PROJECT

EECS 4412 – Data Mining

Submitted By:

Harpreet Kaur Saini -213321917
Shivalika Sharma – 213963608
Komal Patel - 215306129

on

3rd April, 2019

Table of Contents

Objective

The goal of our project is to classify Yelp reviews about different businesses and services as positive, negative or neutral review. This classification helps customers to decide which business or service to choose (they will go for one with positive reviews). The reviews and the classification also helps businesses to realize their flaws and fix them as per customer's opinions.

Text Classification

Text mining is the process of extracting relevant information from natural language text and finding interesting relationships in between the entities extracted. This process gets a bit challenging when we work with high-dimensional data sets with arbitrary patterns of missing data.

Data Preprocessing

Data preprocessing is an important step in data mining process. In our project, we did stop word removal, stemming using Porter Stemmer, converted the training and testing data into a relational table (where words extracted from the reviews are attributes and the frequency or TF-IDF value is the occurence of a word in a review) using StringToWordVector filter.

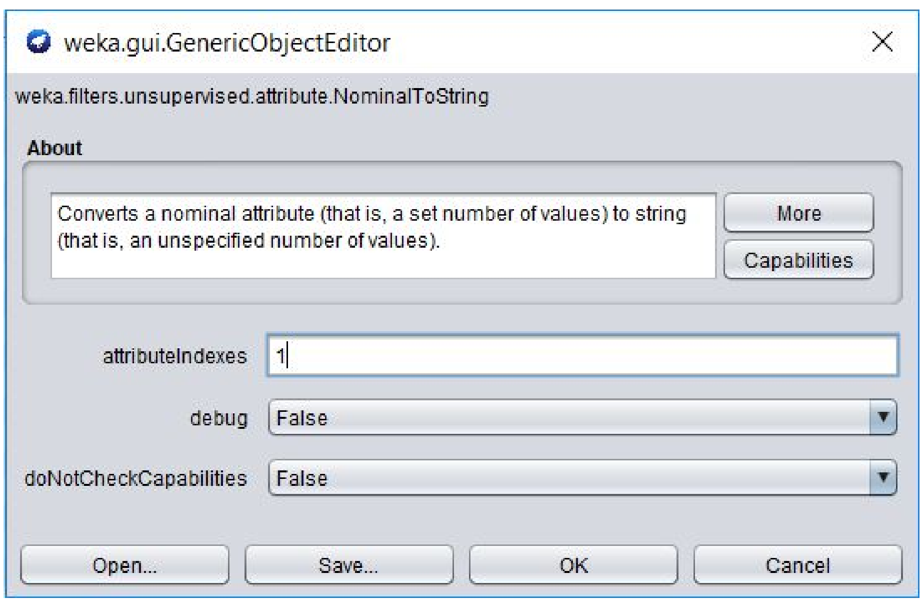
For stop words, we removed the words provided to us in the project description as well as unimportant characters like $, $40, 2pm, !; and words including characters like single and double quotes. Single and double quotes words or characters were needed to be removed because they were giving error when trying to upload to Weka. Other characters and words had to be removed because they were placed high when we were looking for best words to choose. These stop words were removed through a Python script. We applied NominalToString filter to convert the attribute "text" into String type. And finally we applied StringToWordVector filter.

We had to combine the data for both training and testing into one file prior to applying StringToWordVector filter. This is because if we convert training and testing separately, Weka will extract different features for training and test files. Thus when we train the classifier on training data, we won't be able to run it on the test data since the features will be different. We need to have same attributes/features for both training and testing data for classification.

Therefore, we combine both files before applying StringToWordVector filter to extract attributes, which will make training and testing files compatible during classification.

Following are the steps performed for data preprocessing:

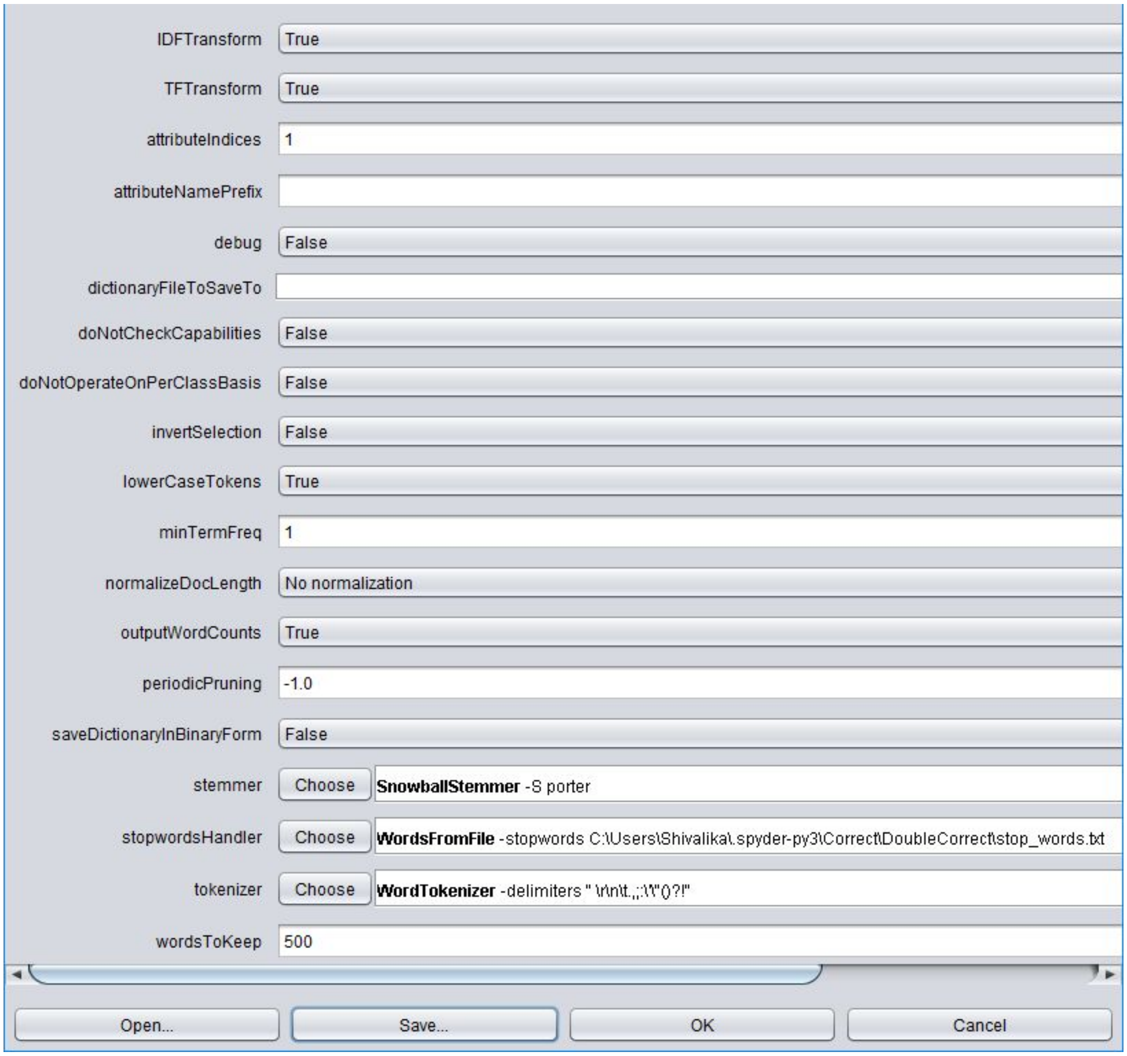
1. Downloaded the train and test data which is in CSV format.
2. Removed the id attribute from both the files since it would not be used for the classification (but kept record of it for later use).
3. Used a python script names **removeOtherStopWords.py** which goes through all the words in the review and remove the words which are not alpha like $, $40, 2pm,!, ' etc. This file is named **train_otherStopWordsRemoved.csv**
4. Loaded the file onto Weka and saved it in the .arff format. Made some changes to .arff as it had some messed up data. Also changed the class attribute into label as class is also a word in one of the reviews and therefore complaining about repetitions of attribute names. This file is named **train_otherStopWordsRemoved_1.arff**
5. Convert all the attributes except the class/label attribute to String type. To do so, we need to apply filter "**NominalToString**". To do that, open **train_otherStopWordsRemoved_1.arff** file into Weka and follow the path: Weka –> Filters –> Unsupervised –> attribute –> NominalToString.
   After selecting NominalToString filter, click the text area of chosen filter to open GenericObjectEditor and specify "attributeIndexes" to all attributes (except class attribute) and click OK and apply the filter. Save the new data into new file as **train_nominalToString_2.arff**



6. Repeat steps 3 to 5 on test data as well. Name the final test data file as **test_nominalToString_2.arff**
7. Now open both **train_nominalToString_2.arff** and **test_nominalToString_2.arff** files in text editor and create a new arff file by first copying everything from **train_nominalToString_2.arff** and then copying @data from

test_nominalToString_2.arff. Thus we combined the @data from both files into one file and saved it as **combined_data.arff**

8. Now open the **combined_data.arff** file in Weka. For tokenizing we need to use the **"StringToWordVector"** filter. The path for this filter is Weka –> Filters –> Unsupervised –> attribute –> StringToWordVector. Specify the settings as below:



We used the above values. We converted all the words into lower case and set outputWordCounts = true. We removed the stop words provided in the Project description using stopwordsHandler option. We performed Porter stemming using stemmer option. We had to install SnowballStemmer for this. At last, we performed TF-IDF and set wordsToKeep = 500. wordsToKeep is important because it keeps the most/top useful words. This is same as performing feature selection. We applied attributeIndices = 1, where 1 means "text" column. We apply the filter with these settings and save the result into the file **combined_data_StringToWordVector.arff.** The filter return 658 attributes for 50,000 instances (since data of train and test is combined). It is 658 instead of 500 attributes because it took 500 attributes for all three class values - positive, negative and neutral, but a lot of attributes were present in more than one category and thus there was an overlap and the resulting attributes came out to be 658. Also, save the result as **combined_data_StringToWordVector.csv** (saving in csv file because will need to separate the train and test data, and doing that in arff file is difficult).

9. Open **combined_data_StringToWordVector.csv** and copy the rows which has "label values = negative or positive or neutral" into file **final_train.csv** and copy the rows with "label values = ?" into file **final_test.csv**, along with the header row with all the attribute names for both files.
10. Now open **final_train.csv** in Weka and save it as **final_train.arff**. Similarly, open **final_test.csv** in Weka and save it as **final_test.arff.**

Thus the above preprocessing steps, gives us final files for train and test data as **final_train.arff** and **final_test.arff** respectively. Both files have 658 attributes, where **final_test.arff** has "label values being equal to ?" and **final_train.arff** has "label values being negative or positive or neutral".

10-fold Cross Validation

Open **final_train.arff** file on Weka and run 10 fold cross-validation for 5 different algorithms to find the error rate and accuracy by evaluating the misclassification rate using confusion matrix for each algorithm. We choose - C4.5, Random Tree, Naïve Bayesian, k-Nearest and Random Forest. This will help us to find out which algorithm is best to make the model for classification.

Below is the 10-fold cross validation summary for each algorithms along with the description why these were chosen for cross validation-

C4.5 (weka.classifier.trees.J48)

This learning algorithm is used for generating pruned or unpruned C4.5 decision tree. It is good at handling both continuous and discrete attributes, training data with any missing attribute values and, is useful when there are attributes with differing costs.

The given image shows the summary and cross validation for the algorithm C4.5 for the train data.

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

Correctly Classified Instances       26155               65.3875 %
Incorrectly Classified Instances     13845               34.6125 %
Kappa statistic                          0.3896
Mean absolute error                      0.2575
Root mean squared error                  0.4461
Relative absolute error                 65.9559 %
Root relative squared error            100.9606 %
Total Number of Instances            40000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.624    0.146    0.613      0.624    0.618      0.475    0.748     0.522     negative
                 0.799    0.360    0.737      0.799    0.767      0.446    0.735     0.709     positive
                 0.227    0.097    0.326      0.227    0.268      0.151    0.565     0.238     neutral
Weighted Avg.    0.654    0.257    0.633      0.654    0.641      0.403    0.709     0.578
```

Random Tree (weka.classifiers.trees.RandomTree)

This algorithm is easy to interpret and explain. The random tree is part of decision tree which can easily handle feature interactions and there is no need to worry about whether there are any outliers or we have a linearly separable data.

The given image shows the summary and cross validation for the algorithm Random Tree for the train data.

```
=== Summary ===

Correctly Classified Instances       22505               56.2625 %
Incorrectly Classified Instances     17495               43.7375 %
Kappa statistic                          0.2484
Mean absolute error                      0.2923
Root mean squared error                  0.5395
Relative absolute error                 74.852  %
Root relative squared error            122.0921 %
Total Number of Instances            40000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.493    0.180    0.503      0.493    0.498      0.315    0.657     0.386     negative
                 0.695    0.412    0.681      0.695    0.688      0.284    0.642     0.644     positive
                 0.242    0.150    0.251      0.242    0.246      0.094    0.546     0.191     neutral
Weighted Avg.    0.563    0.304    0.559      0.563    0.561      0.260    0.630     0.497
```

Naïve Bayesian (weka.classifier.bayes.NaiveBayes)

Naive Bayes works really well with text classification. It is a simple classification method based on Bayes rule. It relies on very simple representation of document, i.e., bag of words. It is fast and robust to irrelevant features, and is very optimal if the independence assumptions hold.

The given image shows the summary and cross validation for the algorithm Naive Bayesian for the train data.

```
=== Summary ===

Correctly Classified Instances       23545               58.8625 %
Incorrectly Classified Instances     16455               41.1375 %
Kappa statistic                          0.3719
Mean absolute error                      0.2743
Root mean squared error                  0.5161
Relative absolute error                 70.2429 %
Root relative squared error            116.8106 %
Total Number of Instances            40000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.645    0.155    0.607      0.645    0.625      0.481    0.793     0.602     negative
                 0.561    0.141    0.834      0.561    0.671      0.431    0.784     0.812     positive
                 0.591    0.285    0.300      0.591    0.398      0.243    0.691     0.283     neutral
Weighted Avg.    0.589    0.169    0.681      0.589    0.612      0.412    0.771     0.665
```

IBK: k-Nearest Neighbor (weka.classifier.lazy.IBk)

K-nearest neighbor supports both classification and regression. The algorithm is based on Instance Based Learning. It stores entire training set and queries data set by locating the k most similar training patterns when making a prediction. Training the dataset is fast and also no data loss occurs.

The given image shows the summary and cross validation for the algorithm k-Nearest Neighbor for the train data.

```
=== Summary ===

Correctly Classified Instances       22180               55.45   %
Incorrectly Classified Instances     17820               44.55   %
Kappa statistic                          0.2167
Mean absolute error                      0.2984
Root mean squared error                  0.5435
Relative absolute error                 76.4171 %
Root relative squared error            123.0112 %
Total Number of Instances            40000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.484    0.210    0.461      0.484    0.472      0.270    0.640     0.366     negative
                 0.704    0.466    0.657      0.704    0.680      0.242    0.621     0.629     positive
                 0.177    0.105    0.259      0.177    0.211      0.085    0.535     0.188     neutral
Weighted Avg.    0.555    0.335    0.536      0.555    0.543      0.223    0.611     0.483
```

Random Forest (weka.classifier.trees.RandomForest)

This is a supervised classification algorithm and used for classification as well as regression. It chooses root node and where the feature nodes are split randomly. The classifier handles missing values and if we have more data, it would not over-fit it.

The given image shows the summary and cross validation for the algorithm Random Forest for the train data.

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

Correctly Classified Instances       28758               71.895 %
Incorrectly Classified Instances     11242               28.105 %
Kappa statistic                          0.4515
Mean absolute error                      0.2953
Root mean squared error                  0.365
Relative absolute error                 75.6135 %
Root relative squared error             82.6118 %
Total Number of Instances            40000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.664    0.069    0.781      0.664    0.718      0.629    0.907     0.801     negative
                 0.954    0.511    0.703      0.954    0.809      0.513    0.874     0.890     positive
                 0.040    0.006    0.564      0.040    0.074      0.115    0.754     0.376     neutral
Weighted Avg.    0.719    0.305    0.700      0.719    0.659      0.476    0.862     0.778
```

Confusion Matrix

Based on the data collected from 10-fold cross validation, the confusion matrix calculations for each algorithm is as follows:

C4.5

Class	**Negative**	**Positive**	**Neutral**	**Overall Classification**	**Producer Accuracy (Precision)**
Negative	6739	2816	1251	10806	62.36%
Positive	2511	17858	1972	22341	79.93%
Neutral	1752	3543	1558	6853	22.74%
Overall Truth	11002	24217	4781	40000	
User Accuracy (Recall)	61.25%	73.74%	32.59%		

- **Overall Accuracy = 65.39%**
- **Misclassification Rate = 13845 / 40000 = 34.61%**
- **Kappa = 0.39**

Random Tree

Class	Negative	Positive	Neutral	Overall Classification	Producer Accuracy (Precision)
Negative	5328	3790	1688	10806	49.31%
Positive	3550	15519	3272	22341	69.46%
Neutral	1716	3479	1658	6853	24.19%
Overall Truth	10594	22788	6618	40000	
User Accuracy (Recall)	50.29%	68.10%	25.05%		

- **Overall Accuracy = 56.26%**
- **Misclassification Rate = 17495 / 40000 = 43.74%**
- **Kappa = 0.248**

Naïve Bayesian

Class	Negative	Positive	Neutral	Overall Classification	Producer Accuracy (Precision)
Negative	6969	1071	2766	10806	64.49%
Positive	3119	12529	6693	22341	56.08%
Neutral	1392	1414	4047	6853	59.05%
Overall Truth	11480	15014	13506	40000	
User Accuracy (Recall)	60.71%	83.45%	29.96%		

- **Overall Accuracy = 58.86%**
- **Misclassification Rate = 16455 / 40000 = 41.14%**
- **Kappa = 0.372**

IBK: k-Nearest Neighbour

Class	Negative	Positive	Neutral	Overall Classification	Producer Accuracy (Precision)
Negative	5229	4403	1174	10806	48.39%
Positive	4301	15735	2305	22341	70.43%
Neutral	1817	3820	1216	6853	17.74%
Overall Truth	11347	23958	4695	40000	
User Accuracy (Recall)	46.08%	65.68%	25.9%		

- **Overall Accuracy = 55.45%**
- **Misclassification Rate = 17820 / 40000 = 44.55%**
- **Kappa = 0.217**

Random Forest

Class	Negative	Positive	Neutral	Overall Classification	Producer Accuracy (Precision)
Negative	7180	3508	118	10806	66.45%
Positive	943	21306	92	22341	95.37%
Neutral	1067	5514	272	6853	3.97%
Overall Truth	9190	30328	482	40000	
User Accuracy (Recall)	78.13%	70.26%	56.43%		

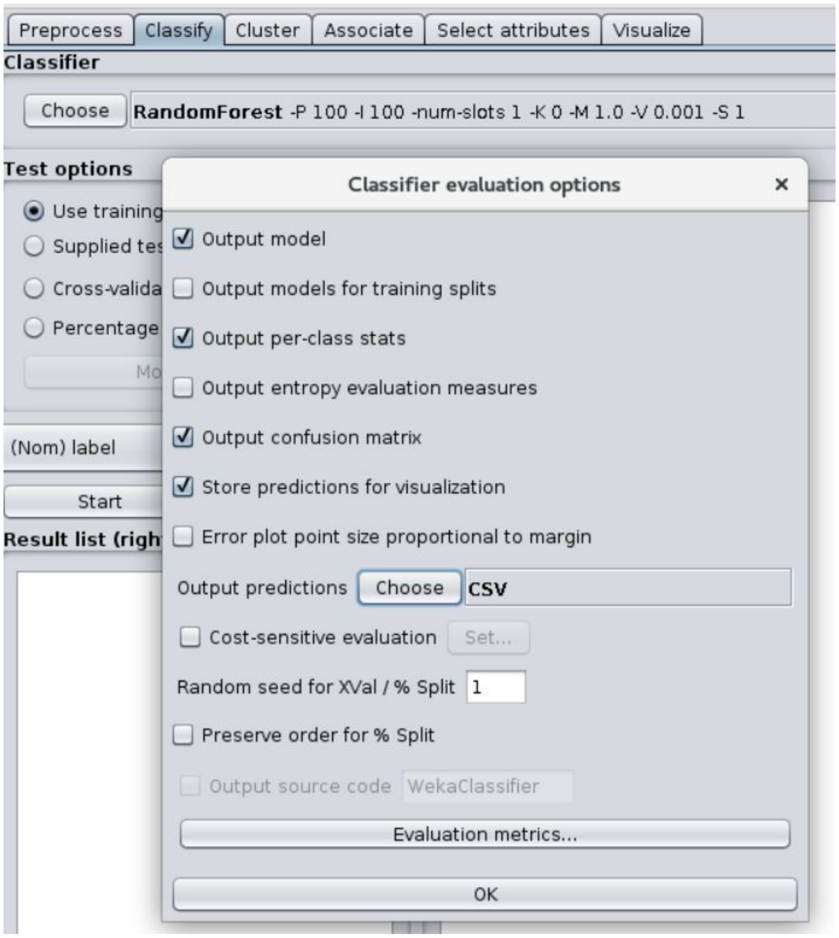
- **Overall Accuracy = 71.89%**
- **Misclassification Rate = 11242 / 40000 = 28.11%**
- **Kappa = 0.451**

Classification

Based on the Confusion matrix calculations, **Random Forest** is the best algorithm for classification because among all the 5 algorithms. It has the highest accuracy of 71.89% and lowest misclassification rate of 28.11%. So we chose Random Forest to build our model on training data. Following are the steps for classification:

Training Data

1. Open **final_train.arff** file in Weka and go to Classify tab. Click Choose □ Weka □ trees □ **Random Forest**. Under "Test Options" select "Use training set" and select the following options from "More options…" button

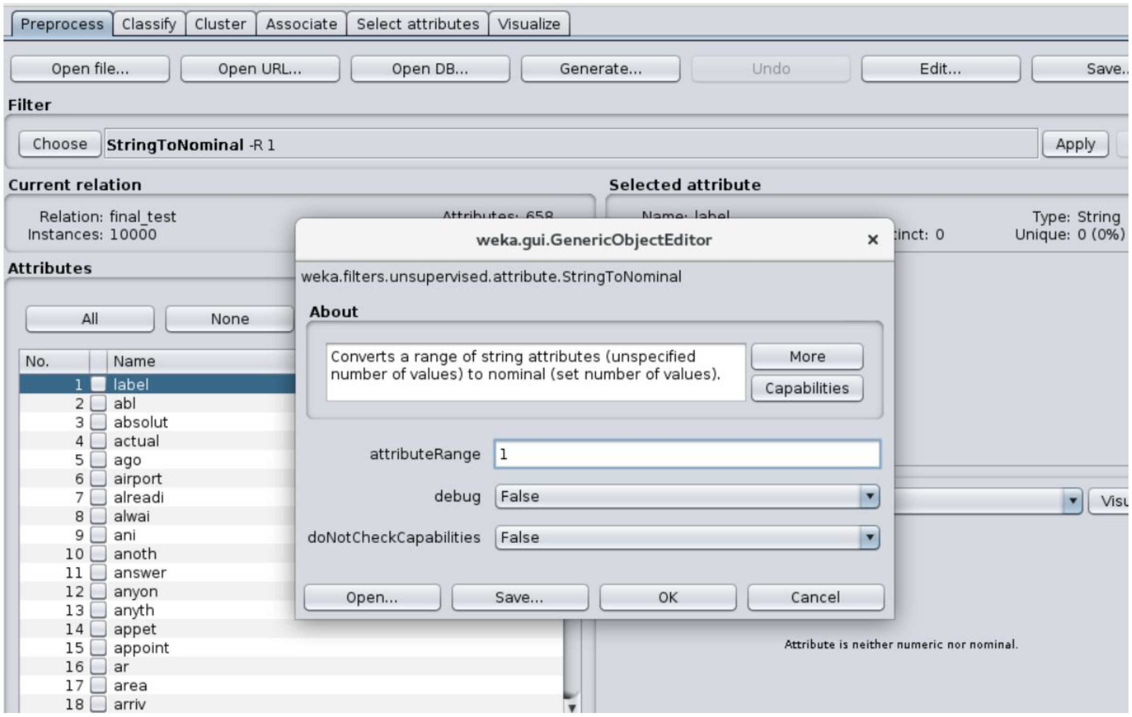
   

   Select the class attribute "**(Nom) label**" from drop-down and click Start button.

2. Once Status is OK, right-click inside the "Result list" and click "Save model" and save it as **RandomForest.model.** This ensures that the model is trained on the training data set and can we used for testing and future data sets.

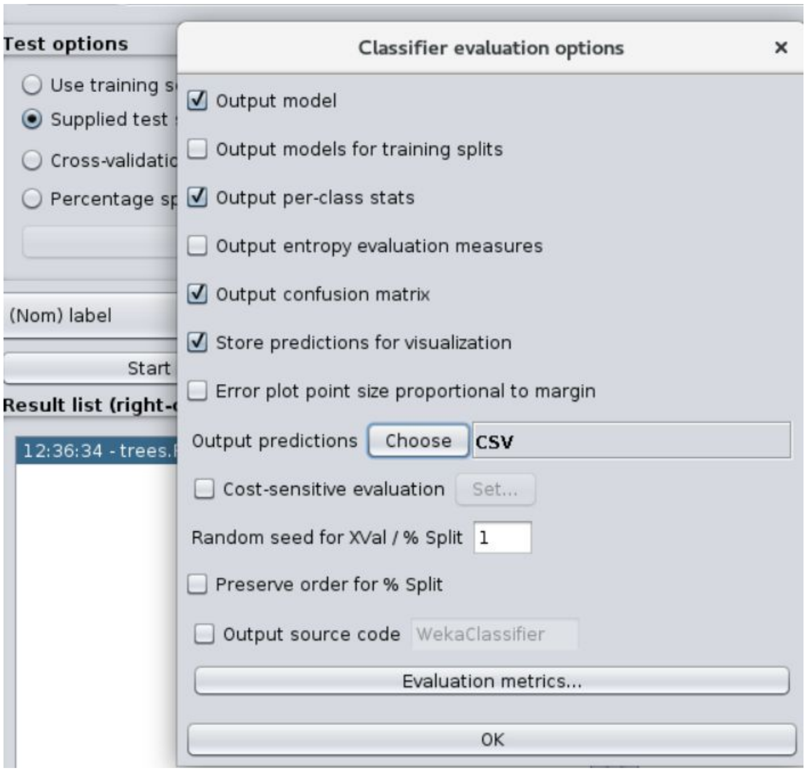
Testing Data

1. Open **final_test.arff** file in Weka and apply filter “**StringToNominal**” on class attribute “label” as attributeIndices = 1 and save the file as **final_test_2.arff**



2. Load the previously created model under the “Classify” tab. Right-click on “Result list” and click “Load model” and select the saved model “**RandomForest.model**”.
3. Now we need to make predictions on testing data. On “Classify” tab, select “Supplied test set” option in “Test Options” pane. Click the “Set” button, click “open file” button and navigate to **final_test_2.arff** file and select it. Select “**(Nom) label**” class attribute from the drop down menu. Click the “Close” button on the window.

4. Click the “More options...” button to bring up options for evaluating the classifier and do the following settings:



Right-click on the list item for your loaded model in the “Results list” pane. Select “Re-evaluate model on current test set”. The predictions for each test instance are then listed in the “Classifier Output” pane.

5. Right-click on the “Result list” and click on “Save Result Buffer” and save **test_classify_output.csv**

Creating final files

1. Open the **test_classify_output.csv** file in Excel and copy the entire “predictions on user test set” section into another file. Open the original “**test.csv**” file and copy the “**ID**” column into the file. Save the file as “**output.csv**” file

inst#	actual	predicted	error	prediction
1	1:?	2:positive		0.48
2	1:?	2:positive		0.49
3	1:?	2:positive		0.87
4	1:?	2:positive		0.8
5	1:?	2:positive		0.94
6	1:?	1:negative		0.8
7	1:?	2:positive		0.59
8	1:?	2:positive		0.69
9	1:?	2:positive		0.45
10	1:?	2:positive		0.67
11	1:?	1:negative		0.62
12	1:?	2:positive		0.69
13	1:?	1:negative		0.45

Fig. Screenshot of predictions on user test set

2. Open the **output.csv** file. Edit the "predicted" column values by removing "1:", "2:", and "3:" from each values so that we are left with class values as "negative", "positive", or "neutral". Once done with editing, copy the columns "predicted" and "ID" and paste it to new file and save it as **"prediction.csv"**. Rename the columns predicted as "CLASS" and ID as "REVIEW-ID" in the **predicted.csv** file and save it. Also save the file **prediction.csv** as tab delimited and then save it as "**prediction.txt**" file, which is the final file for predicted values for test data.
3. Open the **"final_test_2.arff"** file in Weka and save it as csv file. Open the **final_test_2.csv** file and add the "ID" column from the original **"test.csv"** file and copy the class values from **prediction.csv** file into the label column (which has values as ?). Save the file as **final_test_3.csv**. Open the **final_test_3.csv** file in Weka and save it as "**test_output.arff**" file, which is the final file for the testing data.
4. Open the **"final_train.arff"** file in Weka and save it as csv file. Open the **final_train.csv** file and add the "ID" column from the original **"train.csv"** file. Save the file as **final_train_2.csv**. Open the **final_train_2.csv** file in Weka and save it as "**train_output.arff**" file, which is the final file for the training data.

Conclusion

After performing Random Forest Algorithm on the given test data, we conclude that out of 10,000 reviews,

- 7583 are positive (75.83%)
- 2278 are negative (22.78%)
- 139 are neutral (1.39%)

With our chosen model, we were able to achieve 100% accuracy, which means misclassification rate was 0.

We got:

- Kappa Statistic as 1
- Mean absolute error as 0.2404
- Root mean squared error as 0.2823

Most of the reviews in the Yelp were positive, with less than 25% being negative and only a few neutral.

Source Code

removeOtherStopWords.py

```
import csv
import os
import nltk

#
# this program goes through all the reviews/text in the train/test file and remove the words/characters which are not alpha
#

# test_otherStopWordsRemoved.csv is our output file
# change the path of the file accordingly, we used ours..
exists =
os.path.isfile('C:\\Users\\Shivalika\\.spyder-py3\Correct\DoubleCorrect\test_otherStopWordsRemoved.csv')
if exists:
   # if the output file test_otherStopWordsRemoved.csv exists, then remove it
   os.remove("test_otherStopWordsRemoved.csv")
# if the output file test_otherStopWordsRemoved.csv does not exist, then create one
f= open("test_otherStopWordsRemoved.csv","w+")
# test.csv is our input file, to read from
with open('test.csv', 'r') as csvfile:
   csv_reader = csv.reader(csvfile, delimiter=',')
   row_count = 0
   for row in csv_reader:
      if row_count != 0: #this is to not evaluate the first column which is text and class
         # extracting the words into tokens
         tokens = nltk.word_tokenize(row[0])

         result = []
         for a in tokens:
            if a.isalpha():
```

```
            result.append(a) #only adding the tokens which contain all alpha letters to the array
result

        result = " ".join(str(x) for x in result)
        f.write(result+"\n") # writing the result array into the output file
    row_count = row_count + 1
# close both input and output files
csvfile.close()
f.close()
```

Appendix

- To run the program, removeOtherStopWords.py, run the command
  python3 removeOtherStopWords.py
  Give the correct paths for the input and output files. We will get an output file (in the directory mentioned).