DSE 511 Final Project Report

Lancaster Barnstormers:

Hairuilong Zhang

Inzamam Haque

Verónica G. Melesse Vergara

Tom Allemeier

The Bredesen Center, University of Tennessee, Knoxville

DSE 511: Introduction to Data Science & Computing

Instructor: Dr. Jacob Hinkle & Dr. Todd Young

12.09.2020

1. Introduction

The objective of this project is to do a deeper dive into the data provided in the financial database [1] by performing some statistics and machine learning to extract additional patterns from the data. The main idea of our group is to predict a loan status to see if it will be a "good" or "bad" loan based on all the associated features with the loan, such as client characteristics, account statistics, and so on. Generally speaking, this is a binary classification problem. In view of the fact that our original data has a severe class imbalance, with 606 good labels and 76 bad labels, we decided to implement the Synthetic Minority Oversampling Technique (SMOTE) to compare performance between original data and oversampled data. Overall, three classification algorithms, k-Nearest Neighbors (kNN), random forest (RF), and support vector machine (SVM) are applied on original and over-sampled data respectively. Performance will be compared and some insightful findings will be discussed as well.

This project was collaborated through GitHub, and the repository link was attached in the appendix section. By running *execution.py* in the command line, the output will be printed to the screen and figures will be saved as well.

2. Data Preparation

This section will briefly introduce the information of the dataset and the process of preparing the data used for fitting in different classifiers.

This database is about anonymized bank transactions covering six years of bank transactions for 5,369 bank clients with 4,500 accounts. It consists of the following 8 tables: account, client, disposition, permanent order, transaction, loan, credit card, and demographic data. The loan table will be the key dataframe we focus on and

manipulations like join, merge and aggregation will be conducted based on the structure and relationship between these tables. The relationship is illustrated in the following Figure 1.

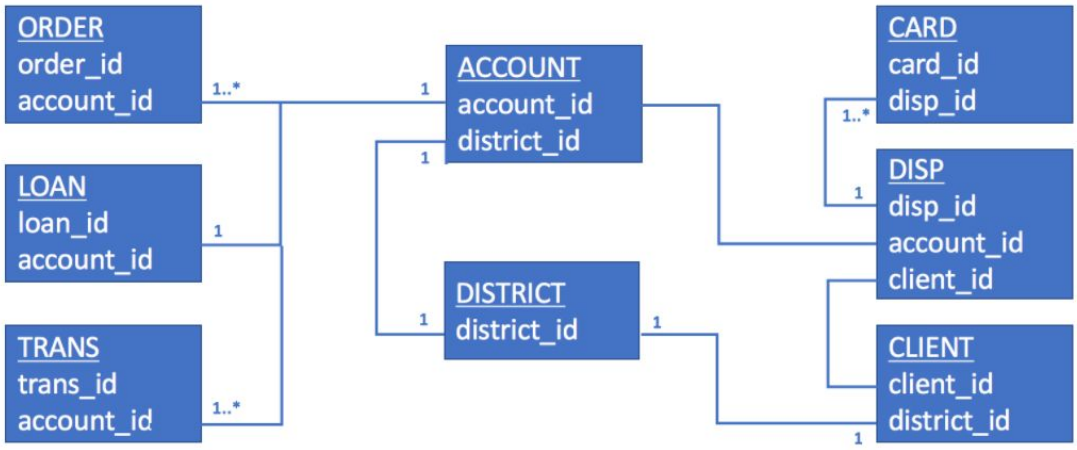

Figure 1. Dataframe Relationship [2]

Creating training features is of great importance to building the classification model. In this project, both static and dynamic features are taken into consideration. Static features refer to account, client and loan characteristics while dynamic feature indicates variables derived from the transaction history of the associated account. In the implementation, minimal, maximal and mean value of the account balance for the cumulative M months (M = 1, 2, 3, 4, 5, 6) before the loan date are computed. These dynamic features are able to reflect the financial condition of the account directly, which plays an important role in determining the status of the loan. Overall, there are 29 numeric features with all categorical and date variables converted into numeric types. This data preparation process is implemented via *prepare_data.py* module. And the detailed instructions of running the program are provided in the attached GitHub repository.

3. Methodology

This section will describe the three learning algorithms we studied for this project as well as the details about the individual implementation: k-Nearest Neighbor, Support Vector Machine, and RandomForest.

3.1. k-Nearest Neighbor (kNN)

In order to implement the k-nearest neighbor algorithm (kNN), we used the *kNeighborsClassifier* function provided in the *Nearest Neighbors* [4] module (*sklearn.neighbors*) of the scikit-learn machine learning library [3]. The code also utilizes scikit-learn's pre-processing, model selection, and metrics modules.

The implementation makes use of the *prepare_data* module to obtain two versions of the data set. One that includes the samples as provided in the financial database, and one that utilizes the oversampling technique described in the previous section (Section 2).

The code gives the user the option to pass a specific number of samples (*k* value) that the algorithm should use to classify the data set.

There are four key routines in the code: *fit_transform()* which normalizes the data set, *train_test_split()* which selects subsets of the data to be used for training versus testing, *fit()* which estimates the parameters for the model given the training data set, and, finally, *predict()* which returns the classification labels of the test data set. Both the training and prediction routines require an estimator. In this case, we are using the *kNearestNeighborClassifier()*.

Once a prediction is obtained, we rely on scikit-learn's *score()* routine to measure the accuracy of the learning algorithm.

3.2. Support Vector Machine (SVM)

The Support vector Machine algorithm was implemented using the *svm.SVC* module [5] of the scikit-learn machine learning library [3]. The code also makes use of scikit-learn's decomposition, pre-processing, model selection, and metrics modules.

First, as like kNN, the *prepare_data* module is used to obtain the original and oversampled data.

Different values of hyperparameters suchs as kernel, gamma - kernel coefficient, C - regularization coefficient, have been tried and tested. The best result was obtained with a *'rbf'* kernel, *'gamma'* set to *'auto'* and *C=3.0.*

The key routines in the code are *fit_transform()* which normalizes the data set, *train_test_split()* which divides the data into subsets for training and testing, *fit()* which estimates the parameters for the model given the training set, and *predict()* which returns the classification labels of the test set. The estimator used for the training and prediction routines in this case is *svm.SVC()*.

Once a model is selected, we use scikit-learn's *roc_auc_score()*, *accuracy_score()* and *classification_report()* to measure the performance of the learning algorithm, as well as to compare the results between the original data and the oversampled data.

Finally, a decision surface is plotted after reducing the dimension using PCA. It is done for both the original and oversampled dataset. The effect of oversampling is clearly visible in these figures.

3.3. Random Forest (RF)

To implement the Random Forest algorithm, we used the *RandomForestClassifier* function provided in the *Ensemble* module *sklearn.ensemble* [6] of the scikit-learn machine learning library [3]. The code also makes use of scikit-learn's model selection and metrics modules.

Like the other algorithms, this implementation uses the *prepare_data* module to obtain the original and oversampled data.

The key routines in the code are *train_test_split()* which divides the data into subsets for training and testing, *fit()* which estimates the parameters for the model given the training set, and *predict()* which returns the classification labels of the test set. The estimator used for the training and prediction routines in this case is *RandomForestClassifier()*.

Once a model is selected, we use scikit-learn's *roc_auc_score()*, *accuracy_score()* and *classification_report()* to measure the performance of the learning algorithm, as well as to compare the results between the original data and the oversampled data.

4. Performance Comparison

We utilized each algorithm described in Section 3 to classify the loan financial data. Our goal was to classify the data available to determine whether the loan would have a

"good" versus a "bad" status. After preprocessing, the resulting data contained 29 features and used two classes.

Figures 2 and 3 show the results obtained from each classifier when applied to the original data set and the oversampled version.

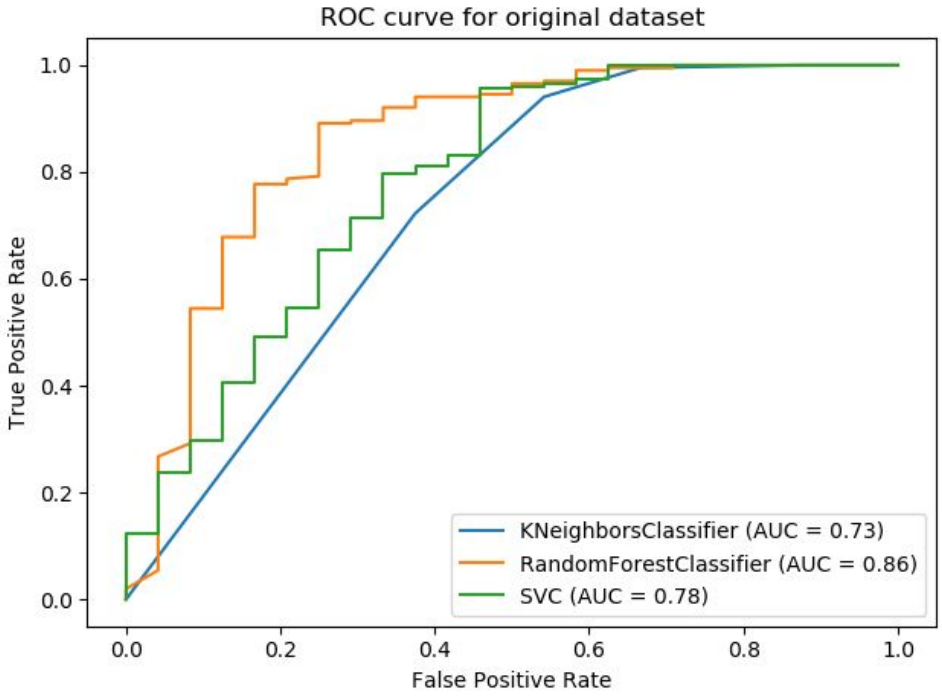

Figure 2. ROC curve comparing the three classifiers: kNN, Random Forest, and SVM applied to the original data set.

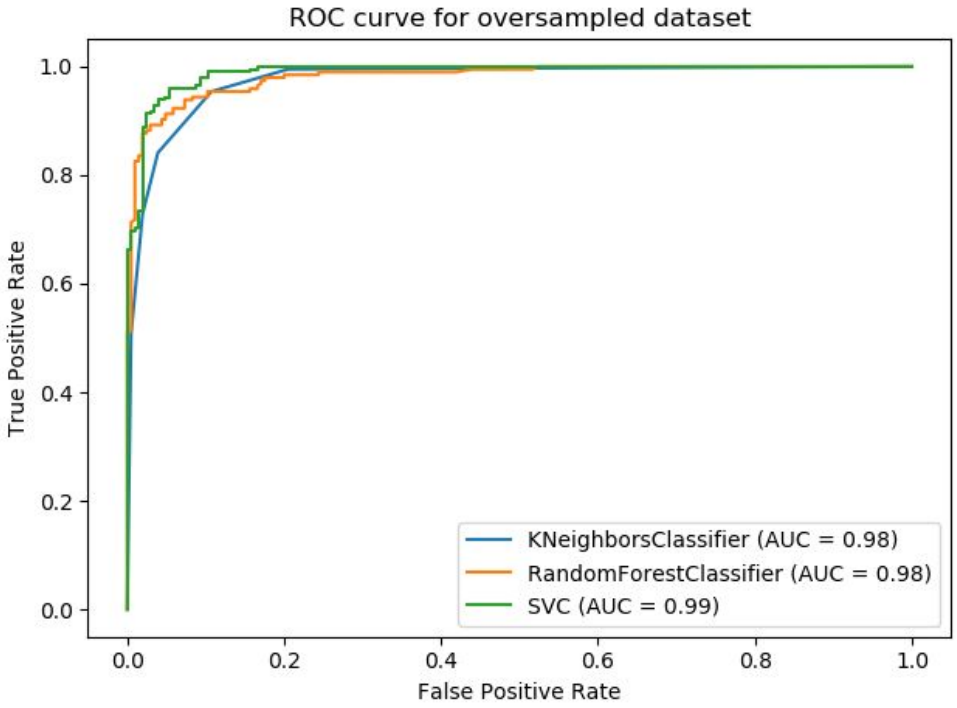

Figure 3. ROC curve comparing the three classifiers: kNN, Random Forest, and SVM applied to the oversampled data set.

For the kNN algorithm, we used k=5 and obtained an accuracy of 92.4% when using the original data set and 90.3% when using the oversampled data set. Results from these experiments are summarized in the confusion matrices shown in Figure 4.

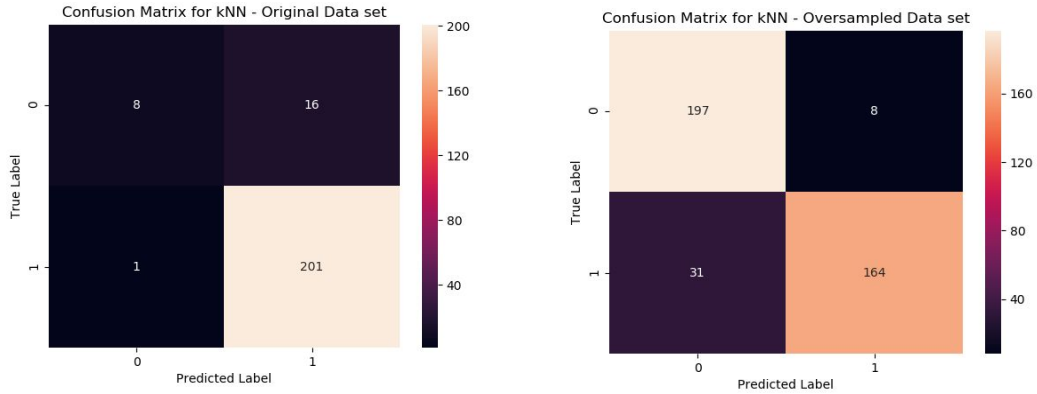

Figure 4. Confusion matrices from kNN (k=5) on the original (left) and oversampled (right) data sets.

For SVM, using the *'rbf'* kernel, *'gamma'* set to *'auto'* and *C=3.0,* returned 91.15% accuracy for the original dataset, which is misleading because the precision and recall were respectively 0.79 and 0.68. But after applying oversampling with the same configuration, a 0.95 value was obtained for accuracy, precision and recall.

The confusion matrices for SVM are given below.

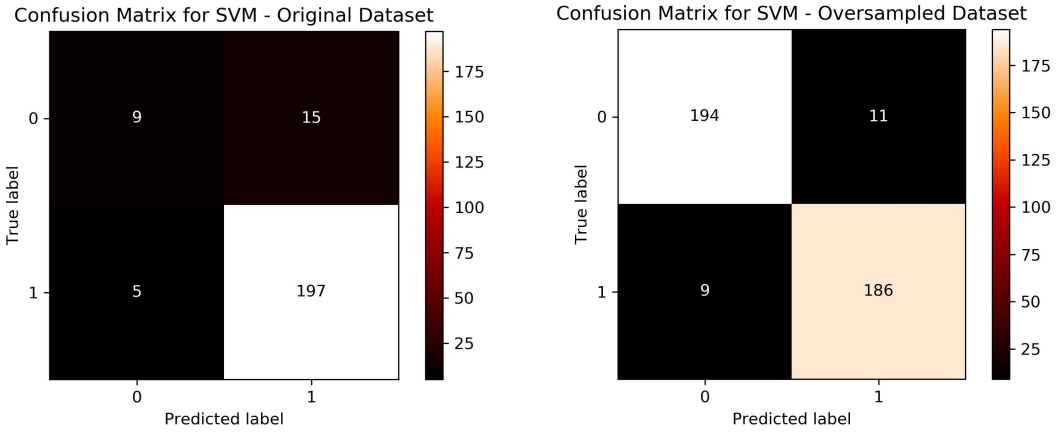

Figure 5. Confusion matrices from SVM on the original (left) and oversampled (right) data sets.

The decision surface with SVM for both original and oversampled datasets are given below.

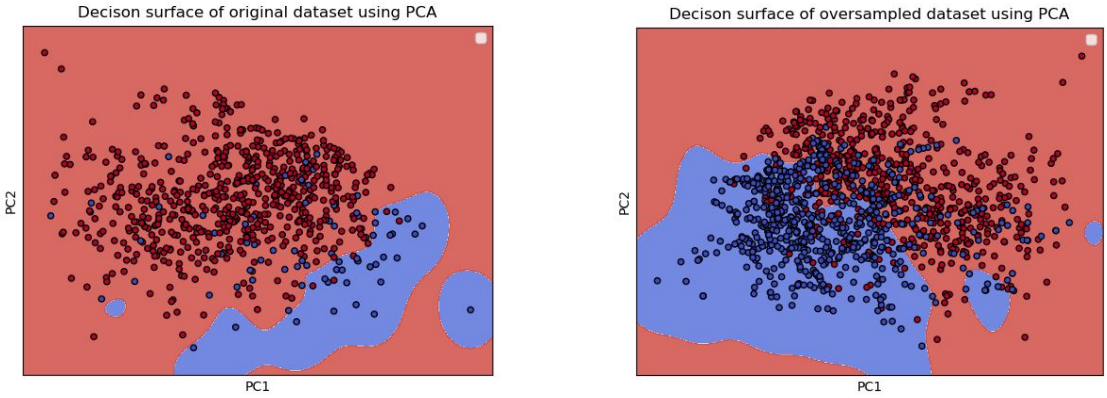

Figure 6. Decision surfaces from SVM on the original (left) and oversampled (right) data sets.

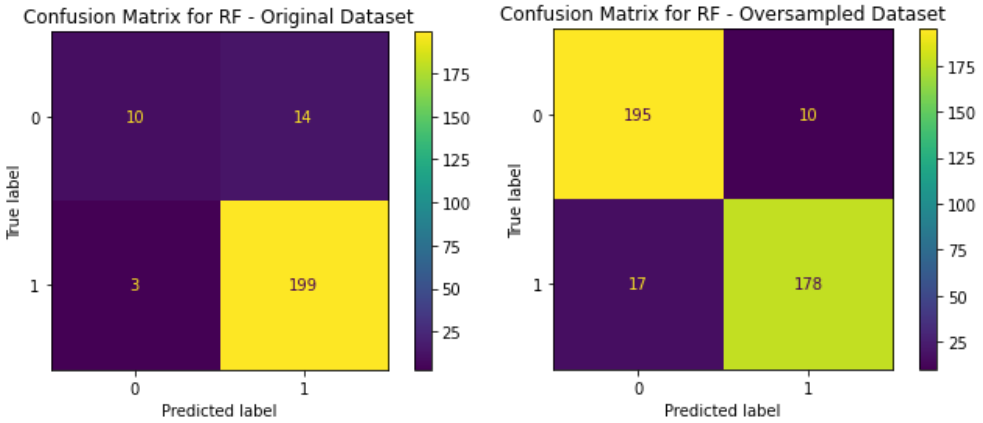

Figure 7. Confusion matrices from RF on the original (left) and oversampled (right) data sets.

The RF achieved accuracies of 92 and 93 percent for the original data and oversampled data respectively. Like with the other methods, the oversampled data had higher precision and recall.

5. Conclusions

Accuracy on original data was highest for kNN at 92.4%, followed by RF at 92% and SVM at 91.15%. Accuracy on the oversampled data was highest for SVM at 95%, followed by RF at 93% and kNN at 90.3%.

Looking at the execution time of each algorithm, we see that RF is the most time consuming of the three taking 1.65 seconds for the original data set and 2.41 seconds for the oversampled data set. The fastest algorithm was kNN applied to the original data set which took 0.012 seconds.

For all algorithms, we observed a higher recall when using the oversampled data set. In addition, we also obtained a higher precision when using the oversampled data set. This is an indication that the model trained using the oversampled data set was able to provide more complete and valid results than that trained with the original data set.

Overall, the SVM model provided the best classification when comparing the parameters measured in this study.

References

1. PKDD'99 Discovery Challenge: http://sorry.vse.cz/~berka/challenge/pkdd1999/berka.htm
2. Predicting bank loan behavior by analyzing bank transactional data with a random forest classifier: http://research.ganse.org/datasci/loanpredict/
3. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
4. Nearest Neighbors, https://scikit-learn.org/stable/modules/neighbors.html#unsupervised-neighbors
5. Support Vector Machine,

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

6. Random Forests,

https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees

Appendix

The source code for this project is available at:

https://github.com/inzamam1190/Lancaster_Barnstormers_DSE511