forest model is built using the scikit-learn library. The model is already available as a module which makes implementation simple.

Evaluation

Results

Classifier	Accuracy	Precision	Recall	F-Score
RNN	98.65%	98.14%	98.93%	98.54%
CNN	54.81%	100%	54.81%	70.81%
Naïve Bayes	56.71%	53.50%	53.23%	53.36%
Random Forest	86.75%	84.58%	87.48%	86.00%
Logistic Regression	60.15%	57.52%	54.91%	56.19%

Figure 20. Result Table for each Classifier based on Evaluation Metrics

As seen from Figure 20, the naive Bayes, logistic regression, and CNN classifiers performed the worst, the random forest classifier performed decently, and the RNN classifier performed excellently.

Naïve Bayes and Logistic Regression

```
              precision    recall  f1-score   support

           0       0.62      0.65      0.63      6002
           1       0.58      0.55      0.56      5223

    accuracy                           0.60     11225
   macro avg       0.60      0.60      0.60     11225
weighted avg       0.60      0.60      0.60     11225
```

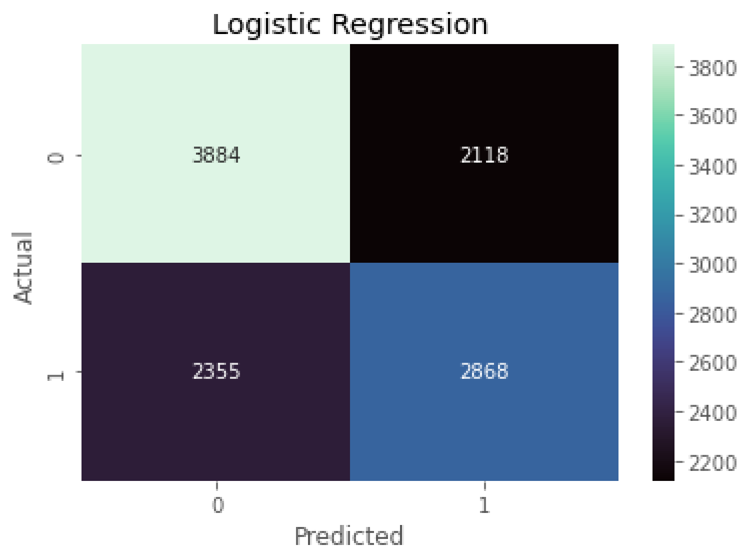


Figure 21. Classification Report and Confusion Matrix for Logistic Regression

```
        Naive Bayes Classifier Results

              precision    recall  f1-score   support

           0       0.59      0.60      0.60      6002
           1       0.54      0.53      0.53      5223

    accuracy                           0.57     11225
   macro avg       0.56      0.56      0.56     11225
weighted avg       0.57      0.57      0.57     11225
```

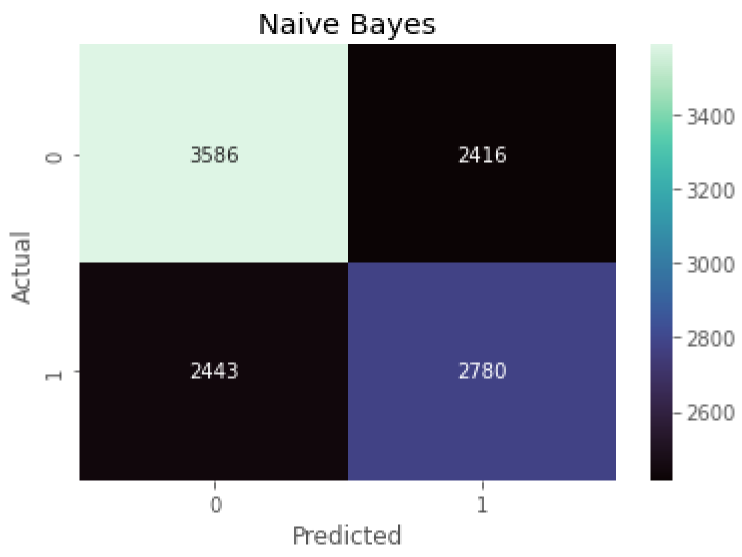


Figure 22. Classification Report and Confusion Matrix for Naive Bayes

The naïve Bayes and logistic regression classifier performed terribly with an accuracy score of 56.71% and 60.15% respectively. For the models themselves, not much can be done as they do not allow much parameter tuning. However, one way to improve the performance of these classifiers would be to use a different feature extraction technique such as bag of words or a TD-IDF vectorizer.

CNN

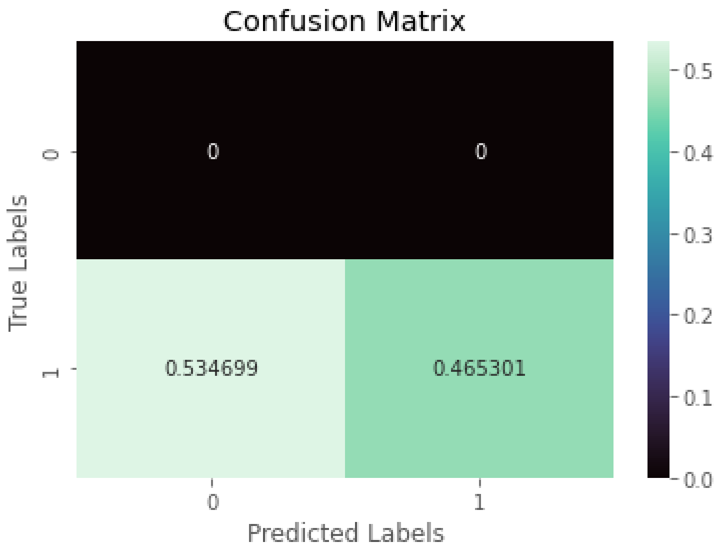
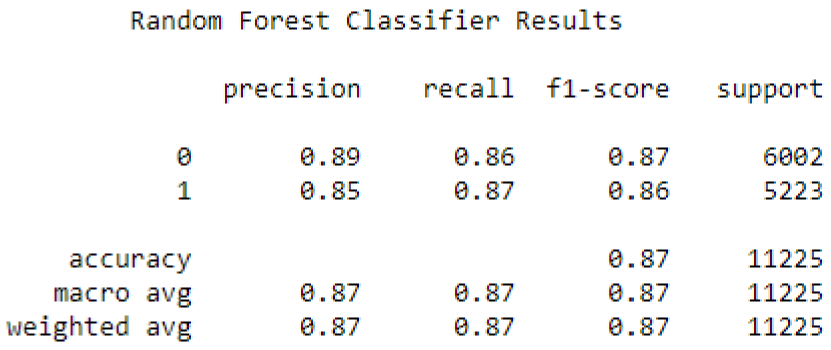


Figure 23. Confusion Matrix for CNN

The CNN classifier also performed terribly with an accuracy score of 54.81%. As discussed previously, it is more suited for image classification tasks, not text classification. The performance of the CNN classifier was to be expected. However, there are potentially some ways to improve the performance of the classifier, such as experimenting with the addition of different neural network layers, loss and activation functions and parameter tuning.

Random Forest



```
                Random Forest Classifier Results

              precision    recall  f1-score   support

           0       0.89      0.86      0.87      6002
           1       0.85      0.87      0.86      5223

    accuracy                           0.87     11225
   macro avg       0.87      0.87      0.87     11225
weighted avg       0.87      0.87      0.87     11225
```

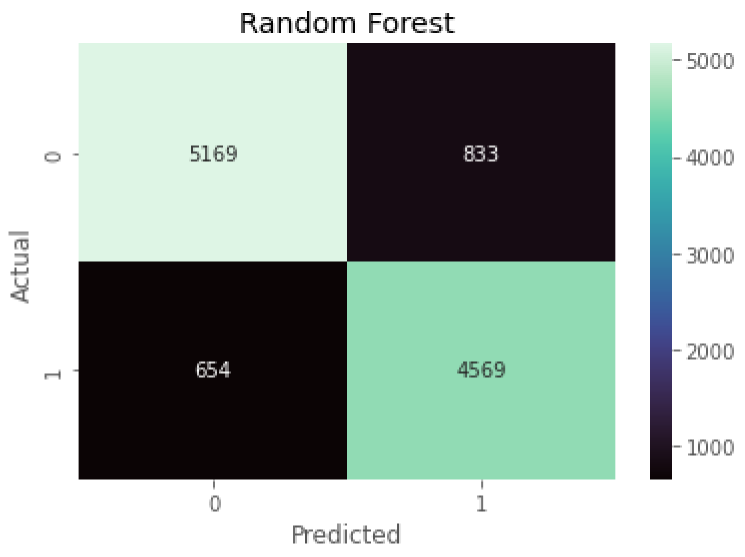


Figure 24. Classification Report and Confusion Matrix for Random Forest

The random forest classifier performed decently with an accuracy score of 86.75% without much training and computation time. The classifier would be ideal if set up time was crucial without sacrificing too much on performance. However, there are ways to improve the performance of the classifier such as again using different feature extraction techniques and perhaps performing cross validation using KFold.

RNN

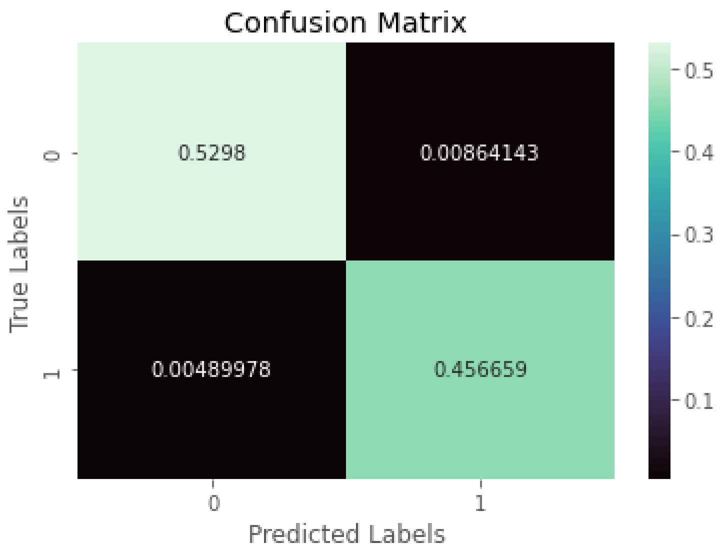
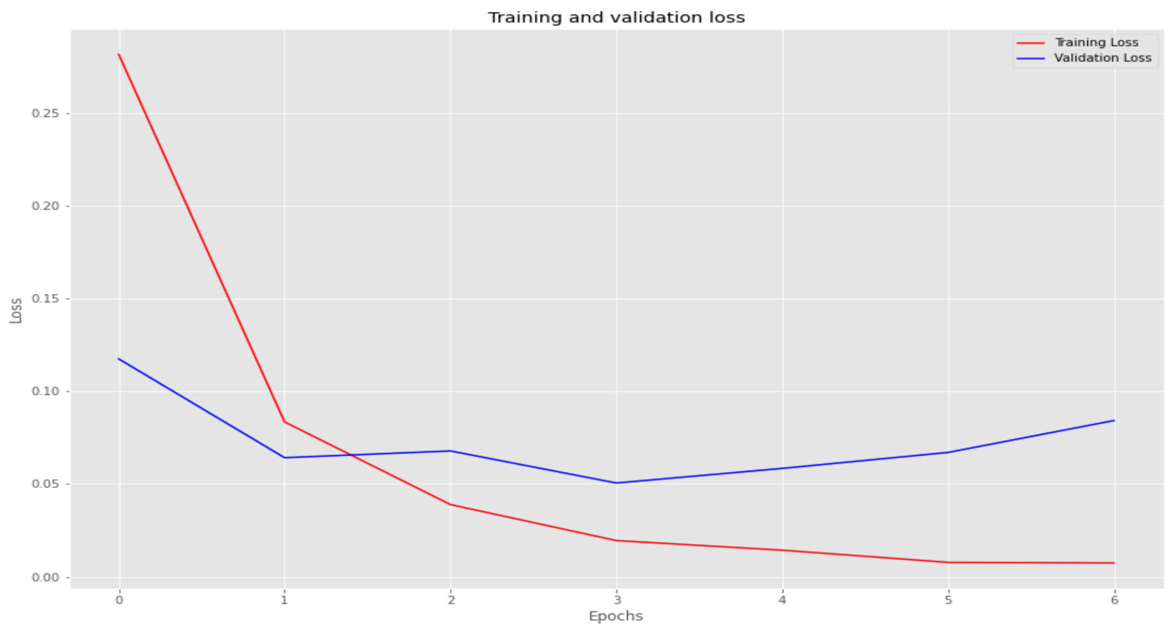


Figure 25. Confusion Matrix for RNN

The RNN classifier performed excellently with an accuracy score of 98.65%. The classifier, however, did require a significant time to set up and long training and computation time. Its performance is credited to the large amount of time experimenting with different neural network layers, loss and activation functions and tuning parameters.



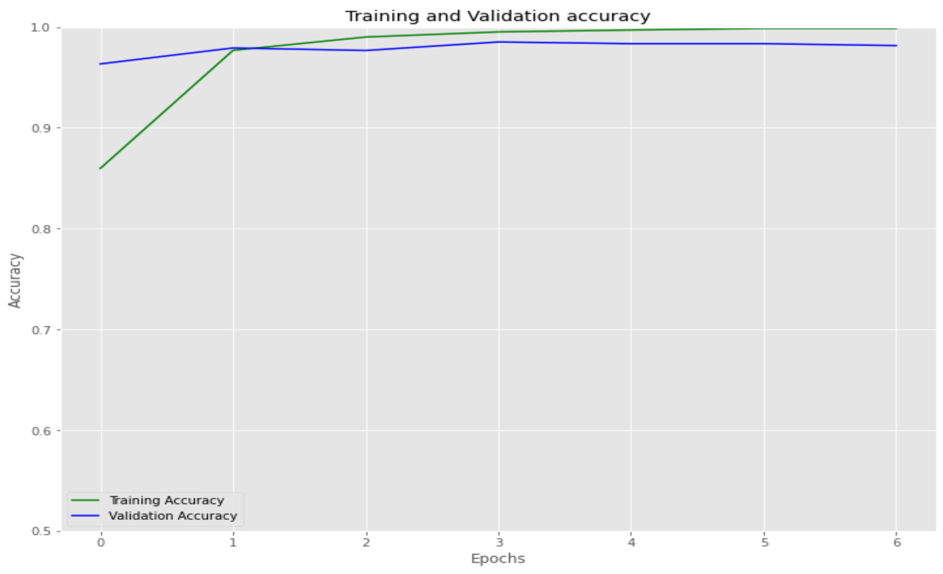


Figure 26. Visualization of RNN training

Conclusion

For this project, the problem of fake news detection was tackled by proposing different machine learning models and natural language processing technique to differentiate between real and fake news. The five machine learning models proposed were the naïve Bayes classifier, logistic regression classifier, random forest classifier, convolutional neural networks and recurrent neural networks.

After evaluation and comparison, the recurrent neural network classifier obtained the highest accuracy of 98.65% as shown in Figure 20. However, recurrent neural networks require a longer set up, training and computation time compared to non neural network classifiers such as the random forest classifier which obtained a decent accuracy of 86.75%, also shown in Figure 20. If resources and time are not of concern, I would recommend implementing recurrent neural networks for fake news detection. However, if time is crucial, implementing a random forest classifier would be the best option without sacrificing too much performance.

References

Afshine Amidi, S. A., n.d. *CS 230 - Recurrent Neural Networks Cheatsheet.* [Online]
Available at: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Anubhav Mishra, S. S., 2021. *Impact of fake news on social image perceptions and consumers' behavoral intentions,* s.l.: Journal of Consumer Marketing.

Bisaillon, C., 2020. *Fake and real news dataset.* [Online]
Available at: https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset?select=Fake.csv

Jamal Abdul Nasir, O. S. K. I. V., 2021. *Fake news detection: A hybrid CNN-RNN based deep learning approach.* [Online]
Available at: https://www.sciencedirect.com/science/article/pii/S2667096820300070

Kaliyar, R. K., 2018. *Fake News Detection Using A Deep Neural Network.* [Online]
Available at: https://ieeexplore.ieee.org/document/8777343

Keras, n.d. *Keras layers API.* [Online]
Available at: https://keras.io/api/layers/
[Accessed 2023].

N. Smitha, R. B., 2020. *Performance Comparison of Machine Learning Classifiers for Fake News Detection.* [Online]
Available at: https://ieeexplore.ieee.org/document/9183072

Robert B.Michael, B. O., 2021. *SpringerOpen.* [Online]
Available at: https://cognitiveresearchjournal.springeropen.com/articles/10.1186/s41235-021-00278-1
[Accessed 4 January 2023].

scikit-learn, n.d. *sklearn.metrics.accuracy_score.* [Online]
Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

scikit-learn, n.d. *sklearn.metrics.precision_recall_fscore_support.* [Online]
Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

Shevon Desai, J. A. O., 2022. *What is "Fake News"?.* [Online]
Available at: https://guides.lib.umich.edu/fakenews
[Accessed 4 January 2023].

TensorFlow, n.d. *tf.keras.Model.* [Online]
Available at: https://www.tensorflow.org/api_docs/python/tf/keras/Model

VanderPlas, J., 2016. Decision Trees and Random Forests. In: *Python Data Science Handbook.* s.l.:O'Reilly Media, Inc..

VanderPlas, J., 2016. Linear Regression. In: *Python Data Science Handbook.* s.l.:O'Reilly Media, Inc..

VanderPlas, J., 2016. Naive Bayes Classification. In: *Python Data Science Handbook.* s.l.:O'Reilly Media, Inc..