Part 2

The aim of this project is to read datasets representing evaluation of a movie from IMdb, preprocess the data, in order to train a machine learning model. This model shall then be able to identify if a certain string of text represents a positive or negative review.

To start with, we import the important libraries for our use, mainly numpy for our vector manipulation, nltk for text processing (tokenization and lemmitization), and scikit-learn for machine learning algorithms. Some additional libraries are also considered, which can be seen in the code.

After checking the available files, we notice that we have 3 sets of files:

- Training dataset: positive and negative reviews
- Test dataset: positive and negative reviews
- Development dataset: positive and negative reviews

We shall then read each of these files using open command to read each of the datasets,as illustrated in the following snip. Note that the jupyter notebook containing the python code must be located in the same folder that contains the (IMDb) datasets.

```
6 path_train_pos='./IMDb/train/imdb_train_pos.txt'
7 dataset_train_pos=open(path_train_pos,encoding='utf8').readlines()
```

Now we take positive and negative reviews of each dataset, and join them into a single set, labelling positive reviews with “1” and negative reviews with “0” in a tuple of the form (review,label). This shall give us 3 new datasets:

- new_train_set, which contains positive and negative reviews of the dataset we shall use to train out model.
- new_dev_set, which contains positive and negative reviews of the dataset for which we shall use to fine-tune the model.
- new_test_set, we shall only use once to evaluate the model fit into the training set as it provides unbiased evaluation. It provides the gold standards used to evaluate the final model.

We don’t need to split the dataset into train, test and development as we already have them split.

Data preprocessing

The next step is data preprocessing, of which absolute word frequency is considered, with the following normalization features:

- Tokenization – converting text into tokens of words.
- Ignore punctuation marks
- Ignore non alphabetic characters
- Removing stopwords
- Filtering out short tokens (single characters)
- Lemmatization – reducing words to common base ‘root’ form, in lowercase.
- Filtering out all words which are not adjectives or verbs, which if part of our feature selection. To do that, we took each token, extracted POS tag and compared the second part of the tuple to consider only adjectives “JJ” and verbs “VERB”.

The following function shall take in text and return clean tokens.

```
# turn a doc into clean tokens
def clean_doc(doc):
    # split into tokens by white space
    tokens = doc.split()
    # remove punctuation from each token
    table = str.maketrans('', '', punctuation)
    tokens = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 1]
    list_tokens=[]
    for token in tokens:
        list_tokens.append(lemmatizer.lemmatize(token).lower())
    # the following part shall the tokens in and return POS tuple with word type tag
    # we shall then be taking adjectives and verbs only
    test = nltk.pos_tag(list_tokens)
    final_tokens = [a[0] for a in test if (a[1] == 'JJ' or a[1] == 'VERB')]
    return final_tokens
```

We start by sending in our training dataset `new_train_set` and print out the most common 50 tokens, and their frequency of appearing. The outcome looks as follows:

```
29803
[('good', 8100), ('great', 5386), ('bad', 5340), ('many', 4127), ('much', 4088), ('little', 3368), ('first', 2927), ('real',
2718), ('old', 2334), ('new', 2268), ('funny', 2089), ('big', 2028), ('young', 1951), ('whole', 1748), ('br', 1745), ('origi
nal', 1706), ('ive', 1683), ('last', 1680), ('u', 1543), ('main', 1414), ('different', 1349), ('sure', 1310), ('american', 1
289), ('special', 1286), ('dont', 1275), ('true', 1239), ('black', 1232), ('hard', 1208), ('second', 1169), ('high', 1159),
('short', 1151), ('poor', 1146), ('cant', 1104), ('classic', 1076), ('give', 1066), ('excellent', 1039), ('beautiful', 102
9), ('right', 1028), ('full', 1018), ('human', 978), ('nice', 972), ('stupid', 965), ('interesting', 960), ('dead', 941),
('terrible', 925), ('wrong', 920), ('im', 918), ('enough', 900), ('long', 890), ('small', 888)]
```

Feature Selection

Now we may proceed to our model feature selection, which is mainly based on the following:

1. Word frequency
2. Ngrams
3. Part of speech tagging, adjectives and verbs

In order to do that, I've selected TfidfVectorizer from scikit-learn library, of which I was able to pass in my features (word frequency and ngrams) and return a matrix of token counts. The third feature selection was applied during data preprocessing as explained earlier. In this regard.

TfidfVectorizer shall take in my defined tokenizer function `clean_doc`, ngram value of which I will be starting with 2 (bigram), and word frequency of 500. We will fit out model with text of `new_train_set` which is referred to here by addressing the first part of `new_train_set` tuples, which will be our X_train.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(tokenizer=clean_doc,ngram_range=(1,2),max_features=500)
matrix = vectorizer.fit_transform([i[0] for i in new_train_set])
a = matrix.toarray()
```

Now we are ready to train our model as per these features, and to do that we use the following code. Note that Y_train is the label part of new_train_set tuples.

```
#now we train our model, this shall take some time.
import time
start = time.time()
X_train=a
Y_train=[i[1] for i in new_train_set]
svm_clf=sklearn.svm.SVC(kernel="linear",gamma='auto')
svm_clf.fit(np.asarray(X_train),np.asarray(Y_train))
end = time.time()

print("Time to train the model is: %f seconds"%(float(end)- float(start)))

```

```
Time to train the model is: 79.993694 seconds
```

Performance

Now that the model is trained, we can check the performance of our model, but before we proceed with the test dataset, we will start with development dataset. This is because we want to avoid overfitting our model on the test dataset, therefore we shall only expose it to the trained model once we are done with our feature selection parameters. We shall try to improve the performance of our model using different number of features and ngrams. Just like we did for training dataset, our X_dev will be from the first (text) part of the development dataset tuple, and Y_dev will be the label part of the tuple (in the snip below Y_dev is implicitly mentioned). To start with, we will use the existing trained model and predict results from the development dataset. The performance with ngrams = 2 and word frequency of 500 is as follows:

```
X_dev = vectorizer.transform([i[0] for i in new_dev_set]).toarray()
predictions = svm_clf.predict(X_dev)
print(sklearn.metrics.classification_report(predictions,[i[1] for i in new_dev_set]))
```

```
              precision    recall  f1-score   support

           0       0.76      0.79      0.77      2394
           1       0.80      0.77      0.78      2606

    accuracy                           0.78      5000
   macro avg       0.78      0.78      0.78      5000
weighted avg       0.78      0.78      0.78      5000
```

Note that the performance is good at this point with these features, but we will still try to improve it for better accuracy. For that, we will try now with ngrams = 3, and word frequency of 2000. We notice a slight improvement on the performance.

```
              precision    recall  f1-score   support

           0       0.77      0.81      0.79      2370
           1       0.82      0.79      0.80      2630

    accuracy                           0.80      5000
   macro avg       0.80      0.80      0.80      5000
weighted avg       0.80      0.80      0.80      5000
```

At this point, we are not sure which features are best suited for our model. In order to reduce the dimensionality of our feature selection, we will use chi-squared method, considering the best 500

features. We start with importing suitable libraries from sklearn (chi2 & SelecKBest). Then our TfidfVectorizer will take ngrams =2, and 5000 features word frequency, of which will then be transformed to the reduced features with .transform function. We fit our model now with these features and then apply chi-square for the best 500 features, and store them in X_feature_best.

```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest

vectorizer_f = TfidfVectorizer(tokenizer=clean_doc,ngram_range=(1,2),max_features=5000)
X_train = vectorizer_f.fit_transform([i[0] for i in new_train_set]).toarray()
Y_train = np.asarray([i[1] for i in new_train_set])

X_feature_best = SelectKBest(chi2, k=500).fit(X_train, Y_train)
X_train_feature_best = X_feature_best.transform(X_train)
```

We can now move to the test dataset, of which we will clean the text part, vectorize with our TfidfVectorizer, and store them in X_test. On the other hand, we shall store the label part of test dataset tuple in as an array in Y_test, just like our training and development datasets.

```
X_test = vectorizer_f.transform([i[0] for i in new_test_set]).toarray()
X_test_feature_best =  X_feature_best.transform(X_test)

Y_test = np.asarray([i[1] for i in new_test_set])
```

Now we are finally ready to train our model and check the performance on the test dataset, by taking the predictions from the test dataset and comparing with our gold labels, in this case Y_test.

```
svm_clf_3=sklearn.svm.SVC(kernel="linear",gamma='auto')
svm_clf_3.fit(np.asarray(X_train_feature_best),Y_train)
predictions_feature_best = svm_clf_3.predict(X_test_feature_best)
print(sklearn.metrics.classification_report(predictions_feature_best,Y_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.83      0.79      2284
           1       0.84      0.77      0.81      2716

    accuracy                           0.80      5000
   macro avg       0.80      0.80      0.80      5000
weighted avg       0.80      0.80      0.80      5000
```

Furthermore, we may calculate precision, recall, F1 measure and accuracy. Note that Y_test_gold represents labels from the test set, which are our ground facts of which we shall compare our predictions with.

The following table summarizes the results of performance measures in the test dataset:

Precision	79.9%
Recall	79.7%
F1-Score	79.7%
Accuracy	79.7%

```
from sklearn.metrics import precision_score,recall_score,f1_score,accuracy_score
Y_test_gold = Y_test
Y_text_predictions = predictions_feature_best
precision=precision_score(Y_test_gold, Y_text_predictions, average='macro')
recall=recall_score(Y_test_gold, Y_text_predictions, average='macro')
f1=f1_score(Y_test_gold, Y_text_predictions, average='macro')
accuracy=accuracy_score(Y_test_gold, Y_text_predictions)

print ("Precision: "+str(round(precision,3)))
print ("Recall: "+str(round(recall,3)))
print ("F1-Score: "+str(round(f1,3)))
print ("Accuracy: "+str(round(accuracy,3)))
```

```
Precision: 0.799
Recall: 0.797
F1-Score: 0.797
Accuracy: 0.797
```

Improvements

Although it seems the model has a good accuracy, however, there may be ways to improve it further by applying a few additional techniques:

1. Analysing the preprocessed data and try to reduce meaningless items (like missed stopwords, useless text, …etc.)
2. Although preprocessing is very important, it might be the case that it was overdone and that some important words were not considered. The same thing applied to feature selection, in our case, I've tried applying only word frequency without additional feature selection, and the accuracy was even better. Since tweets are usually written in slang, cleaning text would remove important words that are key to the analysis.
3. Tried using CountVectorizer instead of TfidfVectorizer and the results were very close.

Bonus part

Code release:

https://github.com/baqheriae/CMT307