BOSTON UNIVERSITY

CLASS PROJECT

Course: MET CS 699 OL Data Mining

(Fall 2 2015)

Author: Sherwin Clarke | Submission date: 8th Dec., 2015

Abstract

A dataset that has a class distribution of two values is said to be class-imbalanced if the main class of interest is underrepresented, that is, represented by only a few tuples, while the other class is represented by the majority of tuples (Han, Kamber & Pei, 2012). An example is a dataset of medical records of patients where “yes” and “no” are class labels that indicate if a person has diabetes and if a person does not have diabetes respectively. In such as dataset the events of interest would be tuples with class label of “yes”. One of the common problems of using a classifier model to predict events of interest in a class-imbalanced dataset is that the classification accuracy of the event of interest tends to be poor while the accuracy of the other class tends to be high.

In this project we examined how to improve the predictive accuracy of a class-imbalanced dataset. We used two techniques to balance the dataset, essentially creating two new datasets, and then we created classification models from those two datasets using four different classifiers. Subsequently, we tested those models on the initial dataset to assess their predictive accuracy in terms of the true positive (TP) rate and observed the results. We started out first by establishing a baseline where we tested the predictive accuracy of each of the four models used on the initial class-imbalanced dataset.

Contents

1. Methodology

This section describes the process used to build the four classifier models with the initial dataset and the oversampled dataset and the under sampled dataset. It also describes how each classifier model was tested.

1.1 Building the classifier models from the initial dataset

The first set of activities were to build the four classifiers from the initial dataset. The initial dataset file "bank-cs699.arff was loaded into Weka Explorer and under the Classify Tab, the Naïve Bayes classifier was selected. The Cross-validation algorithm was then selected as the performance test method to be used after which the classifier model was built and tested. Once the classifier was built the performance results were shown in the output window of the Weka Explorer. The results were recorded for later analysis. This procedure was repeated for the remaining 3 classifiers. The actual procedure is found in Annex-1.

1.2 Creating the oversampled dataset

To oversample the initial dataset the Synthetic Minority Oversampling Technique (SMOTE) algorithm was applied to the original dataset. The SMOTE algorithm increases the minority class tuples to a desired level by replicating the tuples of the minority class of the dataset. The aim was to increase the tuples with minority class label so that it has approximately the same number of tuples as tuples of the majority class. The SMOTE algorithm was applied to the initial dataset by loading it into Weka Explorer, selecting SMOTE as the oversampling technique and setting the Percentage parameter to 671.9. This parameter calculates what percent of the minority class should be replicated. Setting it to a value of 671.9 generates a total of 2801 tuples of class label "yes", which essentially balanced the dataset in terms of class distribution. The dataset was then saved as "oversampled-bank.arff". For procedure, see Annex-2.

1.3 Building the classifier models from the oversampled dataset

The oversampled dataset "oversampled-bank-cs699.arff" was loaded into Weka Explorer and under the Classify Tab, the Naïve Bayes classifier was selected. The Cross-validation algorithm was then selected as the performance test method to be used after which the classifier model was built and the model was saved. This procedure was repeated for the remaining 3 classifiers. A total of 4 classifier models were created and saved using the under sampled dataset. See Annex-3 for procedure.

1.4 Creating the under sampled dataset

The dataset was under sampled by using the Spread sub sample algorithm on the initial dataset. The end result was that the number of tuples of the majority class, that is, tuples with the class label "no", decreased to 363, the same number of tuples of the minority class (tuples with class label "yes"). The dataset was the saved as "underampled-bank.arff". The procedure in Annex-4 describes how this was done.

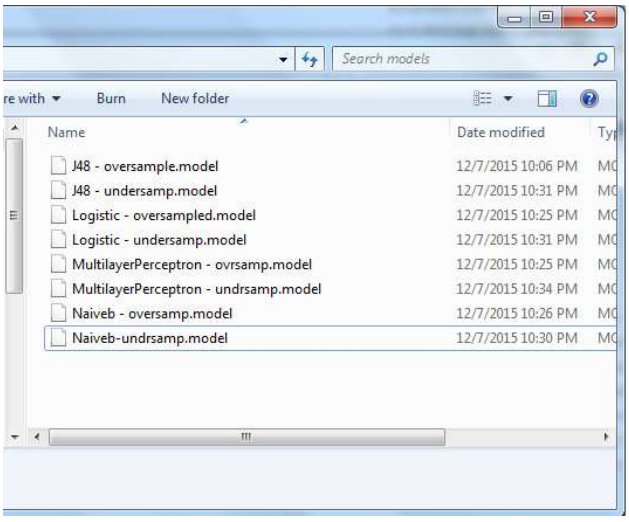
1.5 Building the classifier models from the under sampled dataset

The under sampled dataset "underampled-bank-cs699.arff" was loaded into Weka Explorer and under the Classify Tab, the Naïve Bayes classifier was selected. The Cross-validation algorithm was then selected as the performance test method to be used after which the classifier model was built and tested. After the classifier was built the performance results was shown in the output window of the Weka Explorer. The classifier model was then saved to be used later. This procedure was repeated for the 3 other models. A total of 4 classifier models were created and saved using the under sampled dataset. See Annex-5 for procedure.

1.6 Testing the classifier models on the initial dataset

The initial dataset was loaded into Weka Explorer. Under the Classify tab, Naïve Bayes was selected as the classifier and the classifier was built with default settings and the output shown on the classifier output screen. At this point there was an option to right-click on the Naïve Bayes classifier and load a saved model. The Naïve Bayes model that was built and saved from the oversampled dataset was loaded. Under **Test options**, **Supplied test set** was selected and the initial dataset was selected. The Naïve Bayes model was then executed on the current test set by right-clicking on it and selecting **re-evaluate model on current test set**. Once the performance test was completed the results were recorded from the classifier output window. This procedure was repeated for the remaining 7 classifier that were saved previously (see Fig-1). The actual procedure is described in Annex-6.

Fig-1: Saved models.



2. Performance Test Results

2.1 Test results of models built from initial dataset

Naïve Bayes

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.925	0.548	0.929	0.925	0.927	0.846	No
	0.452	0.075	0.437	0.452	0.444	0.846	Yes
Weighted average	0.87	0.494	0.872	0.87	0.871	0.846	

J48 (Decision Tree)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.959	0.598	0.925	0.959	0.942	0.733	No
	0.402	0.041	0.559	0.402	0.468	0.733	Yes
Weighted average	0.895	0.534	0.883	0.895	0.887	0.733	

MultilayerPerceptron (neural network),

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.946	0.636	0.92	0.946	0.933	0.811	No
	0.364	0.054	0.466	0.364	0.409	0.811	Yes
Weighted average	0.879	0.57	0.868	0.879	0.873	0.811	

Logistic (logistic regression)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.975	0.733	0.911	0.975	0.942	0.881	No
	0.267	0.025	0.581	0.267	0.366	0.881	Yes
Weighted average	0.894	0.652	0.873	0.894	0.876	0.881	

2.2 Test results of models built from oversampled dataset

Naïve Bayes

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.788	0.27	0.958	0.788	0.865	0.832	No
	0.73	0.212	0.309	0.73	0.434	0.832	Yes
Weighted average	0.782	0.263	0.883	0.782	0.815	0.832	

J48 (Decision Tree)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.946	0.226	0.97	0.946	0.958	0.945	No
	0.774	0.054	0.652	0.774	0.708	0.945	Yes
Weighted average	0.927	0.206	0.934	0.927	0.929	0.945	

MultilayerPerceptron (neural network)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.98	0.118	0.985	0.98	0.982	0.949	No
	0.882	0.02	0.853	0.882	0.867	0.949	Yes
Weighted average	0.969	0.107	0.97	0.969	0.969	0.949	

Logistic (logistic regression)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.889	0.377	0.948	0.889	0.917	0.849	No
	0.623	0.111	0.42	0.623	0.502	0.849	Yes
Weighted average	0.858	0.347	0.887	0.858	0.87	0.849	

2.3 Test results of models built from under sampled dataset

Naïve Bayes

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.762	0.198	0.967	0.762	0.852	0.849	No
	0.802	0.238	0.304	0.802	0.441	0.849	Yes
Weighted average	0.767	0.203	0.891	0.767	0.805	0.849	

J48 (Decision Tree)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.814	0.074	0.988	0.814	0.893	0.905	No
	0.926	0.186	0.392	0.926	0.551	0.905	Yes
Weighted average	0.827	0.087	0.92	0.827	0.854	0.905	

MultilayerPerceptron (neural network)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.813	0.003	1	0.813	0.897	0.942	No
	0.997	0.187	0.409	0.997	0.58	0.942	Yes
Weighted average	0.834	0.024	0.932	0.834	0.861	0.942	

Logistic (logistic regression)

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.82	0.171	0.974	0.82	0.891	0.893	No
	0.829	0.18	0.374	0.829	0.516	0.893	Yes
Weighted average	0.821	0.172	0.905	0.821	0.848	0.893	

2.4 Test results of best classification model

MultilayerPerceptron (neural network) created from the under sampled dataset

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.813	0.003	1	0.813	0.897	0.942	No
	0.997	0.187	0.409	0.997	0.58	0.942	Yes
Weighted average	0.834	0.024	0.932	0.834	0.861	0.942	

3. Selection of best classification model

I selected the MultilayerPerceptron (neural network) created from the under sampled dataset as the best classifier because it has the highest true positive (TP) rate of all the classifiers in both the oversampled dataset and the under-sampled dataset.

Even though it does not exhibit the lowest FP rate, the FP rate is of lesser significance than the TP rate in the context of a class-imbalanced dataset where it is more important to correctly classify a tuple as "yes" (e.g. in the case of correctly diagnosing a patient) than incorrectly classifying a tuple as "yes", which would result in a false positive. In other words, a classifier that produces the highest TP rate together with a moderately low FP rate is better than one that produces a moderately high TP rate together with a low FP rate based on the same dataset.

4. Discussion

I have learned that to test a class-imbalanced dataset to produce fairly accurate results the dataset should first be class-balanced and that there are a number of techniques available to do so such as oversampling the dataset or under sampling it. As it relates to oversampling the database, I have learned to use the SMOTE technique to oversample the dataset. It basically adds synthetic tuples of the minority class to the dataset to correct the class-imbalance problem. Similarly, I have learned how to under sample the dataset using the Spread sub sample technique. This technique reduces tuples of the majority class to that of the minority class to correct the class-imbalanced problem.

Additionally, I have gained an appreciation for other classifiers by being able to use them to create classification models from them. Moreover, I have learned how to not only create various classification models but also how to test those models using a different dataset from that which they were built from.

I have learned the importance of using different classifiers to build and test them until some desired outcome is met. And finally this practical learning experience has underscored the importance of creating a classifier model on a dataset and having a separate dataset to test it on to prevent a biased or optimistic performance of the model.

5. Observations

1. The performance test results show that all four classifiers models created from the under sampled dataset produced a higher True Positive (TP) rate as opposed to them being built from the oversampled dataset.

2. The MultilayerPerceptron (neural network) classifier built from the under sampled dataset exhibited the best performance in terms of the TP rate; it produced the highest TP rate.

3. The MultilayerPerceptron (neural network) classifier built from the oversampled dataset exhibited the best overall performance in terms of predictive accuracy

4. The MultilayerPerceptron (neural network) exhibited the highest ROC area under the curve of all the classifiers when tested, which means it has a very good tradeoff between TP rate and FP rate, favoring the TP rate.

5. The J48 (Decision Tree) classifier built from the under sampled database is the second best classifier when tested

6. Logistic (logistic regression) built from the oversampled dataset performed the poorest in terms of TP rate.

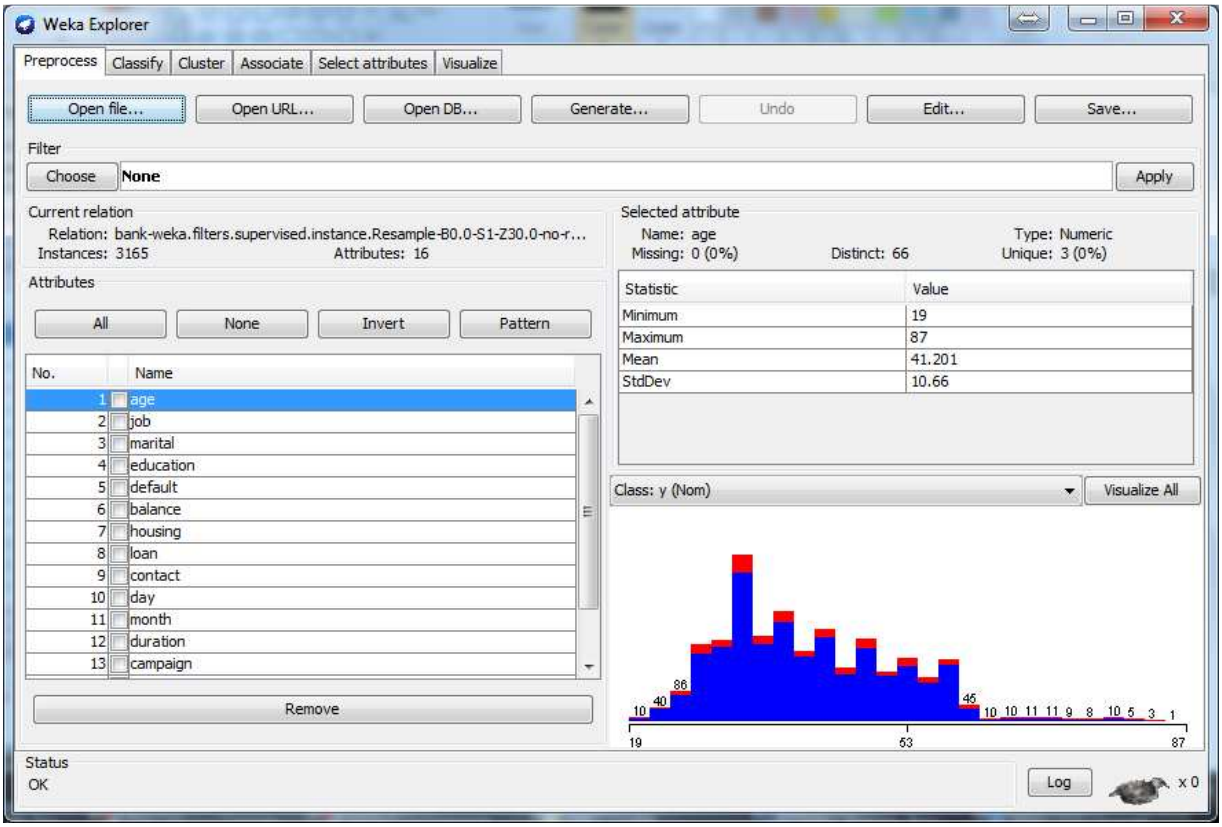
6. Reference

J. Han, M. Kamber, J. Pei, Data Mining Concepts and Techniques (Third edition). Elsevier, 2012.

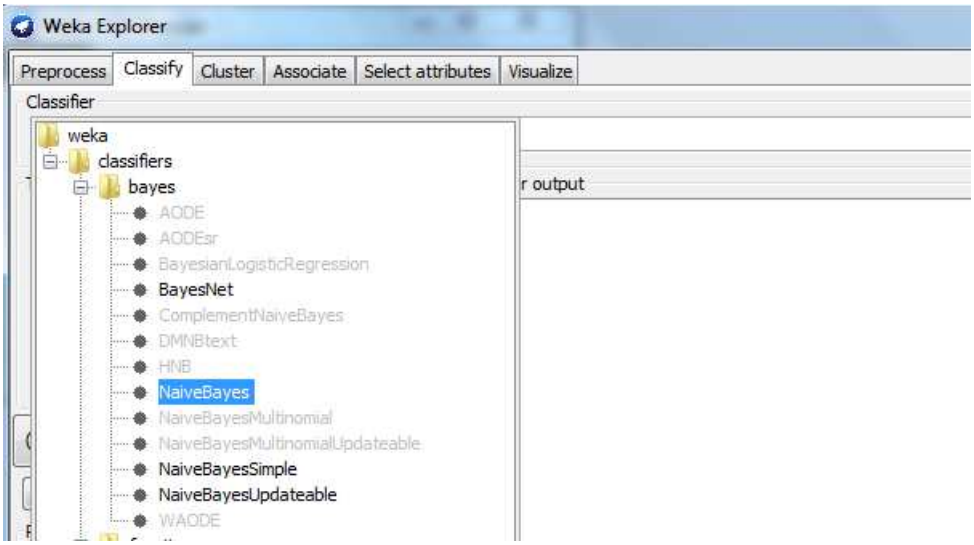
ANNEX 1 – Building the classifiers from the initial dataset

Procedure

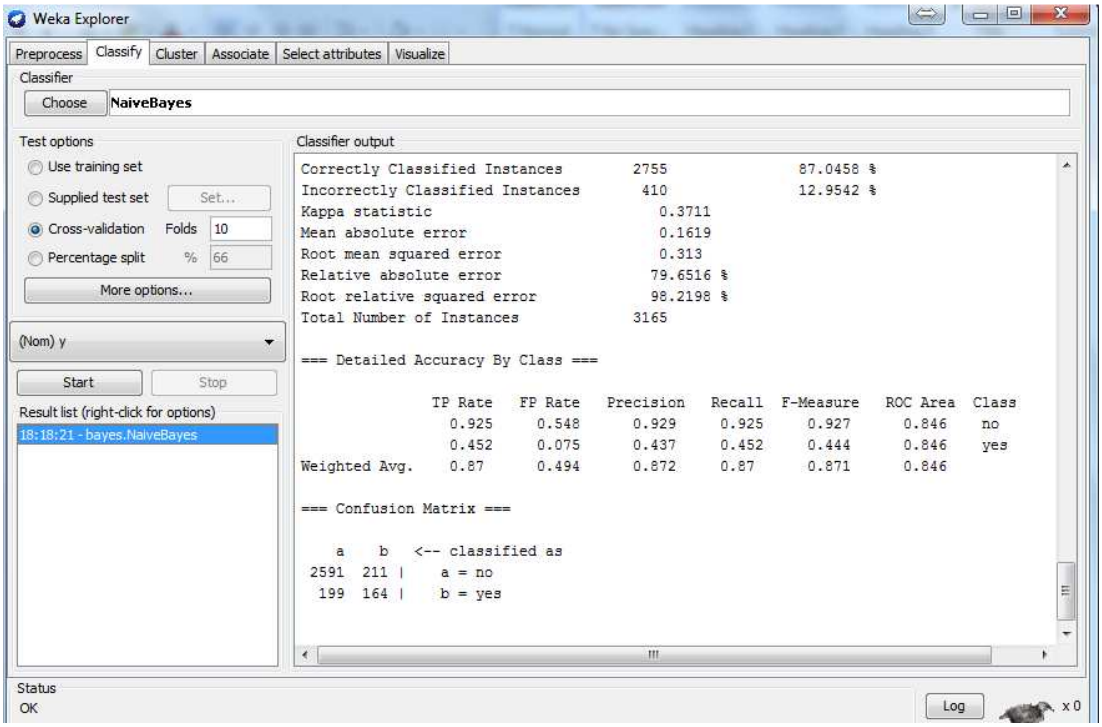
1. Open Weka 3.6 and click on **Explorer**
2. On the Preprocess tab, click on the button **Open file** and load the initial dataset "bank-cs699.arff" from the appropriate location into Weka Explorer



3. Click on the **Classify** tab then click on the button **Choose**. In the location **Weka > Classifiers > Bayes >**, select NaiveBayes



4. In the **Classify** tab, in **Test Options**, select **Cross-Validation**. Leave all else as default and press the **Start** button. Once the classifier is built and tested the results will be shown in the **Classifier Output** screen

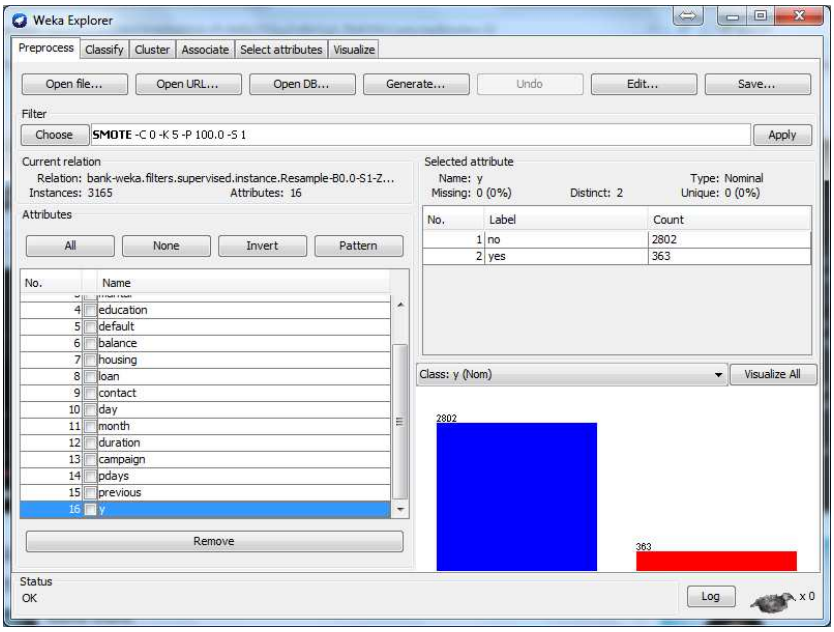


5. Repeat this procedure for the J48 (decision tree), MultilayerPercptron (neural network), and Logistic (logistic regression) classifiers however note that the classifiers are in different locations from the Naïve Bayes classifier.

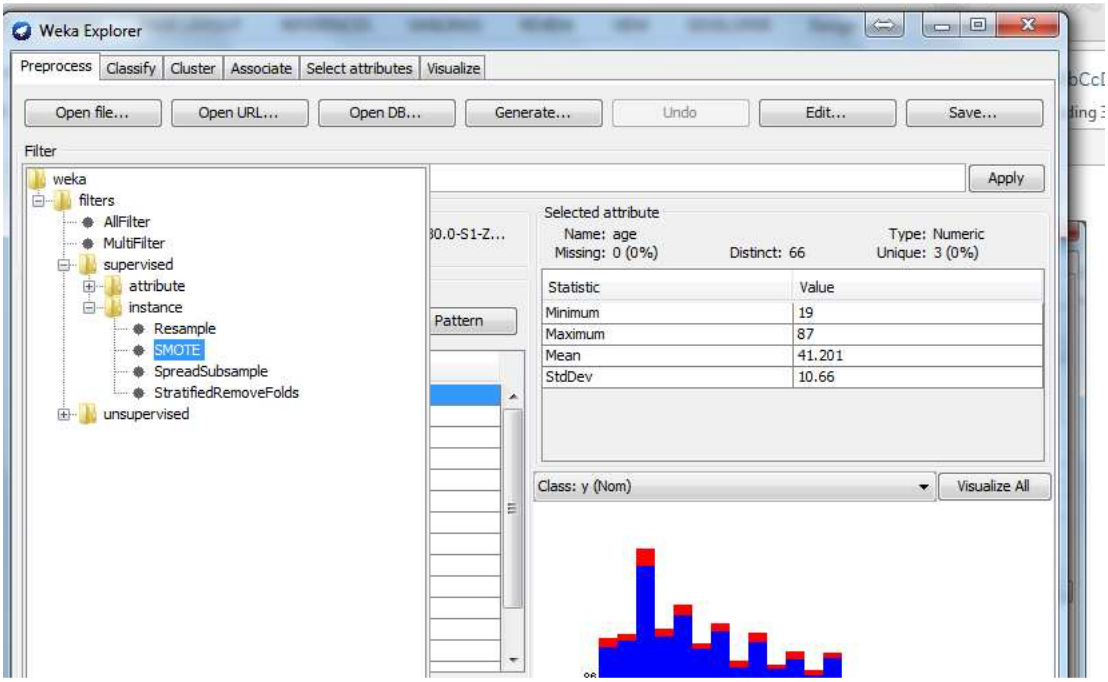
ANNEX 2 – Creating the oversampled dataset

Procedure

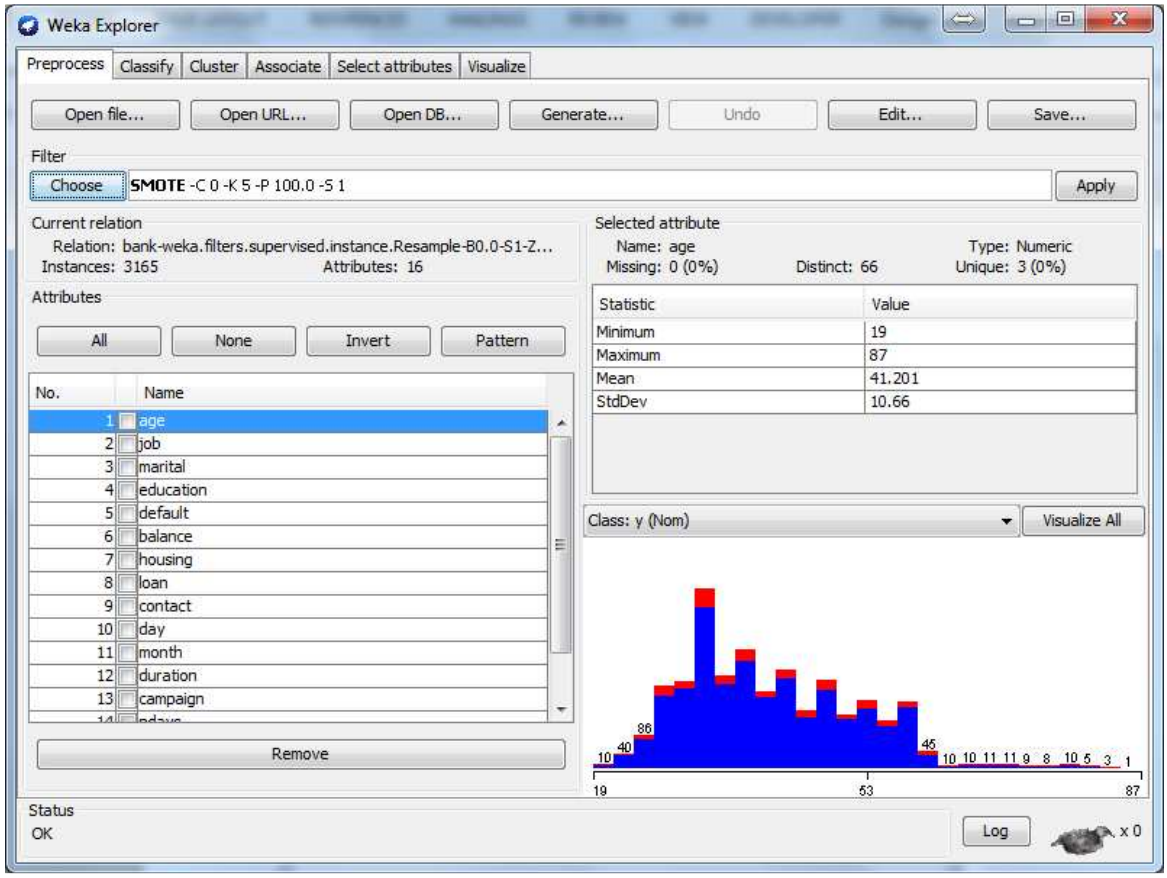
1. Open Weka 3.6 and click on Explorer
2. Load the initial dataset into Weka Explorer
3. Under the **Preprocess** tab, click the button **Choose**



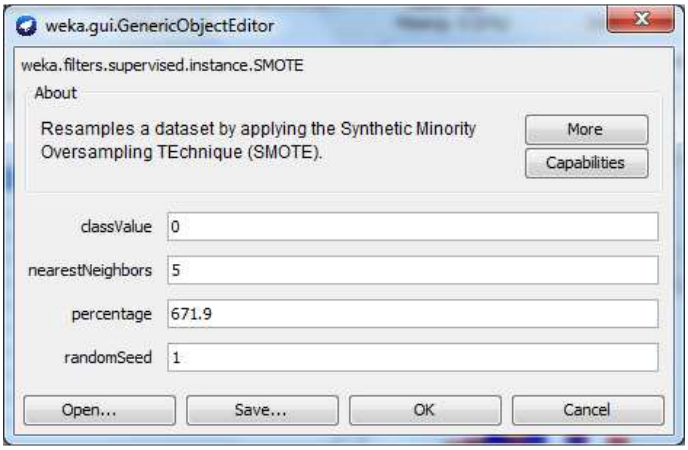
4. In the window that opens, select SMOTE by following the folder path: **Weka > Filters > Supervised > Instance >**



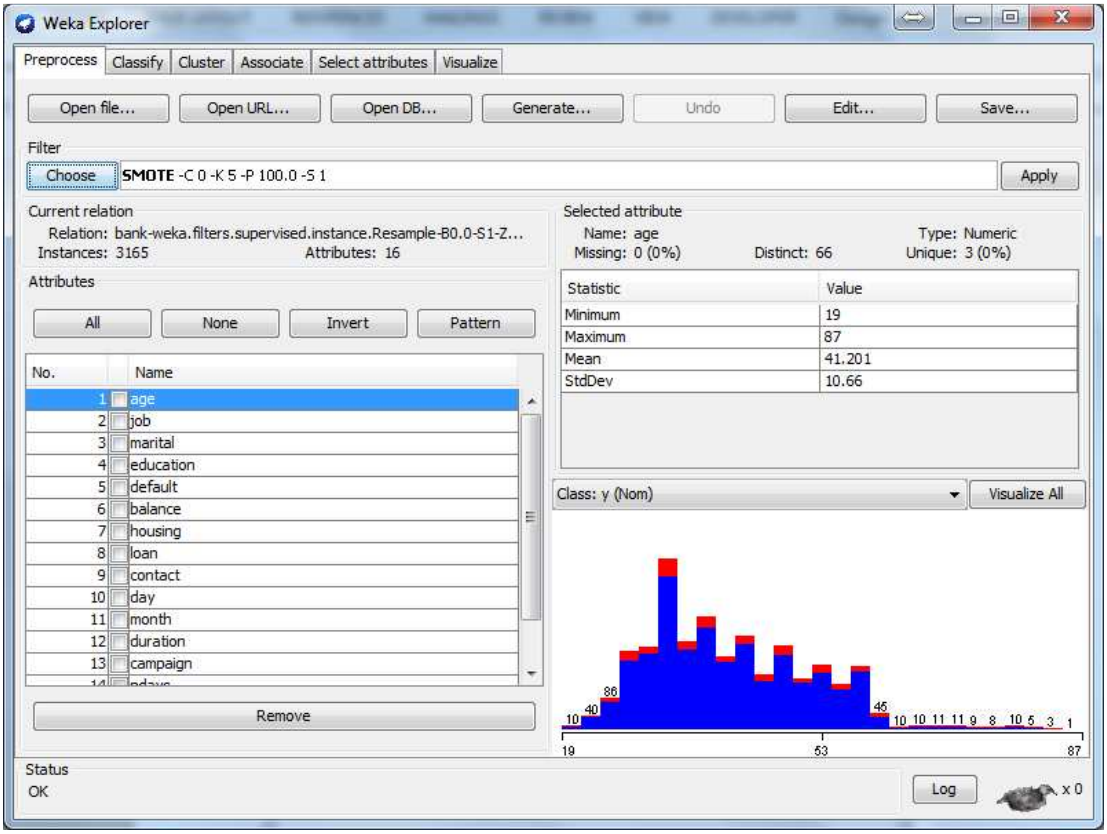
5. In the textbox next to the **Choose** button, click the blank space



6. In the **percentage** textbox change the value to 671.9 and press OK

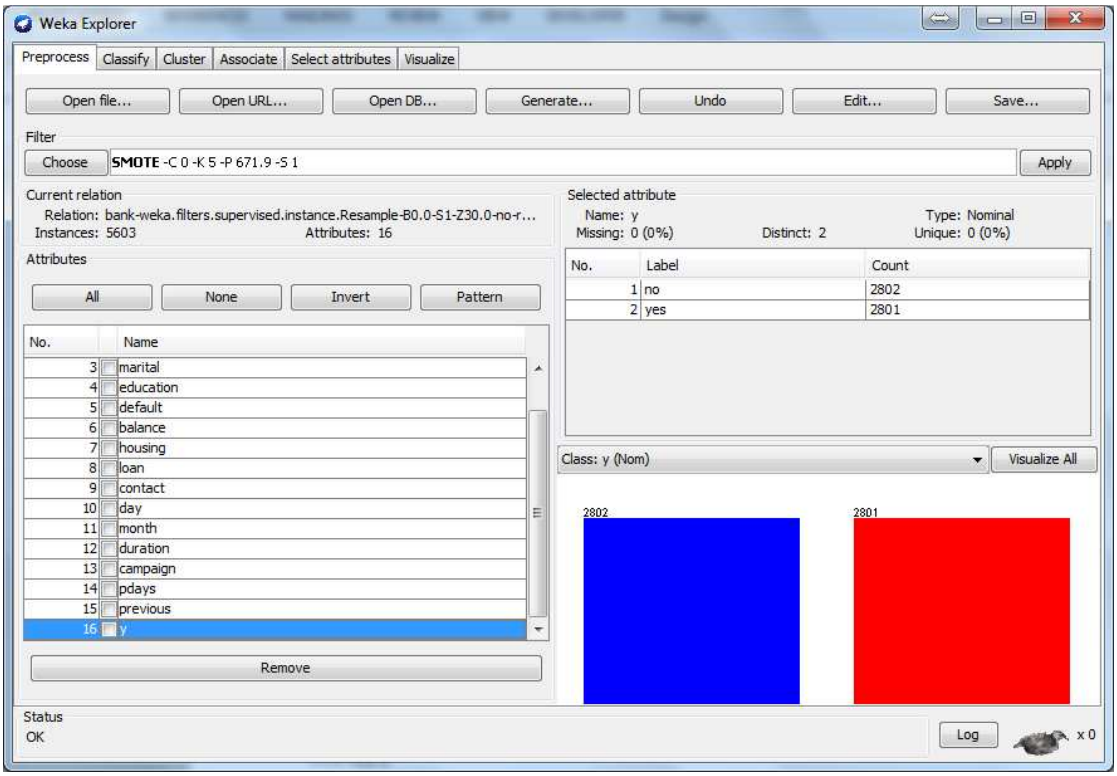


7. Click **Apply**



Note: When the class attribute Y is selected note the tuples with class label “yes” has been increased to 2801 tuples

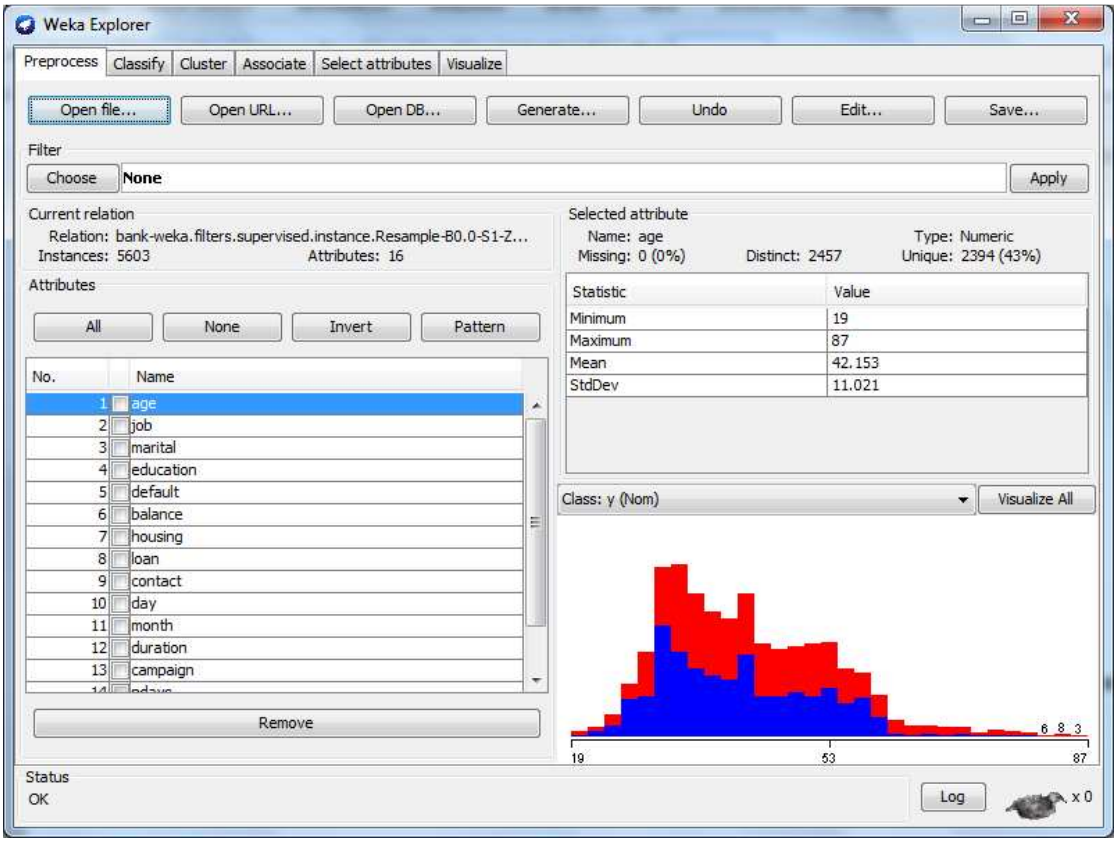
8. Click on the **Save** button and save the dataset as “oversampled-bank.arff” file in an appropriate location



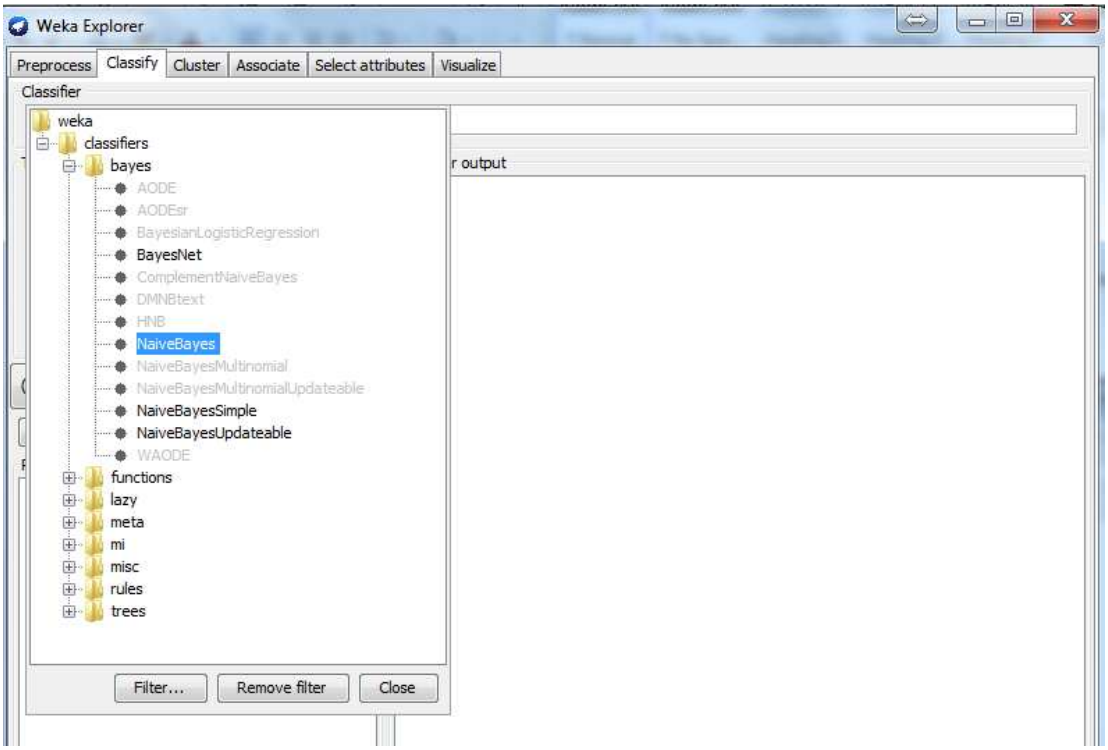
ANNEX 3 – Building the classifiers from the oversampled dataset

Procedure

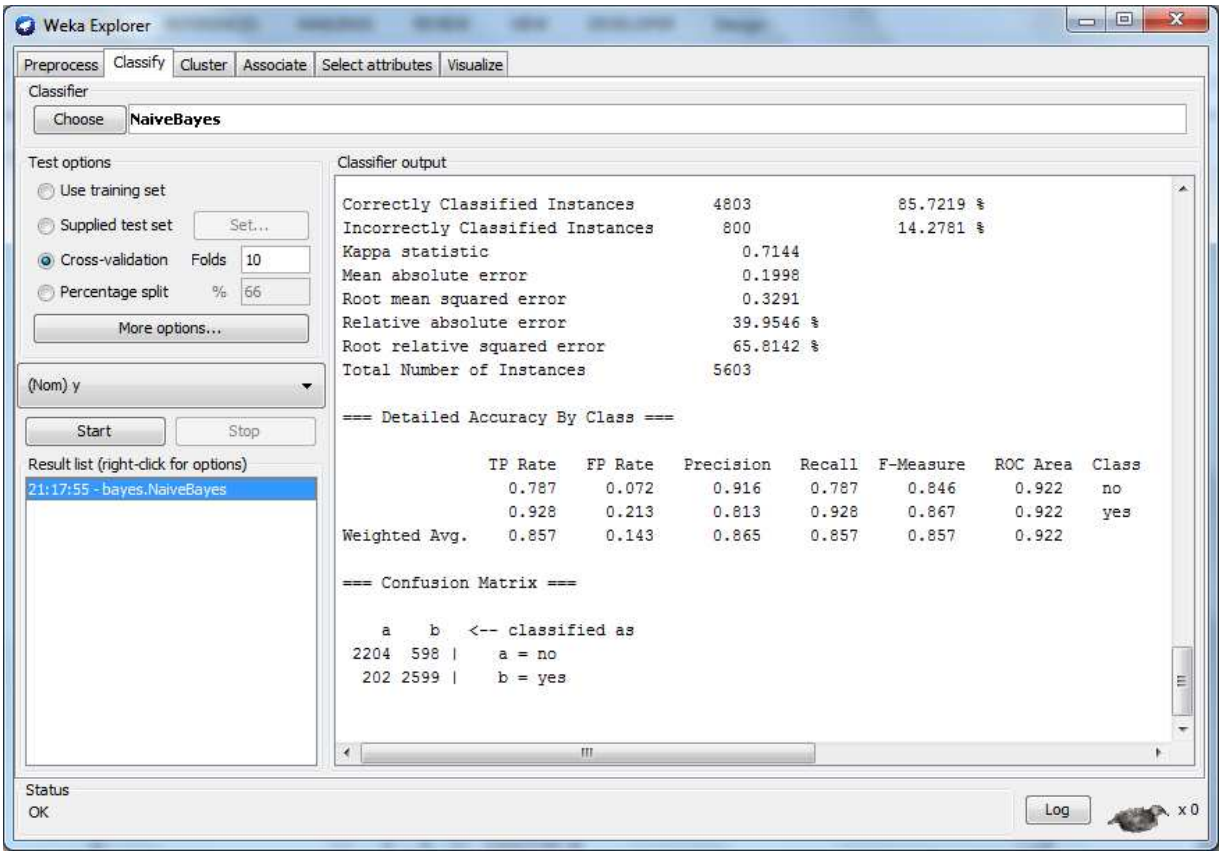
1. Open Weka 3.6 and click on **Explorer**
2. On the Preprocess tab, click on the button **Open file** and load the "oversampled-bank.arff" from the appropriate location into Weka Explorer



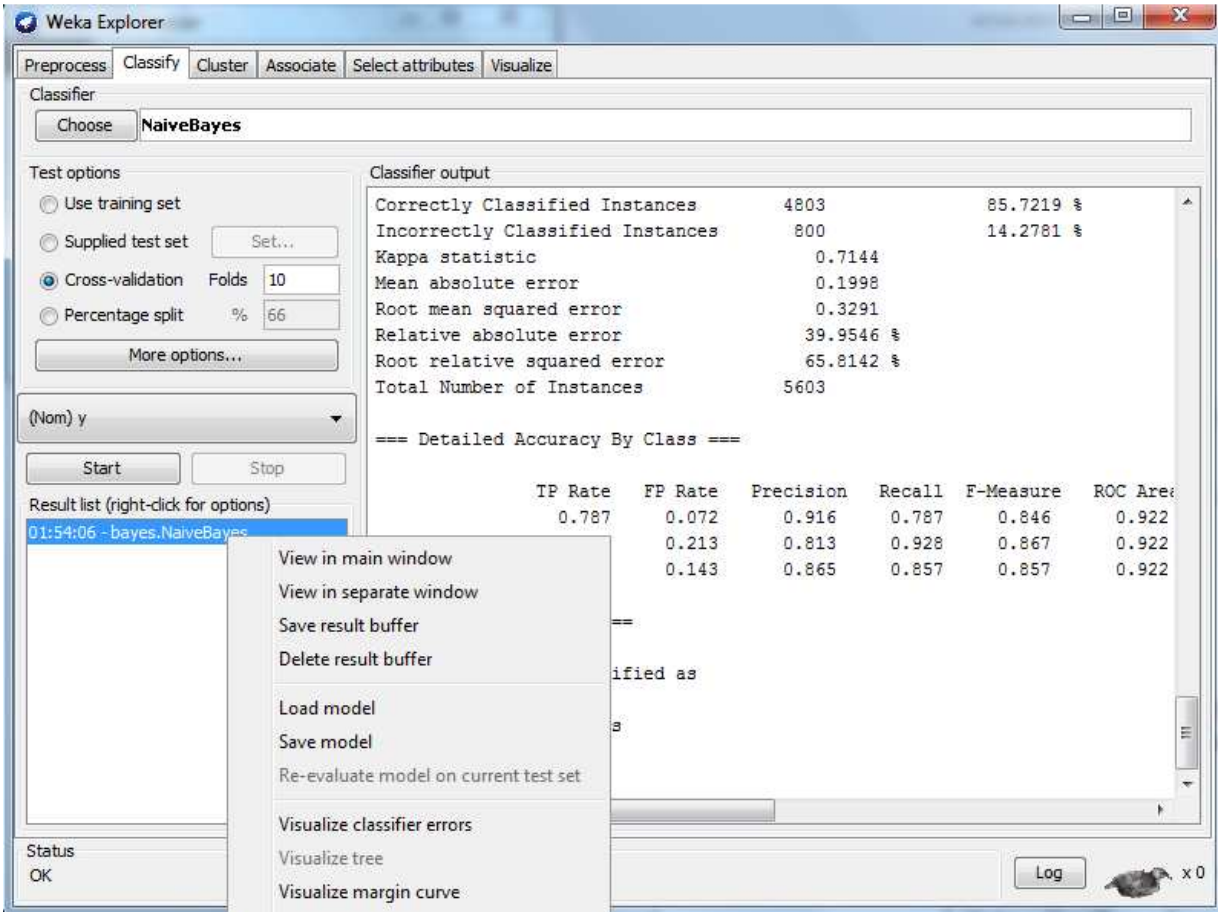
3. Click on **the Classify** tab then click on the button **Choose**. In the location **Weka > Classifiers > Bayes >**, select NaiveBayes



4. In the **Classify** tab, in **Test Options**, select **Cross-Validation**. Leave all else as default and press the **Start** button. Once the classifier is built and tested the results will be shown in the **Classifier Output** screen



5. Right-click on the classifier model and select **save model** and save the model in an appropriate location



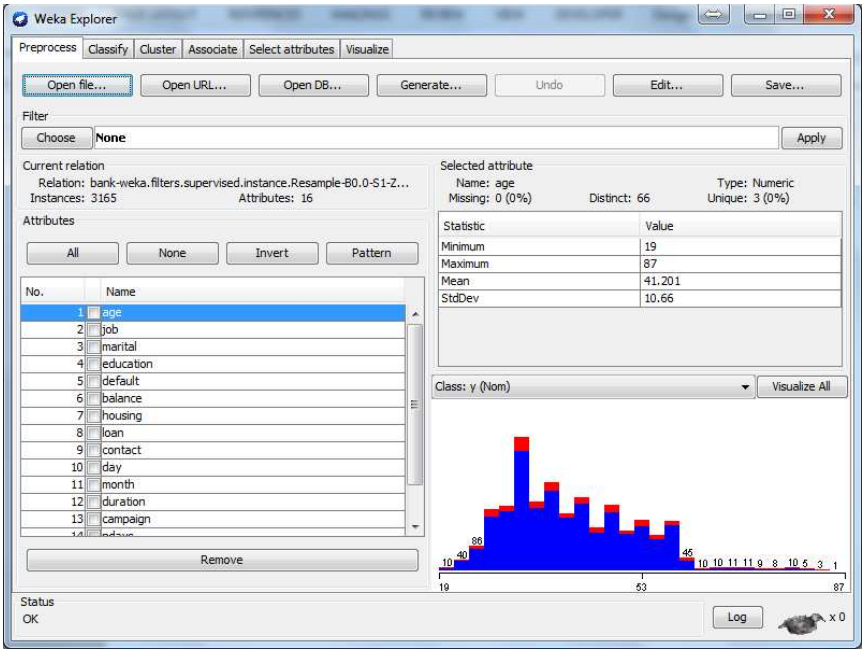
6. Repeat the procedure for the J48 (decision tree), MultilayerPercptron (neural network), and Logistic (logistic regression) classifiers, however note that the classifiers are in different locations. Use the table below to locate the other classifiers

Classification model	**Location**
J48 (decision tree)	Weka > Classifiers > Trees >
MultilayerPercptron (neural network)	Weka > classifiers > Functions >
Logistic (logistic regression)	Weka > classifiers > Functions >

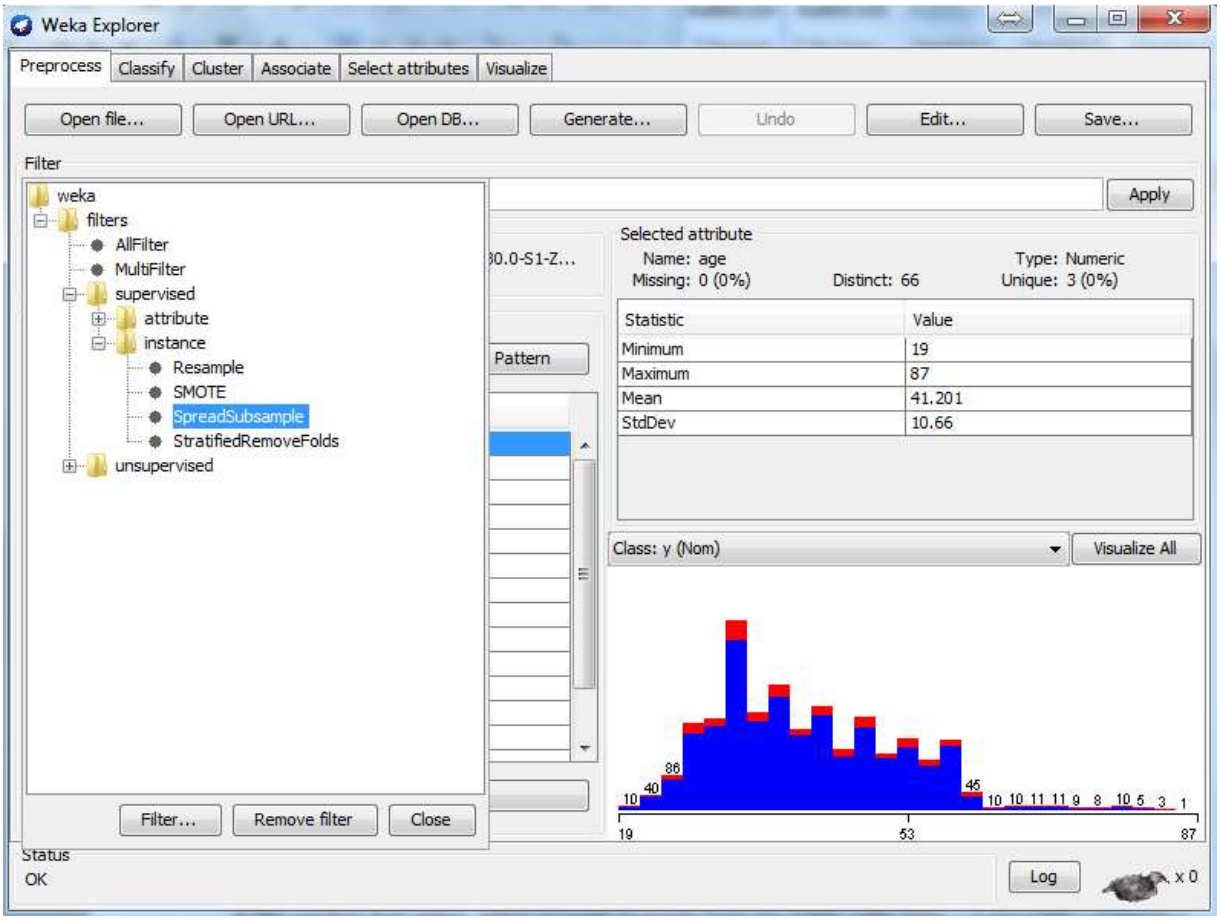
ANNEX 4 – Creating the under sampled dataset

Procedure

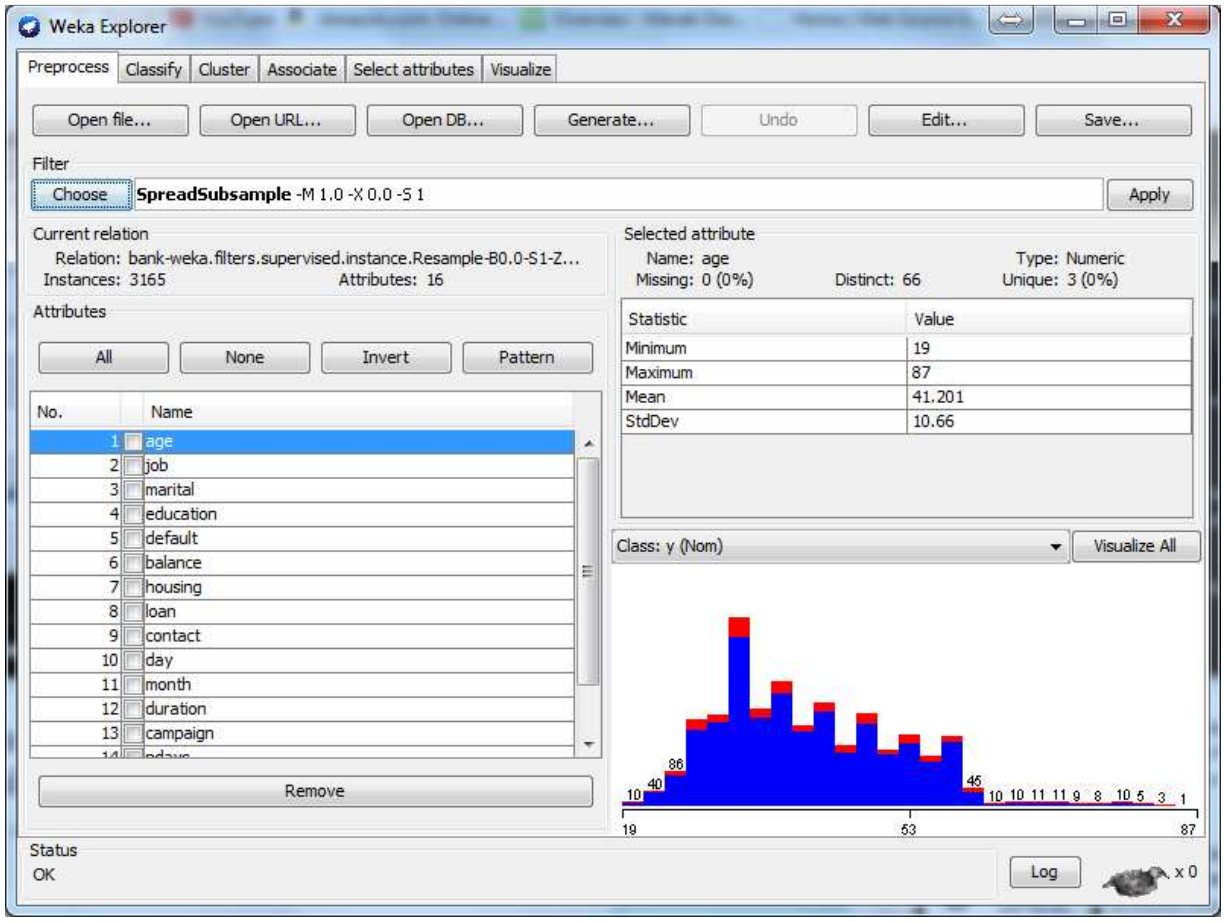
1. Open Weka 3.6 and click on Explorer
2. Load original training dataset, namely the file “bank-cs699.arff” in Weka Explorer



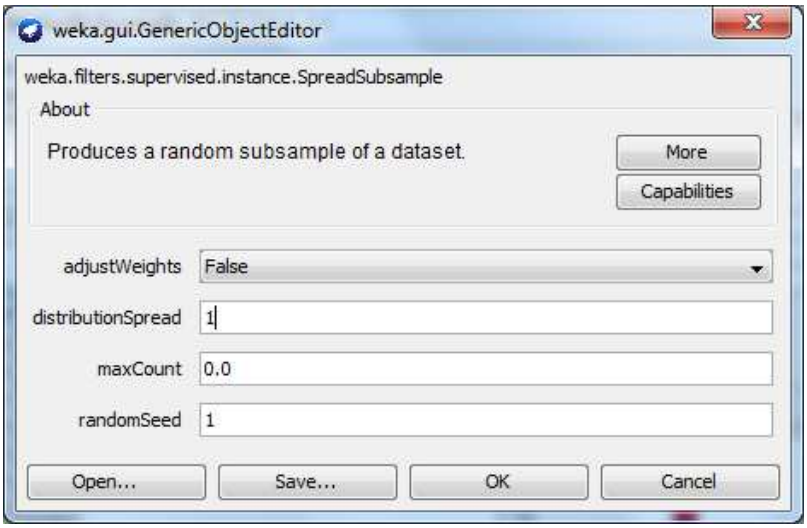
3. Under the **Preprocess** tab, click the button **Choose** and in the window that opens, select **SpreadSubsample** from the folder path: **Filter > Supervised > Instance >**



4. In the textbox next to the button **Choose**, click on the blank space

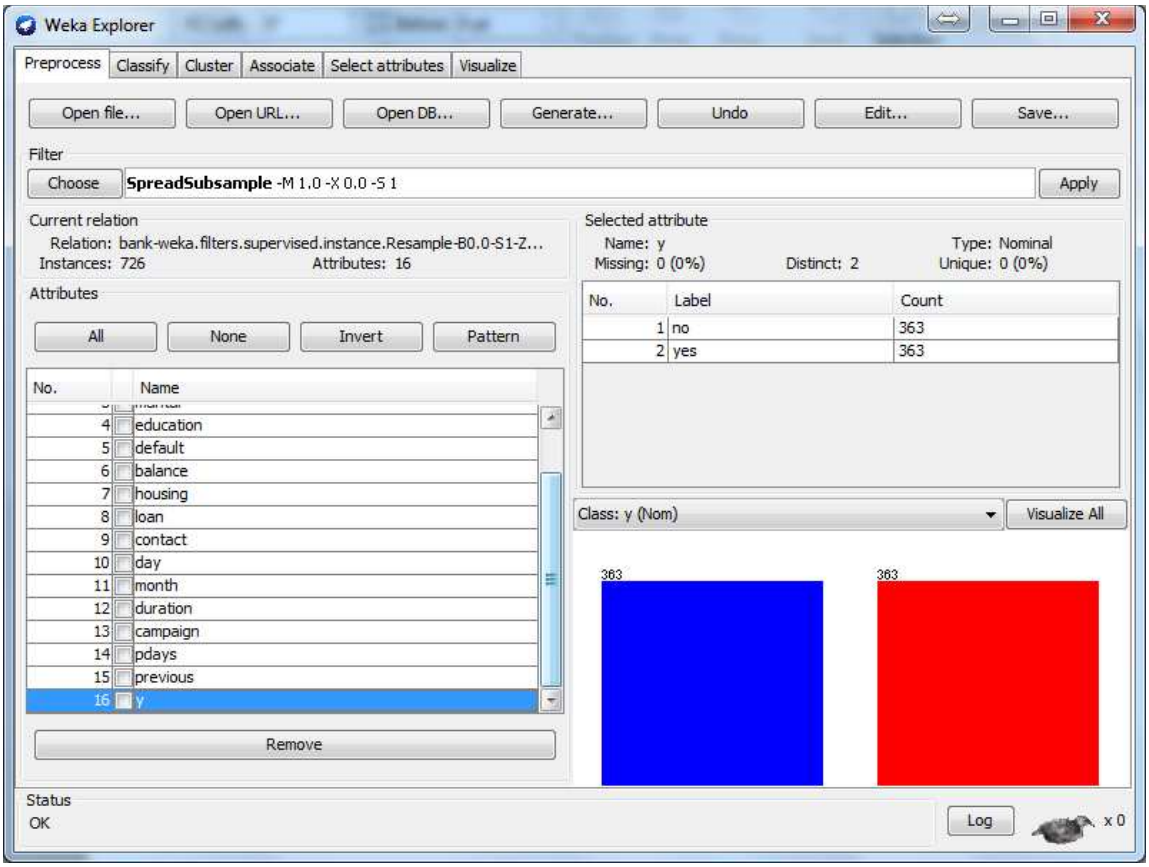


5. A window pops up. In the **distributionSpread** textbox type the value 1 and press OK.
   Note: setting the distribution spread to 1 reduces the number of tuples with “no” class label to 363, which is the same as the number of tuples with “yes” class label.



6. Click on **Apply** and then select the class attribute in the window to view the changes made. Note that the total number of tuples have reduced to 726 and that the number of tuples with “no” class label is now 363, same as the number of tuples with “yes” class label.

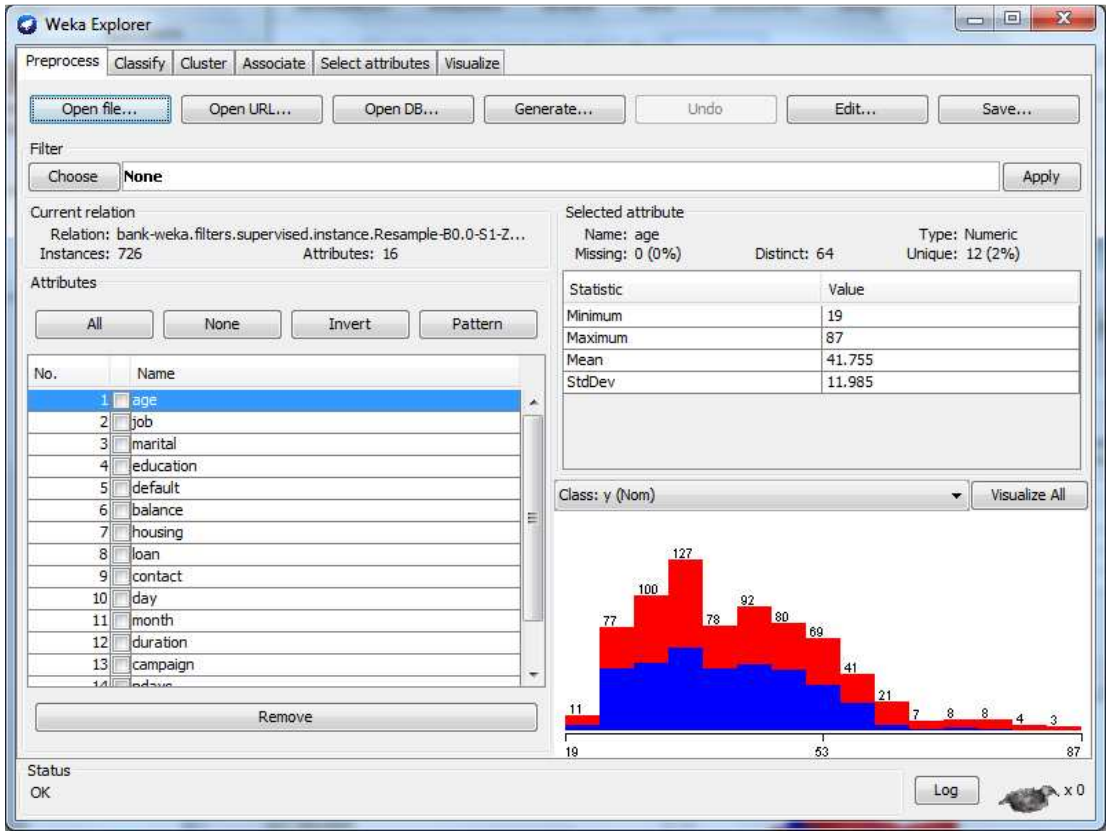
7. Click on the **Save** button and save the dataset as “undersampled-bank.arff” file in an appropriate location



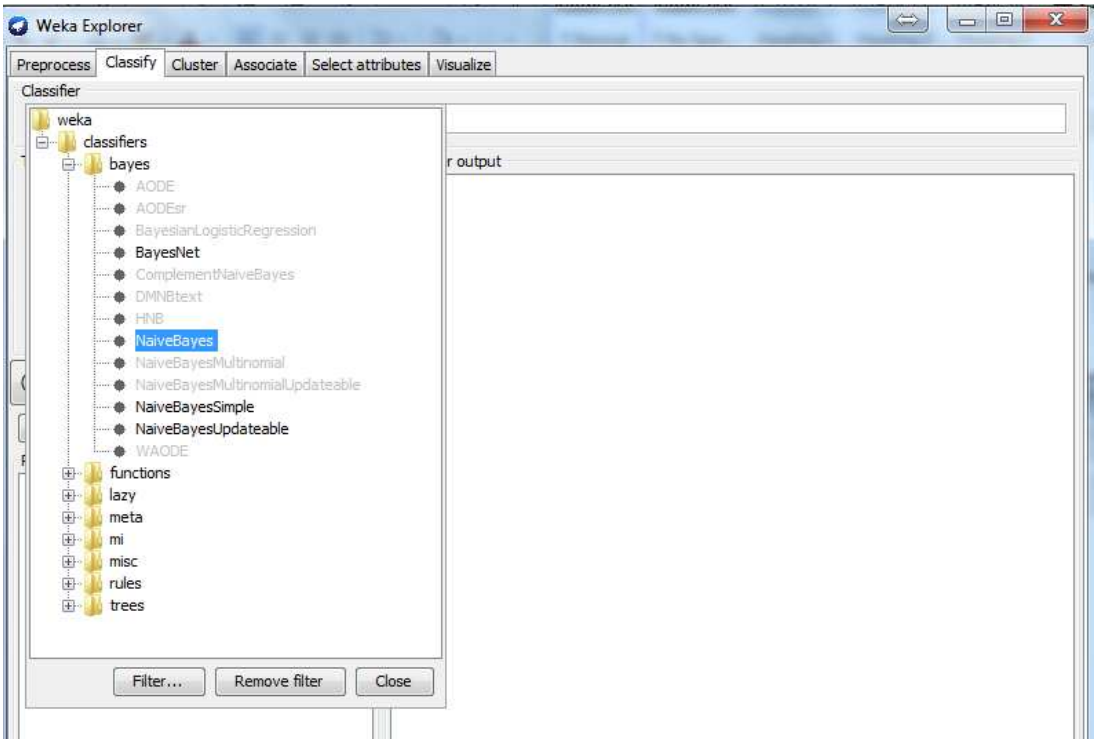
Annex 5 – Building the classifier models from the under sampled dataset

Procedure

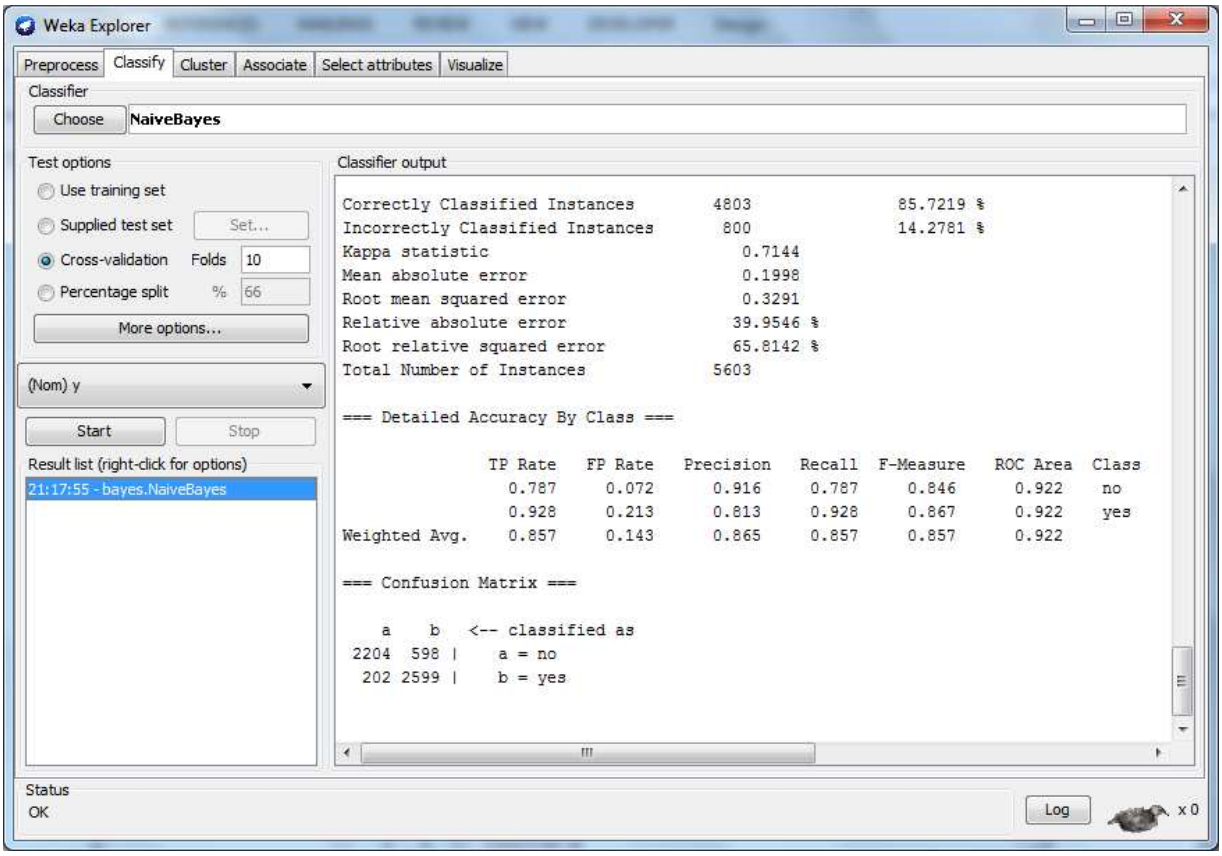
1. Open Weka 3.6 and click on **Explorer**
2. On the Preprocess tab, click on the button **Open file** and load the "undersampled-bank.arff" from the appropriate location into Weka Explorer



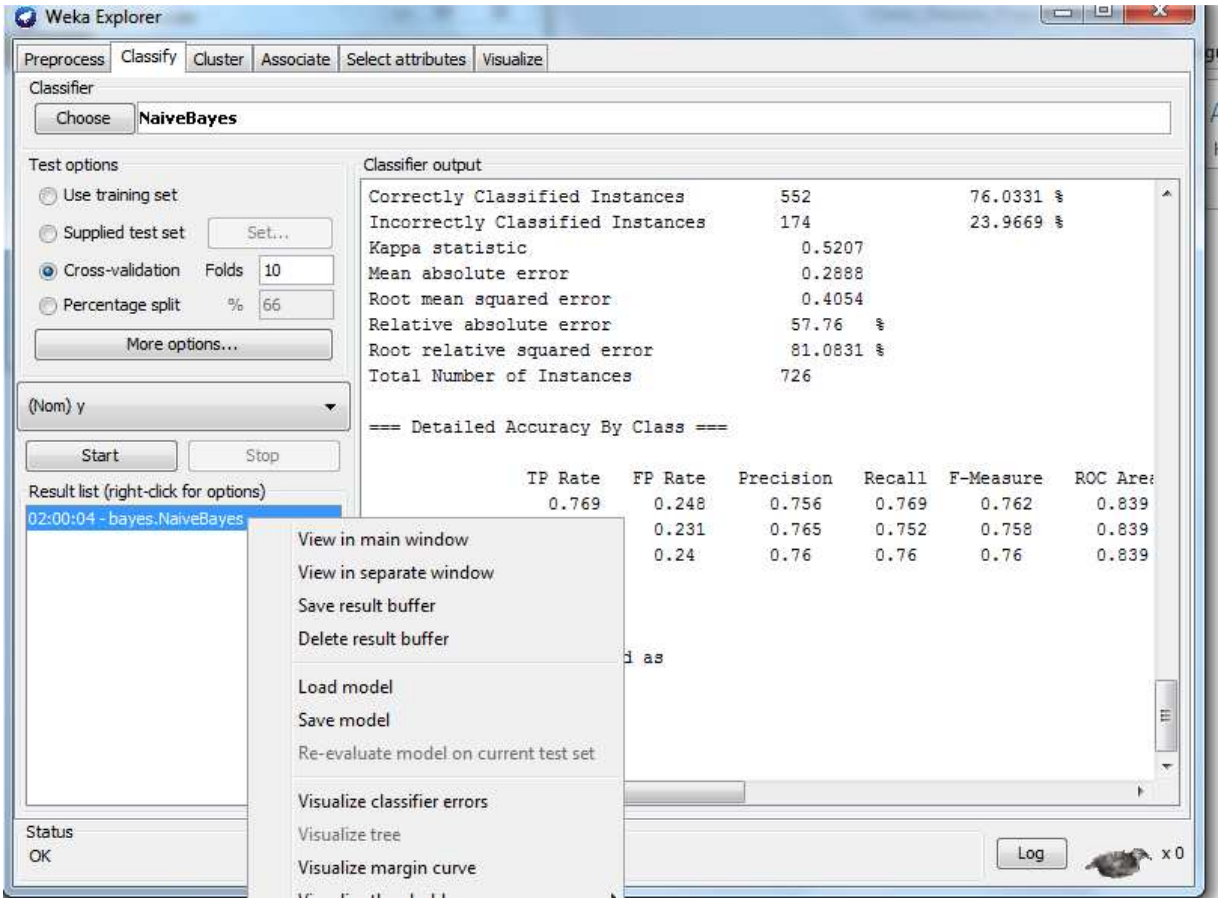
3. Click on the **Classify tab** then click on the button Choose. In the location **Weka > Classifiers > Bayes >**, select **NaiveBayes**



4. In the **Classify** tab, in **Test Options**, select **Cross-Validation**. Leave all else as default and press the **Start** button. Once the classifier is built and tested the results will be shown in the **Classifier Output** screen



5. Right click on the classifier model and select **save model** and save the model in an appropriate location



6. Repeat the procedure for the J48 (decision tree), MultilayerPercptron (neural network), and Logistic (logistic regression) classifiers, however note that the classifiers are in different locations. Use the table below to locate the other classifiers

Classification model	**Location**
J48 (decision tree)	Weka > Classifiers > Trees >
MultilayerPercptron (neural network)	Weka > classifiers > Functions >
Logistic (logistic regression)	Weka > classifiers > Functions >

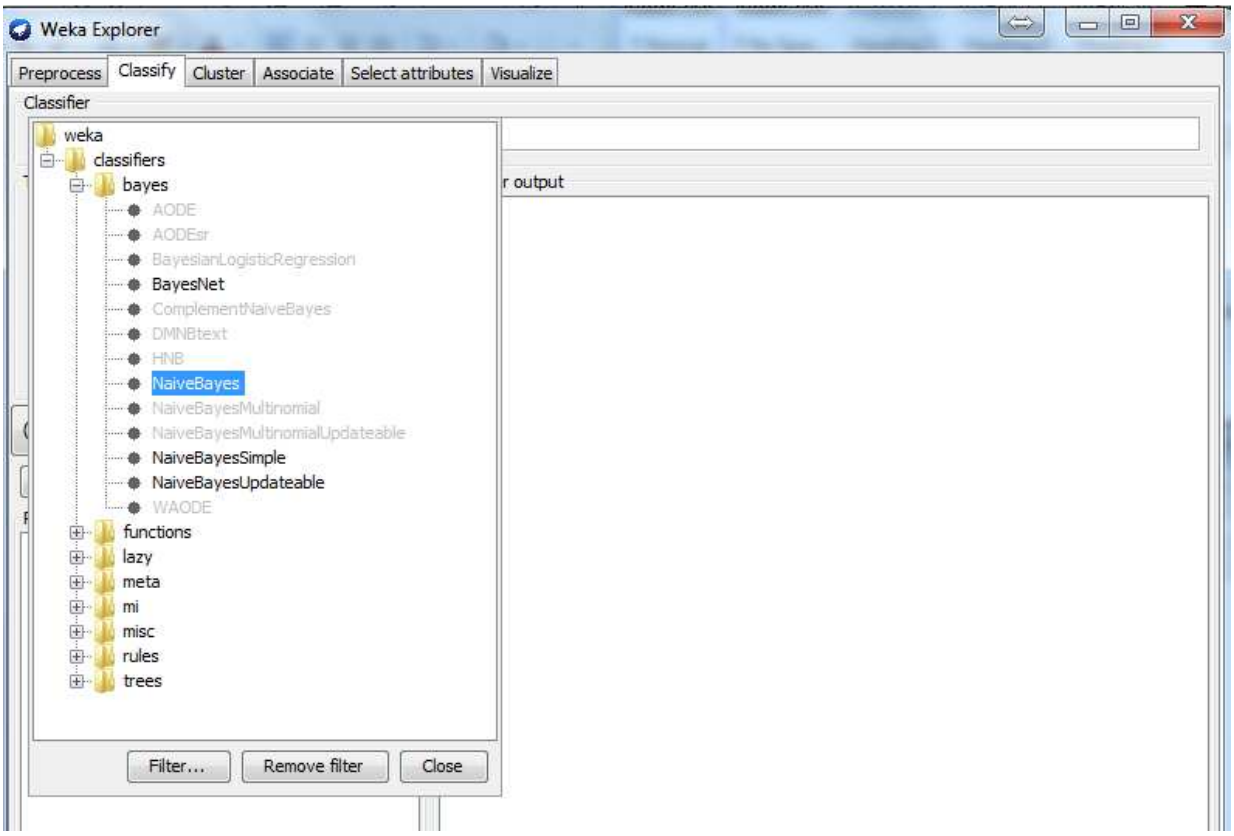
Annex 6 – Testing the classifier models on the initial dataset

Procedure

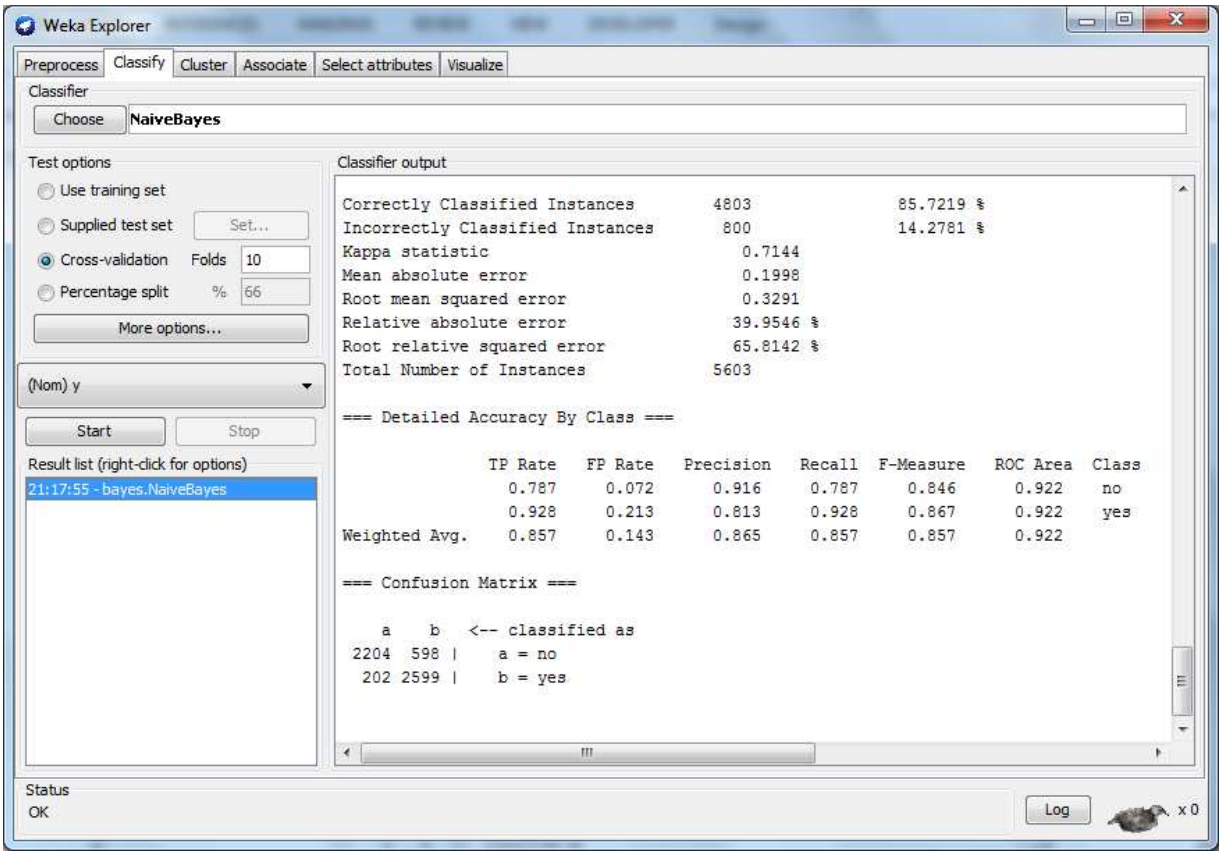
1. Open Weka 3.6 and click on **Explorer**

2. On the Preprocess tab, click on the button **Open file** and load the initial dataset

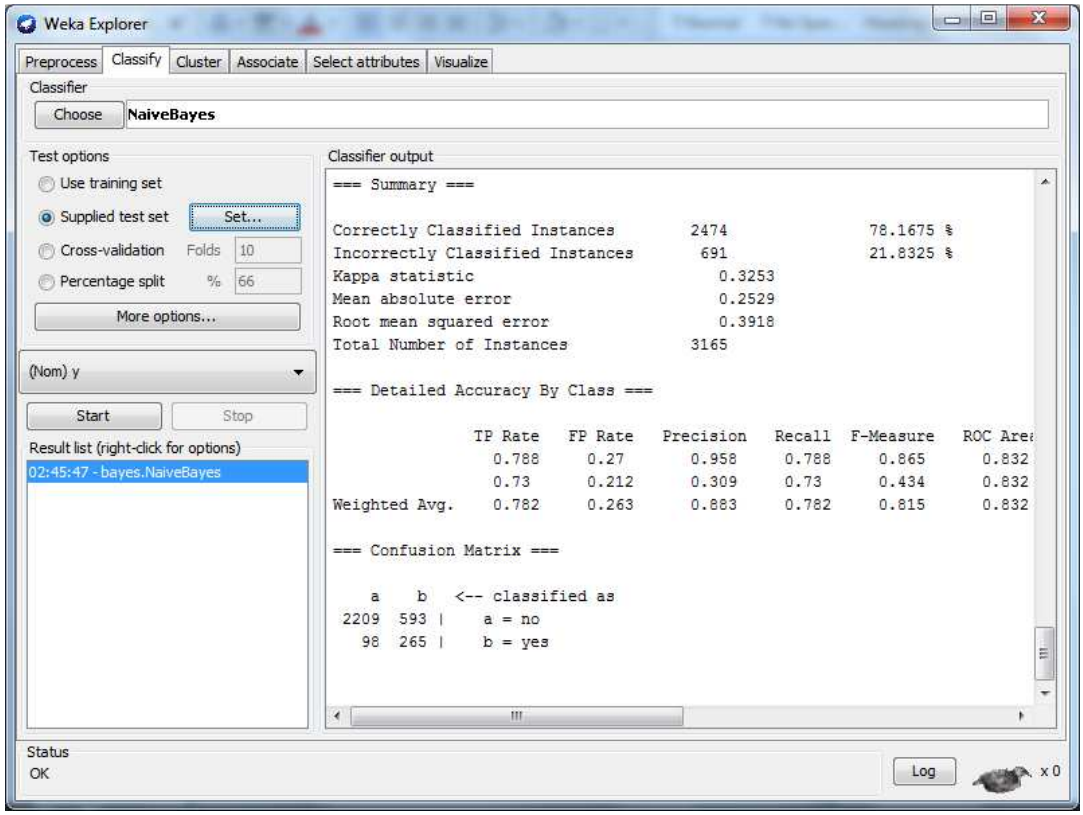
3. Click on the **Classify tab** then click on the button Choose. In the location **Weka > Classifiers > Bayes >**, select **NaiveBayes**



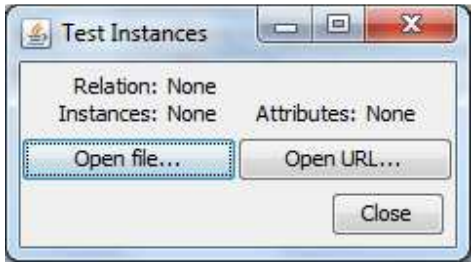
4. In the **Classify** tab, in **Test Options**, select **Cross-Validation**. Leave all else as default and press the **Start** button. Once the classifier is built and tested the results will be shown in the **Classifier Output** screen



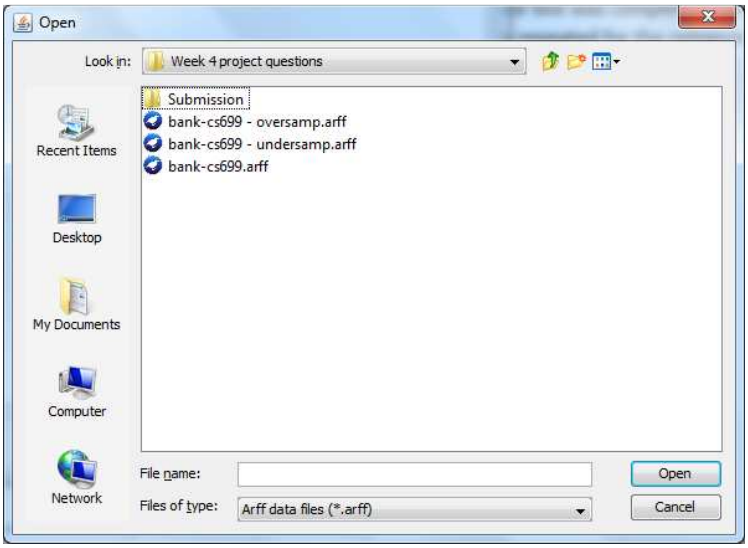
5. Under **Test options**, select **Supplied test set** and click the button **set**



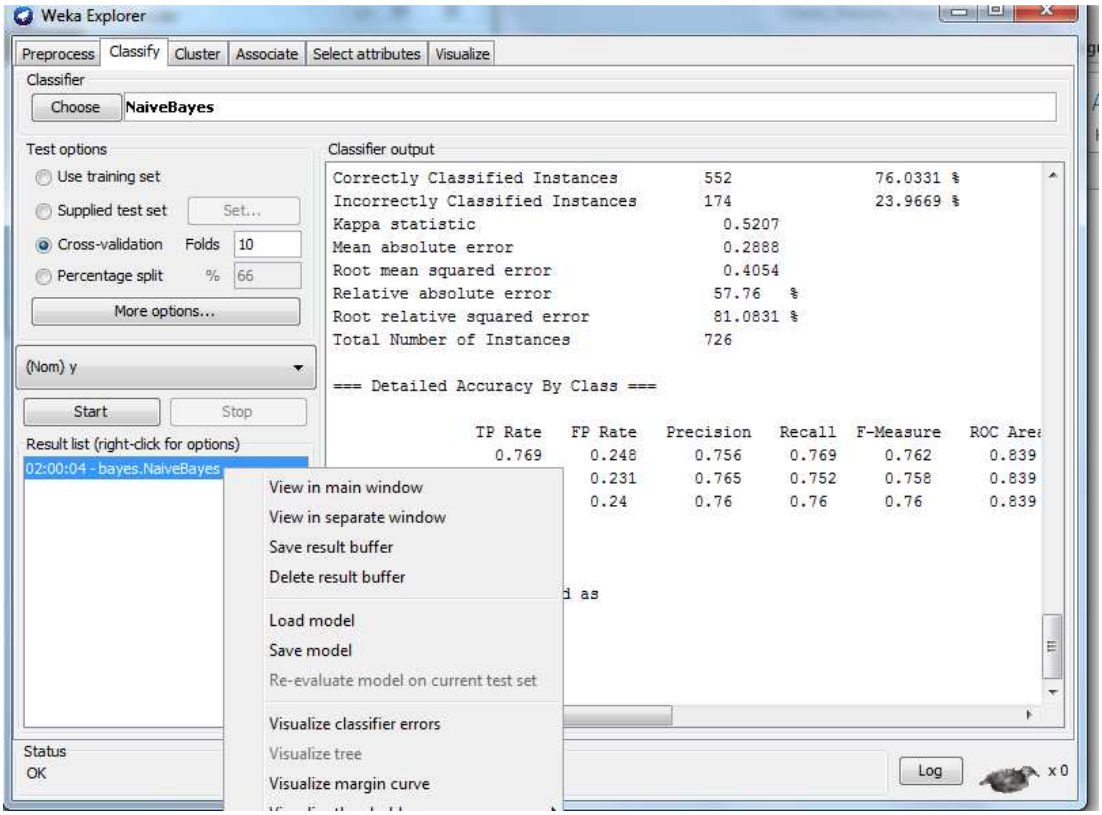
6. In the window that pops up, click on **Open file**



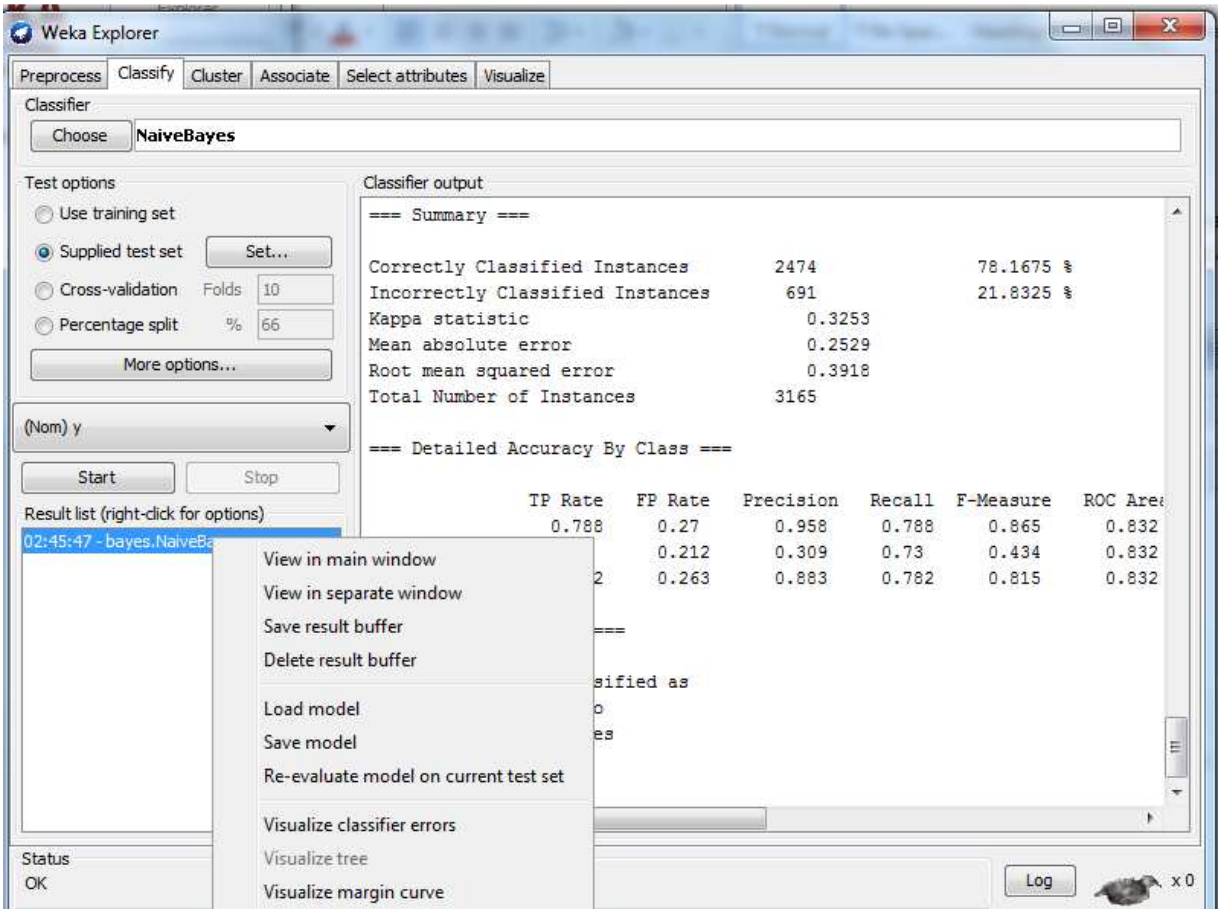
7. Select the initial dataset and click the button **Open**



8. Right click on the classifier model and select **load model** and load the Naïve Bayes model that was built from the oversampled dataset.



9. Right-click again on the classifier model and select **re-evaluate model on current test set.** Once completed the results will display on the **classifier output** screen.



10. Repeat this procedure for the 7 other classifier models that were saved from the oversampled dataset and the under sampled dataset.