**Project**

EECS 4412 – Data Mining

**Submitted By:**

Harpreet Kaur Saini -213321917

Shivalika Sharma – 213963608

Komal Patel - 215306129

**April 3, 2019**

# **Table of Contents**

|  |  |
| --- | --- |
| Table of Contents | 1 |
| Objective | **2** |
| Data Preprocessing | **2** |
| 10-fold cross validation |  |
| Confusion matrix |  |
| classification |  |
| Conclusion |  |
| Source Code |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# **Objective**

The goal of our project is basically that we have to classify Yelp reviews of businesses in the different category. The reviews could be either positive, negative or neutral and based on that the businesses would get better key insight for what their customer has to say about them. Based on the reviews, it would help businesses to make improvement and which could result in better outcomes. The reviews written by customers also provide important information for other customers to make decision about a particular business.

Text mining is concerned with the task of extracting relevant information from natural language text and to search for interesting relationships between the extracted entities. Text classification is one of the basic techniques in the area of text mining. It is one of the more difficult data-mining problems, since it deals with very high-dimensional data sets with arbitrary patterns of missing data.

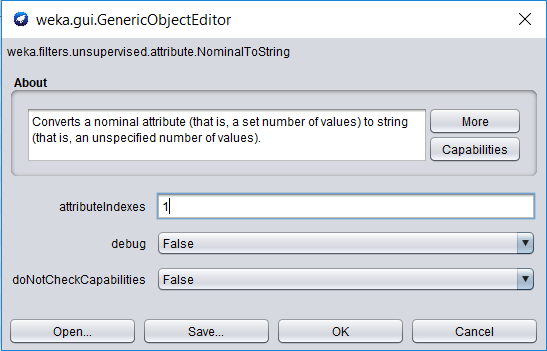
# **Data Preprocessing**

Data preprocessing involves stop word removal, stemming using Porter Stemmer, converting the training and testing data into a relational table (where words are attributes and the frequency or TF-IDF value of a word in a review is the value of the word for the review) using StringToWordVector filter. Also applied NominalToString filter prior to StringToWordVector filter, so as to convert the attribute “text” into String type to apply StringToWordVector filter on “text” attribute. There were other settings and python code executed to preprocess training and testing data. Prior to applying StringToWordVector filter, we need to combine the data for both training and testing data into one file because if we convert training and testing separately, Weka will extract different features for training and test files, so when we train the classifier on training, we won’t be able to run it on the test dataset since the features are different. So for classification we need to have same attributes/features for both training and testing data. Thus we combine the both files before applying StringToWordVector filter to extract attributes from combined files, which will make training and testing files compatible during classification.

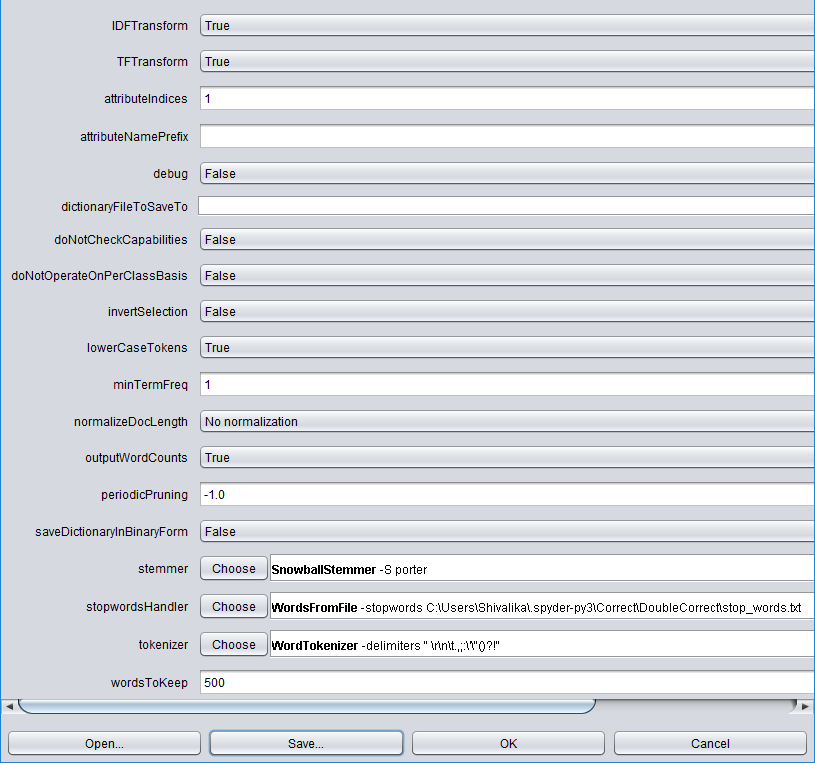
Following are the steps executed for data preprocessing in detail:

1. Downloaded the train data and test data which is in CSV format.
2. Removed the id attribute from both the files since it would not be used for classification.
3. Wrote a script in python **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 attribute to String type. To do so, we need to apply filter “**NominalToString**”. Therefore, open **train\_otherStopWordsRemoved\_1.arff** file in 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**

****

1. Now open both train\_nominalToString\_2.arff and test\_nominalToString\_2.arff files in text editor and create new arff file by copying everything from train\_nominalToString\_2.arff and then copy @data from test\_nominalToString\_2.arff. Thus we combine the @data from both files into one and save as **combined\_data.arff**
2. 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 and specify the settings as below:=> Opened .arff file version into Weka and choose the filter StringToWordVector (more information here about where to find this option and what options were specified)



We used these values. We converted all the words into lower case and set outputWordCounts = true. Removed the stop words provided in Project description using stopwordsHandler option and did Porter stemming using stemmer option. At last, we also performed TF-IDF and set wordsToKeep = 500, which means that we are already performing feature selection here by saying to keep top 500 words. And we applied the filter to attributeIndices = 1, which is @attribute = text. With all the above setting, we apply the filter 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).

Also save the result as **combined\_data\_StringToWordVector.csv**

1. 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 attribute names for both files.
2. 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. For both files has 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”.

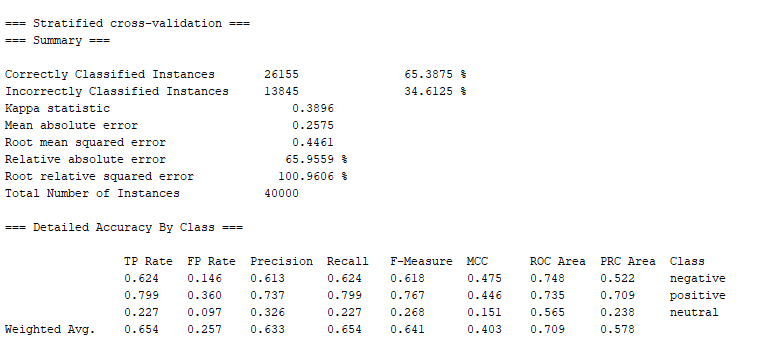
# **10-fold Cross Validation**

Open **final\_train.arff** file on Weka and run the 10 fold cross-validation for 5 different algorithms, namely, C4.5, Random Tree, Naïve Bayesian, k-Nearest, and Random Forest to find the error rate and accuracy by evaluating the misclassification using confusion matrix for each algorithm. 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:

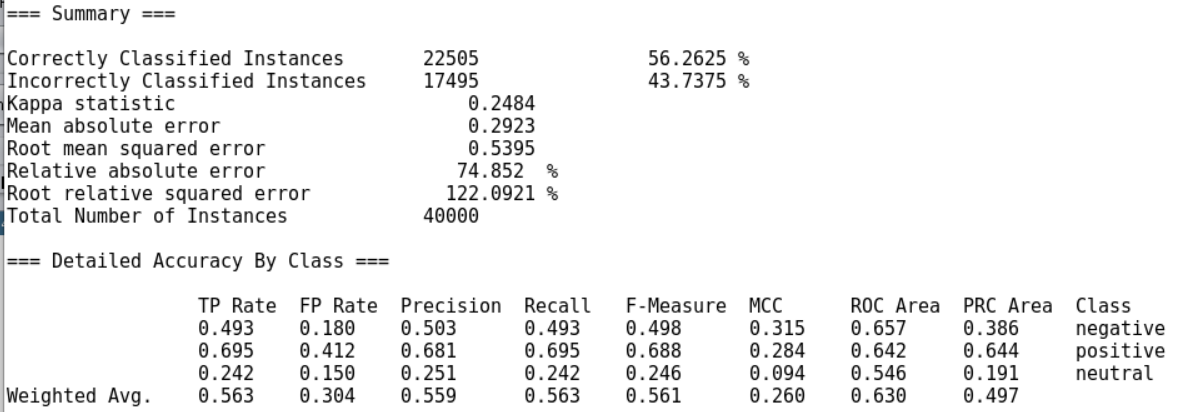
## **C4.5 (weka.classifier.trees.J48)**

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 useful when there are attributes with differing costs.



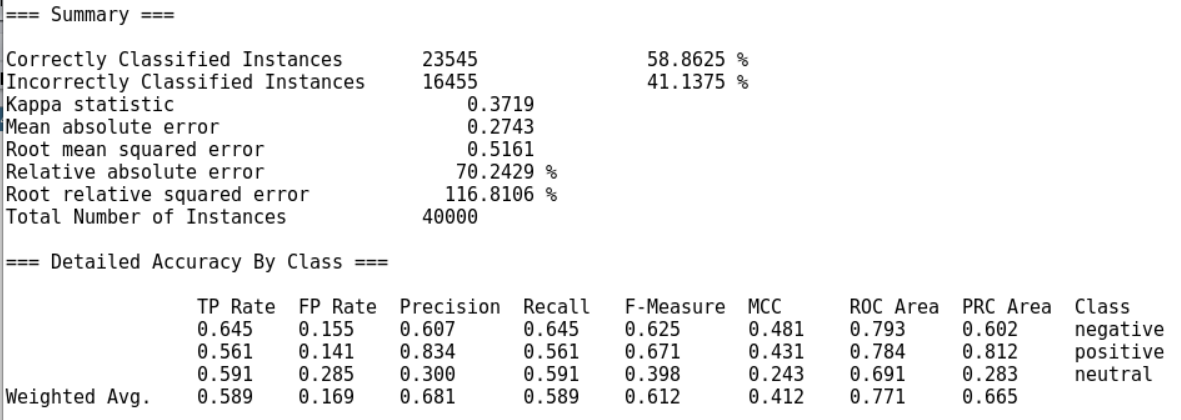
## **Random Tree (weka.classifiers.trees.RandomTree)**

It 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.



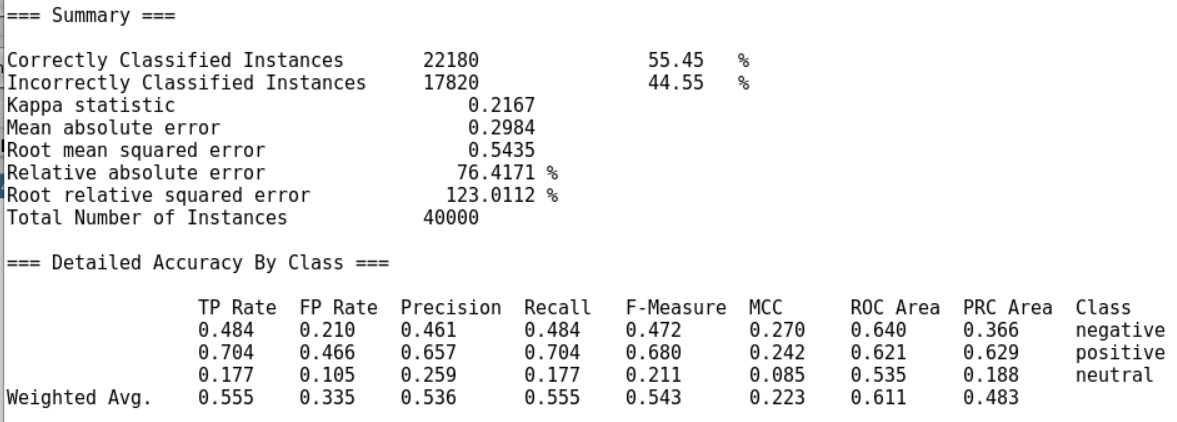
## **Naïve Bayesian (weka.classifier.bayes.NaiveBayes)**

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.



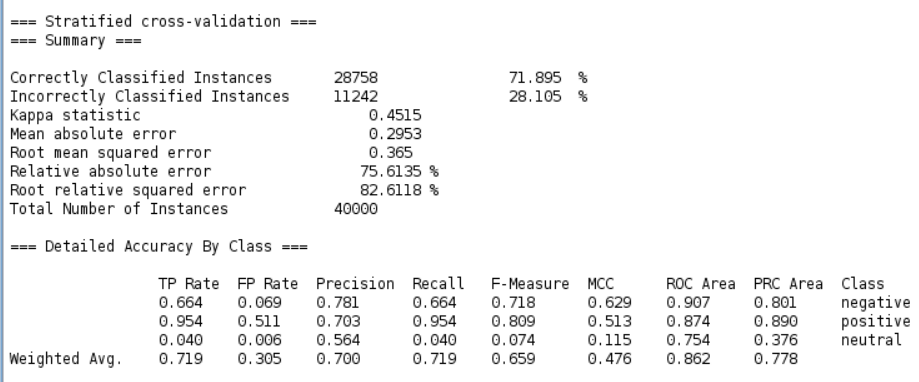
## **IBK: k-Nearest Neighbor (weka.classifier.lazy.IBk)**

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.



## **Random Forest (weka.classifier.trees.RandomForest)**

It 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 the model.



# **Confusion Matrix**

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

## **J48**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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% |  |  |

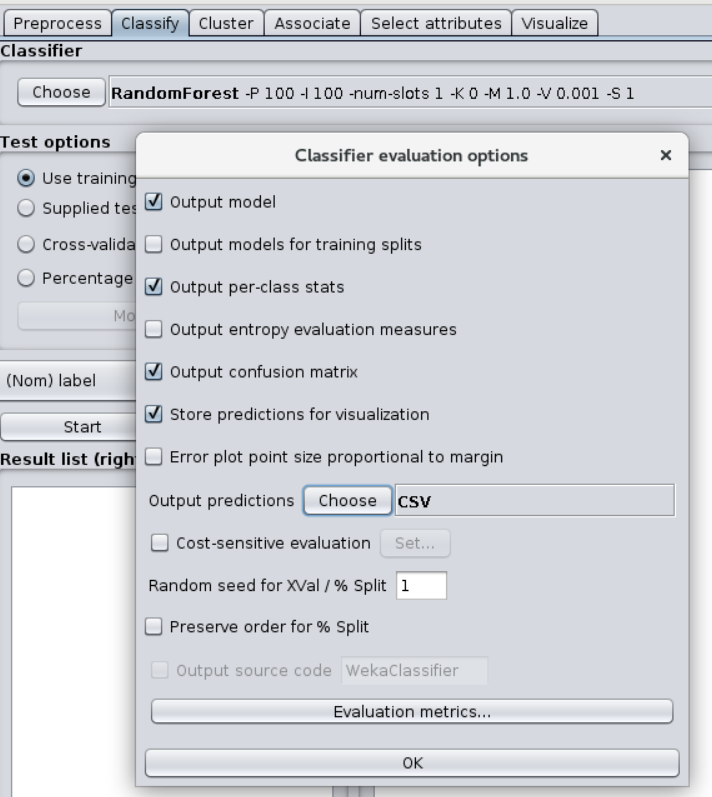
* **Overall Accuracy = 71.89%**
* **Misclassification Rate = 11242 / 40000 = 28.11%**
* **Kappa = 0.451**

# **Classification**

Based on the Confusion matrix calculations, Random Forest is the best algorithm for classification because among all 5 algorithms, Random Forest has highest accuracy of 71.89% and lowest misclassification rate of 28.11%. So we choose 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



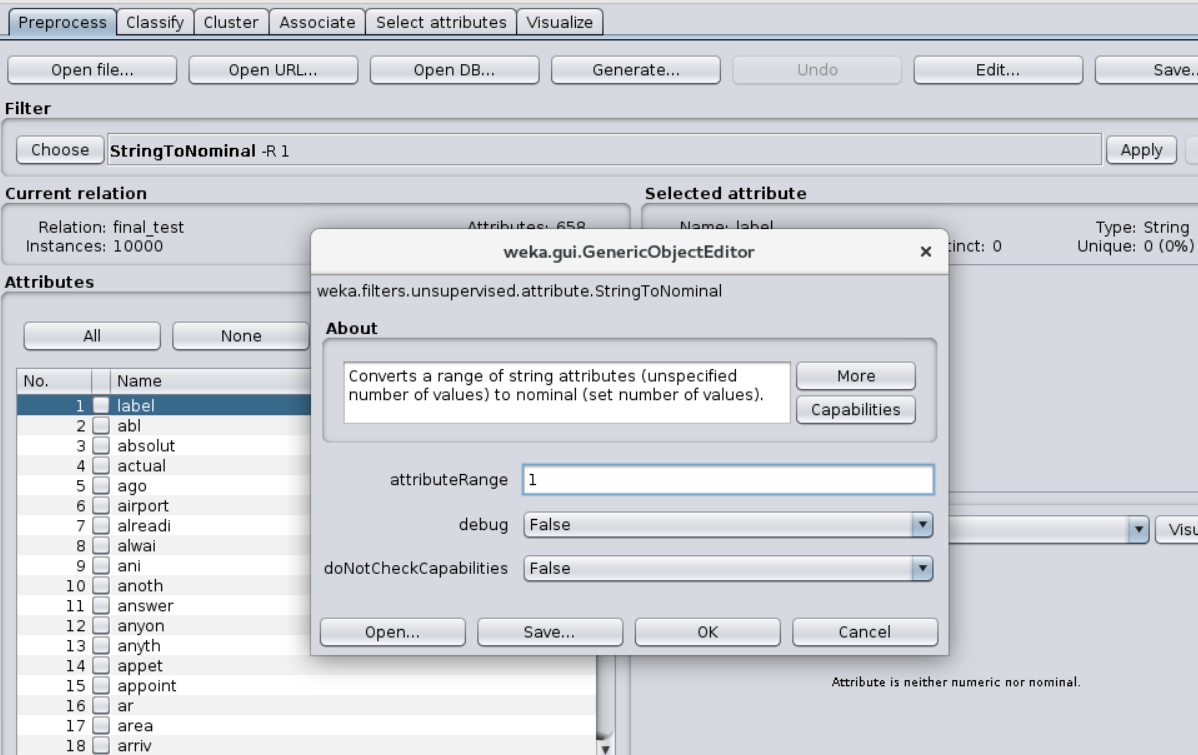
Select the class attribute “**(Nom) label**” from drop-down and click Start button.

1. Once Status is OK, right-click inside the “Result list” and click “Save model” and save it as **RandomForest.model**

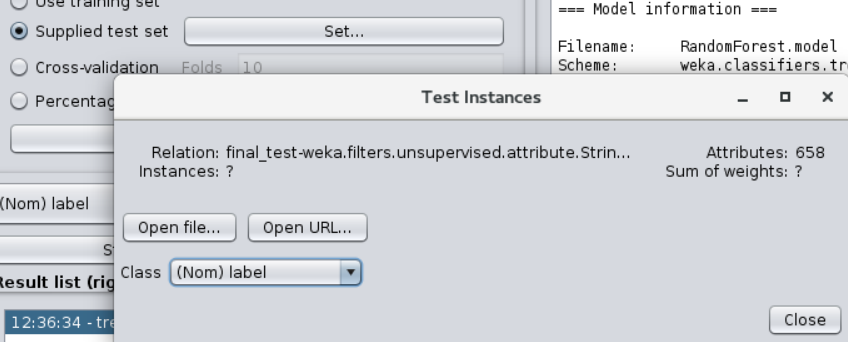
This ensures that the model is trained on the training data set and can we used for testing and future data sets.

**Testing Data**

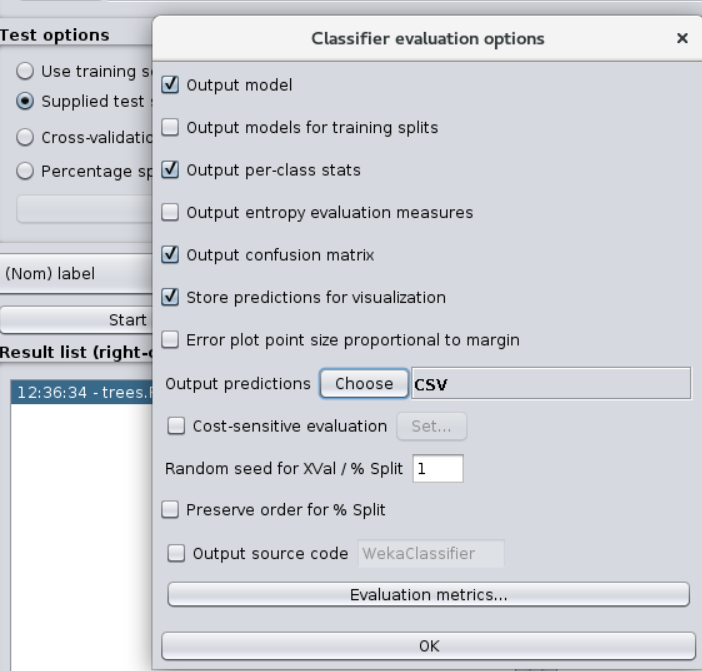
1. Open **final\_test.arff** file in Weka and apply filter “**StringToNominal**” on class attribute “label” as attributeIndices = 1 and save the file as **final\_test\_2.arff**



1. 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**”.
2. 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 and select “**(Nom) label**” class attribute. Click the “Close” button on the window.



1. 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.

1. Right-click on the “Result list” and click on “Save Result Buffer” and save **test\_classify\_output.csv**
2. Open the **test\_classify\_output.csv** file in Excel and copy the entire prediction section

# **Conclusion**

# **Source Code**

## **removeOtherStopWords.py**

import csv

import os

import nltk

exists = os.path.isfile('C:\\Users\\Shivalika\\.spyder-py3\Correct\train\_otherStopWordsRemoved.csv')

if exists:

   os.remove("train\_otherStopWordsRemoved.csv")

f= open("train\_otherStopWordsRemoved.csv","w+")

with open('train.csv', 'r') as csvfile:

   csv\_reader = csv.reader(csvfile, delimiter=',')

   row\_count = 0

   for row in csv\_reader:

       if row\_count != 0:

           # extracting the words into tokens

           tokens = nltk.word\_tokenize(row[0])

           result = []

           for a in tokens:

               if a.isalpha():

                   result.append(a)

           result = " ".join(str(x) for x in result)

#            print (tokens)

#            print ("---------------------------")

#            print (result)

#            print ("===========================")

           f.write(result+"\n")

       row\_count = row\_count + 1

csvfile.close()

f.close()