PROJECT REPORT

EECS 4412: Data Mining

April 6th, 2020

Objective

The review sites i.e. Yelp provides a platform for consumers to get more information about products, businesses or services to help them make a choice and also, helps the businesses get key insights from their customers and make improvements based on feedbacks. The purpose of this project is to build a classification model to label such reviews as Positive, Negative or Neutral.

Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a type of data mining technique that identifies the emotional tone behind a body of text through NLP, computational linguistics and text analysis. We will be using this technique to identify the reviews posted on Yelp as Positive, Negative or Neutral. The different steps of this process are discussed below.

Datasets

Two datasets i.e. Training and Test data are provided to train the model and test.

train3.csv: This dataset contains 56,000 randomly chosen Yelp reviews labelled as positive, negative or neutral.

test3.csv: This dataset contains 14,000 unlabelled reviews from the same website that are to be classified by our model.

Data Preprocessing

The first step to data mining is the preprocessing of the given data i.e. data cleaning, data transformation and data reduction.

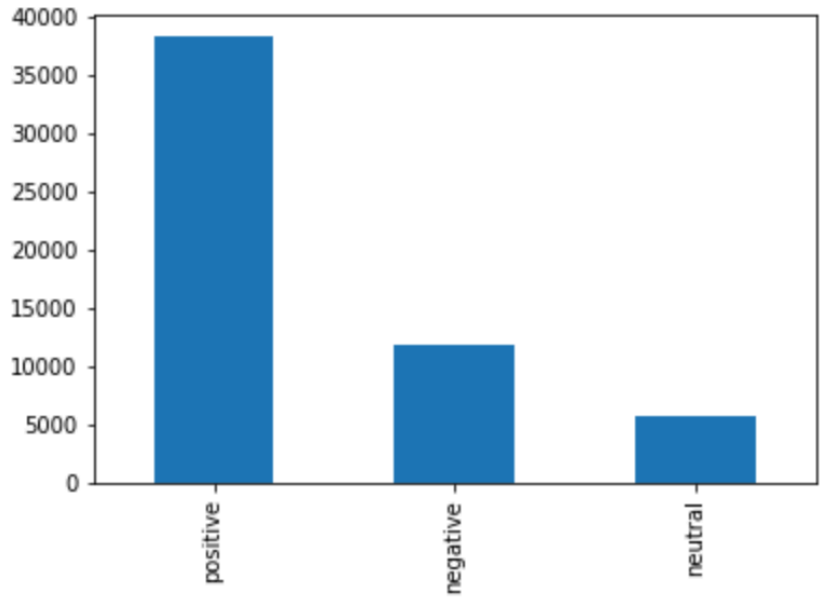
Observations and Challenges: After exploring the given training dataset to identify any potential issues i.e. missing values or imbalances, I noticed that the samples are highly imbalanced.

```
df = pd.read_csv("train3.csv")
```

```
print("The number of reviews for each class is: ",df['class'].value_counts())

df['class'].value_counts().plot(kind='bar')
```

```
The number of reviews for each class is:  positive    38348
negative    11897
neutral      5755
Name: class, dtype: int64
```



The number of positive reviews was significantly higher than both negative and neutral reviews. Feeding such imbalanced data to the classifier can make it biased in favor of the majority class and affect the accuracy.

The approach taken to deal with this situation was to *upsample the minority class*, *neutral,* to match the number of samples in the negative class (11,897) and to *downsample the majority class, positive,* to about 20,000 samples which is much closer to the number of negative samples. Thus, a more realistic distribution was achieved. The functions used to perform this resampling (**upsample_min(df)** and **downsample_max(df)**) are included in the file: **project.py**.

Text Preprocessing: Next step was to remove the words that are not significant for sentiment analysis such as articles, conjunctions and prepositions. The **remove_stop_words(text)** function takes a string as input and converts it to *lowercase*, *removes any stop words as well as words starting with a character that's not an alphabet i.e. $50, 4:00pm etc*. and tokenizes the strings into words. The **load_data()** loads the train3.csv file and converts the dataset into a *two-dimensional tabular data structure, DataFrame* using the pandas llibrary. It also

removes the review ID column and performs *stemming* to reduce the number of insignificant tokens. Further, it performs a 70/30 split and returns the training dataset split into two sets.

Feature Extraction: In this step, the goal is to represent the reviews as a tuple which is basically a vector of attribute values. This is to reduce the number of unimportant tokens (even after stop word removal) by assigning *weight* to them. The **TfidfVectorizer()** class from scikit-learn is used to convert the reviews/text into a matrix of token counts. The **TfidfVectorizer()** class also calculates a token count matrix and produces a normalized **TF-IDF** representation matrix. The parameters used for the function are:

```
TfidfVectorizer(strip_accents='unicode', min_df=3, ngram_range=(1,2))
```

Building a Classification Model

For this multi-class text classification, I have chosen three classifiers that are supposed to be the most fitting for this type of application: **SVM, Logistic Regression** and **Random Forest**. In order to train the model with optimal parameters to achieve the best results, I performed exhaustive *parameter grid search coupled with k-fold cross validation* using GridSearchCV library. The **fit_classifier(x_train, y_train, classifier, param_search, n)** function performs the feature extraction, parameter grid search and model training for each learning algorithms mentioned above, using pipeline. Then the trained models are tested on the test data obtained from the 70/30 split and the obtained classification results were examined. The classification report and accuracy rate of each model were compared and the one with the *highest f1-score* was selected. The f1-score is a measure of the model's accuracy based on both precision and recall and is a more balanced approach than the accuracy rate.

```
Linear SVM accuracy is:  83.68
              precision    recall  f1-score   support

    positive       0.84      0.83      0.84      3516
    negative       0.79      0.71      0.75      3647
     neutral       0.86      0.92      0.89      6007

    accuracy                           0.84     13170
   macro avg       0.83      0.82      0.82     13170
weighted avg       0.83      0.84      0.83     13170
```

```
Logistic Regression accuracy is:  84.68
              precision    recall  f1-score   support

    positive       0.87      0.82      0.84      3516
    negative       0.78      0.77      0.78      3647
     neutral       0.88      0.91      0.89      6007

    accuracy                           0.85     13170
   macro avg       0.84      0.83      0.84     13170
weighted avg       0.85      0.85      0.85     13170

Random Forest accuracy is:  85.69
              precision    recall  f1-score   support

    positive       0.89      0.76      0.82      3516
    negative       0.96      0.77      0.85      3647
     neutral       0.80      0.97      0.88      6007

    accuracy                           0.86     13170
   macro avg       0.88      0.83      0.85     13170
weighted avg       0.87      0.86      0.86     13170
```

Based on the classification report shown above, it is evident that the **Random Forest** model has a more balanced distribution of f1-score as well as the highest accuracy rate amongst them. The model utilizes *Gini index* for further attribute selection. The parameters used to obtain this result are:

```
RandomForestClassifier(n_estimators=500, criterion='gini',n_jobs=-1)
```

Random Forest: Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting. The sub-sample size is always the same as the original input sample size, but the samples are drawn with replacement.

Advantages: Reduction in over-fitting - which is very crucial in this case, as the over-sampling of minority class may result into over-fitting problems - and it is also more accurate compared to other decision-tree based algorithms.

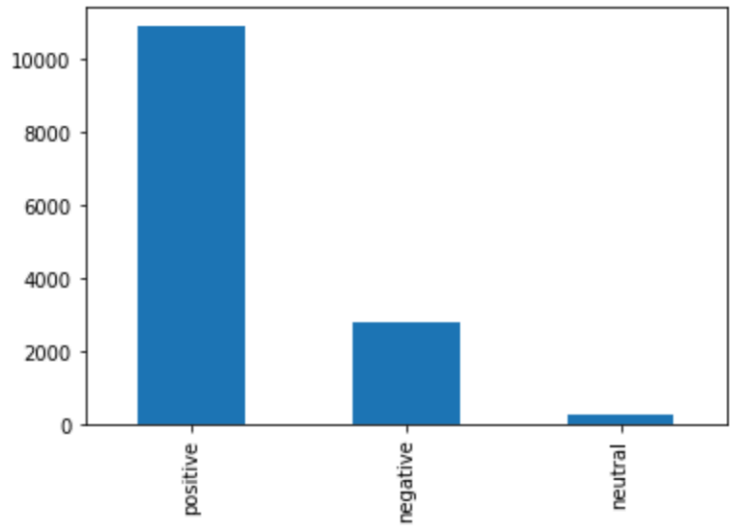
Conclusion

The classification results obtained by using the trained Random Forest model on the test3.csv data are:

```
print("Total reviews in the test dataset: ", len(df_rf_pred))
print("Number of reviews per class:\n",df_rf_pred['CLASS'].value_counts())
```

```
Total reviews in the test dataset:  13999
Number of reviews per class:
 positive    10377
negative     2973
neutral       649
```

```
print(df_rf_pred['CLASS'].value_counts().plot(kind='bar'))
```



It appears that the ratio of positive, negative and neutral reviews classified by the trained model are somewhat similar to the distribution in the training dataset.

The final classification results formatted in accordance with the project specifications are stored in the **prediction.csv** file.

APPENDIX

1. Install Python 3
2. Type **sh lib.sh** in command line to download the Python libraries required for this program.
3. Type **python3 project.py** to run the script. (*it may take 15-20 mins to run*)
4. Type **sh csv2arff.sh** to generate the required output files in arff format.