SEP 788/789 – Deep Learning and Neural Networks

Fake News Detection

Final Report

Instructor: Hamidreza Mahyar

TA: Amir Umani

Name	Student Number	Email
Ruiqiao Wang	**400353790**	wangr228@mcmaster.ca
Siqi Zhao	**400397742**	Zhaos98@mcmaster.ca
Zhuangyuan (Bob) Shen	**400354488**	shenz57@mcmaster.ca

Table of Contents

Problem Statement and Understanding

Project Background and Understanding

Fake news consists of false information or deliberate made-up "facts" spread via means like traditional news networks, newspapers or nowadays more so on social media. Whether the fake news is artificially made or generated by an AI program, both have negative consequences. In the current internet age, fake news is all over the place and they could be quite deceiving and hard to differentiate from the real ones at times. Facebook had long been suffering from the consequences of fake news, not only was it accused of impacting the US presidential elections, it was also fined by the German government by 2 million euros in 2019. According to a Pew Research Center poll from December 2016, 23% of adults in the United States have shared fake news with friends and others, whether consciously or unwittingly. On a societal level, fake news could sabotage the stability of the society, causing tensions and distrust among people and groups. On an individual level, a piece of health-related fake news could have false information that does harm to an individual's health rather than helping it. A recent (and still ongoing) example is the large amount of fake news related to COVID-19, many people poisoned themselves by taking bleach since they were informed by the fake news that it could protect them from COVID-19.

One reason that fake news plagues is that it is very time-consuming to classify fake news from real news if the job were to be performed by humans, not to mention the associated accuracy and cost of doing so. On the other hand, even with the help of an AI program, censoring all the fake news is impossible because the speed of information generated grows exponentially, especially with the help of social media, people could post almost whatever (many of which are hearsay or deliberate false information) on it anytime and most of the posts went uncensored. Yet the speed of even using the best AI programs to scrutinize the information does not match. Additionally, the problems that deals with human language, known as Natural Language Processing (NLP) are one of the most challenging subsets of Machine Learning. The reason being while it is easy to represent an image in terms of a matrix, it is not as easy to encode texts as a number or a vector. Not to mention that NLP often needs to explore the way humans communicate and our consciousness.

The project team is tasked to leverage the data in the **Fake News dataset** with over 7k pieces of news belonging to one of the two classes - real or fake, and the main feature of each data point is **Text** column in the csv file, then make predictions about the authenticity of the news (Label column in the csv file). We will use a variety of machine learning and deep learning algorithms to build models and train classifiers to predict the results. By implementing these algorithms, compare the pros and cons of each algorithm, and evaluate the effect of each algorithm.

In conclusion, for this NLP binary classification problem, first preprocess the data set (see the second part of this report for details), and then use different machine learning and deep learning algorithms (see the third part of this report for details) to train the classifier. Make predictions and evaluations, and finally analyze and compare different algorithms, to get a deeper understanding.

The Project Outcomes:

- Pre-process the data to remove stop words. Stop words are the most occurring words in the language. It's necessary to filter that out first.
- Evaluate various algorithms which can affect best outcome
- Train a model to predict the likelihood of real news.

Project Challenge

- **Huge dimension of the input features - Curse of dimensionality**

  Usually, the length of the news is relatively long, which may lead to a larger dimension of the input features, as known as curse of dimensionality. That means if the dimensionality of the input is so big it needs more data to train the model. Also need good compute power to solve the problem, and may also lead to overfitting

- **Small amount of data – Only 6336 pieces of data points**

  For deep learning and machine learning, a large amount of data is usually required for training to achieve good results, and more than six thousand rows of data are relatively small. And the text dimension is too large also requires a lot of data, making good use of this limited data is a big challenge for this project

Prerequisites & Version

- **Scikit-Learn -- 0.24.2**
- **TensorFlow -- 2.0.0**
- **Keras -- 2.3.1**
- **NLTK -- 3.6.5**
- **gensim -- 3.8.3**
- **H5py -- 2.10.0**

GitHub Repo

The codes can be found in below GitHub repo:
https://github.com/SLAM-CROC/Fake-news-detection
There are Jupyter Notebook Version and Python Project Version Code. Please feel free to check as your preference.

Proposed Approach Outline

Below is the structure of our project proposed approach, each rectangle is implemented by using python functions. The circle is the model we trained, it contains the phase of data preprocessing and different data preparation stage for Machine Learning Model and Deep Learning Model. First, data is cleaned by a group of function, and then drop the news tittle, next preform different word embedding for ML and DL model, finally is training the model and find out the best result model and save it to make prediction.

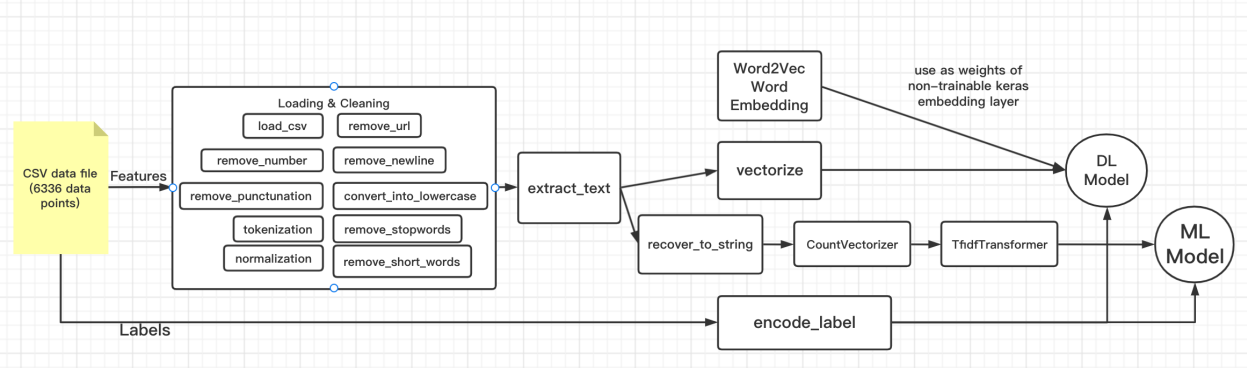

Fig1. Proposed Approach Outline

Data pre-processing and choice of approach to implement

Dataset Visualization

Data Analysis, by running the **visualizeData.py** file, the histogram of top ten most frequent words in real news and fake news will be displayed. To facilitate the understanding of the dataset.

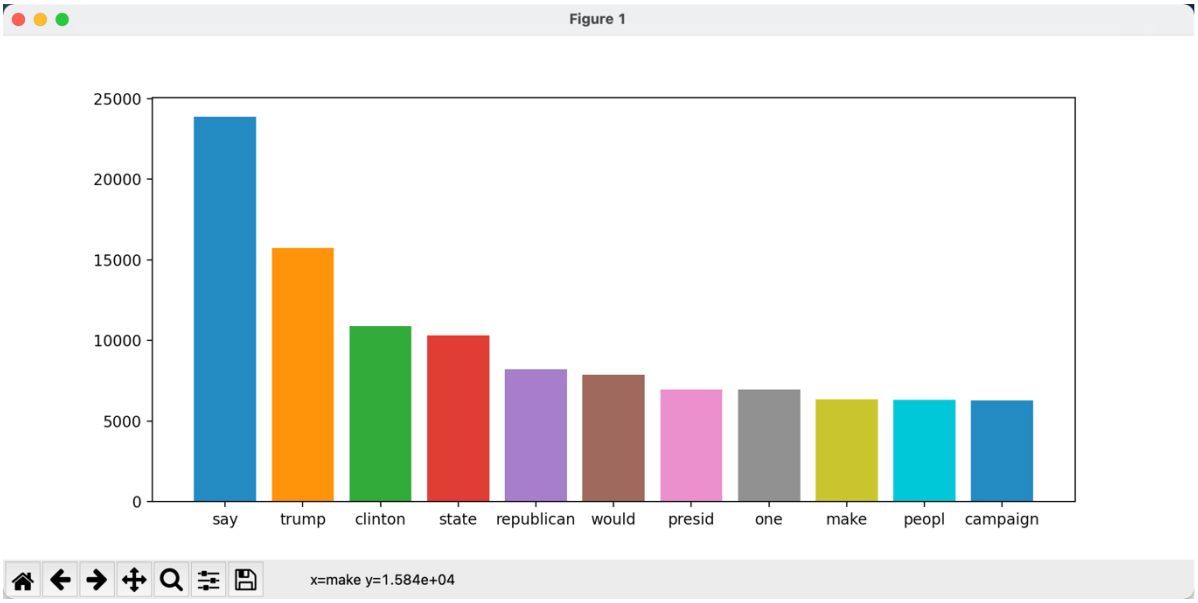

Fig2. Word frequency distribution histogram for REAL news

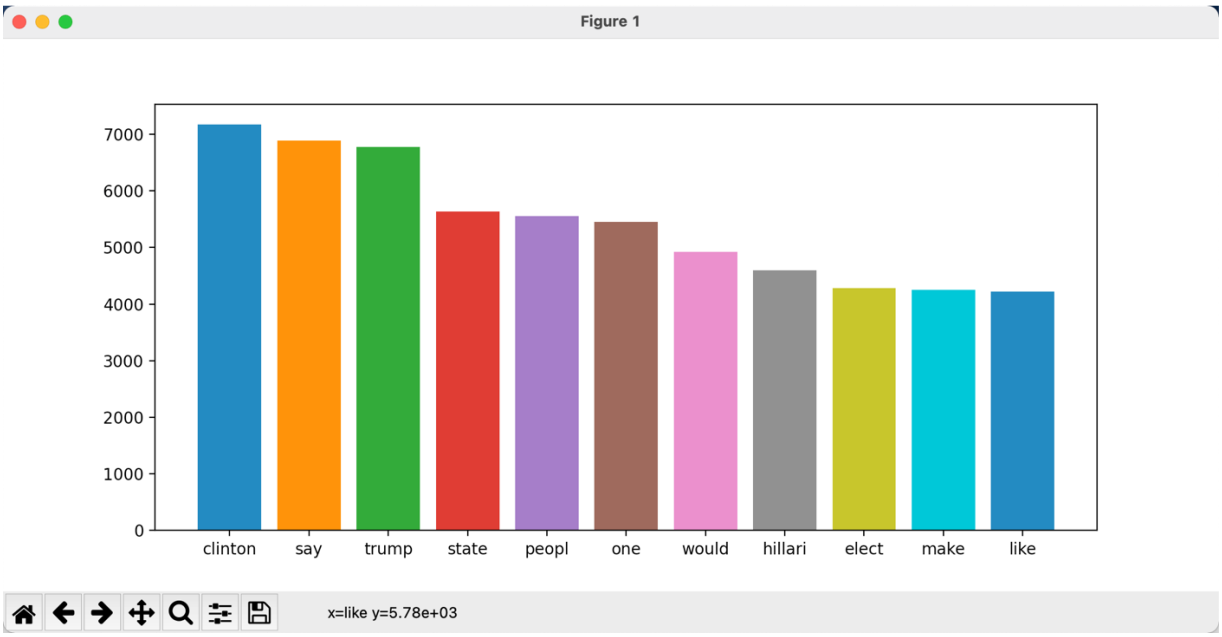

Fig3. Word frequency distribution histogram for FAKE news

Data Loading & Data Cleaning

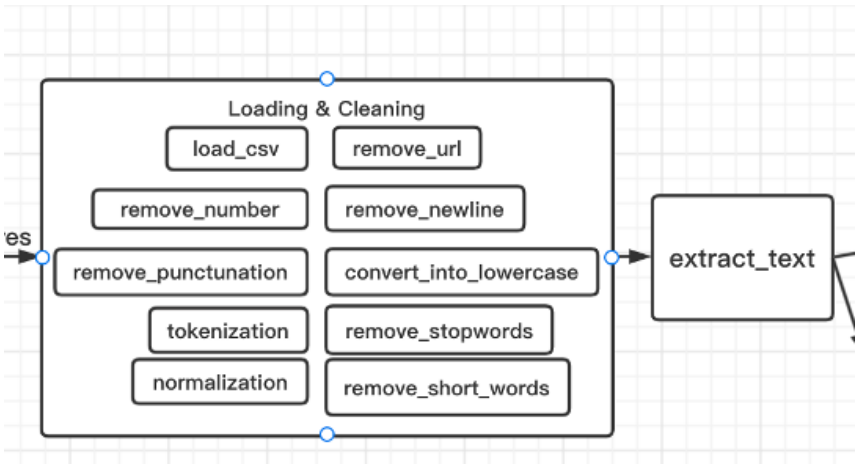

Fig4. Data Pre-Processing Module

For data pre-processing, we used regular expressions, Keras, and the NLTK language processing library to clean data and gensim to use Word2Vec model. And we use Scikit-Learn to perform TF-IDF for Machine Learning Model.

We used a variety of different methods and steps. We encapsulated these methods in the **preprocessing.py** and manipulated the data by calling different methods. We can add or ignore some of these methods according to the different needs of the later training model stage, and we can also compare the impact of different data pre-processing methods on the prediction effect. And figure out which steps or methods are better.

We have implemented the following methods in **preprocessing.py**

clean_data:

1. Load dataset from csv file and delete the missing data points
2. Remove url in the text
3. Remove newline signals from text
4. Remove numbers and punctuations from text
5. Convert all letters into lowercase
6. Tokenization text
7. Remove stop words from text
8. Normalization: including stemming and Lemmatization
9. Remove words whose length are equal or less than 2

extract_text:

Drop news tittle, we will only use news content (text column in the CSV file) as input feature to train the model

encode_labels:

Encode Labels, encode FAKE and Real to 0 and 1

recover_to_string:

Because the data needs to be cleaned, we need to split the text into individual words (Tokenization). For the subsequent training of the machine learning model, we need to restore them to the original text state

split_data:

Split data into training set and test set, we implement it by ourselves not use Scikit-learn Build-in method.

Choice of approach to implement

• **ML algorithm**

- Naive Bayes, K-Nearest Neighbours, Support Vector Machine, Logistic Regression, Decision Tree

• **DL algorithm**

- MLP, CNN, LSTM

Explanation of The Method(s) and Competing Approaches

Model Training

For model training, we used scikit-learn Machine learning framework and Keras deep learning framework to develop the model. We defined different models in **models.py**. And run **ml_training.py** & **dl_training.py** to train the model. ML & DL model training code can also be

found in the Notebook: **Fake news detection project notebook – Machine Learning.ipynb** & **Fake news detection project notebook - Deep Learning.ipynb**

Machine Learning Algorithm

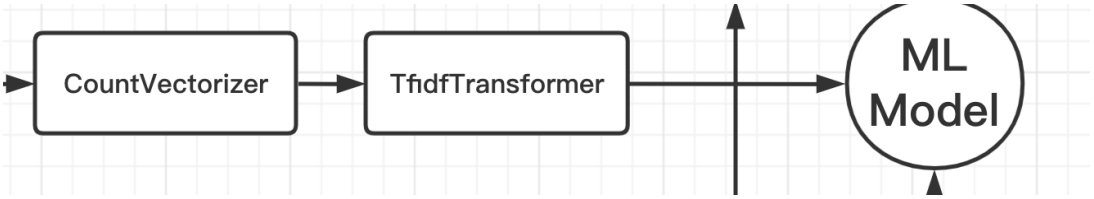

Fig5. Machine Learning Module

Based on the previous data preprocessing, we used the TF-IDF method to achieve the extraction of text features before using Machine Learning Algorithm.

TF-IDF score represents the relative importance of a term in the document and the entire corpus. TF-IDF score is composed by two terms: the first computes the normalized Term Frequency (TF), the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
IDF(t) = log e(Total number of documents / Number of documents with term t in it)

TF-IDF Vectors can be generated at different levels of input tokens (words, characters, n-grams)

Logistic Regression

Logistic Regression is a linear classifier that predicts class probabilities.

Instead of tuning C parameter manually, we can use an estimator which is **LogisticRegressionCV**. We specify the number of cross validation folds **cv=5** to tune this hyperparameter. The measurement of the model is the accuracy of the classification. By setting **n_jobs=-1**, we dedicate all the CPU cores to solve the problem. We maximize the number of iterations of the optimization algorithm.

By calling the **lr_model**, which is in **models,** the model will be trained, and the model will make predictions on the test set, and finally show the accuracy of the model on the test set.

```
models.lr_model(x_train, y_train, x_test, y_test)
```

Naive Bayes Classifier

Naive Bayes Classifier is a widely used generative model for document categorization which is theoretically based on Bayes theorem. Bag-of-words lays the foundation for the basic form of Naive Bayes Classifier. The Naive Bayes algorithm can be described:

$$P(c|d) = (P(d|c)P(c))/P(d)$$

where d denotes the document and c indicates classes and P gives the posterior probability.

In addition, there are several types of Naïve Bayes algorithm, those are Gaussian, Multinomial and Bernoulli. In this project I will be using Multinomial Naïve Bayes since it's the best one to be implemented in this text classification task due to its ability to maintain the number of word occurrences in each document. Fortunately, Sklearn provides an easy-to-implement object called MultinomialNB(), so that we don't have to code the algorithm from scratch. where d denotes the document and c indicates classes and P gives the posterior probability.

By calling the **nb_model**, the model will be trained, and the model will make predictions on the test set, and finally show the accuracy of the model on the test set.

```
models.nb_model(x_train, y_train, x_test, y_test)
```

K-Nearest Neighbours

KNN is an unsupervised machine learning model where a dependent variable is not required to predict the outcome on a specific data. We provide enough training data to the model and let it decide to which particular neighborhood a data point belongs. KNN model estimates the distance of a new data point to its nearest neighbors, and the value of K estimates the majority of its neighbors' votes; if the value of K is 1, then the new data point is assigned to a class which has the nearest distance. In this project, we choose the number of K to be 5.

By calling **knn_model**, the model will be trained, and the model will make predictions on the test set, and finally show the accuracy of the model on the test set.

```
models.knn_model(x_train, y_train, x_test, y_test)
```

Support Vector Machine

Support Vector Machine (SVM) is the supervised learning algorithm for classification and regression. The main objective of SVM is to find a maximal separating hyperplane between vectors that belong to a category and vectors that do not belong to it. The maximum distance between data points helps to classify future data points with more confidence. Here we use a linear kernel

By calling **svm_model**, the model will be trained, and the model will make predictions on the test set, and finally show the accuracy of the model on the test set.

```
models.svm_model(x_train, y_train, x_test, y_test)
```

Decision Tree

A decision tree is a tree-like structure that classifies the data by tracing the path from root to leaf. The attribute with the largest information gain is chosen as the parent node and the subsequent attributes are assigned to the child node.

In a decision tree method, each path starting from the root is representing a sequence of data splitting until a Boolean outcome is reached at the leaf node. In real-life practice, each path in the decision tree is a decision rule that can be easily translated into human languages or programming languages. The capability of decision support is not new in the studies of data mining and machine learning. Technically there are multiple methods capable of supporting decision makings. Nonetheless, a decision tree might be preferred because of its clearness and understandability.

By calling **decision_tree_model**, the model will be trained, and the model will make predictions on the test set, and finally show the accuracy of the model on the test set.

```
models.decision_tree_model(x_train, y_train, x_test, y_test
```

Deep Learning Algorithm

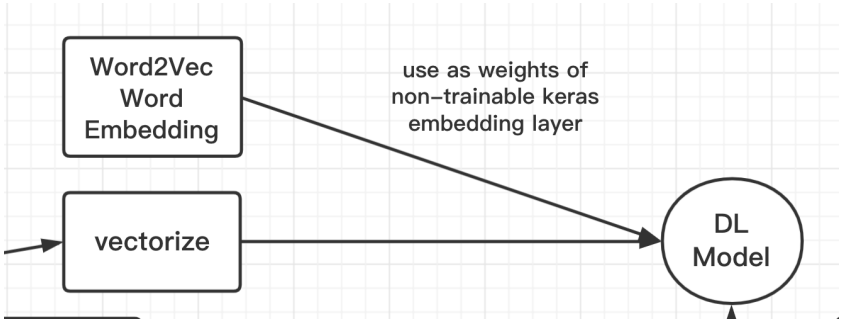

Fig6. Deep Learning Module

Based on the previous data preprocessing, using keras' built-in Tokenizer to use Unique Numbers to Vectorize – represent each word by its index in the words dictionary. And then we are padding the text, make the length of each news the same, if the length is too long, remove the extra part, if the length is not long enough, use 0 to fill in. By exploration, we plot this diagram. This diagram shows the length of the text after data cleaning. We found that most of the text is below 1000, so we choose 1000 for text padding to ensure that the data contains most of the information at the same time saving training time.

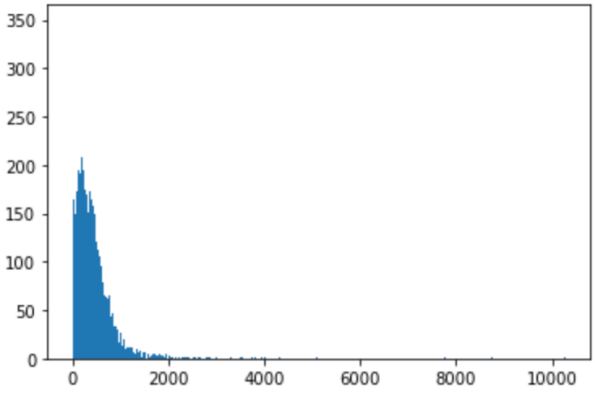

Fig7. Text length distribution graph

Also, we performed word embedding by using Gensim Word2Vec model, getting embedding vectors from word2vec and using its as weights of non-trainable keras embedding layer

And we used Early Stop to prevent over fitting.

MLP

For text classification, the use of MLP will make the model ignore the timing information of the data, but MLP may still be a reliable method, and to compare different deep learning models, MLP is also very valuable for reference, so we also include it in.

By calling the method **mlp_model,** which is in **models**, the model will be started to train, and related log will be showed to us. By analyzing the training log, we can find the problem of our model for example whether it is underfit or overfit and improve the model.

```
models.mlp_model(x_train, y_train)
```

We have following MLP topology.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1000, 100)         4255200
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000, 100)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 100000)            0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               51200512
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513
=================================================================
Total params: 55,456,225
Trainable params: 51,201,025
Non-trainable params: 4,255,200
```

Fig8. MLP topology

CNN

Same as MLP, using CNN to do text classification also will lead to missing time information of data. But CNN also have the ability to solve the text classification task and valuable to implement, and good to compare with RNN classification model.

By calling the method **cnn_model** in **models,** the model will be started to train, and related log will be showed to us. By analyzing the log we can also improve it.

```
models.cnn_model(x_train,y_train)
```

We have following CNN topology

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 1000, 100)         4255200
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1000, 256)         77056
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 334, 256)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 334, 32)           24608
_________________________________________________________________
flatten_2 (Flatten)          (None, 10688)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 10688)             0
_________________________________________________________________
batch_normalization_1 (Batch (None, 10688)             42752
_________________________________________________________________
dense_4 (Dense)              (None, 256)               2736384
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 257
=================================================================
Total params: 7,136,257
Trainable params: 2,859,681
Non-trainable params: 4,276,576
```

Fig9. CNN topology

LSTM

For processing natural language, RNN has obvious advantages compared with MLP and CNN. RNN are designed for modeling sequences and are capable of remembering past information and processing new events accordingly, which is a clear advantage when working with sequence data. LSTM as a special RNN, LSTM can solve the vanishing gradient challenge. So, it's necessary to try LSTM on our text classification problem.

By calling the method **lstm_model** in **models**, the model will be started to train, and related log will be showed to us. By analyzing the log we can also improve it.

```
models.lstm_model(x_train, y_train)
```

We have following LSTM topology

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 100)         4255200
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 4,372,577
Trainable params: 117,377
Non-trainable params: 4,255,200
```

Fig10. LSTM topology

Metrics/Competing Approach

For our classification problems, the combination of the result predicted by the model and the true category of the sample can be divided into **true positive (TP), false positive (FP), false negative (FN)** and **true negative (TN).**

Based on **TP, FP, FN, TN**, we can also measure and evaluate our training result by

Precision = $\frac{TP}{TP+FP}$ Proportion of the samples that the model judges to be positive

Recall = $\frac{TP}{TP+FN}$ Indicates how many positive cases I judged in all positive cases

F1-Sorce = $\frac{2\times Precision\times Recall}{Precision+Recall}$

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

Confusion matrix:

Actual / Predict	Fake	Real
Fake	**TN**	**FN**
Real	**FP**	**TP**

Experimental Evaluation and Results

Result and Analysis

Through experiments, we got this table. It contains various models and the accuracy rates obtained by these models.

Model	Accuracy
Naive Bayes	**0.8333333333333334**
K-Nearest Neighbours	**0.8698412698412699**
Support Vector Machines	**0.9333333333333333**
Logistic Regression	**0.9317460317460318**
Decision Tree	**0.8047619047619048**
MLP	**0.9009523582458496**
CNN	**0.8876190185546875**
LSTM	**0.9141269612312317**

Table1. Experimental Result

Through analysis, we can conclude that in the machine learning model, support vector machines and logistic regression have better accuracy, and in the deep learning model, LSTM has a higher accuracy. And it can be found that support vector machines and logistic regression models have better results than LSTM. This may be the result of using different methods in the Word Embedding stage. This proves that TF-IDF has a better effect than Word2Vec.

Prediction

Finally, we save the best model as a **.h5** file, such as the **model_LSTM_test.h5 file** in the code folder. In the **predict.py** script, we load the model and use the model to make predictions on new data. The results are shown in the figure below.

```
Loaded csv file, and there are 36 blank text have been removed from dataset
6299 data points in total
There are 295 url have been removed from text
Newline symbols have been removed from text
Numbers have been removed from text
Punctuations have been removed from text
All text have been converted into lowercase
Preformed tokenization
Stopwords have been removed
Text has been normalized
Short words have been removed
News Tittle: You Can Smell Hillary's Fear
Likelihood of REAL: [[0.01471508]]
['FAKE']
```

Fig11. Prediction result

We can see that the program displays the logs in the data processing stage. Finally, for **News: You Can Smell Hillary's Fear**, the probability of predicting that the news is REAL is given, which is **0.01471508**, so it is a fake news.

Difficulties and Solutions

Huge dimension of the input features, lead to overfitting

As mentioned in the previous challenge section, the input features have huge dimensionality, at the beginning of the project. we didn't do the unsurprised word embedding. The validation accuracy of each model is only round 70%. But the training accuracy is already near 100%. Because we do not perform word embedding and cannot capture the similarities between two words. after we use word2vec in gensim model or TfidfTransformer () in Scikit-Learn, we create the word vector for every word, and we can find the similar word.

```
In [30]: w2v_model.wv.most_similar("say")

Out[30]: [('tell', 0.6622174978256226),
          ('ask', 0.5990400314331055),
          ('agre', 0.5693711638450623),
          ('batric', 0.5014475584030151),
          ('acknowledg', 0.49850720167160034),
          ('admit', 0.49767521023750305),
          ('speak', 0.49394840002059937),
          ('respond', 0.4900415241718292),
          ('believ', 0.4846227169036865),
          ('insist', 0.48254138231277466)]
```

Fig12. Words similar to say found through the Word2Vec model

Lack of data

For the problem of lack of data, we use cross-validation to make good use of these limited data

Improvement or Future Work

In order to solve the problem of too little data, we can also improve our project in the following two aspects:

NLP Data Augmentation
By NLP Data Augmentation, we can increase the amount of training data and improve the generalization ability of the model and also add noise data to improve the robustness of the model. ICLR 2019 workshop paper [EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks] introduced several NLP data enhancement techniques, which are worthy of our study and use in our projects.

Train More Model
Trying more models, such as Transformer, Google Bert, may enable us to get better results