SEP 788/789 – Deep Learning and Neural Networks

Fake News Detection

Report 3

Explanation of the methods and competing approaches

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

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**. The code structure can be seen by following screenshot, the other python file has been introduced in the preprocessing section. Full code can be found on project GitHub repository:

https://github.com/SLAM-CROC/Fake-news-detection

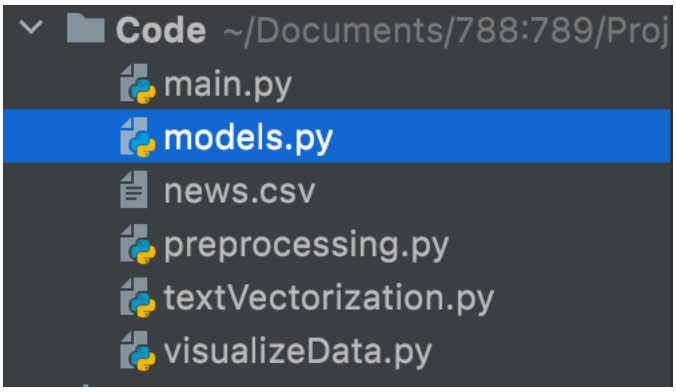

Fig1. Code structure including Data Preprocessing and Model training

1. Machine Learning Algorithm

In the last stage, data preprocessing, we use the Keras and NLTK data preprocessing libraries to clean data and create a dictionary for the words in the text, and vectorize the text according to the word dictionary. Finally padding or cutting the text to the same length.

For example:

"I am a teacher" → [1, 2, 3, 4]
"I am a student" → [1, 2, 3, 5]

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)

1.1. Logistic Regression

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)
```

In the next stage, we will evaluate the model and compare different models

1.2. Naive Bayes Classifier

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)
```

1.3. 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)
```

1.4. 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)
```

1.5. Decision Tree

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
```

2. Deep Learning Algorithm

Based on the previous data preprocessing, by keras' built-in Tokenizer, we set the dictionary size to 2000, and padding all text to a length of 1000. And before sending the data to the neural network for training, we all add an embedding layer to them. Turns positive integers (indexes) into dense vectors of fixed size.

2.1 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)
```

For now, we have following MLP topology, and that will might be change in the next dev phase after analysing and evaluating.

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 1000, 32)          64000
_________________________________________________________________
dropout_7 (Dropout)          (None, 1000, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 32000)             0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               8192256
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 514
=================================================================
Total params: 8,256,770
Trainable params: 8,256,770
Non-trainable params: 0
```

Fig2. MLP initial topology

2.2 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)
```

For now, we have following CNN topology, and that will might be change in the next dev phase after analysing and evaluating and using hyperparameter tuning.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 1000, 32)          64000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1000, 256)         24832
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 334, 256)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 334, 32)           24608
_________________________________________________________________
flatten_1 (Flatten)          (None, 10688)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 10688)             0
_________________________________________________________________
batch_normalization_1 (Batch (None, 10688)             42752
_________________________________________________________________
dense_3 (Dense)              (None, 256)               2736384
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 514
=================================================================
Total params: 2,893,090
Trainable params: 2,871,714
Non-trainable params: 21,376
```

Fig3. CNN initial topology

2.3 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)
```

For now, we have following LSTM topology, and that will might be change in the next dev phase after analysing and evaluating and using hyperparameter tuning.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 1000, 64)          128000
_________________________________________________________________
lstm_6 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense_7 (Dense)              (None, 256)               16640
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 514
_________________________________________________________________
activation_4 (Activation)    (None, 2)                 0
=================================================================
Total params: 178,178
Trainable params: 178,178
Non-trainable params: 0
```

Fig3. LSTM initial 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**