DUBLIN CITY UNIVERSITY

ELECTRONIC AND COMPUTER ENGINEERING

EE514 Assignment

Fake News Detection

Author

Michael Lenehan michael.lenehan4@mail.dcu.ie

Student Number: 15410402

15/12/2019

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://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf` and IEEE referencing guidelines found at `https://loop.dcu.ie/mod/url/view.php?id=448779`.

Signed: _____ Date: 15/12/2019

Michael Lenehan

Contents

1 Introduction

With the explosion of information which has come with the prevalence of social media, there has also come an explosion of misinformation. Machine Learning techniques allow the opportunity to automatically determine whether the information being served up is genuine, or satirical, or in simpler terms “real”, or “fake” news.

With social media being one of the largest sources of shared misinformation, the ability to flag suspect articles could lead to a large decrease in the amount of misinformation in circulation. However, the classification of these articles would need to be robust enough to avoid misclassifying, and mislabelling, articles as “fake” or malicious.

Using a recently assembled dataset of satirical and non-satirical headlines, a model can be trained in order to satisfy these requirements. This assignment aims to investigate the implementation of such a model, and to examine the accuracy which can be achieved.

2 Preprocessing

In the preprocessing stage, the dataset must first be loaded into the program, the required data parsed from the dataset, and any inconsistencies within the dataset must be accounted for. Iconsistencies can include missing data, or any redundant or unnecessary data.

The provided “fake-news” dataset is stored in the “JSON” format. This hierarchical data type makes parsing relatively simple. The “loadData.py” file contains two methods, one for loading the data into a pandas DataFrame object, and one for defining the target values and the features within the dataset.

Listing 1: “loadJson” function

```
def loadJson(jsonFile, printMsgs=False):
        jsonInput = pd.read_json(jsonFile, lines=True)
        if(printMsgs == True):
                print(jsonInput.head())
                print("Shape:", jsonInput.shape)
                print("\nFeatures:", jsonInput.columns)
        return jsonInput
```

The function shown in Listing 1 utilises the pandas python library, and it’s provided “read_json” function. This function loads a json file, with the given file path, to a pandas DataFrame object. As the provided dataset contains each json object on a new line, the “lines” parameter of the “read_json” function must be set to “True”, telling the function to treat each newline as a new json object.

Listing 2: “splitFeaturesAndTargets” function

```
def splitFeaturesAndTargets(data, printMsgs=False):
        x = data[data.columns[1:]]
        y = data[data.columns[0]]
        if(printMsgs == True):
                print("Feature Matrix:", x.head())
                print("\nResponse Vector", y.head())
        return x, y
```

The function shown in Listing 2 performs the split of the features from the dataset. The features chosen are the “headlines”, and the “article links”. The targets are chosen as the “is_sarcastic” value, which is the ground truth, i.e. indicates whether the headline is deemed to be sarcastic, and therefore fall into the category of “fake news”

2.1 Train and Test Split

In order to accurately train and test a classification model, the dataset must be divided into a training set, and a test set. This removes the possibility that the model will "memorize" the entire dataset, giving a 100% accuracy. While performing this split, an unbiased training and test set must be created. For example, training a model off entirely off "sarcastic" items would not provide good accuracy during the testing phase.

Listing 3: sklearn "train_test_split" function

```
X_train, X_test, y_train, y_test = train_test_split(
x, y , shuffle=True, test_size=0.25, random_state=1)
```

The scikit-learn python library contains a "train_test_split" function, which, as shown in Listing 3, can be used to split the feature and target sets in the proportions given by the "test_size" parameter. The "shuffle" parameter allows for the shuffling of the data, which will allow for less biased split. The "random_state" parameter sets the seed used by the random number generator, allowing for repeatability within the testing[1].

The function call in Listing 3 creates a train test split of 75% to 25%, which is shuffled to remove bias, with a random state of 1 for repeatability between tests.

Listing 4: "countRealFake" function

```
def countRealFake(targetTrainSet, targetTestSet):
    print("Real Headlines in Training Set: %s" % sum(
        targetTrainSet == 0))
    print("Fake Headlines in Training Set: %s" % sum(
        targetTrainSet == 1))
    print("Real Headlines in Test Set: %s" % sum(
        targetTestSet == 0))
```

```
    print("Fake Headlines in Test Set: %s" % sum(
        targetTestSet == 1))
```

In order to count the number of real and fake headlines within each set, the "countReal-Fake" function is used. This function takes two datasets, namely the target training set, and the target test set, and prints the total number of fake and real headlines in each, represented by a "1" or a "0" respectively.

Listing 5: "exportTestTrain" function

```
def exportTestTrain(X_train, X_test, y_train, y_test,
    printMsgs=False):
        Export = X_train.to_json(r'./XTrain.json', orient
            ='records')
        Export = X_test.to_json(r'./XTest.json', orient='
            records')
        Export = y_train.to_json(r'./YTrain.json', orient
            ='records')
        Export = y_test.to_json(r'./YTest.json', orient='
            records')
        if(printMsgs == True):
                print("Training Features output to XTrain
                    .json")
                print("Test Features output to XTest.json
                    ")
                print("Training Targets output to YTrain.
                    json")
                print("Test Targets output to YTest.json"
                    )
```

In order to export the feature and target training and test sets, the "exportTestTrain"

function is used. As shown in Listing 5, this function uses the pandas library "to_json" function to output the json object to a file. The "orient" parameter is set to "records" in order to make the file more human readable.

2.2 Feature Extraction

For the purposes of feature extraction, a "Bag of Words" tokenizer is utilised to count the occurrences of words within the headlines. The sklearn "CountVectorizer" function provides this functionality[2]. There are a number of tunable parameters with this function, however, the main concern for this assignment is the inclusion or removal of "stop words". The headlines, and the words of which they comprise are the features within this dataset.

TFIDF stands for Term-Frequency-Inverse Document Frequency. Term Frequency is given in terms of each headline, i.e. how frequent a term is within the headline, whereas inverse document frequency is the number of occurrences of the word acrross all documents. Tf-Idf can be implemented in python using the sklearn "TfidfTransformer" method, which can be configured to utilise, or ignore, idf. A comparison of the inclusion or exclusion of idf is contained in the Model Selection section.

3 Exploratory Data Analysis

By analysing the "XTrain.json" feature training set, some important information can be found. For example, the file contains both the article headline, and the article link. Training utilising the article link will give 100% accuracy, as all fake news headlines are sourced from the satirical news website "The Onion", while all real news headlines are sourced from the "Huffington Post". As such, models should only be fit using the article headlines.

Listing 6: “countTopWords” Function

```
def countTopWords(featureTrainCounts, count_vect, n):
    sum_words = featureTrainCounts.sum(axis=0)
    words_freq = [(word, sum_words[0, idx])
                  for word, idx in count_vect.vocabulary_
                      .items()]
    words_freq = sorted(words_freq, key=lambda x: x[1],
        reverse=True)
    return words_freq
```

In order to count the most frequently occurring words in each feature, the “countTopWords” function is used. This function, as in Listing 6, takes the matrix output of the “CountVectorizer” function, adds the occurrences of each word, and sorts the values into a list of tuples of word-frequency pairs[3]. This list can then be returned to the calling function.

In order to plot the most frequently occurring words within the training set, the “plotBagOfWords” function is used. This function , as shown in Listing 7, takes a plot title, the output from the “countTopWords” function, and the number of terms, and plots them in a bar plot of words against frequency. This bar plot is plotted horizontally for clearer readability.

Listing 7: “plotBagOfWords” Function

```
def plotBagOfWords(plotTitle, data, n):
    sns.set(style="darkgrid")
    x = []
    y = []
    for N in range(n):
        x += [data[N][0]]
        y += [data[N][1]]
```

```
        print(x)
        print(y)
    ax = sns.barplot(x=y, y=x, color="blue")
    ax.set(xlabel='Frequency', ylabel='Word', title=
        plotTitle)
    plt.show()
```

To test the difference in allowing or removing stop words - presumed "uninformative" words - the "bagOfWords", "countTopWords", and "plotBagOfWords" functions must be called with the "stopWords" parameter set to "True" and "False". The following plots show the differences when stop words are and aren't included.

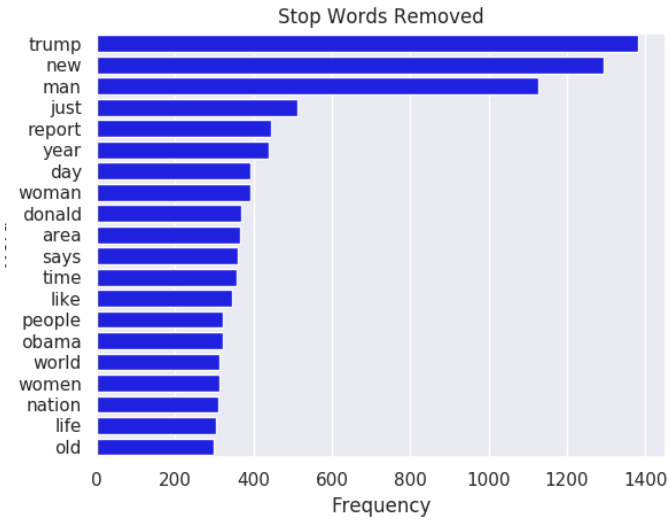

Figure 1: Most Frequent Terms - Stop Words Removed

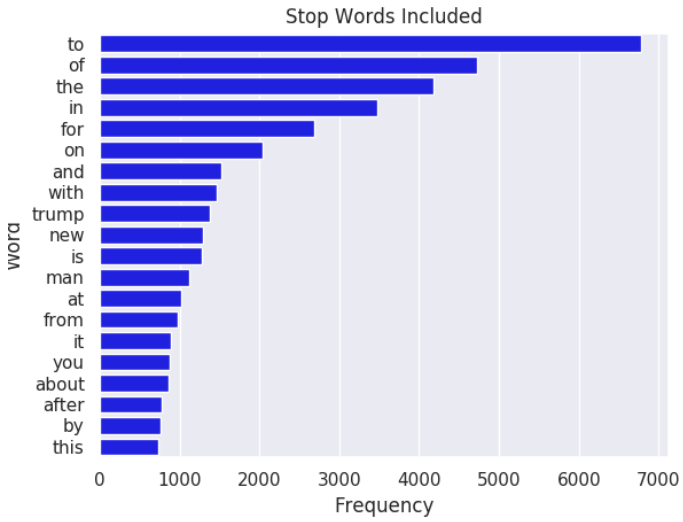

Figure 2: Most Frequent Terms - Stop Words Included

In order to verify these results, the "yellowbrick" library was used to vizualize the data[4]. The results from this verification can be seen in Figures 3 and 4.

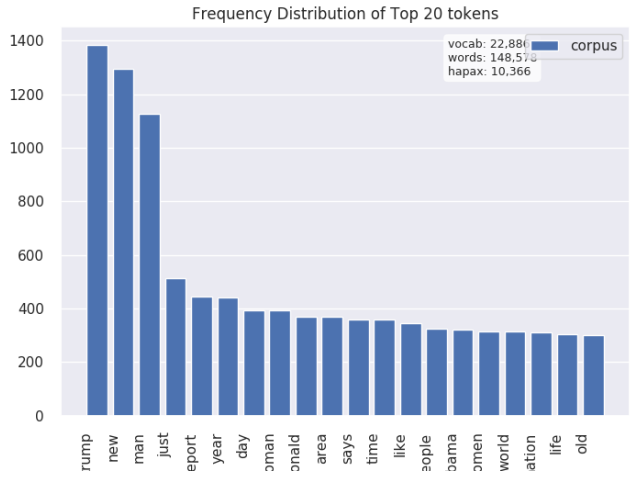

Figure 3: Most Frequent Terms - Stop Words Removed - Yellowbrick

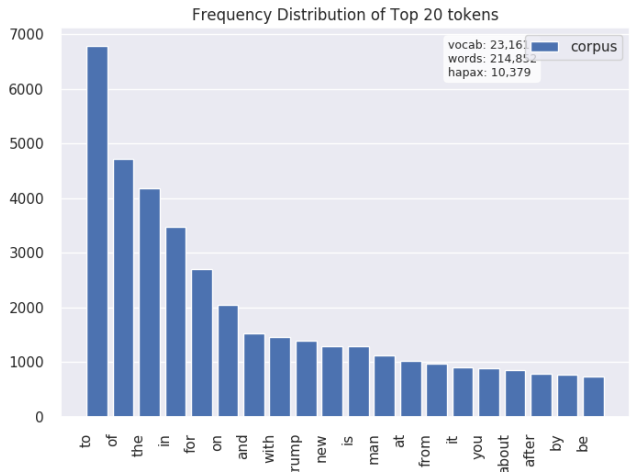

Figure 4: Most Frequent Terms - Stop Words Included - Yellowbrick

As seen in Figure 1, the most frequent word within the training set, with stop words removed, is “Trump”. This gives some context to the nature of the content within the dataset. However, in Figure 2, the most frequent word within the training set without removing stop words is “to”. This word gives no contextual information, and as such, shows the benefits of removing stop words within the Bag of Words model.

Listing 8: “countHeadlineLength” Function

```
def countHeadlineLength(features, target):
    fakeLen = np.array([])
    realLen = np.array([])
    for index, value in target.items():
        # print("Index: {}, Value: {}".format(index,
            value))
        if(value == 1):
            fakeLen = np.append(fakeLen, len(features.
                headline.loc[index]))
```

```
        else:
            realLen = np.append(realLen, len(features.
                headline.loc[index]))
        # print(features.headline.loc[index])
    print(np.mean(fakeLen))
    print(np.mean(realLen))
    plotHeadlineLength(fakeLen, realLen)
```

In addition to the frequency of the words, the length of the headlines can be assessed. The "countHeadlineLength" function, as shown in Listing 8, takes the feature and target datasets as inputs, iterates over the target set, and creates two numpy arrays, one containing the length of the "real" article headlines, and one with the "fake" articel headlines. The mean value of these arrays can then be taken. In order to visualize this information, two boxplots are drawn, using the seaborn library "boxplot" function. The "plotHeadlineLength" function takes in the real and fake headline length arrays are passed in to the function, which converts each to a pandas DataFrame object in order to plot them.

Listing 9: "plotHeadlineLength" Function

```
def plotHeadlineLength(fake, real):
    dataFake = pd.DataFrame(fake)
    dataReal = pd.DataFrame(real)
    sns.set(style='darkgrid')
    ax = sns.boxplot(data=dataFake, orient="h")
    ax.set(xlabel='Headline Length', ylabel='', title="
        Fake Headline Length")
    plt.savefig('fakeHeadlineLength.png')
    plt.show()
    bx = sns.boxplot(data=dataReal, orient="h")
    bx.set(xlabel='Headline Length', ylabel='', title="
```

```
    Real Headline Length")
plt.savefig('realHeadlineLength.png')
plt.show()
```

The resulting boxplots from Listing 9 for the fake and real headlines are shown below in Figures 5, and 6 respectively.

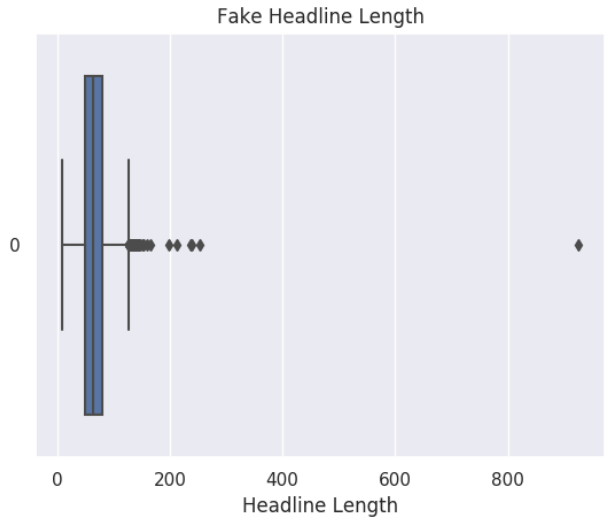

Figure 5: Lengths of the “Fake” Headlines

The mean length of a fake headline is approximately 65.48 characters. There is a maximum length of 926 characters, and a minumum length of 8 characters.

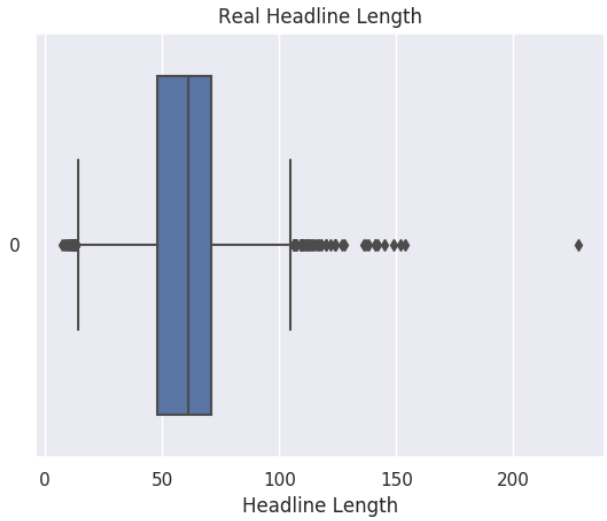

Figure 6: Lengths of the “Real” Headlines

The mean length of a real headline is approximately 59.46 characters. There is a maximum length of 228 characters, and a minimum length of 7 characters.

From the above analysis it can be seen that the length of headlines for fake headlines are, on average, longer, with a greater value outlier. As such, the length of real headlines are shorter, with smaller outlier values.

4 Supervised Classification

Supervised Classigication involves training a supervised model based on the aforementioned features, using a validation set. Due to an error which was occurring - values within the target set changing to “NaN” - the chosen cross validation method of “Leave One Out” cross validation could not be correctly implemented, however, the tested implementation is commented within the code. As such, only the held-out test set could be used for the purpose of validation.

The “Pipeline” function is used to quickly create a model, allowing for the vectorizer, tfidf function, and classifier to be applied to the training sets for the features and targets. The models are saved using the “pickle” library’s “dump” function. Each optimised model is saved after fitting.

5 Model Selection

For the purposes of classification within this assignment, the naive Bayes, and Support Vector Machine (SVM) with Stochastic Gradient Descent are utilised. The sklearn function “MultinomialNB” provides the naive Bayes classifier for the model**skBayes**. Using this classifier, a test accuracy of approximately 83% was achieved.

For the SGD classifier, the sklearn function “SGDClassifier” was used. This function takes the loss function, in this case a linear SVM is used by implementing the “hinge” loss function. The standard linear SVM “l2” penalty is also used. This classifier gives a test accurracy of approximately 79%.

By utilising the sklearn “GridSearchCV” function, a number of parameters can be tested for each model.

For the Naive Bayed model, the defined parameters were as follows:

Listing 10: Bayes Model Parameters

```
parameters = {
    'vect__stop_words': (None, "english"),
    'vect__ngram_range': [(1, 1), (1, 2)],
    'tfidf__use_idf': (True, False),
    'clf__alpha': (1e-5, 1),
    'clf__fit_prior': (True, False),
}
```

By utilising this function, a “best case” model can be created. The model described by the parameters above has a best score of approximately 85%. The parameters required for this performance were as follows:

- Classifier:
  - Alpha = 1
  - Fit Prior = False
- Feature Selection:
  - Tf-Idf:
    - Use Idf = False
    - ngram Range = (1,2)
  - Count Vectorizer:
    - Stop Words = None

For the SGD model, the defined parameters were as follows:

Listing 11: SGD Model Parameters

```
parameters = {
    'vect__stop_words': (None, "english"),
    'vect__ngram_range': [(1, 1), (1, 2)],
    'tfidf__use_idf': (True, False),
    'clf__alpha': (1e-2, 1e-3),
    'clf__loss': ('hinge', 'log'),
    'clf__max_iter': (1, 2, 3, 4, 5),
    'clf__shuffle': (True, False),
}
```

By utilising this function, a “best case” model can be created. The model described by the parameters above has a best score of approximately 81%. The parameters required for this performance were as follows:

- Classifier:
  - Alpha = 0.001
  - Loss Function = Log
  - Max Iterations = 1
  - Shuffle = True
- Feature Selection:
  - Tf-Idf
    - Use Idf = True
    - ngram Range = (1,1)
  - Count Vectorizer
    - Stop Words = None

6 Model Evaluation

In order to evaluate both models, the sklearn “metrics” library was used, in particular, the “classification_report”, and “confusion_matrix” functions. The classification reports, shown in Figures 7 and 8 show the Precision, Recall, and F1 Score of each of the “best case” models, as described in the preceding section.

	precision	recall	f1-score	support
0	0.81	0.89	0.85	3705
1	0.87	0.77	0.82	3450
accuracy			0.83	7155
macro avg	0.84	0.83	0.83	7155
weighted avg	0.84	0.83	0.83	7155

Figure 7: Metrics Report for the Bayes Model

Precision is defined as the models ability to minimize false positives, i.e. only label positive data as positive data[5]. As can be seen in the metrics report for the Bayes model, the precision for "fake" articles is 87%, while the precision for the "real" articles is 81%. Comparing this to the SGD Model, it can be seen that the precision for "real" articles was much higher in the case of the SGD Model, however there is a much lower value for precision for the "fake" articles.

Recall is defined as the models ability to find all positive matches[5]. Within the Bayes model, the recall for the "real" articles is 89%, while for "fake" articles there is a value of 77%. Comparing with the SGD Model, there is a value of only 69% for the "real" articles, with a value of 90% for the "fake" articles.

Finally, the F score is a weighted average of the precision and recall values. The Bayes has an F score of 85% for the "real" articles, with a value of 82% for the "fake articles". The SGD Model has a value of 77% for the "real" articles, with a value of 80% for the fake articles.

	precision	recall	f1-score	support
0	0.88	0.69	0.77	3705
1	0.73	0.90	0.80	3450
accuracy			0.79	7155
macro avg	0.80	0.79	0.79	7155
weighted avg	0.81	0.79	0.79	7155

Figure 8: Metrics Report for the SGD Model

From these classification reports, it is clear that the Bayes classifier is superior for this classification task. It has a better average precision and recall for both article types. While the SGD classifier had better precision for "real" articles, and better recall for "fake" articles than the Bayes classifier, the overall performance results in a lower oeverall accurracy.

The confusion matrices, as shown below in figures 9 and 10 confirm the evaluation above. The number of false positives within the Bayes classifier is much lower than that of the SGD Classifier, however the number of false negatives within the SGD classifier is much lower than that of the Bayes classifier. Again, the difference leaves a greater accuracy for the Bayes classifier, making it a more optimal solution for this assignment.

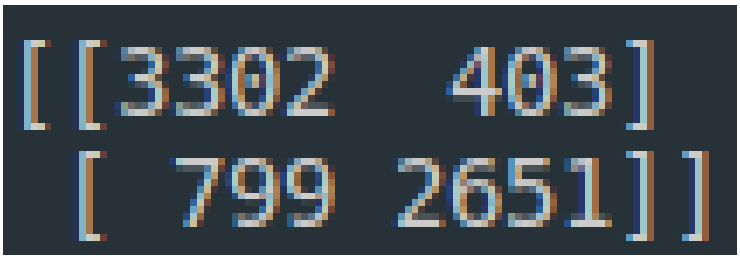

Figure 9: Confusion Matrix for the Bayes Model

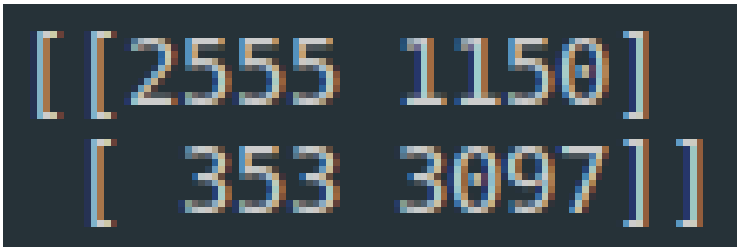

Figure 10: Confusion Matrix for the SGD Model

7 Conclusion

While there were a number of requirements not completely satisfied within this assignment, there are a number of conclusions which can be drawn.

In general the Multinomial Naive Bayes Classifier model achieved much greater accuracy than the Stochastic Gradient Descent model. With an approximate difference of 4% in the accuracies achieved. The difference achieved could have been much greater, had a correct implementation of Leave One Out Cross Validation been utilised.

The evaluation of the models showed the differences in precision and recall for the "best case" models, and the confusion matrices provided show the number of false positives and false negatives for each model.

Overall, the Naive Bayes Classifier model achieved better overall accuracy due to it's higher recall for "real" articles, and higher precision for "fake" articles. These attributes allowed for the much lower false positives, and relatively low false negatives.

8 Appendix

The following is the main code body for the assignment. All other code can be found in the code submission, or on github at `https://github.com/mLenehan1/EE514-DataAnalysis\AndMachineLearning`

In order to execute the code, place all required python files in the working directory, along with the dataset, and run "python3 assignmentMain.py" via the command line.

```
from loadData import loadJson
from loadData import splitFeaturesAndTargets
from loadData import countRealFake
from exportData import exportTestTrain
from bagOfWords import bagOfWords
from dataAnalysis import countHeadlineLength
from sklearn.model_selection import LeaveOneOut
from sklearn.feature_extraction.text import
    CountVectorizer
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
    TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import numpy as np
import pickle
from sklearn.metrics._plot.confusion_matrix import
    plot_confusion_matrix
from matplotlib import pyplot as plt
```

```
data = loadJson('fake_news.json')
x, y = splitFeaturesAndTargets(data)

X_train, X_test, y_train, y_test = train_test_split(
    x, y, shuffle=True, test_size=0.25, random_state=1)

countHeadlineLength(X_train, y_train)

countRealFake(y_train, y_test)


# loo = LeaveOneOut()
# print(loo.get_n_splits(X_train))
# print(loo)
# for train_index, test_index in loo.split(X_train.
    to_frame(), y_train.to_frame()):
#     print("Train:", train_index, "Test:", test_index)
#     X_train, X_test = X_train[train_index], X_train[
    test_index]
#     y_train, y_test = y_train[train_index], y_test[
    test_index]
#     print(X_train, X_test, y_train, y_test)

exportTestTrain(X_train, X_test, y_train, y_test)

count_vect, X_train_counts = bagOfWords(X_train, False,
    True, True)

tf_transformer = TfidfTransformer(use_idf=False).fit(
```

```
    X_train_counts)
X_train_tf = tf_transformer.transform(X_train_counts)
print(X_train_tf.shape)

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(
    X_train_counts)
print(X_train_tfidf.shape)

text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])

text_clf.fit(X_train.headline, y_train)
filename = 'naiveBayesModel'
pickle.dump(text_clf, open(filename, 'wb'))

docs_test = X_test.headline
predicted = text_clf.predict(docs_test)
print("Bayes")
print(np.mean(predicted == y_test))

print(metrics.classification_report(y_test,
                                    predicted,
                                        target_names=None)
                                        )

cm = metrics.confusion_matrix(y_test, predicted)
```

```
print(cm)

parameters = {
    'vect__stop_words': (None, "english"),
    'vect__ngram_range': [(1, 1), (1, 2)],
    'tfidf__use_idf': (True, False),
    'clf__alpha': (1e-5, 1),
    'clf__fit_prior': (True, False),
}

gs_clf = GridSearchCV(text_clf, parameters, cv=10, n_jobs
    =-1)
gs_clf = gs_clf.fit(X_train.headline, y_train)
filename = 'naiveBayesModelBest'
pickle.dump(gs_clf, open(filename, 'wb'))


print(gs_clf.best_score_)
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, gs_clf.best_params_[
        param_name]))


text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SGDClassifier(loss='hinge', penalty='l2',
                          alpha=1e-3, random_state=42,
                          max_iter=5, tol=None)),
])
```

```
text_clf.fit(X_train.headline, y_train)
filename = 'SGDModel'
pickle.dump(text_clf, open(filename, 'wb'))
predicted = text_clf.predict(docs_test)
print("SGD")
print(np.mean(predicted == y_test))

print(metrics.classification_report(y_test,
                                    predicted,
                                        target_names=None)
                                        )
cm = metrics.confusion_matrix(y_test, predicted)
print(cm)

parameters = {
    'vect__stop_words': (None, "english"),
    'vect__ngram_range': [(1, 1), (1, 2)],
    'tfidf__use_idf': (True, False),
    'clf__alpha': (1e-2, 1e-3),
    'clf__loss': ('hinge', 'log'),
    'clf__max_iter': (1, 2, 3, 4, 5),
    'clf__shuffle': (True, False),
}

gs_clf = GridSearchCV(text_clf, parameters, cv=10, n_jobs
    =-1)
gs_clf = gs_clf.fit(X_train.headline, y_train)
filename = 'SGDModelBest'
pickle.dump(gs_clf, open(filename, 'wb'))
```

```
print(gs_clf.best_score_)
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, gs_clf.best_params_[
        param_name]))
```

References

[1] *Sklearn.model$_s$election.train$_t$est$_s$plit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[2] *Sklearn.feature$_e$xtraction.text.countvectorizer*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer.

[3] C. Boujon, *How to list the most common words from text corpus using scikit-learn?*, Sep. 2019. [Online]. Available: https://medium.com/@cristhianboujon/how-to-list-the-most-common-words-from-text-corpus-using-scikit-learn-dad4d0cab41d.

[4] *Token frequency distribution¶*. [Online]. Available: https://www.scikit-yb.org/en/latest/api/text/freqdist.html.

[5] *Sklearn.metrics.precision$_r$ecall$_f$score$_s$upport*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support.