Sentiment Analyser

March 14, 2019

1 Sentiment Rating

```
In [1]: import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
        import matplotlib.pyplot as plt # Plotting labelled data
        from nltk.corpus import stopwords # dealing with stop words
        from textblob import Word # dealing with lemmatization
        from sklearn.feature_extraction.text import TfidfVectorizer # leading with term freque

        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
        from sklearn.metrics import accuracy_score
```

```
In [2]: import matplotlib.pyplot as plt
        import seaborn as sns
        % matplotlib inline
```

```
In [3]: df = pd.read_csv("C:/Users/Jatin/Downloads/Major Project/Dataset/Sentiment Analysis/tra
        df.head()
```

```
Out[3]:    PhraseId  SentenceId                                             Phrase  \
        0         1           1  A series of escapades demonstrating the adage ...
        1         2           1  A series of escapades demonstrating the adage ...
        2         3           1                                           A series
        3         4           1                                                  A
        4         5           1                                             series

           Sentiment
        0          1
        1          2
        2          2
        3          2
        4          2
```

```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156060 entries, 0 to 156059
Data columns (total 4 columns):
PhraseId      156060 non-null int64
SentenceId    156060 non-null int64
Phrase        156060 non-null object
Sentiment     156060 non-null int64
dtypes: int64(3), object(1)
memory usage: 4.8+ MB
```

Since we are only interested in the **Phrase** and **Sentiment** of the review, therefore we will strip the dataframe.

```
In [5]: df = df[["Phrase", "Sentiment"]]
        df.head()
```

```
Out[5]:                                                Phrase  Sentiment
        0  A series of escapades demonstrating the adage ...          1
        1  A series of escapades demonstrating the adage ...          2
        2                                           A series          2
        3                                                  A          2
        4                                             series          2
```

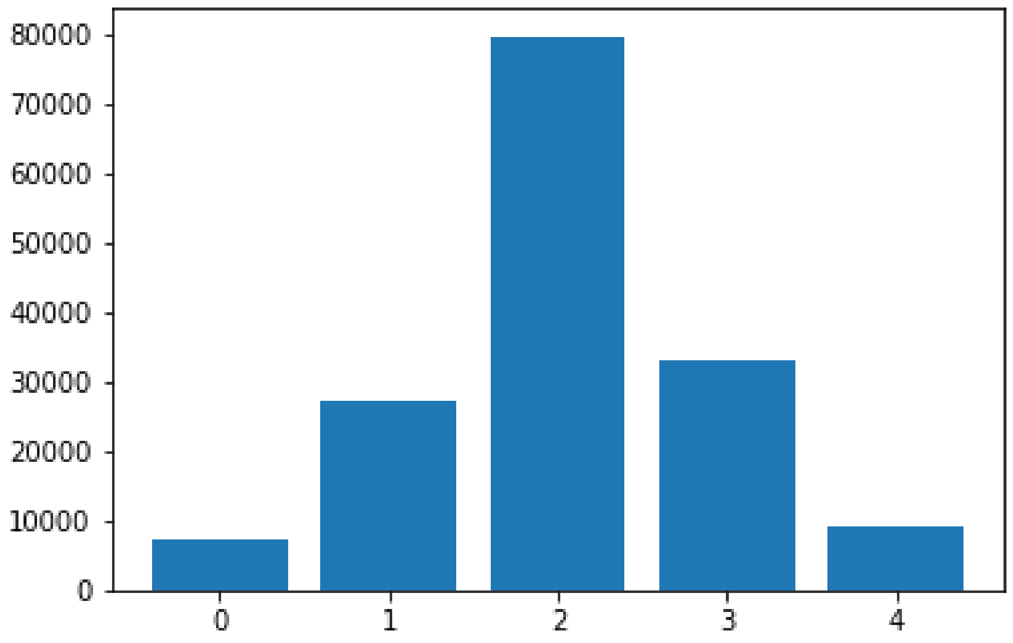
```
In [6]: num_counts = df["Sentiment"].value_counts()
        num_counts
```

```
Out[6]: 2    79582
        3    32927
        1    27273
        4     9206
        0     7072
        Name: Sentiment, dtype: int64
```

The target column **Sentiment** contains the labels for the reviews. There are five classes in which a review is classified.
0 - negative
1 - somewhat negative
2 - neutral
3 - somewhat positive
4 - positive

```
In [7]: plt.bar(x = num_counts.index,height = num_counts)
```

```
Out[7]: <BarContainer object of 5 artists>
```



From the bar graph it is seen that the majority of the reviews are classified to be of neutral sentiment.

1.1 Preprocessing of text

Text Preprocessing is a process of converting and cleaning of the dataset so that it can be easily fed into the machine learning algorithm for its proper functioning and giving higher accuracy.

1.1.1 Lower casing

```
In [8]: df['Phrase'] = df['Phrase'].apply(lambda x: " ".join(x.lower() for x in x.split()))
        df.head()
```

```
Out[8]:                                               Phrase  Sentiment
        0  a series of escapades demonstrating the adage ...          1
        1  a series of escapades demonstrating the adage ...          2
        2                                           a series          2
        3                                                  a          2
        4                                             series          2
```

1.1.2 Removing Punctuations

```
In [9]: df['Phrase'] = df['Phrase'].str.replace('[^\w\s]','')
        df.head()
```

```
Out[9]:                                               Phrase  Sentiment
        0  a series of escapades demonstrating the adage ...          1
```

```
1  a series of escapades demonstrating the adage ...          2
2                                          a series          2
3                                                 a          2
4                                            series          2
```

1.1.3 Removal of Stop Words

```
In [10]: stop = stopwords.words('english')
         df['Phrase'] = df['Phrase'].apply(lambda x: " ".join(x for x in x.split() if x not in
         df.head()

Out[10]:                                              Phrase  Sentiment
         0  series escapades demonstrating adage good goos...          1
         1    series escapades demonstrating adage good goose          2
         2                                             series          2
         3                                                             2
         4                                             series          2
```

1.1.4 Tokenization

Tokenization refers to dividing the text into a sequence of words or sentences. In this example, we have used the textblob library to first transform phrases into a blob and then converted them into a series of words.

```
In [11]: from textblob import TextBlob
         TextBlob(df['Phrase'][1]).words

Out[11]: WordList(['series', 'escapades', 'demonstrating', 'adage', 'good', 'goose'])
```

1.1.5 Lemmatization

Lemmatization is a more effective option than stemming because it converts the word into its root word, rather than just stripping the suffices. It makes use of the vocabulary and does a morphological analysis to obtain the root word. Therefore, we usually prefer using lemmatization over stemming.

```
In [12]: df['Phrase'] = df['Phrase'].apply(lambda x: " ".join([Word(word).lemmatize() for word
         df.head()

Out[12]:                                              Phrase  Sentiment
         0  series escapade demonstrating adage good goose...          1
         1     series escapade demonstrating adage good goose          2
         2                                             series          2
         3                                                             2
         4                                             series          2

In [13]: xtrain, xtest, ytrain, ytest = train_test_split(df.Phrase, df.Sentiment, test_size = 
```

1.2 Count Vectorizer

```
In [14]: cv = CountVectorizer(max_features = None)
         cv.fit(xtrain)

Out[14]: CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                 strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                 tokenizer=None, vocabulary=None)

In [15]: xtrain_cv = cv.transform(xtrain)
         xtrain_cv

Out[15]: <124848x14887 sparse matrix of type '<class 'numpy.int64'>'
                 with 486877 stored elements in Compressed Sparse Row format>

In [16]: xtest_cv = cv.transform(xtest)
         xtest_cv

Out[16]: <31212x14887 sparse matrix of type '<class 'numpy.int64'>'
                 with 121688 stored elements in Compressed Sparse Row format>
```

2 Model selection

- Logistic Regression
- Linear Support Vector Machine
- Multinomial Naive Bayes

Different Models are tested as:

2.1 Multinomial Naive Bayes

```
In [29]: mnb_cv = MultinomialNB(alpha = 0.5)
         mnb_cv.fit(xtrain_cv, ytrain)

Out[29]: MultinomialNB(alpha=0.5, class_prior=None, fit_prior=True)

In [30]: print('Accuracy of Naive Bayes using Count Vectorizer: ', mnb_cv.score( xtest_cv , yt

Accuracy of Naive Bayes using Count Vectorizer:  0.6117839292579778
```

Inverse Document Frequency

The intuition behind inverse document frequency (IDF) is that a word is not of much use to us if it's appearing in all the documents.

Therefore, the IDF of each word is the log of the ratio of the total number of rows to the number of rows in which that word is present.

IDF = log(N/n), where, N is the total number of rows and n is the number of rows in which the word was present.

TfidfVectorizer can lowercase letters, disregard punctuation and stopwords.

```
In [31]: tv = TfidfVectorizer(max_features = None)
         xtrain_tv = tv.fit_transform(xtrain)
         xtest_tv = tv.transform(xtest)

In [32]: mnb_tv = MultinomialNB()
         mnb_tv.fit(xtrain_tv, ytrain)

Out[32]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

In [33]: print('Accuracy of Naive Bayes using Tf-Idf Vectorizer: ', mnb_tv.score( xtest_tv , y
```

```
Accuracy of Naive Bayes using Tf-Idf Vectorizer:  0.5874022811739075
```

2.2 Logistic Regression

** Count Vectorizer**

```
In [34]: clf_cv = LogisticRegression(C = 1)
         clf_cv.fit(xtrain_cv, ytrain)

Out[34]: LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [35]: print('Accuracy of Logistic Regression using CountVectorizer: ', clf_cv.score( xtest_
```

```
Accuracy of Logistic Regression using CountVectorizer:  0.631680123029604
```

Tf-idf Vectorizer

```
In [36]: clf_tv = LogisticRegression(C = 1)
         clf_tv.fit(xtrain_tv, ytrain)

Out[36]: LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [37]: print('Accuracy of Logistic Regression using Tf_Idf Vectorizer: ', clf_tv.score( xtes
```

```
Accuracy of Logistic Regression using Tf_Idf Vectorizer:  0.6186082276047674
```

2.3 Linear SVC

CountVectorizer

```
In [38]: from sklearn.svm import LinearSVC
         svm_cv = LinearSVC(dual = False)
         svm_cv.fit(xtrain_cv, ytrain)

Out[38]: LinearSVC(C=1.0, class_weight=None, dual=False, fit_intercept=True,
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)

In [40]: print('Accuracy of SVM using CountVectorizer: ', svm_cv.score( xtest_cv , ytest))

Accuracy of SVM using CountVectorizer:  0.6361335383826733
```

Tf-idf Vectorizer

```
In [41]: svm_tv = LinearSVC(dual = False)
         svm_tv.fit(xtrain_tv, ytrain)

Out[41]: LinearSVC(C=1.0, class_weight=None, dual=False, fit_intercept=True,
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)

In [42]: print('Accuracy of SVM using Tf-Idf Vectorizer: ', svm_tv.score( xtest_tv , ytest))

Accuracy of SVM using Tf-Idf Vectorizer:  0.6380879149045239
```