Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the DCU Academic Integrity and Plagiarism at

https://www.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v2.pdf

Name: Michael O'Hara Student Number: 16414554

Date: 12/12/20

Contents

1. Pre-processing

1.1 Editing the JSON

The first thing which must be done to the data set is the pre processing of the data to make sure it is readable. The data in the dataset must be passed in and read from start to finish and locate any inconsistencies such as missing fields or redundant data.

The dataset provided was in the form of a .JSON file. The first thing which had to be done was convert the dataset into a valid data set object. In order to do this the original .JSON file was open in "NotePad++" and using this a "[" was added to start of the file and a "]" was added to the end of the file. The next thing that was done was "Find all & Replace" it was used to find all instances of "}" and replace it with "}," and then the find the last "," and remove it as the last object does not need one as there won't be anything after it. After this it will be saved as new file called "edit_fake_news.json"

1.2 Splitting the data

Now that the data can be properly read in by a python program the "actual" pre-processing can begin. In the "MLSplit.py" the first thing that happens is the edited json file gets read in to a variable df by the python package pandas [1], using the "read-json" method. Once this is done then the code prints the shape of the json and the head of the json to the command line.

The next part is done using sklearn.model_selction package specifically the train_test_split [2] method. This splits the dataset into random training and test sets. This method was used to also split of a separate validation set as well. This was done by first splitting it into test data and training_and_validation (together in 1) data which is done at a ratio of 25% and 75% respectively then splitting the training and validation into separate smaller datasets at a ratio of 80% and 20%.

```
#split off training
X_training_and_valid, X_test, Y_training_and_valid, Y_test = train_test_split(df['headline'], df['is_sarcastic'], test_size = 0.25, random_state = 117)

#split off validation from training
X_train, X_val, Y_train, Y_val = train_test_split(X_training_and_valid, Y_training_and_valid, test_size=0.20, random_state=117)
```

After the splits are completed the data is then taken and concatenated into 3 separate .csv files to make for easier data analysis and model selection. .csv was used rather than json as I felt this was easier for myself to be able to inspect the smaller datasets visually if there was something which got messed up in the pre-processing.

```
#Adds training data to csv
pd.concat([X_train,  Y_train], axis = 1).to_csv(train,index=False)

#Adds test data to csv
pd.concat([X_test, Y_test], axis=1).to_csv(test, index=False)

#Add valid data to csv
pd.concat([X_val, Y_val], axis=1).to_csv(val, index=False)
```

1.3 Feature Extraction

At this stage, the features which are needed for the remainder of analysis are extracted. The first thing done in this section is to carry out the bag-of-words to transform the document. This can be done using sklearn.feature_extraciton.text package. This can be done by running CountVectorizer on the data. This is the process of the removing the “stop words” from the data. The program will print out the most common words before and after removing the stop words and after removal we can see that the top 20 words change as the stop words were the most common words.

```
cv_no_stop_words = CountVectorizer(stop_words='english')
cv_nsw_res = cv_no_stop_words.fit_transform(X_train)
print(cv_nsw_res.toarray())
```

TFIDF is also used at this stage in the evaluation. TFIDF is the acronym for Term Frequency Inverse Document Frequency. TF is how frequent a term is inside a headline and IDF is the number of occurrences of that word across the dataset. This can be run in python using TFIDFVectorizer() similar to CountVectorizer.

2. Data Analysis

In this section we do some basic analysis on the training dataset. This is done with a combo of the training csv file and the natural language toolkit. Using the nltk [3] we can download a package of English stop words to be used in this section.

```
nk.download("stopwords")
```

Firstly, we can use the python module Counter in a list comprehension to count the top 20 most common words and the amount of times the occur in the headline column of the training csv. This then gets outputted to the command line.

```
top20stopwords = Counter(" ".join(train['headline']).split()).most_common(20)
print(top20stopwords)
```

Then use the list of stop words to run a similar list comprehension only this time we find the top 20 most common words which are not in the list of stop words and the number of times they occur and out put that to the command line.

```
list_of_stopwords = nk.corpus.stopwords.words('english')

remove_stopwords = [word for word in " ".join(train['headline']).split() if word not in
list_of_stopwords]

top20words = Counter(" ".join(remove_stopwords).split()).most_common(20)
print(top20words)
```

There is commented out code in this file which was meant to be used to output these as bar charts but this wasn't functioning at the time of submission so was left commented out to allow the program to run

The other part of data analysis that was done in this section was to compare the lengths of real and fake headlines. This was done by creating 2 numpy [4] arrays, one for real headlines and one for fake headlines. Then we run a for loop to read through the training csv and performing checks on the is_sarcastic column and adding the length of the headline to the corresponding array to match if the headline is real or fake.

```
#comparing the headline length
fakeLength = np.array([])
realLength = np.array([])

for line in train_reader:
        if(line[1] == '1'):
                fakeLength = np.append(fakeLength, len(line[0]))
        else:
                realLength = np.append(realLength,  len(line[0]))
```

Then we do some basic calculations on the arrays to get the minimum, mean and maximum values for the real and fake headlines and out this to the command line and a boxplot to be displayed.

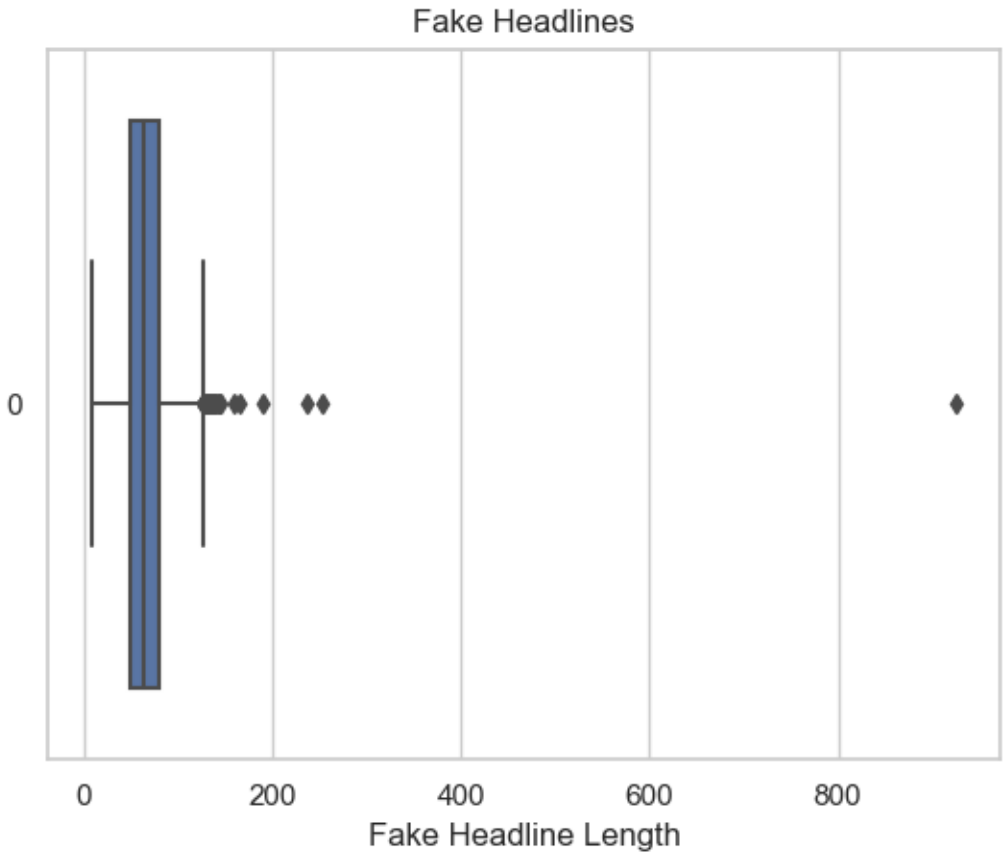
```
#show stats of headline length
print("//-------------------------------------------")
print("Details of fake headline:")
print("Mean fake length:")
print(np.mean(fakeLength))
print("Max fake length:")
print(np.max(fakeLength))
print("Min fake length:")
print(np.min(fakeLength))
```

```
//-------------------------------------------
Details of fake headline:
Mean fake length:
65.50993538949164
Max fake length:
926.0
Min fake length:
8.0
//-------------------------------------------
```

```
//-------------------------------------------
Details of real headline:
Mean real length:
59.711450551900995
Max real length:
154.0
Min real length:
8.0
```

The above images show the results of these calculations for both fake and real headlines

```
#plot to boxplot
sns.set(style='whitegrid')
fakePlot = sns.boxplot(data=fakeLength, orient='h')
fakePlot.set(xlabel='Fake Headline Length', title='Fake Headlines')
plt.savefig('C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/figs/LengthOfFakeHeadlinePlot.png')
plt.show()
```


Fake Headlines
0
0
200
400
600
800
Fake Headline Length

3. Supervised Classification

The supervised classification section of the assignment involves training a model using the validation set of data. The TFIDF method fit_transform was performed on the training data and then the method transform() was performed on the validation set.

```
#TFIDF on files
tfidf = TfidfVectorizer()
tfidf_Train = tfidf.fit_transform(X_train)
tfidf_Val = tfidf.transform(X_val)
```

The classification run was the Multinomial Naïve Bayes was run on the training set and then a prediction was made on the validation set. The accuracy was then printed to command line. I then used a for loop in order to loop through the predictions in order to determine how many were correct and incorrect and outputted these to the command line as well.

```
#Naive Bayes on training data
supervised = MultinomialNB()
supervised.fit(tfidf_Train, Y_train)
predictions = supervised.predict(tfidf_Val)

print("Accuracy of MultinomialNB: ")
print(metrics.accuracy_score(Y_val, predictions))

correct = 0
incorrect = 0
#check the predictions
for i, pre, lab in zip(X_val, predictions, Y_val):
        if pre != lab:
                incorrect+=1
        else:
                correct+=1

print("Number of correct predictions: " + str(correct))
print("Number of incorrect predications: " + str(incorrect))
```

The results of the above code are outputted and show below.

```
Running Supervised_Class:
//--------------------------------------------
Accuracy of MultinomialNB:
0.8327509899836943
Number of correct predictions: 3575
Number of incorrect predications: 718
Finished Supervised_Class:
//--------------------------------------------
```

4. Model Selection

The models which were considered for this section are as follows, Multinomial Naïve Bayes [5], Logistic Regression [6], Support Vector Classification [7] and Random Forest Classifier [8]. These were able to be implemented using the following methods from the corresponding packages in python.

```
#models
classifier1 = MultinomialNB()
classifier2 = LogisticRegression(solver='newton-cg', multi_class='multinomial')
classifier3 = RandomForestClassifier(n_estimators=25)
classifier4 = svm.SVC(gamma='scale')
```

These classifiers are then passed into a for loop which will run tfidf fit() on the training set and the run each of the classifiers on the validation set. It will then output the accuracy to the command line.

```
#store classifiers and names
list_classifiers = [classifier1,classifier2,classifier3,classifier4]
names = ["MultinomialNB", "LogisticRegression", "RandomForestClassifier", "SVC"]


for i in range(4):
        classifier = list_classifiers[i]
        classifier_name = names[i]

        classifier.fit(tfidf_Train, Y_train)
        prediction = classifier.predict(tfidf_Val)
        accuracy = metrics.accuracy_score(Y_val, prediction)


        print("//-------------------------------------------")
        print(classifier_name)

        print("The accuracy of " + classifier_name + " is:")
        print(accuracy)
```

After this is finished a classification_report [9] will be generated for each of the classifiers with the heading of real and fake and the accuracy of the predications being made.

The reports generated for each of the classifiers are displayed below

Multinomial Naïve Bayes:

```
MultinomialNB
The accuracy of MultinomialNB is:
0.8327509899836943
              precision    recall  f1-score   support

Real Headline       0.80      0.90      0.85      2245
Fake Headline       0.88      0.75      0.81      2048

     accuracy                           0.83      4293
    macro avg       0.84      0.83      0.83      4293
 weighted avg       0.84      0.83      0.83      4293
```

Logistic Regression:

```
LogisticRegression
The accuracy of LogisticRegression is:
0.8476589797344515
              precision    recall  f1-score   support

Real Headline       0.86      0.85      0.85      2245
Fake Headline       0.84      0.84      0.84      2048

     accuracy                           0.85      4293
    macro avg       0.85      0.85      0.85      4293
 weighted avg       0.85      0.85      0.85      4293
```

Support Vector Classification:

```
SVC
The accuracy of SVC is:
0.8485907290938738
              precision    recall  f1-score   support

Real Headline       0.86      0.84      0.85      2245
Fake Headline       0.83      0.85      0.84      2048

     accuracy                           0.85      4293
    macro avg       0.85      0.85      0.85      4293
 weighted avg       0.85      0.85      0.85      4293
```

Random Forest Classifier:

```
RandomForestClassifier
The accuracy of RandomForestClassifier is:
0.803633822501747
              precision    recall  f1-score   support

Real Headline       0.82      0.80      0.81      2245
Fake Headline       0.79      0.80      0.80      2048

     accuracy                           0.80      4293
    macro avg       0.80      0.80      0.80      4293
 weighted avg       0.80      0.80      0.80      4293
```

Each of the models then gets saved by the python package pickle to make them easier to use in the evaluation stage. Due to the amount of time taken to run the models it was easier to save all of the models with pickle during the running of the program to ensure they are all saved and there is no need to rerun the program.

```
        #save model
        if(classifier_name == "MultinomialNB"):
                with
open('C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/pickle/save_model_Multinomial
NB.pickle', 'wb') as file:
                        pickle.dump(classifier, file)
        elif(classifier_name == "LogisticRegression"):
                with
open('C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/pickle/save_model_LogisticReg
ression.pickle', 'wb') as file:
                        pickle.dump(classifier, file)
        elif(classifier_name == "RandomForestClassifier"):
                with
open('C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/pickle/save_model_RandomFor
estClassifier.pickle', 'wb') as file:
                        pickle.dump(classifier, file)
        elif(classifier_name == "SVC"):
                with
open('C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/pickle/save_model_SVC.pickle',
'wb') as file:
                        pickle.dump(classifier, file)
```

5. Model Evaluation

Based on the reports generated by the model evaluation it was determined that the classifier with the highest accuracy was Logistic Regression with an accuracy of 85%. Because of this it was this model which was selected to be evaluated on the test data.

Using pickle we load in the model and then use it to predict the test data. The test data has been read in and also had the TFIDF method transform run on it.

```
#tfidf on test
tfidf = TfidfVectorizer()
tfidf_Train = tfidf.fit_transform(X_train)
tfidf_Test = tfidf.transform(X_test)

#open with pickle
with
open('C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/pickle/save_model_LogisticRegression.pickle', 'rb') as file:
        best_model = pickle.load(file)
```

Once the model has been run on the test data the classification report and confusion matrix are generated, in the same way they were generated in the model selection section.

```
print(classification_report(Y_test, prediction))

print(metrics.confusion_matrix(Y_test, prediction))
```

The results to these are as follows:

Classification Report:

```
Running Evaluate:
//-------------------------------------------
The accuracy of the model on the test data is:
0.8419287211740042
              precision    recall  f1-score   support

           0       0.86      0.84      0.85      3772
           1       0.82      0.85      0.84      3383

    accuracy                           0.84      7155
   macro avg       0.84      0.84      0.84      7155
weighted avg       0.84      0.84      0.84      7155
```

Confusion Matrix:

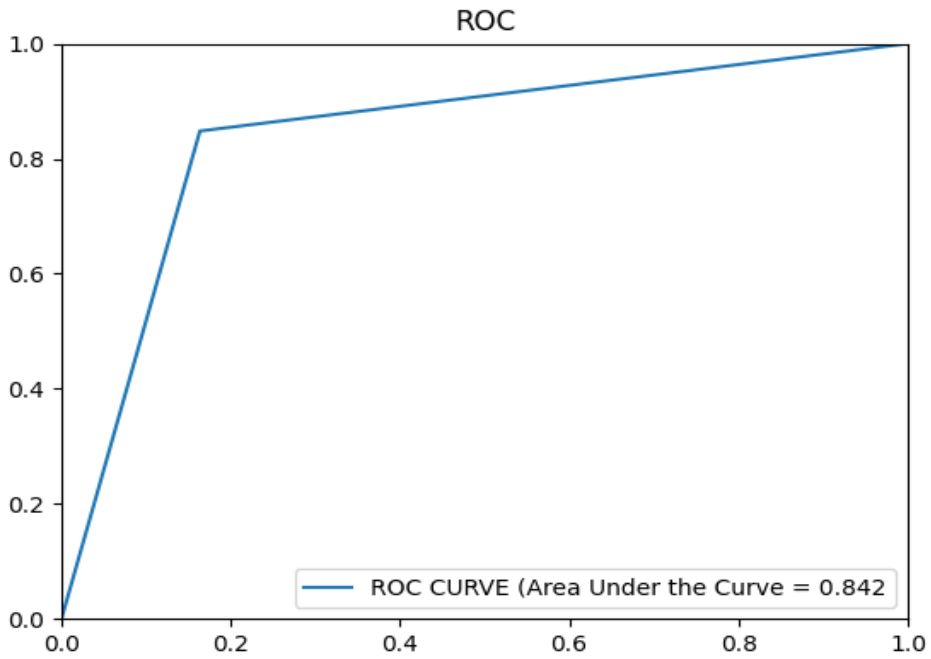
```
Confusion Matrix of model:
[[3155  617]
 [ 514 2869]]
```

The final piece of evaluation that was carried out was to generate a Receiver Operating Characteristic curve and to then calculate the Area under the curve. This was done in a similar manner to the generation of the boxplots in a previous section.

```
#roc curve and area under curve value
fpr, tpr, thresholds = roc_curve(Y_test, prediction)
Area_under_curve = roc_auc_score(Y_test, prediction)

plt.plot(fpr, tpr, label='ROC CURVE (Area Under the Curve = %0.3f' % Area_under_curve)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC')
plt.legend(loc='lower right')
plt.savefig('C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/figs/ROC.png')
plt.show()
```

The code above is how this is done and the results are shown below:



6. Conclusions and Findings

In conclusion it was found that when each of the models are run on the validation data it was given that the minimum accuracy was 80% which comes out at about 1/5 predictions would be incorrect.

Logistic Regression and SVC both achieved the same accuracy when run on the validation set which was 85%. The only difference between the 2 that was noticed during the development of the code, was that the execution of the SVC takes far longer than Logistic Regression. When working with a larger dataset this difference in computation time could be days.

Logistic Regression however offered the most balanced numbers for precision and recall, these allow for better numbers in terms of false positives.

This assignment shows that a model can be created to predict the “realness” of a headline. It can be done using a number of different classifiers which offer similar results but in order to attain a higher level of precision a number of features would need to be tweaked.

There are also a few features which I was unable to implement which would have both affected results and quality of life. One of these was then plotting the stop words to bar charts. The implementation is still unfinished but remains unstable and causes the code to crash.

The other thing that I would have like to implement would have been to eliminate the need for a user to edit the path the files use when running the code, I just used fixed file paths at the beginning in order to get it working.

7. Appendix

In order to run this code, there is a file included with the submission called “Run.py” must be executed at the command line using “ Python Run.py” this file imports all the other files into itself and takes advantage of the fact that when the files get import they get automatically run. Once this has completed the code will simply output “Execution Finished”

```
import MLSplit
import Train
import Data_Analysis
import Supervised_Class
import Select_Model
import Evaluate


print("Execution finished")
```

The object of this design was to minimize the need of user input and fully automate the process. The one thing that does require user input however is the directories in each of the files need to be altered in order to work on a new machine rather than the one development took place on.

An example of the variables which contain path files is shown as below:

```
#path for files
train = 'C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/csv/train.csv'
test = 'C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/csv/testing.csv'
val = 'C:/Users/Nova6/Desktop/College/MCTY/EE514_DAML/csv/valid.csv'
```

Another note to be made about the code is that sometimes on the running of the code the SVC model will sometimes cause the code to hang but a keyboard interrupt (CTRL + C) will make the code recover and continue to run correctly.

8. References:

[1] “pandas documentation — pandas 1.1.5 documentation.” https://pandas.pydata.org/docs/ (accessed Dec. 16, 2020).

[2] “sklearn.model_selection.train_test_split — scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html?highlight=train#sklearn.model_selection.train_test_split (accessed Dec. 16, 2020).

[3] “Natural Language Toolkit — NLTK 3.5 documentation.” https://www.nltk.org/ (accessed Dec. 16, 2020).

[4] “NumPy.” https://numpy.org/ (accessed Dec. 16, 2020).

[5] “sklearn.naive_bayes.MultinomialNB — scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB (accessed Dec. 16, 2020).

[6] “sklearn.linear_model.LogisticRegression — scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression (accessed Dec. 16, 2020).

[7] “sklearn.svm.SVC — scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC (accessed Dec. 16, 2020).

[8] “3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random#sklearn.ensemble.RandomForestClassifier (accessed Dec. 16, 2020).

[9] “sklearn.metrics.classification_report — scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html?highlight=classification%20repo#sklearn.metrics.classification_report (accessed Dec. 16, 2020).