Username on Miner: **Michael_Scarn**
Rank: 109
Accuracy: 0.88

Loading Data into Pandas Data frame:

Used *read_table()* pandas function to read text data from train and test files.

data = pd.read_table("1643662645_8986752_1567602457_1187546_train_file.dat", names= ["Sentiment", "Review"])

Where 'names' parameter indicates the list of column names that are assigned. The First column is named 'Sentiment' indicating the +1 or -1 sentiment of the review. The second column is the 'Review' which contains 18506 Reviews.

To handle rows with null data I have used *fillna ()* pandas function to replace null/NaN with text as "No Review" (As we require 18506 rows in the final output file with the predicted sentiment score of +1/-1)

Preprocessing/Cleaning Data:

1.**Converting Text to Lowercase**: used *pandas.Series.str.lower* function to convert all characters to lowercase as it helps reduce the size of the vocabulary of our text data.
2. **Remove Numbers**: used *pandas.Series.str.replace* function to remove digits from the review.
Regex used – '\d+', \d matches any digit character (0-9) and + quantifier means 1 or more times.
3. **Remove Punctuation**: used *pandas.Series.str.replace* function to remove punctuation from the review.
Code: *data.Review=data.Review.str.replace('[{}]'.format(string.punctuation), '')*

Where *string. Punctuation* is a pre-initialized string used as string constant. *string.punctuation* will give us all sets of punctuation.
The regex matches the characters in the set (!"#$%&'()*+, -./:;<=>?@[\]^_`{|}~)
4.**Remove leading and trailing whitespaces**: Used *pandas.Series.str.strip function* to strip whitespaces from left and right sides.
4. **Remove stopwords**: Used *stopwords* from nltk library to remove stop words from the review data.
5.**Stemming**: used *SnowballStemmer* algorithm from nltk library to get root form of a word in the review data.
6. **Lemmatization:** used *WordNetLemmatizer* from nltk library to perform lemmatization with part of speech parameter as verb.

Training the model:

To split the data array into training and testing data, *'train_test_split'* function from sklearn model is used.

Code:
```
X = data['Review']
y = data['Sentiment']
X_train, X_test, y_train, y_test = train_test_split(  X, y, test_size = 0.25)
```

I found the best result with the default test size parameter of 0.25 value specifying the size of testing dataset. To check the accuracy of the trained model I've leveraged *np.mean()* numpy function on correctly predicted sentiment values.

Code:

```
predicted sentiment = tf_idf_model.predict(test_data_tfidf_features)
correctly_predicted = predictedsentiment == y_test
accuracy = np.mean(correctly_predicted) * 100
```

Parameters used in 'CountVectorizer': *analyzer = 'word'* which enables the feature to be made of n-gram*, ngram_range = (1,2)* which indicates the range of n-values for different word n-grams to be extracted (here (1,2) means unigrams and bigrams), *max_features* which enables to build a vocabulary that only considers the top max_features ordered by term frequency.

fit_transform() is used on the training data so that we can scale the training data and learn the scaling parameters of that data.

I have used *toarray()* numpy function to make handling of feature vector easy.

Tf-Idf Transformation

Implemented using *TfidfTransformer* from sklearn library to transform the feature vectors from bag of words implementation to a normalized tf-idf representation.

Code:

```
tfidf =  TfidfTransformer()
tfidf_features = tfidf.fit_transform(features).toarray()
```

Logistic regression classifier

Implemented using *LogisticRegression* from sklearn library.

Parameter with which I got more accuracy: C=1.1 which is the inverse of regularization strength. This parameter retains strength modification of regularization by being inversely positioned to the Lambda regulator. It's basically a penalty parameter term which helps regulate against overfitting.

C: inverse of regularization strength	Accuracy
1.1	0.883
1.0	0.877
10.0	0.874
100.0	0.865
0.1	0.843
0.001	0.538
1e-05	0.536

Testing the model

The same preprocessing steps are used on the test data file.
The vectorizer is applied to test data review set to create a word vector.
Then I applied the tf-id transformer so that word vectors are transformed in the same way as the training data set.

Using *predict()* function the model predicts the sentiment of the all-test reviews

Runtime for preprocessing and loading : 15 seconds, for bag of words and tfidf transformation it took 18 seconds and for training and predicting it took around 35 seconds.

Effects on accuracy with different representations

Representation	Accuracy
Bag of words with ngram_range = (1,2) and tfidf	0.883
Bag of words with ngram_range = (1,1) and tfidf	0.865
Bag of words with ngram_range = (1,2)	0.849
Bag of words with ngram_range = (1,1)	0.828
Bag of words with ngram_range = (2,2) and tfidf	0.821
Bag of words with ngram_range = (2,2)	0.817
Bag of words with ngram_range = (3,3) and tfidf	0.657
Bag of words with ngram_range = (3,3)	0.655

References :

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
https://pandas.pydata.org/docs/reference/
https://numpy.org/doc/stable/reference/