Thanh Huynh | thanh146
Richard Park | rpark

Project
EECS 4412
Winter 2020
Due Monday April 6 at 11:00 PM

I. Abstract

Most consumers read online reviews before vising a business or using a service. The reviews not only provide customers with important information for making their decisions, but also help businesses get key insights from their customers and make improvements based on customer feedbacks. However, reading numerous reviews can be tedious and time-consuming. Therefore, it is necessary for reviews to have a specific class label to obtain useful information faster.

II. Introduction

In this project, a training data set that contains 56,000 Yelp reviews of businesses (such as restaurants, home services, auto services, etc.) is given. The reviews in the training are labeled as positive, negative or neutral. A classifier based on the training data is built and learnt by using Weka. A test set of 14,000 Yelp reviews without sentiment/class labels is also provided. The goal is to predict the sentiment/class labels for the 14,000 Yelp reviews by identifying critical attributes from the training data set and using data mining algorithms.

The report discuss the steps are made to preprocess the data and which WEKA built-in classification algorithm used as well as the predictions on sentiment/class labels are achieved.

III. Data Preprocessing

1. *Emoji Process:*

   Some reviews contain emojis (i.e. :), :(, :D, etc.) that could be important in determining whether that review is positive, negative, or neutral. First, we created a python script to parse through the training data to find all the emojis that can be used in reviews. Then, we created a file named emoji.xlsx containing all possible emojis and gave them a word pertaining to the emotion of which the emoticon represents (i.e. happy, sad, etc.) and a score that will be used to replace it. After that, the script will get the total score of the all available emoji in a review that associated with it in the emoji file. Source code can be located in the project.py file in the function: Emoji_Process().

2. *User Rating Process:*

   Many of the reviews have ratings that user rated in the review. These ratings are useful in classify the sentiment/class labels. Ratings are given in these formats: # stars, # out of #, # of #, #/#. We separate them into two parts, one is with the number ratings (e.g. 5/10, 4 out of 5, etc.) and another one is with the star ratings. Some people put multiple ratings for individual parts, so our script gets the average of all the ratings. For example, some reviews will say, "the service is 4 stars, but the food is 3 stars". In which our program will give an average of 3.5 stars in the star review attribute for that review. Source code can be located in the project.py file in the function: Rating_Process().

3. *Punctuation Process:*
   Expression of thought could also be characterized using strong punctuation. It is very commonly the case that when using a lot of "!" and/or "?" the user is trying to convey those messages of excitement and/or concern. We found that these are notable attributes in which help identify the classification of the reviews. The code is found under the section: Punctuation() function in the project.py file.
4. *Capitalized Words Process:*
   Similarly with the Punctuation process, the use of all capitalized words is another attribute we found using a python script. For example, the use of many capitalized words such as "NOT HERE" would have this attribute count increase by 2, which would help the model in conjunction with the rest of the attributes show a greater level of intensity. Source code can be found in the project.py file in the function: Capitalized().
5. *Sentiment Words Process:*
   Two data set contains positive and negative words that are commonly used in reviews are used to help with processing the words. All the positive and negative words are matched in each review with the two data set. The total number of positive and negative words are counted. The neutral words are the words those are not appeared in the positive and negative words data set, the frequency count of these words are added each time they appear. For example, if "okay" was found in 1000 neutral reviews, the score of neutral_words is increased by 1000. Source code: Sentiment() in project.py file for the score, and WordStats.py for the creation of the dictionary.
   Text sentiments were used to get sentiment scores from the text. After the text are separated into individual words, the sentiment scores for each word are achieved by an external API: TextBlob.
6. *Stop words:*
   Stop words are words that are commonly used that are not relevant to the goal that is trying to be achieved. Words such as pronouns (i.e. I, you, they, etc.), "a", "the" are probably appear many times in all the reviews but those words are not useful to classify the sentiment/class labels. Therefore, it is needed to remove all words in the stop words file provided. After removing the stop words, the remaining words are filtered again by their frequency and only the top 1000 number of words are kept.
7. *Stemming:*
   Porter Stemming is a method to get the base of an inflected word. That way, all forms of a word can be treated the same. For example, "eat" and "eating" can be treated as just "eat". This allows us to group words faster since they would be transformed to the same word. The source code for the implementation of Porter Stemming is found in the file: ProcessData.py
8. *Dataset Cleaning:*
   There are a lot of meaningless special characters in the dataset, such as "#", "*", "@", etc. There are also some useless strings such as dates in the dataset. Therefore, regular expression is used to filter out of these characters. Filtering out was not just the single use of a regex but also the timing of the regex as sometimes such as in the Emoji-processing some of the special characters were first kept and used to identify the emoji in the string, then discarded as they were no longer needed. Many functions throughout the program handled this cleaning process.

Thanh Huynh | thanh146
Richard Park | rpark

9. *Weka Preprocess:*

   Within Weka, we removed the attribute *"ID", "text"*, and *"caps"* because those three attributes are not as useful as other attributes and do not help in producing higher accuracy.

IV. Learning Algorithm

These are the data mining algorithm in WEKA those are used with 10-fold Cross-validation to verify their accuracy to evaluate and choose classification algorithms. Since our final data after preprocessing contains only numeric values and some of the attributes are missing values, we only go with algorithms those can deal better with numeric values and missing values. 10-fold Cross-validation option divides the entire data set into 10 different subsets and each subset has a corresponding training data set and a test data set. The classification algorithm runs 10 times on various training data of 10 subsets to build a final model. Therefore, this is the best option to give accuracy rates for all the algorithm. The table below shows the performance of each classification algorithm.

Algorithm	**ROC Area**			**PRC**			**Classification Rate**
	Positive	**Negative**	**Neutral**	**Positive**	**Negative**	**Neutral**	
BayesNet	0.876	0.908	0.715	0.933	0.757	0.231	77.8446 %
NaiveBayes	0.850	0.884	0.697	0.914	0.687	0.191	75.5232 %
Logistic	0.878	0.910	0.735	0.931	0.762	0.224	79.4196 %
Neural Network	0.884	0.916	0.719	0.936	0.774	0.202	79.7786 %
SMO	0.740	0.864	0.462	0.808	0.600	0.097	79.5054 %
IBk	0.683	0.703	0.526	0.777	0.394	0.110	68.6268 %
DecisionTable	0.850	0.896	0.654	0.909	0.736	0.179	78.5518 %
JRip	0.725	0.789	0.531	0.800	0.626	0.119	79.0071 %
J48	0.839	0.882	0.644	0.893	0.690	0.168	79.0321 %
RandomForest	0.874	0.909	0.696	0.928	0.760	0.185	79.1232 %
RandomTree	0.717	0.735	0.536	0.797	0.433	0.114	69.9304 %

Figure 1.1 Analysis of the Result on Train Dataset

The Precision-Recall Value (PRC) indicates an example labelled as positive/negative/neutral is indeed positive/negative/neutral. Receiver Operating Characteristics (ROC) Area is the area under the probability curve. It tells how much model capable of distinguishing between classes.

From Firgure 1.2, we can see that the train data set is an imbalanced data set. Therefore, PRC value is more useful thatn the ROC Area value to identify which learning algorithm we should use. From Figure 1.1, BayesNet seems to be the best fit learning algorithm to be chosen. In average, the PRC value for each class label of BayesNet is better than the rest of the learning algorithms.

In order to verify that our assumption in chosing BayesNet algorithm, we run the test data set on the models those have been classified with all the algorithms in Figure 1.1. The result is shown in Figure 1.3.

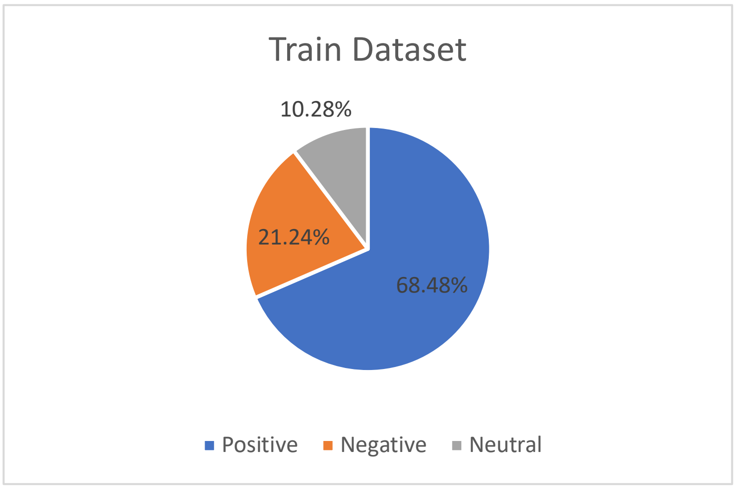

Figure 1.2 Distribution of train data set class labels

Algorithm	**Positive**	**Negative**	**Neutral**
BayesNet	10040 (71.71%)	3110 (22.21%)	850 (6.07%)
NaiveBayes	9978 (71.27%)	2949 (21.06%)	1073 (7.66%)
Logistic	11142 (79.59%)	2733 (19.52%)	125 (0.89%)
Neural Network	10395 (74.25%)	3602 (25.73%)	3 (0.02%)
SMO	11236 (80.26%)	2764 (19.74%)	0 (0%)
IBk	10304 (73.6%)	2427 (17.34%)	1269 (9.06%)
DecisionTable	11440 (81.71%)	2502 (17.87%)	58 (0.41%)
JRip	11486 (82.04%)	2502 (17.87%)	12 (0.09%)
J48	10939 (78.14%)	2999 (21.42%)	62 (0.44%)
RandomForest	10911 (77.94%)	2922 (20.87%)	167 (1.19%)
RandomTree	9572 (68.37%)	2907 (20.76%)	1521 (10.86%)

Figure 1.3 Analysis of Class Label Distribution in Test data set

From Figure 1.3, even though NaiveBayes, Ibk and RandomTree predictions have the closest distribution of class labels as the train data set, these predictions are not accurate because of their low ROC Area values and PRC values as explained above. BayesNet prediction is the closest to the train data set and the prediction is the most accurate out of all the algorithm. In conclusion, we chose BayesNet algorithm to classify the test data set.

V. Appendix

1. Make sure all these libraries/packages are installed for python:
    - pandas
    - re
    - pickle
    - xlrd
    - TextBlob
2. Preprocessing and Classifying on training dataset:
    - Run *Project.py* in folder *venv* (follow the steps mentioned in the comments in *main()* method)
    - Run *trainScript.py* in folder *venv*
    - Convert the *train_final.csv* in folder *final_data* into *train_final.arff* using Weka
    - Load train_final.arff into Weka
    - Under Preprocess tab, remove attribute "ID", "text", and "*caps*".
    - Under Classify tab, run learning algorithm with Cross-Validation 10-folds.
    - Save the model achieved.
3. Preprocessing and Classifying on test dataset:
    - Run *Project.py* in folder *venv* (follow the steps mentioned in the comments in *main()* method)
    - Run *testScript.py* in folder *venv*
    - Convert the *test_final.csv* in folder *final_data* into *test_final.arff* using Weka
    - Load *test_final.arff* into Weka
    - Under *Preprocess* tab, remove attribute *"ID", "text",* and *"caps"*
    - Under *Classify* tab, load the model that has been saved in the previous step *(Preprocessing and Classifying train dataset)*.

VI. Reference

1. *Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews. "Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, Washington, USA"*