IE 582: STATISTICAL LEARNING FOR DATA MINING

Project Report

Date: February 15th, 2021

Hande Mızrak

2020702114

Nur Gündoğdu

2020702075

Yasemin Yumak

2020702183

TABLE OF CONTENTS

1. INTRODUCTION

In the final project, the data are split into two sets, training and test. Training set is used to build models and test set is used to evaluate performance of these models. Training set includes 2074 objects and test set includes 2073. Both have 60 predictor variables and 1 target variable. All predictors are numeric and there is no missing value. Therefore, it is not necessary to use encoding and imputation methods with these data sets. Only target variable is categorical consisting of two levels which are "a" and "b" and the aim will be to build a classification model for the given data based on the methods described in the course. At the same time, it is seen from the training set 1565 objects are in class A comparing to 509 objects in class B, so this problem is considered as class imbalance problem having a ratio of 3:1.

Balanced error rate and area under the ROC curve are used as performance measures. Error rate is calculated by dividing incorrect predictions (false positives and false negatives) by the total number of predictions (true positive, true negative, false positives and false negatives). Since the classes in this problem are unbalanced, balanced error rate is chosen which is calculated as the average of false positive rate and false negative rate in order to make model minimize the error rate on each class. Area under ROC curve (AUC) is also widely used performance metric for binary classification problems. True positive rate (TPR) against false positive rate (FPR) is plotted at different threshold values. Then, predictions having maximum TPR and minimum FPR values give the best results. These two measures will be tried to be maximized which means multi-objective optimization. It is tried to be find a nondominated or Pareto optimal solution.

Different approaches have been tried to build the most suitable model for this problem. They are penalized regression (PRA), decision trees (DT), random forests (RF), stochastic gradient boosting (SGB), and extreme gradient boosting (EGB). For each model, the best parameter for this problem should also be determined. Penalty for PRA, the minimal number of observations per tree leaf and complexity parameter for DT, the number of features for RF and depth, learning rate, number of trees for SGB and EGB. In order to improve the proposed models, k-fold cross validation and hyperparameter tuning are used. After trying different approaches with different parameters, random forest algorithm with parameter mtry = 16, ntree = 100 gave the best result for this data set according to specified performance metrics.

2. RELATED LITERATURE

There are a lot of research about binary classification problems and also about approaches to imbalanced datasets. Imbalance between classes may occur in many places such as product defects in industrial areas, detection of spam mails or the diagnosis of severe diseases. In these cases, models may perform well for all test set, but may not give good enough results for minor class although it is crucial to predict the cases belonging to minor class such as diagnosis a disease. For this problem, some methods are developed. They are examined in two groups: internal and external methods. Internal methods focus on new algorithms while external methods focus on the datasets. Giving more weight to the cost of incorrect classification of rare event or support vector machines are some examples for internal ones. Resampling operations are made to reduce imbalance between classes as external methods. First option is to reduce samples belonging to the class having higher occurrence rate. It is called undersampling and can be made randomly or based on some criteria such as eliminating outliers. Second option is to increase the number of samples belonging to less occurring class. It is called oversampling. It can be made randomly again or by selecting the samples on the border between the two classes.

These two methods combined through similarity-based undersampling and normal distribution-based oversampling (SUNDO) approach (Cateni, Colla, & Vannucci, 2014). Through this method, first of all dataset is split into two groups as train and test without changing the proportion between classes in each set. Then, training set is also divided according to the classes and occurrence rate is specified for each set. Undersampling is applied for majority class and oversampling is applied for minority class until reaching specified rate. Lastly, they are combined again to train model.

A post-processing technique for Support Vector Machine (SVM) algorithms where after solution vector for the problem is obtained, the bias is considered as a parameter to be tuned, is introduced for binary classification problems in another article (L. Gonzalez-Abril, C. Angulo, H. Nuñez, & Y. Leal, 2017). This approach is especially important when importance of correct prediction for one class is higher than the other one as in disease diagnosis or bankruptcy prediction. Therefore, true positive rate will have more importance than other accuracy metrics. Such a technique can also be used for imbalance problems by giving more emphasis on minority class.

3. APPROACH

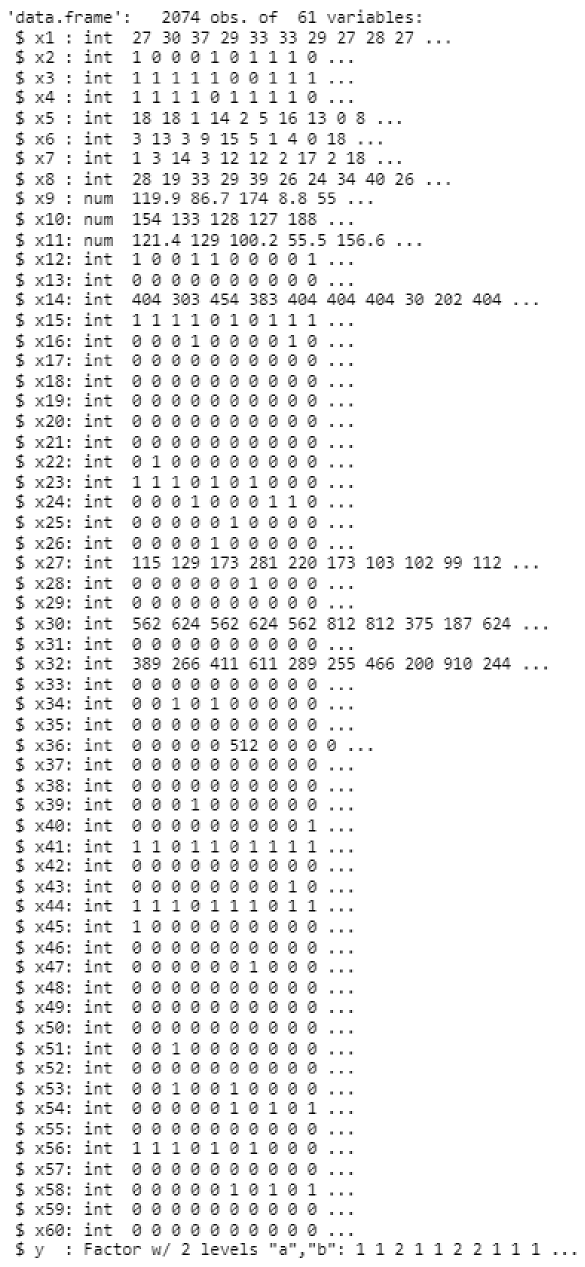
In this project, different approaches have been tried to build the most suitable model for this problem. They are penalized regression (PRA), decision trees (DT), random forests (RF), stochastic gradient boosting (SGB), and extreme gradient boosting (EGB). In order to understand how the models work on the training data 4 different base models are tried using their default parameters. By looking at class error rate of the models we realized that the models were not good enough in separating classes. When distribution of the class labels is analyzed class imbalance problem is detected on the training data. In order to solve this problem, 4 most commonly used methods are found from the literature which are SMOTE, Under-sampling, Over-sampling and ROSE.

After an appropriate analysis, it is found that Under-Sampling gives better results compared to other methods. The analysis will be explained in the result section in detail. Then, the best model has been tried to find using under sampling method. For this part, random forests (RF), stochastic gradient boosting (SGB), and extreme gradient boosting (EGB) have been trained and RF gave the best submission score. In order to improve the proposed model (RF), 10-fold cross validation and hyperparameter tuning are used in this project.

Using caret package and cross validation, the best parameters for RF is found as mtry (number of features) :16, nodesize:14 and ntree: 100. Then, the final RF model is trained using the whole training data (without splitting) and using the best parameters. And then, this final model is used to make predictions on the test data. Cross validation results for different hyperparameters will be given at the end of the report in the appendix section.

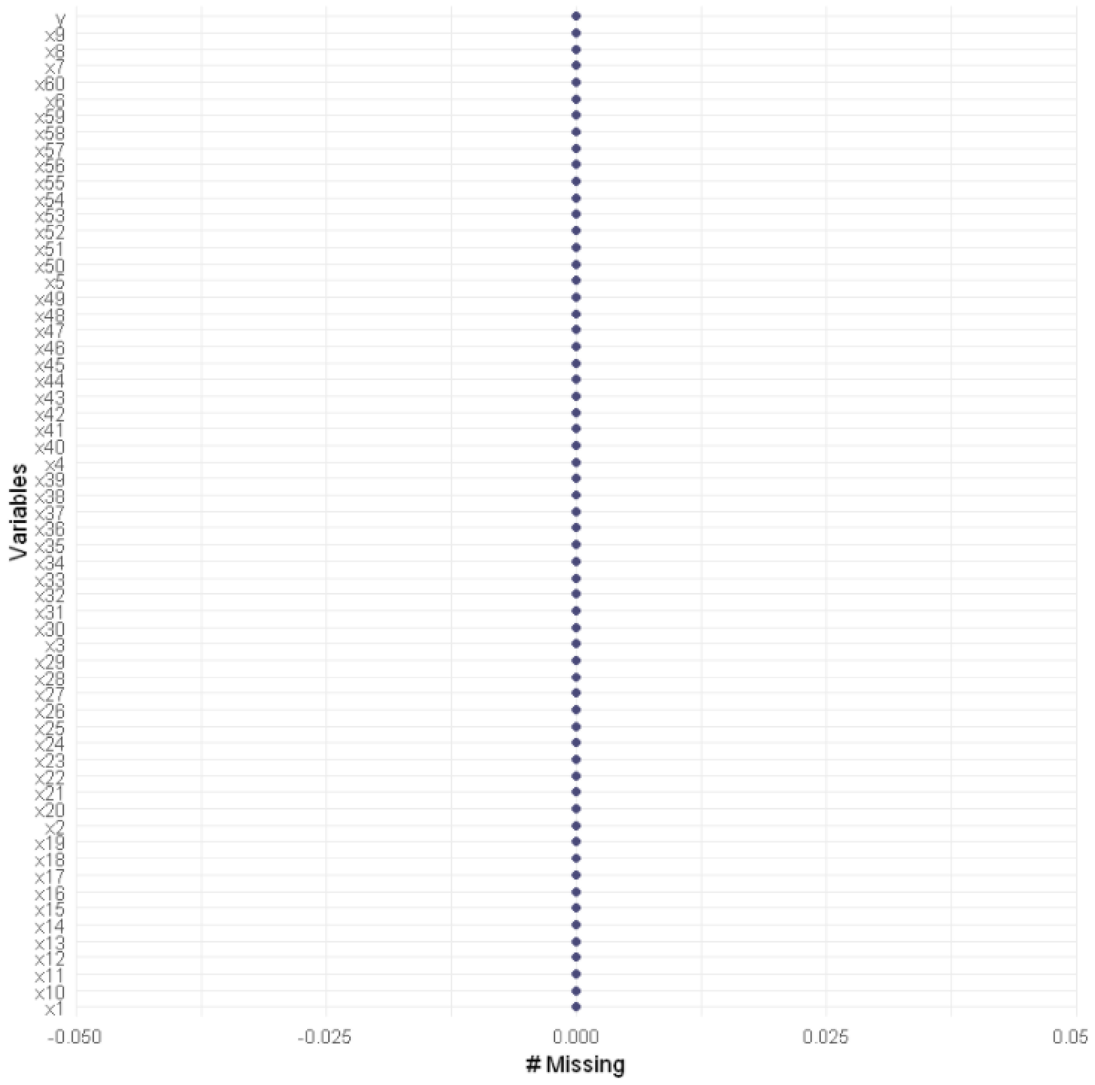
4. RESULTS

Firstly, our train data is read and the data is the following:



```
'data.frame':   2074 obs. of  61 variables:
 $ x1 : int  27 30 37 29 33 33 29 27 28 27 ...
 $ x2 : int  1 0 0 0 1 0 1 1 1 0 ...
 $ x3 : int  1 1 1 1 1 0 0 1 1 1 ...
 $ x4 : int  1 1 1 1 0 1 1 1 1 0 ...
 $ x5 : int  18 18 1 14 2 5 16 13 0 8 ...
 $ x6 : int  3 13 3 9 15 5 1 4 0 18 ...
 $ x7 : int  1 3 14 3 12 12 2 17 2 18 ...
 $ x8 : int  28 19 33 29 39 26 24 34 40 26 ...
 $ x9 : num  119.9 86.7 174 8.8 55 ...
 $ x10: num  154 133 128 127 188 ...
 $ x11: num  121.4 129 100.2 55.5 156.6 ...
 $ x12: int  1 0 0 1 1 0 0 0 0 1 ...
 $ x13: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x14: int  404 303 454 383 404 404 404 30 202 404 ...
 $ x15: int  1 1 1 1 0 1 0 1 1 1 ...
 $ x16: int  0 0 0 1 0 0 0 0 1 0 ...
 $ x17: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x18: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x19: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x20: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x21: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x22: int  0 1 0 0 0 0 0 0 0 0 ...
 $ x23: int  1 1 1 0 1 0 1 0 0 0 ...
 $ x24: int  0 0 0 1 0 0 0 1 1 0 ...
 $ x25: int  0 0 0 0 0 1 0 0 0 0 ...
 $ x26: int  0 0 0 0 1 0 0 0 0 0 ...
 $ x27: int  115 129 173 281 220 173 103 102 99 112 ...
 $ x28: int  0 0 0 0 0 0 1 0 0 0 ...
 $ x29: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x30: int  562 624 562 624 562 812 812 375 187 624 ...
 $ x31: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x32: int  389 266 411 611 289 255 466 200 910 244 ...
 $ x33: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x34: int  0 0 1 0 1 0 0 0 0 0 ...
 $ x35: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x36: int  0 0 0 0 0 512 0 0 0 0 ...
 $ x37: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x38: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x39: int  0 0 0 1 0 0 0 0 0 0 ...
 $ x40: int  0 0 0 0 0 0 0 0 0 1 ...
 $ x41: int  1 1 0 1 1 0 1 1 1 1 ...
 $ x42: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x43: int  0 0 0 0 0 0 0 0 1 0 ...
 $ x44: int  1 1 1 0 1 1 1 0 1 1 ...
 $ x45: int  1 0 0 0 0 0 0 0 0 0 ...
 $ x46: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x47: int  0 0 0 0 0 0 1 0 0 0 ...
 $ x48: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x49: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x50: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x51: int  0 0 1 0 0 0 0 0 0 0 ...
 $ x52: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x53: int  0 0 1 0 0 1 0 0 0 0 ...
 $ x54: int  0 0 0 0 0 1 0 1 0 1 ...
 $ x55: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x56: int  1 1 1 0 1 0 1 0 0 0 ...
 $ x57: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x58: int  0 0 0 0 0 1 0 1 0 1 ...
 $ x59: int  0 0 0 0 0 0 0 0 0 0 ...
 $ x60: int  0 0 0 0 0 0 0 0 0 0 ...
 $ y  : Factor w/ 2 levels "a","b": 1 1 2 1 1 2 2 1 1 1 ...
```

Since we do not know the name of the attributes, we decided to continue with the default data types for feature variables. Then, we checked if there are missing values in the columns of the training set.



Since there is not any missing value in the given data, we skipped the imputation process. To understand how the models work on the training data 4 different base models are tried using their default parameters. These models are random forest, gradient boosting, penalized logistic regression, and decision tree. The score of these models is the following:

Model	**Submission Score**
Random Forest	0.845
Decision Tree	0.834
Penalized Logistic Regression	0.818
Gradient Boosting	0.856

Table 1: Submission Score of the Base Models

After that, the distribution of class labels are visualized.

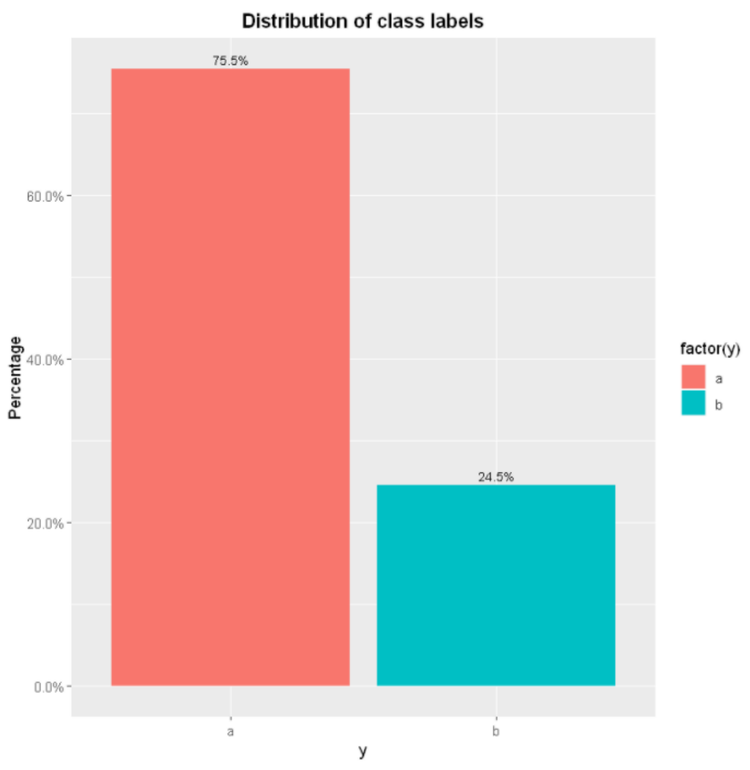


Figure 1: Distribution of the Class Labels on the Train Data

From Figure 1, we can easily say that our data have unbalanced class distributions. In order to solve this problem, we reviewed the literature and found 4 most used methods which are SMOTE, Under-sampling, Over-sampling and ROSE.

In order to see how they work, we split the train data into two parts: train and validation using split ratio of 0.7. Then, we used SMOTE, ROSE, downSample, and upSample functions to find a balance between two classes. The class distributions we obtained after implementing these methods are the following:

Method	Class a	Class b
Original Training Data	1096	356
SMOTE	1424	1068
ROSE	713	739
downSample	356	356
upSample	1096	1096

Table 2: The class distributions after implementing SMOTE, ROSE, Under Sampling, and Over Sampling

Since base Random Forest model gives one of the best results at the beginning of the project, we trained Random Forest models using the data created by the functions given in Table 2. Then, we used Area Under the Curve "AUC" metric to compare the data balancing methods' performance on the validation set. The AUC results on the validation set are the following:

Method	**Random Forest AUC Scores**
Original Data	0.743
SMOTE	0.748
ROSE	0.678
downSample	0.793
upSample	0.751

Table 3: Random Forest AUC Scores of the Methods

From this result, we decided to use downSample function for further analysis. We think that downsampling gave the best result since the data in the dominant class may have more noise and mislead the algorithm. To find the best model, 3 different models are tried: xgboost, gradient boosting and random forest. We trained all of them using downSample function and the submission scores are in the table given below:

Model	**Submission Scores with downSample function**
Random Forest	0.868
Xgboost	0.841
Gradient Boosting	0.855

Table 4: Submission Score of the models after implementing Under Sampling

Since we get more accurate results with the RF we decided to focus on this model. "caret" package is used to tune hyperparameters of the Random Forest and 10 fold cross validation is used. The best parameters and the improved scores for Random Forest are the following:

Model	**mtry**	**nodesize**	**ntree**
Random Forest	16	14	100

Table 5: The best parameters of the RF model after 10-fold CV

Model	**Improved Submission Score with downSample function**
Random Forest	0.889

Table 6: The best submission score of RF model

Throughout this project, we focused to utilize state-of-the-art machine learning algorithms that proved their performance in both the academia and the industry. We got best performance results from Random Forest, Xgboost and Gradient Boosting methods. Since these methods create more robust models and prevent overfitting by using different samples of training, specifying random feature subsets and combining these separate trees, we expected them to perform best. Although these networks are superior to other basic methods with complex model strategies, we were aware of they need adequate hyperparameter tuning. In hyperparameter search phase, we tried to approximate our validation set to blind test set by investigating the submission scores. For example, although we got better results with some new hyperparameters on the validation set we determined, we got worse results in submission. Thus, we wanted to simulate the blind test set with carefully allocating our validation set from all training set. By doing this, we aimed to utilize our submission trial number without submitting random models that did not prove their performance on our carefully selected validation strategy. In addition, these methods have randomness strategies during training. We could observe different results for each run although we specify same hyperparameters and random seed. Thus, it was another obstacle for us to finalize our approach with different hyperparameters with this limited submission numbers.

For the final results, we expected Xgboost could give better results with its proven performance in the literature. However, we got better results with random forests. We believe that we might need more hyperparameter tuning trial chance for Xgboost to get competing results since they are harder to tune than Random Forest. The reason for Random Forest performed better could also be due to the high noise level in the data that Xgboost could overfit. Because we know that Random Forests perform well on a data which tends to have a lot of statistical noise, it could result better for this task.

5. CONCLUSION AND FUTURE WORK

In this project we worked with training and test dataset as already split. Data has 60 variables (features) and a categorical dependent variable we wanted to predict using classification methods. Our performance measures for the models were balanced error rate and area under the ROC curve. Firstly, we decided on the classification methods to build methods that we learnt in the lectures: random forest, penalized regression, decision tree, and stochastic gradient boosting.

The first results with these methods were unsatisfactory since we were dealing with an imbalanced dataset. Hence, we tried sampling methods to alter the training data and shape it into a balanced dataset, make things easier for the classification method. We tried undersampling, oversampling, SMOTE, and sampling methods in the ROSE package.

We applied classifiers with each sampling method and compared the results. Undersampling gave the best score with the random forest model. We then used cross validation to find the best hyperparameters and tuned the model. We noticed that when we increased the complexity of the model, the results have gone worse. Hence, our final model is applying random forest instead of a more advanced model such as xgboost or catboost.

As for the future work, we can utilize the sampling method to come up with a more advanced setting, such as using undersampling and downsampling at the same time to create more data while maintaining the balance. Secondly, we can try a wider hyperparameter set for our models. We opted to keep it simple with the parameter sets in this project since simpler models seemed to work better. However, we might have missed a parameter setting that performs better than our model.

Another addition to our model can be dimensionality reduction. There are 60 features in the model, and it may cause curse of dimensionality. We can further improve our model by trying dimensionality reduction methods such as Principal Components Analysis and Multidimensional Scaling, then apply sampling and classifiers.

Finally, we can extend the classification models we tried with methods such as Bagged Decision Trees, Extra Trees, Support Vector Machine, Artificial Neural Networks, and k-Nearest Neighbors. We need to make sure that the minority class is represented in the model with all these models and find out which one performs best with our binary classification problem.

6. REFERENCES

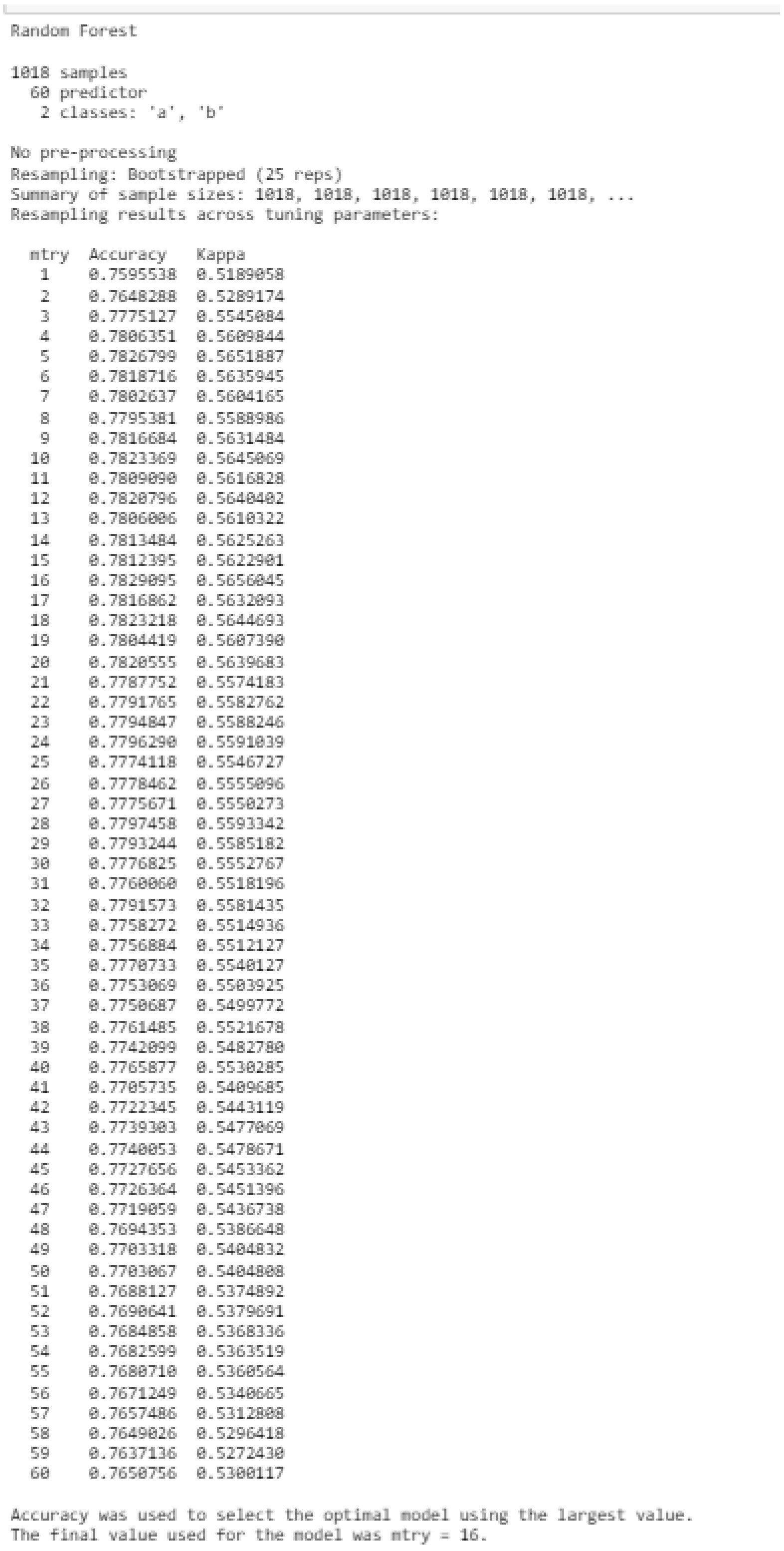
Cateni, S., Colla, V., & Vannucci, M. (2014). A method for resampling imbalanced datasets in binary classification. *Neurocomputing*, 32-41.

L. Gonzalez-Abril, C. Angulo, H. Nuñez, & Y. Leal. (2017). Handling binary classification problems with a priority class by using Support Vector Machines. *Applied Soft Computing*, 661-669.

7. APPENDIX

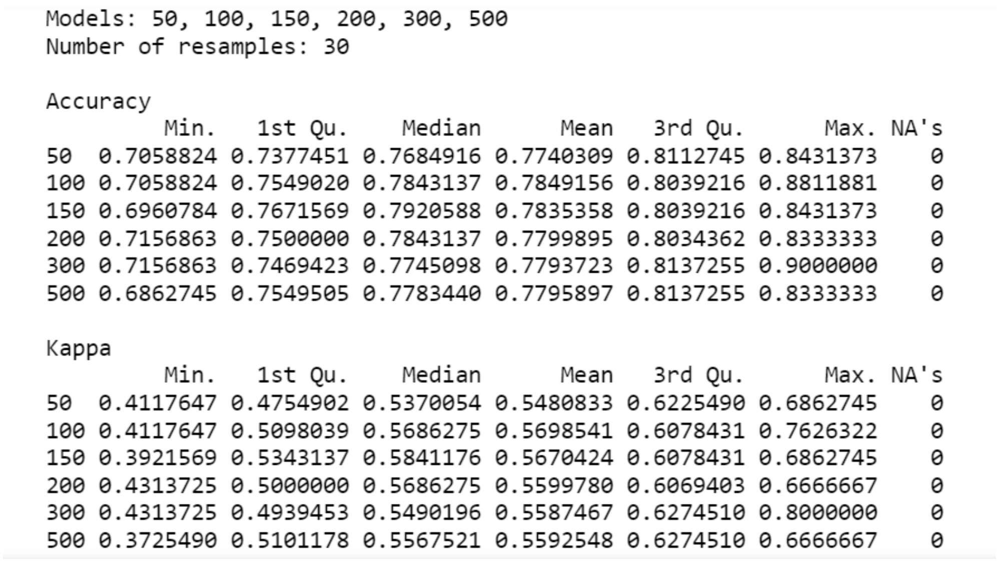
Cross Validation Results for parameters:

For mtry:



```
Random Forest

1018 samples
  60 predictor
   2 classes: 'a', 'b'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 1018, 1018, 1018, 1018, 1018, 1018, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   1    0.7595538  0.5189058
   2    0.7648288  0.5289174
   3    0.7775127  0.5545084
   4    0.7806351  0.5609844
   5    0.7826799  0.5651887
   6    0.7818716  0.5635945
   7    0.7802637  0.5604165
   8    0.7795381  0.5588986
   9    0.7816684  0.5631484
  10    0.7823369  0.5645069
  11    0.7809090  0.5616828
  12    0.7820796  0.5640402
  13    0.7806006  0.5610322
  14    0.7813484  0.5625263
  15    0.7812395  0.5622901
  16    0.7829095  0.5656045
  17    0.7816862  0.5632093
  18    0.7823218  0.5644693
  19    0.7804419  0.5607390
  20    0.7820555  0.5639683
  21    0.7787752  0.5574183
  22    0.7791765  0.5582762
  23    0.7794847  0.5588246
  24    0.7796290  0.5591039
  25    0.7774118  0.5546727
  26    0.7778462  0.5555096
  27    0.7775671  0.5550273
  28    0.7797458  0.5593342
  29    0.7793244  0.5585182
  30    0.7776825  0.5552767
  31    0.7760060  0.5518196
  32    0.7791573  0.5581435
  33    0.7758272  0.5514936
  34    0.7756884  0.5512127
  35    0.7770733  0.5540127
  36    0.7753069  0.5503925
  37    0.7750687  0.5499772
  38    0.7761485  0.5521678
  39    0.7742099  0.5482780
  40    0.7765877  0.5530285
  41    0.7705735  0.5409685
  42    0.7722345  0.5443119
  43    0.7739303  0.5477069
  44    0.7740053  0.5478671
  45    0.7727656  0.5453362
  46    0.7726364  0.5451396
  47    0.7719059  0.5436738
  48    0.7694353  0.5386648
  49    0.7703318  0.5404832
  50    0.7703067  0.5404808
  51    0.7688127  0.5374892
  52    0.7690641  0.5379691
  53    0.7684858  0.5368336
  54    0.7682599  0.5363519
  55    0.7680710  0.5360564
  56    0.7671249  0.5340665
  57    0.7657486  0.5312808
  58    0.7649026  0.5296418
  59    0.7637136  0.5272430
  60    0.7650756  0.5300117

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 16.
```

For ntree:



```
Models: 50, 100, 150, 200, 300, 500
Number of resamples: 30

Accuracy
         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
50  0.7058824 0.7377451 0.7684916 0.7740309 0.8112745 0.8431373    0
100 0.7058824 0.7549020 0.7843137 0.7849156 0.8039216 0.8811881    0
150 0.6960784 0.7671569 0.7920588 0.7835358 0.8039216 0.8431373    0
200 0.7156863 0.7500000 0.7843137 0.7799895 0.8034362 0.8333333    0
300 0.7156863 0.7469423 0.7745098 0.7793723 0.8137255 0.9000000    0
500 0.6862745 0.7549505 0.7783440 0.7795897 0.8137255 0.8333333    0

Kappa
         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
50  0.4117647 0.4754902 0.5370054 0.5480833 0.6225490 0.6862745    0
100 0.4117647 0.5098039 0.5686275 0.5698541 0.6078431 0.7626322    0
150 0.3921569 0.5343137 0.5841176 0.5670424 0.6078431 0.6862745    0
200 0.4313725 0.5000000 0.5686275 0.5599780 0.6069403 0.6666667    0
300 0.4313725 0.4939453 0.5490196 0.5587467 0.6274510 0.8000000    0
500 0.3725490 0.5101178 0.5567521 0.5592548 0.6274510 0.6666667    0
```