COMP42415: Text Mining and Language Analytics
Summative Assignment | Student No. 001059181

Machine Learning Models to Dectect Reliability of News Articles

1. Dataset

The dataset "fake_news_dataset.csv" has 20800 news articles from a variety of sources. It has 5 columns with "id" (numbering), "title" (title of the article), "author" (information about authors), "text" (body of the article), and "label" (reliability of the article i.e. 0 is reliable (real news) and 1 is unreliable (fake news).

Suitability. As the task is to develop and evaluate machine learning models to predict the reliability of a news article, a dataset that has data on whether a news article is reliable or not is necessary to train the models. In this exercise, the "fake_news_dataset.csv" has information on the title, the author, the text as well as a label that indicates reliability. Therefore, the dataset provides useful information suitable to build such models.

Problems. First, are some missing data in this dataset. Thus, the dataset requires cleaning prior to using it for training. Second, there are concerns of representation because the majority of the dataset contains English articles. This means that the model trained will have language constraints in its ability to predict the reliability of articles in other languages. Third, no information is provided regarding the process of collecting the data. Therefore, it is not possible to assess potential biases, exclusions that may have taken place when collecting the data (for example, possible geographical representation).

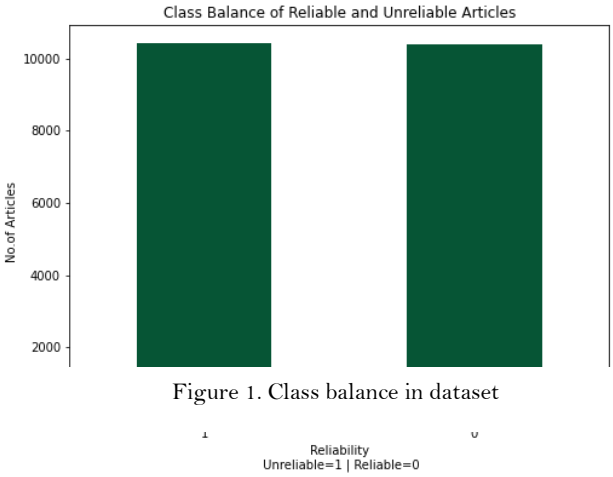

Figure 1. Class balance in dataset

Class balance. It is best if the class distribution in the dataset is balanced as this provides sufficient data to train the model more effectively. In the dataset there are 10,413 unreliable (fake news) articles and 10,387 reliable (real news) articles. Therefore, the number of reliable and unreliable classes are almost equal in the dataset.

2. Data Preparation

Splitting dataset into training and test data. The dataset was split into training and testing samples following the given 70/30 split using the train_test_split function (from sklearn.model_selection). Presumably because of the class balance in the dataset, the split resulted in a somewhat balanced class balance in both the training and test data. For instance, when split using random state(42), the class balance was as follows:

	Training Data	Testing Data
Reliable	7239	3148
Unreliable	7321	3092
Total Samples	14560	6240

Table 1. Class balance when split into training and test sets

To get as close a class balance as possible, the stratify argument was used. This provided an even closer class balance as seen below:

	Training Data	Testing Data
Reliable	7271	3116
Unreliable	7289	3124
Total Samples	14560	6240

Table 2. Class balance when split into training and test sets using "stratify"

The datasets were prepared for data cleaning as follows. The title and text columns in each set was concatenated. The id and author columns were discarded as they provide minimal insight during the training process. The concatenated columns (now titled "alltext") were converted to string datatype to aid pre-processing that requires the data to be of a string type. Missing values were identified and removed). This was carried out subsequent to concatenation as, if not, due to there being missing values in the "author" column (which was discarded), more datapoints than required would have been removed in this process. This resulted in a training dataset with 14143 datapoints and a test dataset of 6060 datapoints. Both datasets were converted to lists to facilitate pre-processing.

The presence of uppercase and lowercase letters, punctuation, and stopwords that can cause noise when training a model were noted. Therefore, the following steps were taken to clean the model in order to remove noise and normalise the text. The training and test datasets were tokenised (to apply cleaning more uniformly to all the words), converted to lowercase, and stored in lists. A for loop was used to remove the following: hyphens and dots (to normalise abbrieviations such as N.F.L to NFL); all punctuation, symbols, and numbers using regular expressions (regex); and all stopwords (using the stopwords from the nltk.corpus). The datasets were stored as dataframes and all texts with less than 50 characters were removed in order to remove noise, and as such small texts will not contribute significantly to training the models. All articles were also truncated to 200 words (for performance reasons).

3. Text Representation

The following text representation methods were used.

Naïve Bayes and KNN Models. Both models handle the text as "bags of words" as it converts text to numerical feature vectors.[1] Therefore, The text ("alltext column") in the training and test data were converted to lists for the Naïve Bayes Classifier and the KNN Classifiers. Thereafter, the input was converted to TF-IDF vectors using a pipeline.

[1] Scikit Learn "Working with Text data". https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

CNN Model. The testvectors.csv word embeddings were used to prepare document embeddings as input data for the CNN model. Word embeddings use cosine similarity to measure semantic similarity, and is not as computationally intensive as BOW.

LSTM Model. A training sample of 9808 texts and a test sample of 4203 texts were created (70-30%). Thereafter, a vocabulary was created using the training set. In creating the vocabulary, words that occur at least one time were used, and the no.of unique tokens in the vocabulary were 69856. An embedding layer that converts a text to a list of embeddings of a specific size was included as a layer in the model itself.

4. Machine Learning Models

To build the Naïve Bayes Model, the input was converted to TF-IDF vectors using a pipeline, and thereafter, a Multinomial Naïve Bayes is used.

To build the K-Nearest Neighbors Models, the input was converted to TF-IDF vectors, and KNeighborsClassifier(n_neighbors=_) was set to 3 and 7 respectively for each model.

To build the Long Short-Term Memory (LSTM) Model, two iterators were created for the training and testing (validation) data. A class FakeNewsNet was used. This LSTM Model takes the vocabulary size as the length of the vocabulary built during text representation, 10 embedding dimensions, 16 features in the hidden state of the LSTM (batch size), 2 recurrent layers for the RNN. It is also bidirectional with a dropout layer of 0.2 (i.e. 20% dropout) and 1 output layer. The output layer has 1 neuron which uses a Sigmoid activation function (by using a Sigmoid function with one neuron, it is possible to get an output of 0 or 1 as the Sigmoid function provides an output between 0 and 1, which is then rounded to the nearest integer to identify the predicted class). The model uses the Adam optimiser and Binary Cross Entropy between the target and the output as the loss. Figure 2 indicates that while the training loss reduces, the validation loss somewhat plateaus around 2 epochs onwards. Similarly, the validation accuracy too plateaus after 2 epochs.

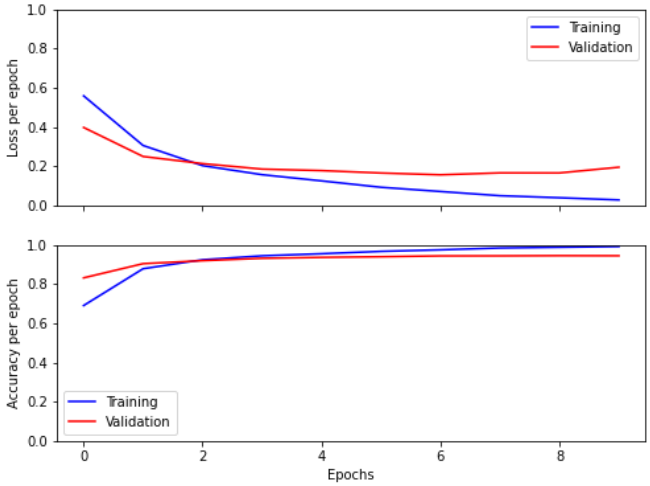

Figure 2. Loss and Accuracy values of LSTM over 10 epochs

To build the CNN model,[2] a one-dimensional (as input is text) convolutional layer taking 300 inputs and 1 kernel, and a second 1 dimensional layer taking 32 inputs and 1 kernel were defined. The model was created for an embedding the size k=300.

5. Experimental Results

[2] It was not possible to fully implement the CNN model. Therefore, the relevant code relating to the CNN model is extracted and included at the end of the python notebook.

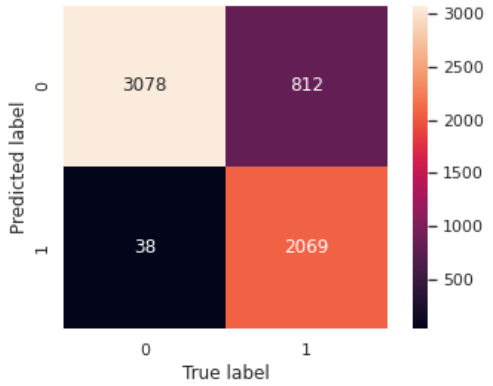

Figure 3. Confusion Matrix: Naïve Bayes Classifier

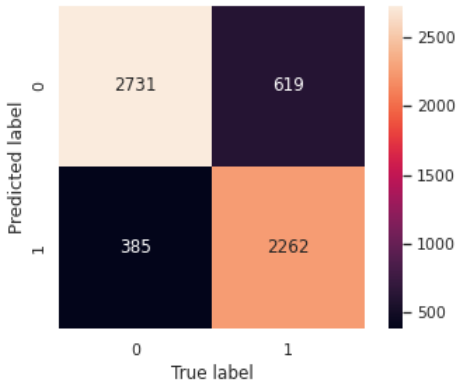

Figure 4. Confusion Matrix: KNN Classifier (K=3)

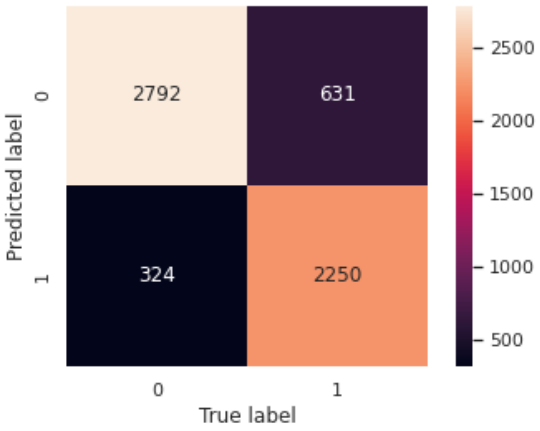

Figure 5. Confusion Matrix: KNN Classifier (K=7)

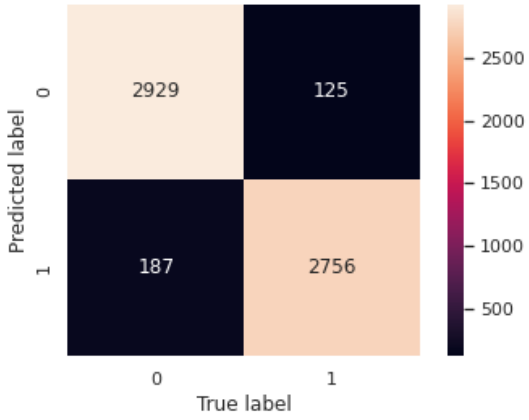

Figure 6. Confusion Matrix: LSTM

	Accuracy	F1-score	Precision	Recall
Model 1: Naïve Bayes Model				
Overall (total)	0.858262 (86%)			
0		0.88	0.79	0.99
1		0.83	0.98	0.72
Model 2: KNN Model (K=3)				
Overall (total)	0.832583 (0.83%)			
0		0.84	0.82	0.88
1		0.82	0.85	0.79
Model 3: KNN Model (K=7)				
Overall (total)	0.840754 (84%)			
0		0.85	0.82	0.90
1		0.82	0.87	0.78
Model 4: LSTM Model				
Overall (total)	0.947974 (95%)	0.947931	0.947765	0.948300
0		0.95	0.96	0.94
1		0.95	0.94	0.96

Table 3. Performance Statistics of the Trained Models

As seen in Table 3, the Naïve Bayes classifier predicts with 86% accuracy. However, the KNN classifiers have a slightly lesser prediction accuracy: when KNN=3, the accuracy is 83% whereas when KNN=7 increases slightly to 84%. The LSTM has the highest accuracy at 95% (at 6 epochs). Therefore, the LSTM appears to outperform the other models. The LSTM also records the highest precision and F1-scores of all the models.

The recall statistics (i.e. proportion of actual positives that were identified correctly) and the F1 score (the mean of precision and recall) provide insight into how the models managed to predict the classes. For instance, it appears that all the models performed consistently better in predicting when an article is reliable. While the Naïve Bayes model has the highest recall statistics when predicting reliable news, the LSTM model is the best to predict both reliable and unreliable news as it predicts with beteen 94-96% accuracy.

6. Discussion

While the Naïve Bayes and KNN models are capable of faster processing, the LSTM Model takes a longer time to train over several epochs when the texts are larger. Therefore, for performance reasons, the length of each text was reduced to 200 words. This reduction was not applied solely to the dataset used to train the LSTM as performance evaluation between all models would then be skewed. Based on experiments run, there is possibility for a slight increase in accuracy if the LSTM model is run on the full dataset available (without truncating each text to only 200 words).

Nevertheless, all models predict with an accuracy of over 80%. The script also includes a defined predict function to predict whether an article is reliable or not.