EECS 4412: Project

Priya Mishra, Maitry Patel, Krishna Patel

212976627, 215296519, 213320403

INTRODUCTION

The objective of this project is to study a dataset that contains Yelp reviews of businesses such as restaurants, home services, auto services, etc. The reviews in the training dataset are labeled as positive, neutral or negative. In this project we chose a classifier based on the different classifiers we tried on the training dataset in order to perform predictions on the test dataset. We used the software "Weka" to learn a classifier. However, before doing the classification step we performed a number of preprocessing steps. These steps were necessary for the classification steps because the data that we started off with initially, had a lot of unnecessary characters such as meaningless words, punctuations, etc. During the preprocessing step, we removed the unnecessary data, stop words and then stemmed the words that were similar so that the data used for the classification process was more simplified than it originally was and contained only those words that were significant in the reviews.

The significant words mainly included descriptive words that had a meaning and they described a sentiment that a person felt when they wrote the review. Therefore, studying these words and knowing whether they have been used in certain reviews helped in determining the sentiment that a person felt on receiving a service provided by one of the businesses on Yelp. This in turn helps classifying a review as positive, negative, or neutral. Different classification algorithms were applied on the preprocessed data to help determine which algorithm will provide the most optimal results for the prediction model. In order to decide the best algorithm we calculated the misclassification error rate for each and chose the one that had least misclassification error rate in order to increase the accuracy of the prediction model. Lastly, we applied the classification model to the test data in order to perform the prediction for the class values of the test data.

This type of predictive analysis is typically performed on such datasets to help organisations obtain a general idea of how their business is doing in the market and what can be done to improve their business decisions and value in the future. For instance in this case the predicted data can be studied to observe whether they have more positive, negative or neutral feedback. Furthermore, the organization can focus on either one of the class labels. For instance, focusing on the negative label could help determine what their business lacks and what they can do in order to improve their product or service. Similarly, referring to the positive feedback can help them determine the strengths of the organization which can be used for further strengthening their strengths and using them to get more customers, clients.

PREPROCESSING

We began the preprocessing step by combining the training data and the test data using a script in Python. (The script is submitted as a separate document - preprocessing.py). This script combines the training and the testing data together, and then further modifies the positions of the attributes "ID", "text", and "class" in order to make them compatible. To elaborate, since the

train file has attributes in the format - text, class and ID, while the test file had attributes in the format - ID, text, the script reorders the attributes(text, ID, class) so that the train and test files are compatible. The script also works on the preprocessing of the "text" attributes of the train and test datasets. The following steps are a part of the preprocessing done by this script:

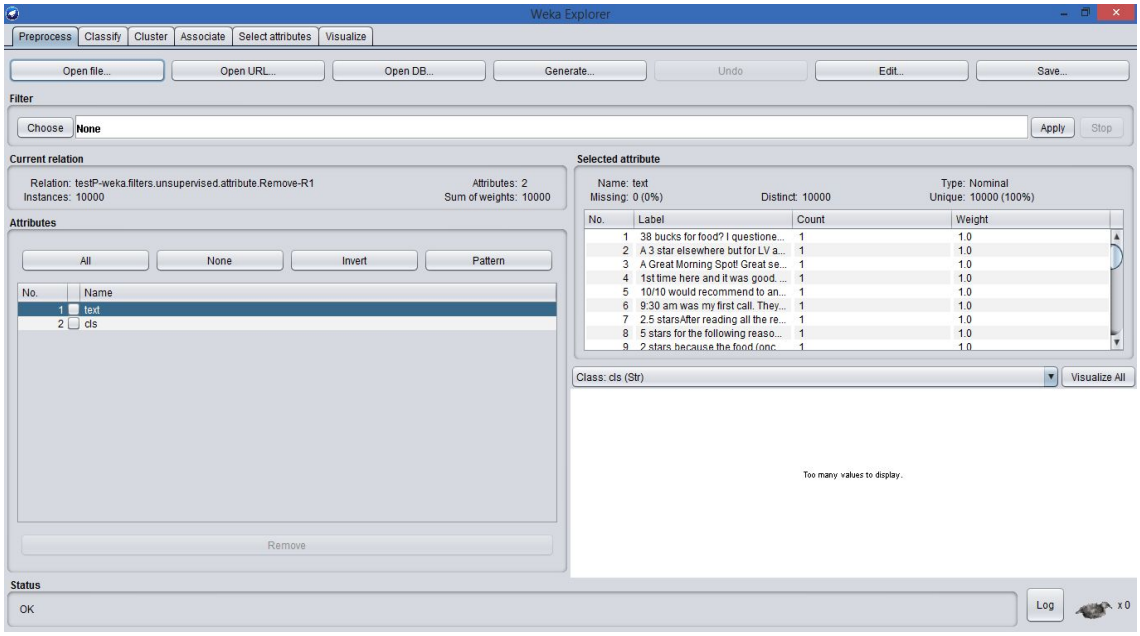
1. All the non alphabet characters are removed from the "text" attributes of each review from training and testing file. The non alphabet characters are not words and they do not provide a significant contribution toward the prediction of the class of a review.

2. Some of the punctuations like ",", ";" were written stuck with two words, without any whitespace. Therefore, replacing the punctuations with empty strings could lose some important words. For example, if a review said that "I like their mango shakes,vanilla shakes", then replacing the comma with an empty string would make "shakesvanilla" a word, which does not make any sense, and loses the two words "shakes", and "vanilla". Therefore, we considered it important to replace commas with spaces. We did the same with ";", ":", "(", ")", "-", "_", "/", and "\", for the same reason.

3. We went through each word in a loop, to make sure there are no words that contain anything but english alphabets. All the other characters including numbers, were taken out. This was done for both, training and test files.

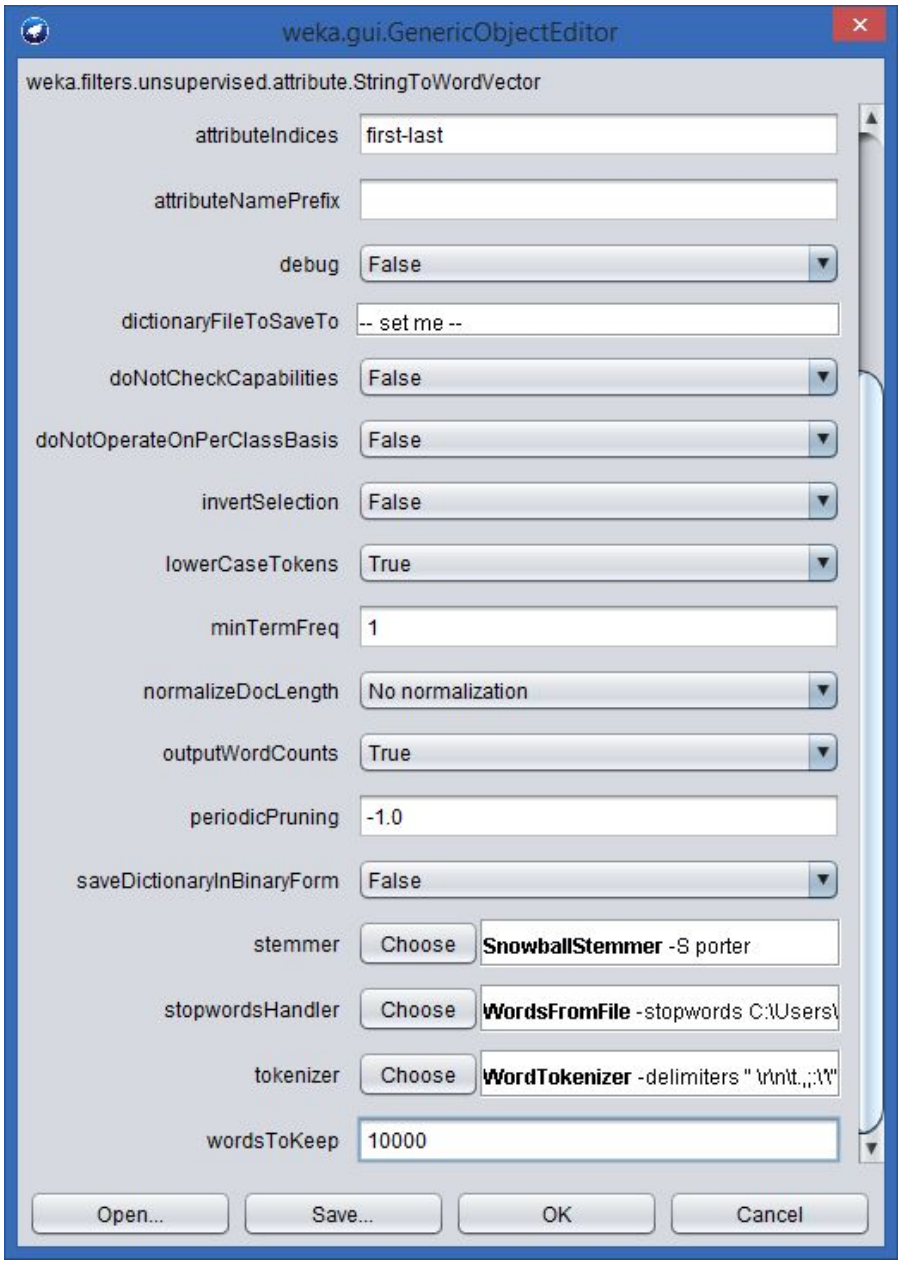
4. Then the script writes the resulting review tuples to a new file in the same format: "text", "ID", "class". For the test data, the script added a "?" in the class attribute in order to make it compatible with the train data. The resulting file did not open in Weka due to an error (too many attributes), which we fixed, by adding double quotes around the text attributes of the reviews. Then, Weka was able to interpret the review as one attribute ("text" attribute).

After the script gave a new csv file combining the train and test data as described, we performed further preprocessing and readied our data for the attribute selection process. The steps along with screenshots are presented below.



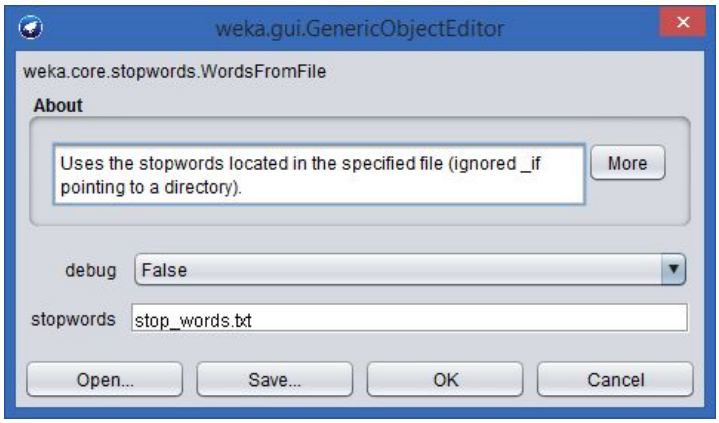
We gave the file outputted by preprocessing.py to Weka, did the preprocessing (stemming and removing stop words), and we chose to keep the top 10,000 words in the files to make a matrix, where the rows represent different reviews and the columns are all the 10,000 words, ID, and class. We chose to keep 10,000 words as attributes in order to have plenty of words to apply the attribute selection method (information gain). Since Weka generates words based on the number of times the word appears in the data file, this word generation was not enough to select attributed. We wanted to see which of the 10,000 words impact the class of the reviews the most. We saved the file that Weka generated, as an arff file.

To execute the above process in weka: Open file > select combined.arff THEN Choose > filter > unsupervised > attributes > select StringToWordVector
Click on StringToWordVector to open GenericObjectEditor and make following changes.

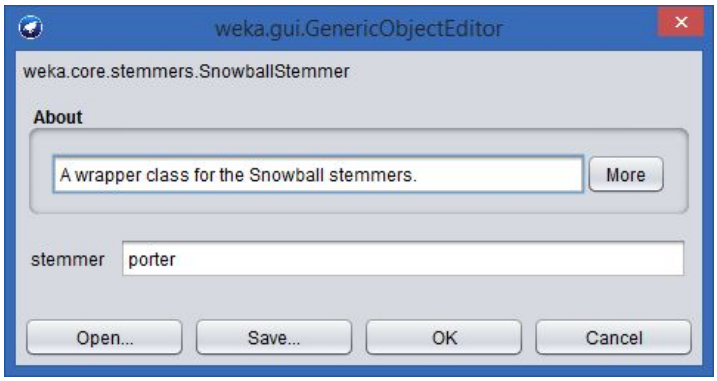


To apply stop words: In stopwordsHandler > choose “WordsFormFile” > click on it and provide the file “stopWords.txt” that contains all the stop words. This helped us to remove all the stop words that are not required in our data. In our stop words file, we had added one letter words for each alphabet because most of the alphabets were coming as frequently used words. We also added words like “th”, “rd”, “st”, “hrs”, “mins” etc, to the stopwords list. We chose to do so because, “mins”, which stands for “minutes”, could be used in good or bad context, so it does not make sense to use it for prediction. For example, a review could say that “They made us wait 15 mins for a table at the restaurant”, and another could say that “The service was very quick, the waitress got our food in only 5 mins”. In both of these scenarios, the expected class is totally

different. The first review suggests the class to be negative, while the second suggests the class to be positive. Therefore, words like these, do not make a good impact on the classification and so we chose to add them in our stop words.



We used SnowballStemmer- S porter to stem the dataset, which helped us to trim the recurrence of words and improve the quality of data.



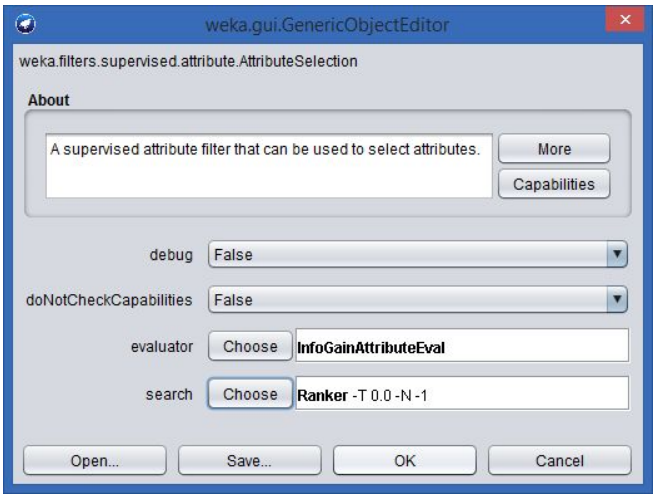
ATTRIBUTE SELECTION

For attribute selection, we applied the Information Gain Attribute Selection method on the train dataset. First, we tried another method as well -chi squared attribute selection- but when we compared the words that the two methods generated as being the most important words, we found a very large similarity in the choice of words and therefore, we decided to stick to any of the two methods, hence, information gain.

To do so: open file > TrainDataSet.arff

Now Choose > weka > filters > supervised > attribute > select “AttributeSelection”

Open Generic editor to select “Information Gain” and Search method “Ranker” > OK



After selecting this click "Apply". This will provide all the attributes in order of their importance on the classification from top most important words to bottom most important words. The order of the attributes will be according to the importance each attribute holds in that particular dataset in order to classify the review as what it is. Once we obtained the attributes in the order of how significant they are according to the information gain attribute selection mehod, we decided to have three different sets of significant attributes. The first set contained 250 top significant attributes, the second set consisted of top 500 attributes and the last one consisted of the top 1000 attributes. Based on the results obtained by applying different classification algorithms to these different sets of attributes we chose the set of attributes that provided the least misclassification cost.

CLASSIFICATION

For classification we provided the preprocessed data in the arff format to Weka and tried multiple classifying algorithms to determine which algorithm does the classification with the most accuracy and has the least misclassification error rate. When we tried the different algorithms, we did so on the top 1000 most important attributes, the top 500 most important attributes, and on the top 250 most important attributes (which were selected using the information gain method on the training data). We tried on 3 different datasets with different number of attributes, to see which dataset gives us the least misclassification rate on each classifier.

The classification algorithms we tried on each dataset are: **Random Forest, Logistic, Naive Bayes and Support Vector Machine**. To do so, we used **10-fold cross validation** on the 3 training datasets and saved the models and outputs of each for prediction purposes.

The purpose for trying the different number of attributes with each algorithm was to help us determine how many number of attributes would be the right choice to obtain a more accurate prediction model.
For obvious reasons, we removed the "ID" attribute before applying the classifiers on the dataset, so that the classifier does not make decisions based on the ID of a review. On completion of the process of running all the classifiers with each set of data, we obtained the confusion matrix for each run. Then, we used the confusion matrix that we obtained for each classifier, to determine their misclassification error rates. The calculation for the misclassification error rate was done manually and based on the minimum misclassification rate from amongst our 12 runs, we decided which algorithm would be best for our data.

We obtained the following confusion matrix after using the random forest classification algorithm on the data set that contains 250 attributes (top 250 ones based on the information gain attribute selection method):

Confusion Matrix for 250 attributes data set with Random Forest Classification Algorithm

a	b	c
7251	2935	600
1815	19732	794
1560	4334	959

To calculate the misclassification error rate, we added all the misclassifications and divided them with the total number of reviews which is 40,000 in this case. So for the above matrix the misclassification error rate would be as follows:

$$\textbf{Misclassification Error Rate} = \frac{2935 + 600+1815+794+1560+4334}{40000} = \textbf{0.27725}$$

We obtained the following results for each trial that we performed:

Random Forest Classification Algorithm

	Number of Attributes in Training Dataset		
	1000 Attributes	**500 attributes**	**250 Attributes**
Misclassification Error Rate =	**0.273**	**0.27445**	**0.27725**

Logistic Classification Algorithm

	Number of Attributes in Training Dataset		
	1000 Attributes	**500 attributes**	**250 Attributes**
Misclassification Error Rate =	**0.23335**	**0.23895**	**0.24905**

Naive Bayes Classification Algorithm

	Number of Attributes in Training Dataset		
	1000 Attributes	**500 attributes**	**250 Attributes**
Misclassification Error Rate =	**0.38265**	**0.3641**	**0.341125**

Support Vector Machine Classification Algorithm

	Number of Attributes in Training Dataset		
	1000 Attributes	**500 attributes**	**250 Attributes**
Misclassification Error Rate =	**0.23025**	**0.2409**	**0.252475**

We calculated the above misclassification error rates with the help of the confusion matrix and based on these calculations we decided to choose the **"Support Vector Machine"(SVM)** misclassification algorithm for the further process as it has the **lowest misclassification error rate**. We opted to use the data set that has **1000 attributes** for further work, as it had the lowest misclassification error rate in combination with the support vector machine classification algorithm.

The following information is the output obtained by running the support vector machine classifier on our final data set which consists of 1000 attributes:

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances	30679	76.6975 %
Incorrectly Classified Instances	9321	23.3025 %
Kappa statistic	0.5806	
Mean absolute error	0.289	
Root mean squared error	0.3747	

Relative absolute error 73.9983 %
Root relative squared error 84.797 %
Total Number of Instances 40000

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.793	0.088	0.769	0.793	0.781	0.698	0.891	0.706	negative
	0.907	0.289	0.799	0.907	0.849	0.637	0.817	0.783	positive
	0.270	0.050	0.529	0.270	0.358	0.294	0.637	0.283	neutral
Weighted Avg.	0.767	0.194	0.745	0.767	0.747	0.595	0.806	0.676	

=== Confusion Matrix ===

a	b	c	<-- classified as
8565	1589	652	a = negative
1082	20262	997	b = positive
1489	3512	1852	c = neutral

Now that we had saved the model of SMO classifier (on 1000 attributes) to work on prediction of the classes of the test data.
To do so: right click on Result List > Load model > select filename.model > open
In test option select “supplied test set” and set test.arff file.
Select More Option > uncheck everything > Output Predictions > choose > plain text.
Then apply “Re-evaluate model on current test set”. This will give us predictions generated based on the support vector machine model that we selected from classifier.

After we got the predictions in the output tab of Weka, we copied the predictions and pasted them in a text file. We had to make prediction.txt file which consists of the ID and the prediction of the review, and therefore we had to extract just the prediction from the output, for which we used a script outputPredictionScript.py (which is submitted as a separate file and its description is in the Appendix).

CONCLUSION AND DISCUSSION

The motive for this project was to perform sentiment analysis on the reviews provided by Yelp to determine whether the review is positive, negative or neutral. Such predictions can be used by the companies when they get reviews about their products or services. Usually, the researchers in the companies do not have the time to go through all the reviews and track the number of positive and negative reviews they receive. Therefore, this kind of a prediction model can be used by the concerned to acknowledge how many positive, negative and neutral reviews they have. Once they know which reviews are negative, they can go through those reviews and find

out how they can improve their business. This saves them a lot of time. At the same time they can also use the positive and neutral feedbacks for determining their strengths, and for further magnifying their strengths to grow as a business. Therefore, it is very important for a person performing a prediction model task to try to be as accurate as possible, since the prediction accuracy makes a very big difference for someone who wants to study such datasets and learn from them to base important decisions on them.

We faced quite a few challenges during the process of creating a prediction model.
Firstly, we did not know that each step that we perform on Weka will take far more time than we expect as compared to other software applications. It took us about 8 hours to run one of the classifiers on the training data with 1000 attributes. Another issue we faced was making the datasets compatible after applying information gain on the training data. We did not realize that when we apply information gain on the training data and rearrange the 10,000 attributes in the rank of their importance for prediction, we would have to do the same on the test data, but manually.
After we selected the attributes (250, 500, and 1000) for the train data to build a model from, we realized that the test data has its attributes arranged alphabetically. This did not let the datasets be compatible and therefore we had to spend hours making the test data compatible with the train data, by manually removing from the test data, all the 9000 attributes that were not in the new training dataset.
We also found it challenging to select the number of attributes to keep as we did not have much of an idea about how many attributes would be the best to keep. In addition, choosing the classifiers to try was another challenge. We researched online and were not able to find much on the types of classifiers that work best with text classification. Most of the sites said that any classifier would be helpful in performing text classification, and so we went ahead and made decisions based on our own knowledge and experiences. However, many of the times when the classifiers were running, they took more than two hours to run, which made it very time consuming for us to make decisions about our choice of classifiers.
In conclusion, the project was very helpful in terms of getting to learn how data scientists make decisions. It seems as if the job of a data scientist is just to understand what data they have, decide what they want to learn from it and apply it in their workplace. However, this project made us realize how much more a data scientist needs to plan and think about when the data is at hand. We realized that finding a classifier that works best, along with the optimal number of attributes that are chosen using the best attribute selection methods, making the training and the test data compatible, and taking care of every little error in a database with millions of data is not as straightforward as it seemed. Careful planning and time must be given in order to come up with a model that has a misclassification error rate close to 0.

Appendix

We have 2 programs submitted as separate files (Python). The files are preprocessing.py and outputPredictionScript.py.

- To use preprocessing.py: type the command “python preprocessing.py train.csv test.csv”. The two files to give to the script are the train and test data files given to us.
- To use outputPrediction.py: type the command “python outputPrediction.py output.txt”. The text file to give to this program is submitted as a separate file. This file consists of the prediction for the class of the test data, given by our classifier model. This script was only used to strip off the extra information from the output. We only needed the classification to put it in the file to submit (prediction.txt), which is what this script does.