CS 315 Final Project Report

Quynh Tran

Credit Card Approval Prediction using Scikitlearn

Table of Contents

I. INTRODUCTION

As we all know, credit cards are loans that consumers pay back every month based on the amount they spend. In the financial sector, almost all financial institutions use credit scores to quantify risk quantitatively. Using this common ground, I hope to utilize the project to help identify defaulter and their repaying abilities. This project employs classifier algorithms to classify applications into good and bad categories to aid in decision-making processes: whether to grant credit cards to applicants or not. Intending to incorporate what we have discussed in class, I aim to minimize losses and bad debts through preventive measures.

I started this project with some questions:

- In what ways can Machine Learning be realistically applied to different fields? Then when I got approved for my third credit card, I started to wonder if ML played a part in the decision of my bank.
- How do I build a classification model using my knowledge of python and some other tools that I have learned in class like Scikitlearn.
- I briefly got exposed to the negative effect of imbalanced data, so I am curious about how to combat this problem in real life

After coming up with the initial questions, I am intrigued to learn about the process of determining the best classification model that yields the highest accuracy and overcoming highly imbalanced data. The final result is expected to be used to predict if an applicant is allowed to get a loan or approved for a credit card.

Some of the biggest challenges come from the data processing stage as there is more than one available approach to data resampling. Additionally, most of the data's features are categorical, so they will need to be converted to numerical.
To achieve this goal, I decided to oversample my data from the minor class using both SMOTE (Synthetic Minority Oversampling Technique) and ADASYN (Adaptive Synthetic). Four models are used to classify the oversampled data: Naive Bayes, K-nearest Neighbors, Random Forest Trees, and Support Vector Machines (SVMs). After

obtaining the results, the winner model is selected by its accuracy score and ROC AUC (the area under the ROC curve).

As a whole, SVM and Random Forest are the most suitable models for data that has been resampled by SMOTE. When ADASYN resamples data, SVM fails to perform, so Random Forest is ultimately the winner.

II. DATA MINING TASK

The input data is a csv file named Application_Data that contains 25,128 rows and 21 columns. The six features that are categorial (Applicant gender, Income type, Education type, Family Status, Housing type, Job title) are then converted to numerical (see Figure 2) for classification purposes. The output is expected to reflect the accuracy score and ROC AUC score of each classification and type of oversampled data (see Figure 3).

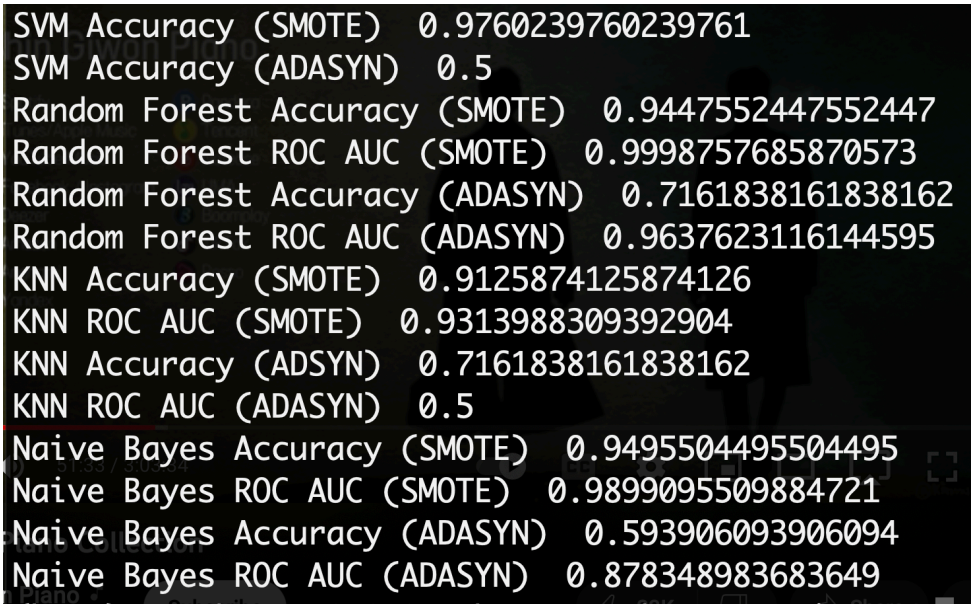
Figure 1. Input Data Information

```
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Applicant_ID          25128 non-null  int64
 1   Applicant_Gender      25128 non-null  object
 2   Owned_Car             25128 non-null  int64
 3   Owned_Realty          25128 non-null  int64
 4   Total_Children        25128 non-null  int64
 5   Total_Income          25128 non-null  int64
 6   Income_Type           25128 non-null  object
 7   Education_Type        25128 non-null  object
 8   Family_Status         25128 non-null  object
 9   Housing_Type          25128 non-null  object
 10  Owned_Mobile_Phone    25128 non-null  int64
 11  Owned_Work_Phone      25128 non-null  int64
 12  Owned_Phone           25128 non-null  int64
 13  Owned_Email           25128 non-null  int64
 14  Job_Title             25128 non-null  object
 15  Total_Family_Members  25128 non-null  int64
 16  Applicant_Age         25128 non-null  int64
 17  Years_of_Working      25128 non-null  int64
 18  Total_Bad_Debt        25128 non-null  int64
 19  Total_Good_Debt       25128 non-null  int64
 20  Status                25128 non-null  int64
```

Figure 2. Processed Data Information

```
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Applicant_ID          25128 non-null  int64
 1   Applicant_Gender      25128 non-null  int64
 2   Owned_Car             25128 non-null  int64
 3   Owned_Realty          25128 non-null  int64
 4   Total_Children        25128 non-null  int64
 5   Total_Income          25128 non-null  int64
 6   Income_Type           25128 non-null  int64
 7   Education_Type        25128 non-null  int64
 8   Family_Status         25128 non-null  int64
 9   Housing_Type          25128 non-null  int64
 10  Owned_Mobile_Phone    25128 non-null  int64
 11  Owned_Work_Phone      25128 non-null  int64
 12  Owned_Phone           25128 non-null  int64
 13  Owned_Email           25128 non-null  int64
 14  Job_Title             25128 non-null  int64
 15  Total_Family_Members  25128 non-null  int64
 16  Applicant_Age         25128 non-null  int64
 17  Years_of_Working      25128 non-null  int64
 18  Total_Bad_Debt        25128 non-null  int64
 19  Total_Good_Debt       25128 non-null  int64
 20  Status                25128 non-null  int64
```

Figure 3. Accuracy score and ROC AUC Output from Terminal



The mining questions that I have for this data are:

- How much data is to be used for training and testing purposes?
- What method to use to overcome imbalanced data? In case there are multiple, how do I select the best one?
- Classification models to apply to this data to get desired realistic results
- What evaluation metrics should be reported to reflect real-world application perspectives?

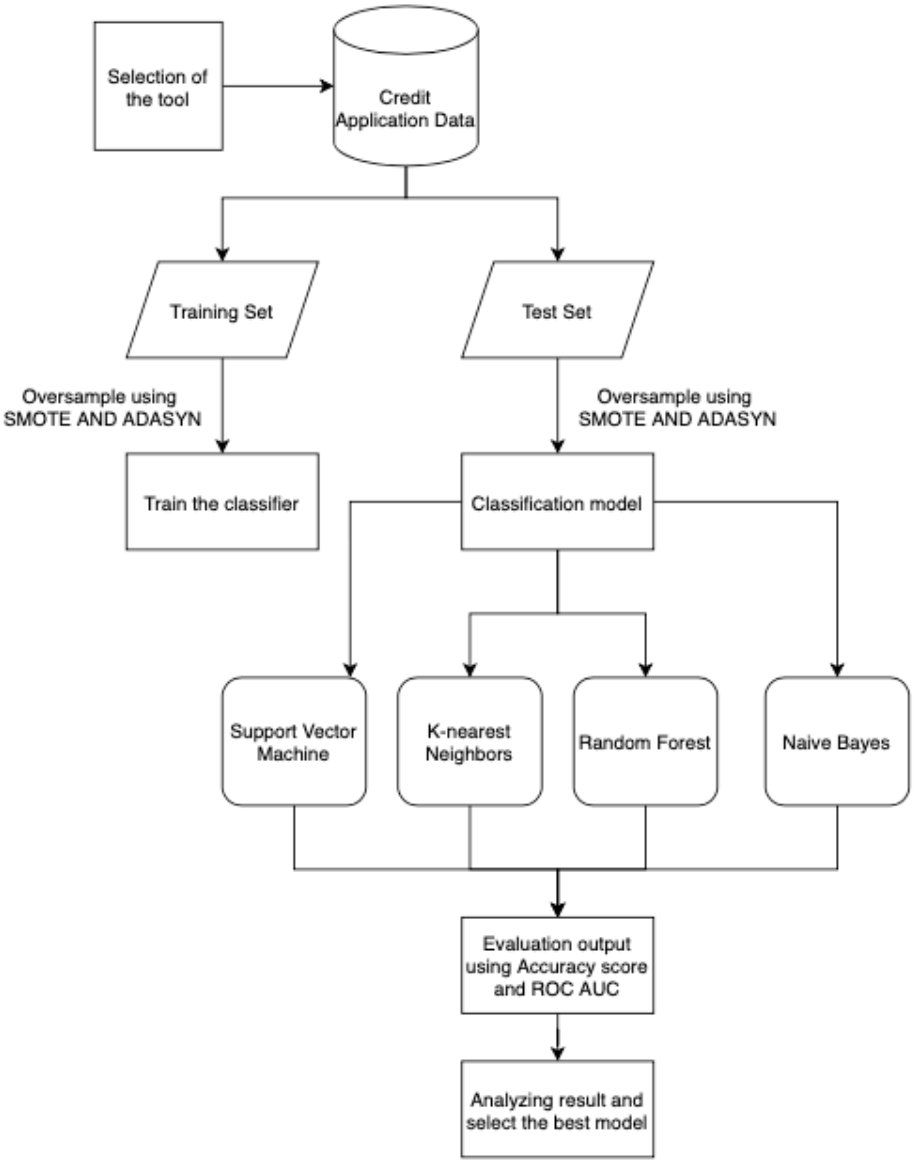
A key challenge is to combat data imbalances that may lead to biased predictions. This is because the feature 'Status' in the data indicates that the number of approved applications is significantly higher than rejected ones. Thus, choosing the appropriate methods to resample the data to ensure the quality of the original data but also avoiding bias or imbalance data is an imperative but also difficult task for me. Additionally, classifying applicants with abstract definitions of the categories is a real problem that I have to overcome as for most of the homework, the categories are clearly stated. Regarding employing the right metrics to evaluate the model output, I Googled and learned that different measurements can be used to measure the success

of a classification model such as accuracy score, F1 score, and the area under the ROC curve. Thereby, selecting the most meaningful measurement is another major challenge.

III. TECHNICAL APPROACH

The framework of this project is represented in the diagram below (Figure 4)

Figure 4. System Flow Diagram



Following the framework, below are the key steps of my project:

1. Converting categorical features to numerical. This task can be done using Preprocessing package form Scikitlearn. However, Job_Title feature contains about 18 occupations, so a dictionary is created to map the occupation title to a corresponding number. The intermediate result of this pre-processing step has all numerical values (see Figure 2.)

2. Splitting data to training and testing datasets and processing them.

- After removing the data labels (Applicant ID and Status), 80% of the data is used as training data and 20% is used as the testing set. This step is essential to avoid data leakage and unrealistic results.

- Standardize the train data before resampling the data.

- Oversample the train data and label; and test data and label using SMOTE and ADASYN algorithms.

3. Apply SVM to the balanced train and use the model to predict the balanced test data. Compute accuracy score on the test data and ROC AUC.

4. Similarly, repeat with Random Forest, KNN, and Gaussian Naive Bayes classifiers.

For KNN, I tried 3 different values for k (number of neighbors) and obtained:

- k = 5

```
KNN Accuracy (SMOTE)  0.8987012987012987
KNN ROC AUC (SMOTE)  0.9118018245490772
KNN Accuracy (ADSYN)  0.7607392607392608
KNN ROC AUC (ADASYN)  0.5
```

- k = 10

```
KNN Accuracy (SMOTE)  0.9104895104895104
KNN ROC AUC (SMOTE)  0.9219252675396531
KNN Accuracy (ADSYN)  0.7607392607392608
KNN ROC AUC (ADASYN)  0.5
```

- k = 15

```
KNN Accuracy (SMOTE)  0.9122877122877123
KNN ROC AUC (SMOTE)  0.9292723460355828
KNN Accuracy (ADSYN)  0.7607392607392608
KNN ROC AUC (ADASYN)  0.5
```

The accuracy score of SMOTE data appears to increase as k increases, so I decided pick k = 20 as it yields the highest accuracy (see Table 2).

Regarding resample method, I decided to oversample the minor class as we want more data instead of removing some data from the data set. ADASYN and SMOTE the most common approach to resolving imbalance, so I tried both and compared the accuracy that they yield. Besides, different packages and classifiers from Scikitlearn were imported to help with the data processing. It was very intriguing to learn about what Scikitlearn can provide to solve various data mining tasks. Pandas was another tool that help save time when loading and transforming data.

Throughout the project, I have learned about the importance of the difference in usability of Predict() and Predict_proba functions in Scikitlearn library. I did some research about this and one person on Kaggle advised that Predict() predictions should not be used to determine the score of ROC AUC. Scikitlearn website states: "[1]in binary classification, a sample may be labeled by predict as belonging to the positive class even if the output of predict_proba is less than 0.5; and similarly, it could be labeled as negative even if the output of predict_proba is more than 0.5." In other words, Predict() can easily fail to predict labels during the process of building the ROC curve and evaluating different thresholds.

It was difficult to select the metrics that would be use to evaluate the classifiers' performance; however, due the properties and application of this study, I prioritized accuracy score and ROC AUC.

Furthermore, runtime is an important factor when I implemented my code, so I had to modify my SVM algorithm to ensure the runtime is appropriately shortened. To compute ROC AUC for SVM, it requires that the *probability* property of the model is enabled to True (default is False). This, however, is very costly in terms of execution time. A run takes up to 16min, so I commented this out in my source code file.

---

[1] https://scikit-learn.org/stable/modules/svm.html#scores-probabilities

Figure 5. Classification Algorithm Pseudo Code

CLASSIFICATION ALGORITHM PSEUDO CODE

```
import package from sklearn
model = classifier
model.fit( train data, test data )
y_pred = model.predict( train label )
accuracy = accuracy_score( test label, y_pred)

roc = model.predict_proba( train data)[ :,1]
roc_auc = roc_auc_score( test label, roc)
```

The Pseudo in Figure 5 is applied twice for each classification model. The first time is for data that is oversampled using SMOTE. And the second run is for data that is oversampled using ADASYN.

IV. EVALUATION METHODOLOGY

The input data for this project came from Credit Card Approval Prediction (Cleaned Version). Some jobs were done to this dataset like removing duplicate and null values, calculating and adding Status column based on total good debt and bad debt from the original data, which came from Credit Card Approval Prediction (Original). The original data was collected and translated from the website of a Chinese financial institution. It has two tables that connect by applicant id. The cleaned version, which is used as the input data for this project, is a nicer version that contains 21 columns and 25,128 rows without any null or duplicated values. Yet, the data for Status column is significantly skewed as more than 99% of the values are 1 and only less than 1% is 0 (see Table 1). If used to build a classifier, this highly imbalanced dataset results in a "perfect" classifier that is 100% accurate on the training data, but totally inaccurate on the testing data. In other words, the imbalance of the data will inevitably produce a biased predictive model.

The metrics that were employed to evaluate the output of my data mining tasks are the accuracy score and the area under the ROC curve. From a real-world applications perspective, the goal of this project is to help financial institutions with the decision making process regarding approving or rejecting a loan application. Therefore, it is vital

that the results must be easy to explain to non-technical stakeholders. Moreover, every class in the input data has a certain effect on the repaying ability of an individual, so all classes are equally important. It is also crucial to know the reason to reject an application, so we should also equally care about both positive and negative classes. In other words, both true negatives and true positives should be taken into account and consideration. Thus, accuracy score and ROC AUC are the ultimate metrics to use for the result evaluation of this project.

V. RESULTS AND DISCUSSION

SMOTE and ADALYN oversampling methods indeed generate different sample sizes (see Table 1). Thus, the difference leads to a variation in performance of classification models.

Table 1. Number of 1 and 0 values from Status column

	1	0
Original dataset	25,007	121
Original train data label	20,002	100
Oversampled Train label - ADASYN	20,011	20,002
Oversampled Train label - SMOTE	20,002	20,002

Table 2. Accuracy Score & ROC AUC of 4 Classifiers

		SMOTE	ADASYN
SVM	Accuracy score	0.976	0.5
	ROC AUC	0.996	0.989
Random Forest	Accuracy score	0.947	0.716
	ROC AUC	0.999	0.964
KNN (k = 20)	Accuracy score	0.913	0.716
	ROC AUC	0.931	0.5
Naive Bayes	Accuracy score	0.949	0.593
	ROC AUC	0.989	0.878

As shown in Table 2 and Figure 3, the four chosen classifiers work well on the data sets that are generated by SMOTE as all accuracy scores and ROC AUC scores stay above 90%. Meanwhile, the scores obtained from the 4 classifiers are generally lower for ADASYN data, ranging from 50-80%. SVM and KNN both only result in 50% accuracy score and ROC AUC score respectively. KNN appears to be the best classification model for this dataset.

Generally, ADASYN is known to be an optimized version of SMOTE that is created to overcome some drawbacks of SMOTE. However, in this case, the results show us the opposite. The reason behind this can be that ADASYN is putting too much attention on the decision boundary where the region is less dense. These low-density areas, however, are typically outliers. Thus, this causes the performance of the model to decrease.

A concern regarding the implemented SVM algorithm that is mentioned earlier in Technical Approach is the execution time once the parameter probability is enabled to True. The extensively long runtime is expected as Scikitlearn indicates on its website that this process is very costly when the training is relatively large (over 20,000 instances in this case).

Regarding KNN classifier, it works very well on SMOTE-generated data. By trying 4 different values of k, I noticed that the highest accuracy score that the model achieves is about 93% at k = 20. ADASYN-generated data seems to not be impacted by the adjustment of k. Even ADASYN data has lower accuracy compared to SMOTE, 77.6% is relatively high for common cases. As mentioned above, the lower performance of ADASYN can be caused by the way ADASYN focuses on low-density areas.

VI. LESSONS LEARNED

I have learned about the serious effect of imbalanced data on the performance of the data model. For this case, using imbalanced data to build a naive data model can yield a perfect accuracy score of 100%. however, this model is not usable when applied to test or unseen data. This project provides me an opportunity to dive deep into oversampling methods: ADASYN and SMOTE and how to implement them using python and sklearn packages. On the other hand, as a complement lesson, I have learned that data exploration is fundamental to learning thoroughly about the dataset that later on will help with the analysis and code construction

Even though my model performs relatively well, there are still lots of room for development and improvement. Some improvements that could have been done to obtain better results or help leverage my skills:

- I would try to work with the preprocessed dataset that is unlabelled and contained nulled values to leverage my skills in terms of data processing This will prepare me for my professional career as well when I have to deal with live data
- Hypertuning model parameters techniques can be applied to some models to improve accuracy scores. For example, KNN, packages like GridSearchCV can be used to find the optimal k. Tuning parameters of SVM can also improve the accuracy score and ROC AUC event though it comes with a cost- longer runtime.
- More classification models can be applied to the data set, so we can compare them with our winner model.
- I could have saved some data from the original input data to test the model that I created as unseen data. Furthermore, cross-validation methods can be applied to the model to ensure its accuracy. This would let us estimate how well the model performs in reality.

VII. ACKNOWLEDGEMENTS

Class notes, Kaggle, and the Scikitlearn website are the three main sources that I use to build my classification models. My System Flow diagram (Figure 4) is derived from a paper from Research Gate.

WORKS CITED

1.4. Support Vector Machines. scikit. (n.d.). Retrieved December 12, 2022, from https://scikit-learn.org/stable/modules/svm.html#scores-probabilities

Adasyn: Adaptive Synthetic Sampling Approach for ... - IEEE xplore. (n.d.). Retrieved December 10, 2022, from https://ieeexplore.ieee.org/document/4633969

Credit card approval prediction. Kaggle. (n.d.). Retrieved December 12, 2022, from https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction

Torvekar, N. (2019, January). *Predictive analysis of Credit Score for credit card defaulters*. Predictive analysis of credit score for credit card defaulters. Retrieved December 9, 2022, from https://www.researchgate.net/publication/332557433_Predictive_analysis_of_credit_score_for_credit_card_defaulters