Credit Score Classification

Mamnuya Rinki
George Mason University
Fairfax, VA, USA
mrinki@gmu.edu

Sai Sharanya Garika
George Mason University
Fairfax, VA, USA
sgarika@gmu.edu

Kaartikeya Reddy Poli
George Mason University
Fairfax, VA, USA
kpoli@gmu.edu

Vishal Reddy Basani
George Mason University
Fairfax, VA, USA
vbasani3@gmu.edu

ABSTRACT

With the increasing amount of credit-related data being collected globally, there is a growing need to streamline how we evaluate an individual's creditworthiness. Credit worthiness applies to insurance, employment, mortgages, auto loans, and more. To address the challenge of streamlining credit worthiness, our proposal aims to develop machine learning models capable of categorizing credit scores based on their financial history and basic bank details. By automating this classification process, our objective is to significantly reduce manual efforts and improve the efficiency of credit assessment. Implementing efficient models will improve accuracy and consistency in credit scoring. This will lead to enhanced decision-making capabilities and optimized risk management strategies in the finance sector. We analyze various machine learning models and evaluate the most suitable model on a dataset that is rich in personal and financial information. We explore the classification results of Random Forest (RF), K-Nearest Neighbor (KNN), Gaussian Naive Bayes (GNB), and Extreme Gradient Boosting (XGBoost, also referred to as XGB) Classifiers. Additionally, we consider the classification with combinations of applying Synthetic Minority Over-sampling Technique (SMOTE), not applying SMOTE, applying outlier removal, and not applying outlier removal. This information can be used to understand relevant methods for financial institutions to consider when analyzing credit scores.

Key words: Credit Score, Classification, SMOTE, Credit, Classify, Outliers, KNN, Extreme Gradient Boost, Gaussian Naive Bayes, Random Forest

1. INTRODUCTION

Credit score classification is valuable for employment, mortgages, and more. This research identifies multiple models that can be used for classification and evaluates each model. These metrics can provide an overview of model performance for different variations of Random Forest, K-Nearest Neighbor, Gaussian Naive Bayes, and XGBoost Classifiers, with or without SMOTE, and with or without outliers. Our research contributes to the classification field by narrowing down the strengths of different models in credit score classification from a dataset rich in personal and financial information, without nationality or business related or biases.

This paper explores a dataset and the steps taken to prepare the data for our models. We discussed feature engineering, feature selection, and the application of SMOTE. The performance results and confusion matrices for each model variation are provided. There is a discussion on related works in the field and conclusions on the best models for classifying credit scores.

2. DATA

The dataset consists of 28 columns and 100,000 rows [4]. The dataset includes descriptions of the customer. There are columns for personal information like name, occupation, and age. There is financial information of the customer, like number of loans, the interest rate of a credit card the customer holds, and outstanding debts.

The column for credit scores was fundamental in training our supervised classification models. The credit score column places the customer in 3 classes of "Poor," "Standard," and "Good." It is shown that 53% of the credit score column consists of "Standard" credit scores. This information was used to classify and predict which customers can be classified with "Poor," "Standard," or "Good" credit scores.

2.1. Attributes of Data

The dataset contained different attributes representing personal and financial information of customers. There were 28 columns total, including the target column of

credit scores. Each column provided unique information that could be utilized to determine credit worthiness of each customer [2].

2.1.1. Column List

- Row identification number
- Customer's identification number
- Month of the year
- Name
- Age
- Social security number
- Occupation
- Annual income
- Monthly net income
- Number of bank accounts
- Number of credit cards
- Interest rate of credit cards
- Number of loans
- Type of loan
- Average number of days delayed from the due date of payment
- Number of delayed payments
- Percentage of change in the credit card limit
- Number of credit card inquiries
- Credit mix
- Outstanding debt
- Credit card utilization ratio
- Credit history age
- Whether the payment received was the minimum amount due
- Total equated monthly installment
- Amount invested monthly
- Payment behavior
- Monthly balance
- Credit score

2.2. Pre-Processing

The dataset was stored in a pandas dataframe from a .csv file, and each column's data type and frequent values were examined.

Columns that contained string data were mapped and altered to numerical values representing the original ordered categorical variables. In the "Month" column, there are string representations of the month name, which we changed into numerical values. For example, 1 indicated January and 12 indicated December.

The "Occupation" column consisted of some string values such as "-". We replaced these values with null values.

Any entries of solely " " were replaced with empty strings. This applied to 8 columns. If there were columns with missing values, these columns were grouped by the "Customer ID" column. Then, they were replaced with null values, mean values, or mode values, calculated per column.

The "Credit History Age" contained values in the format "YY Years and MM Months". These values were transformed to consist of an integer representing the total number of months.

Next, LabelEncoder() from Python's scikit-learn library was implemented on the columns "Occupation," "Credit Mix," and "Payment of Min Amount," to convert the values in these columns into unique numerical identifiers. The "Credit Score" column was altered to map "Standard" to class 0, "Good" to class 1, and "Poor" to class 2. This converted the categorical string data in this column to numerical data.

Columns that were used to group the data together and provided personal customer information that should not be significant in predicting credit score classes for other customers were removed from the dataset. These columns were the customer identification, name, social security, row identification number, and loan type. This information was not incorporated into our models for predictions.

2.3. Feature Engineering and Selection

To enhance the dataset, we incorporate 3 new features. The new features are the total number of accounts, debt per account, and the debt-to-income ratio. These features are calculated in the following manner:

(1) Total Number of Accounts: The summation of an individual's number of bank accounts and number of credit card accounts.
(2) Debt per Account: An individual's overall outstanding debt divided by the total number of accounts.
(3) Debt-to-Income Ratio: An individual's debt per account divided by their reported annual income.

Before adding these new features, we dropped some personal identifying columns such as the social security number and name of an individual. After dropping columns, 26 columns remained. With our additional engineered columns, the dataset consists of 29 columns total.

We reduced the features to 14 features through feature selection to decrease the runtime. We chose 14 features to preserve about half of the most important columns. We did not include the "Credit Score" column in these 14 features since this was the target variable. To select 14 features, we implemented SelectKBest() from scikit-learn library with k=14 to select the top 14 columns. To determine how to find the top 14 features, we used the Mutual Information score. This score measures the dependency between the features. The mutual information score was the highest for the following 14 columns:

- Annual Income
- Monthly net income
- Number of credit cards
- Interest Rate of Credit Cards
- Average number of days delayed from the due date of payment
- Number of credit card inquiries
- Credit mix
- Outstanding debt
- Credit history age
- Whether the payment received was the minimum amount due
- Total equated monthly installment
- Total number of accounts
- Debt per account
- Debt-to-income Ratio

The columns with the lowest dependence, and therefore not included in the data for our models were:

- Month of the year
- Age
- Occupation
- Number of bank accounts
- Number of loans
- Number of delayed payments
- Percentage of change in the credit card limit
- Credit card utilization ratio
- Amount invested monthly
- Payment Behaviour
- Monthly Balance

3. METHODS

After performing feature engineering and selection on the dataset, we implement various models and experiments. We implement 4 classification models, check accuracy, and check performance for the data with SMOTE, without SMOTE, with outliers, and without outliers.

3.1. Outlier Detection and Removal

A K-Nearest Neighbor algorithm was implemented to perform outlier detection, with the number of neighbors, k, set to 5. After ensuring the columns were numerical, distances between all points were calculated. A threshold was established of 1,000 units and any data points over 1,000 units away from other points were deemed outliers.

Once outliers beyond the threshold were established, they were removed from a copy of the dataset and stored separately from the dataset including outliers.

3.2. Retrieval of Class Counts

To further evaluate the balance of the dataset, we check the number of rows that consist of a certain class. With the outliers included we found the amounts of each class were imbalanced, as shown in Table 1.

Class 0	Class 1	Class 2
53174	17828	28998

Table 1: Class Counts including Outliers.

With the outliers excluded, we found the amounts of each class remained imbalanced, as shown in Table 2.

Class 0	Class 1	Class 2
50999	17003	27822

Table 2: Class Counts excluding Outliers.

We observed that there are significantly many rows with the class 0 label, indicating a ”Standard” credit score. Therefore, we incorporated experiments using SMOTE to observe the model performance after synthesizing examples of the underrepresented classes.

3.3. Constants

It is relevant to note the parameters for the SMOTE applied in this work are the default parameters provided by Python’s imbalanced learn package. These parameters include k neighbors = 5 and the sampling strategy = ’auto.’

The datasets with and without outliers were split into training and testing sets, where the testing sets consisted of 33% of the respective data set. These datasets did not use SMOTE.

For each model, a SMOTE instance was declared to resample the dataset including the outliers and the dataset excluding the outliers. The dataset with and without outliers were split into training and testing sets. The testing sets consisted of 33% of the datasets with and without outliers.

Each classifier’s parameters were searched to determine which parameters provided the highest accuracy for the unfiltered dataset with outliers and the filtered dataset without outliers. The parameter search was conducted by GridSearchCV() from Python’s scikit-learn library with 3-fold cross validation, selecting parameters based on accuracy.

After determining the best parameters, the best combination of parameters for classifiers on the datasets with and without outliers were fitted and tested.

3.4. PCA Attempt

Performing dimensional reduction was attempted with principal component analysis (PCA), however, the dataset consisted of 14 columns excluding the target column. PCA reduced accuracy by 5% and was therefore not implemented throughout the project.

3.5. Random Forest Classifier

The first model fine-tuned was a random forest classifier using GridSearchCV() from Python’s scikit-learn library. Random forest is a bagging classifier, meaning multiple decision trees are built in parallel as the algorithm progresses. The parameters searched were:

- n estimators: 100, 200
- max _depth: 10, 30, 50
- min samples leaf: 1, 3, 5

3.6. Gaussian Naive Bayes Classifier

The second model fine-tuned was a Gaussian Naive Bayes Classifier using GridSearchCV() from Python’s scikit-learn library. The parameters searched were:

- var smoothing: 1e-9, 1e-7, 1e-5

3.7. K-Nearest Neighbor Classifier

The third model fine-tuned was a K-Nearest Neighbor Classifier using GridSearchCV() from Python’s scikit-learn library. The parameters searched were:

- n neighbors: 5, 15, 25, 50
- weights: uniform, distance
- metric: euclidean, Manhattan

3.8. XGBoost Classifier

The fourth model fine-tuned was a XGBoost Classifier using GridSearchCV() from Python’s scikit-learn library. XGBoost is a boosting classifier, meaning a decision tree is built sequentially as the algorithm progresses. The parameters searched were:

- n estimators: 100, 200
- max depth: 10, 30, 50
- min child weight: 1, 3, 5

3.9. Evaluate Performance

The confusion matrices for the models evaluated on the dataset with outliers and without SMOTE, dataset with outliers with SMOTE, without outliers and without SMOTE, without outlier and with SMOTE are produced. Additionally, the accuracy, precision, recall, f1 score, and support are recorded for the classifications.

4. EXPERIMENTS AND RESULTS

4.1. Random Forest Classifier

For the dataset with SMOTE and outliers, the best parameters were max depth=50, n estimators=200, and min samples leaf=3.

For the dataset with SMOTE and without outliers, the best parameters were max depth=30, n estimators=100, and min samples leaf=1.

The performance metrics for the random forest classifier with outliers and without SMOTE are shown in table 3. The confusion matrix generated is shown in figure 3.

Table 3: RF Report with outliers, without SMOTE

Class	Precision	Recall	F1-Score
0	0.85	0.76	0.80
1	0.71	0.82	0.77
2	0.77	0.85	0.81
Accuracy			0.80
Macro Avg	0.78	0.81	0.79
Weighted Avg	0.80	0.80	0.80

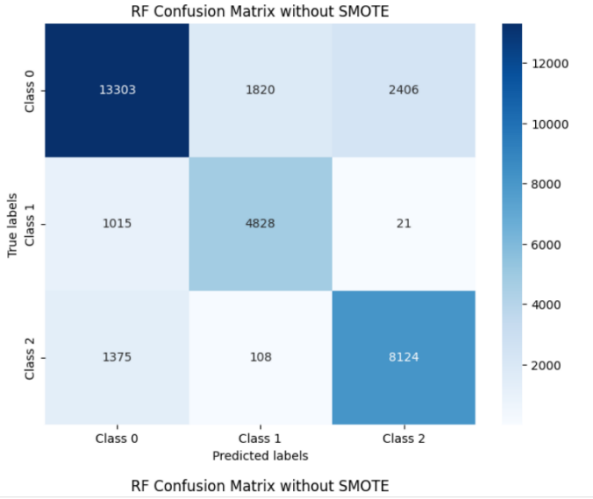

Fig. 3 RF Confusion Matrix with outliers, without SMOTE.

The performance metrics for the random forest classifier without outliers and without SMOTE are shown in table 4 and the confusion matrix generated is shown in figure 4.

Table 4: RF Report without outliers, without SMOTE

Class	Precision	Recall	F1-Score
0	0.85	0.75	0.80
1	0.71	0.81	0.76
2	0.76	0.85	0.80
Accuracy			0.79
Macro Avg	0.77	0.81	0.79
Weighted Avg	0.80	0.79	0.79

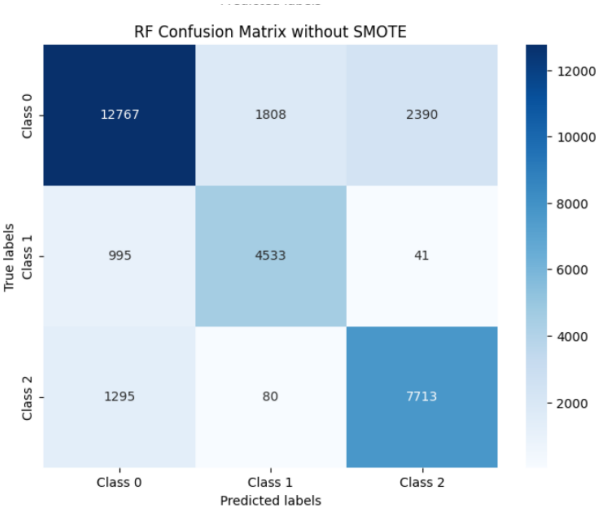

Fig. 4 RF Confusion Matrix without outliers, without SMOTE.

The performance metrics for the random forest classifier with outliers and with SMOTE are shown in table 5 and the confusion matrix generated is shown in figure 5.

Table 5: RF Report with outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.84	0.76	0.80
1	0.87	0.93	0.90
2	0.86	0.88	0.87
Accuracy			0.86
Macro Avg	0.86	0.86	0.86
Weighted Avg	0.86	0.86	0.85

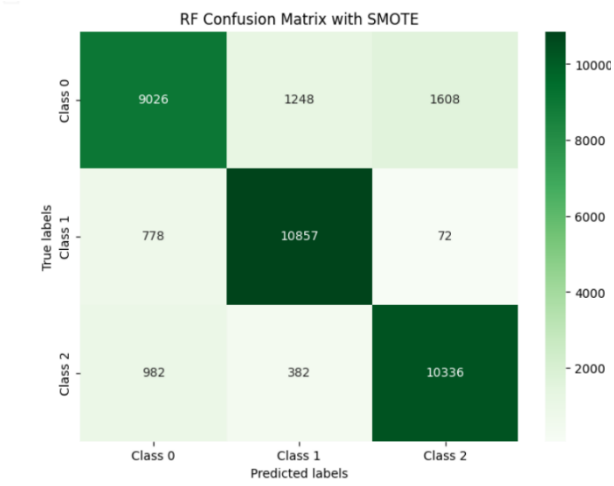

Fig. 5 RF Confusion Matrix with outliers, with SMOTE

The performance metrics for the random forest classifier without outliers and with SMOTE are shown in table 6 and the confusion matrix generated is shown in figure 6.

Table 6: RF Report without outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.85	0.75	0.80
1	0.87	0.93	0.90
2	0.85	0.90	0.87
Accuracy			0.86
Macro Avg	0.86	0.86	0.86
Weighted Avg	0.86	0.86	0.86

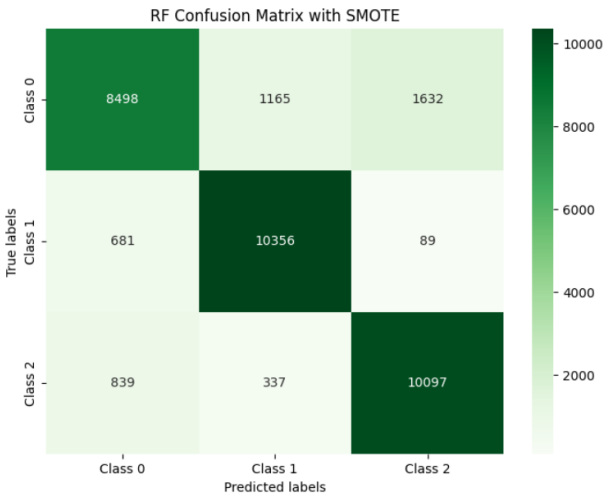

Fig. 6: RF Confusion Matrix without outliers, with SMOTE.

4.2. Gaussian Naive Bayes Classifier

For the dataset with SMOTE and outliers, the best parameter was var smoothing=1e-9.

For the dataset with SMOTE and without outliers, the best parameter was var smoothing=1e-9.

The performance metrics for the Gaussian Naive Bayes Classifier with outliers and without SMOTE are shown in table 7. The confusion matrix generated is shown in figure 7.

Table 7: GNB Report with outliers, without SMOTE

Class	Precision	Recall	F1-Score
0	0.79	0.42	0.55
1	0.38	0.84	0.52
2	0.64	0.71	0.67
Accuracy			0.58
Macro Avg	0.60	0.66	0.58
Weighted Avg	0.67	0.58	0.58

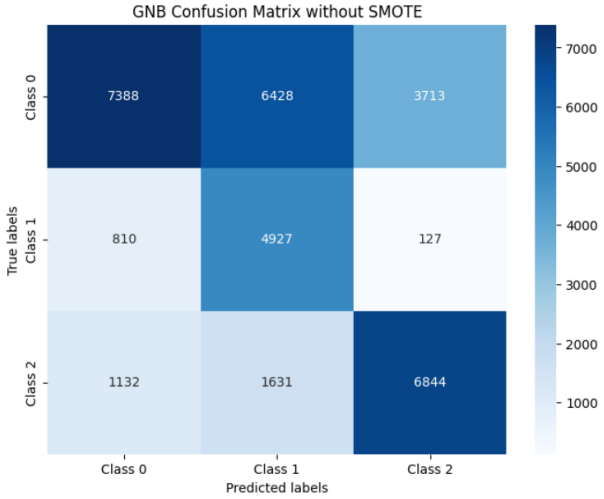

Fig. 7 GNB Confusion Matrix with outliers, without SMOTE.

The performance metrics for the Gaussian Naive Bayes Classifier without outliers and without SMOTE are shown in table 8. The confusion matrix generated is shown in figure 8.

Table 8: GNB Report without outliers, without SMOTE

Class	Precision	Recall	F1-Score
0	0.79	0.37	0.51
1	0.35	0.85	0.50
2	0.63	0.70	0.67
Accuracy			0.55

Macro Avg	0.59	0.64	0.56
Weighted Avg	0.67	0.55	0.55

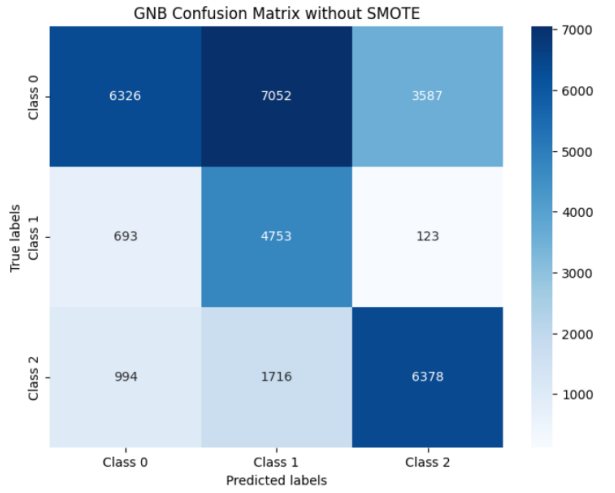

Fig. 8 GNB Confusion Matrix without outliers, without SMOTE.

The performance metrics for the Gaussian Naive Bayes Classifier with outliers and with SMOTE are shown in table 9. The confusion matrix generated is shown in figure 9.

Table 9: GNB Report with outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.65	0.42	0.51
1	0.61	0.86	0.71
2	0.75	0.71	0.73
Accuracy			0.66
Macro Avg	0.67	0.66	0.65
Weighted Avg	0.67	0.66	0.65

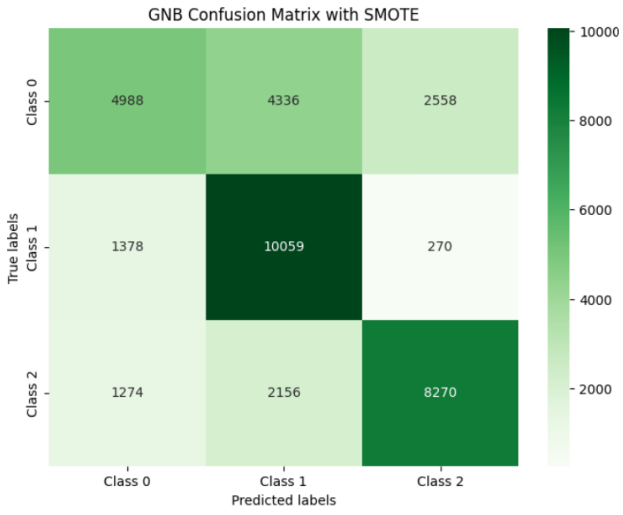

Fig. 9 GNB Confusion Matrix with outliers, with SMOTE

The performance metrics for the Gaussian Naive Bayes Classifier without outliers and with SMOTE are shown in table 10. The confusion matrix generated is shown in figure 10.

Table 10: GNB Report without outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.66	0.37	0.47
1	0.59	0.88	0.70
2	0.75	0.70	0.72
Accuracy			0.65
Macro Avg	0.66	0.65	0.63
Weighted Avg	0.67	0.65	0.63

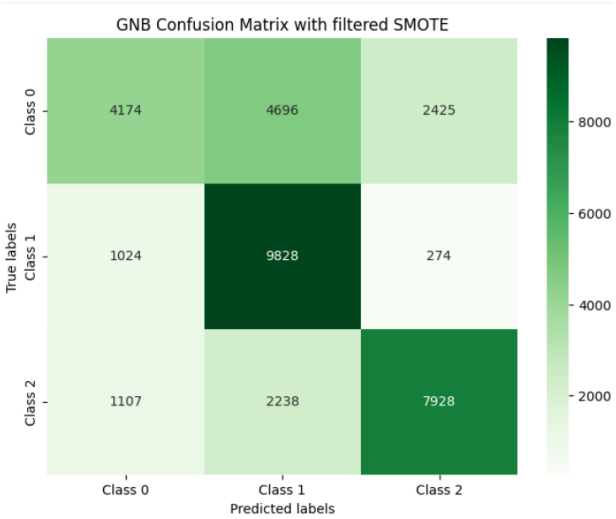

Fig. 10 GNB Confusion Matrix without outliers, with SMOTE

4.3. K-Nearest Neighbor Classifier

For the dataset with SMOTE and outliers, the best parameters were n_neighbors=5, metric=manhattan, and weights=distance.

For the dataset with SMOTE and without outliers, the best parameters were n neighbors=5, metric=euclidean, and weights=distance.

The performance metrics for the K-Nearest Neighbor Classifier with outliers and without SMOTE are shown in table 11. The confusion matrix generated is shown in figure 11.

Table 11: KNN Report with outliers, without SMOTE

Class	Precision	Recall	F1-Score
0	0.79	0.77	0.78
1	0.71	0.75	0.73
2	0.76	0.77	0.76
Accuracy			0.76
Macro Avg	0.75	0.76	0.76
Weighted Avg	0.77	0.76	0.77

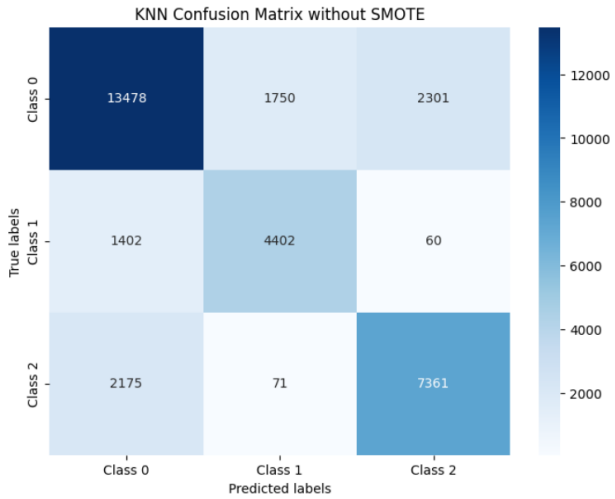

Fig. 11 KNN Confusion Matrix with outliers, without SMOTE.

The performance metrics for the K-Nearest Neighbor Classifier without outliers and without SMOTE are shown in table 12. The confusion matrix generated is shown in figure 12.

Table 12: KNN Report without outliers, without SMOTE.

Class	Precision	Recall	F1-Score
0	0.79	0.78	0.78
1	0.72	0.75	0.73
2	0.76	0.77	0.77
Accuracy			0.77
Macro Avg	0.76	0.77	0.76
Weighted Avg	0.77	0.77	0.77

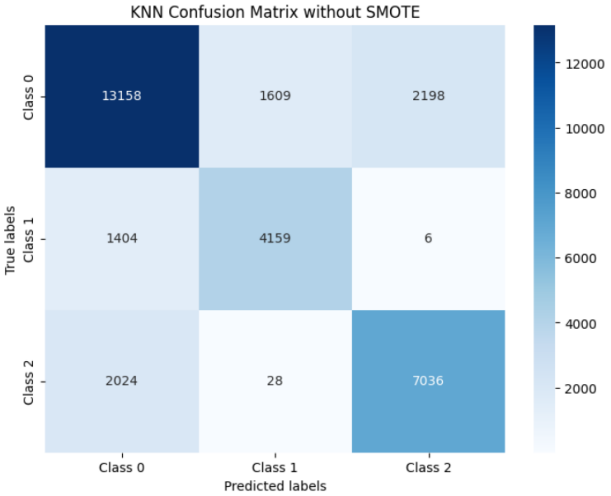

Fig. 12 KNN Confusion Matrix without outliers, without SMOTE.

The performance metrics for the K-Nearest Neighbor Classifier with outliers and with SMOTE are shown in table 13. The confusion matrix generated is shown in figure 13.

Table 13: KNN Report with outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.77	0.77	0.77
1	0.89	0.89	0.89
2	0.86	0.87	0.86
Accuracy			0.84
Macro Avg	0.84	0.84	0.84
Weighted Avg	0.84	0.84	0.84

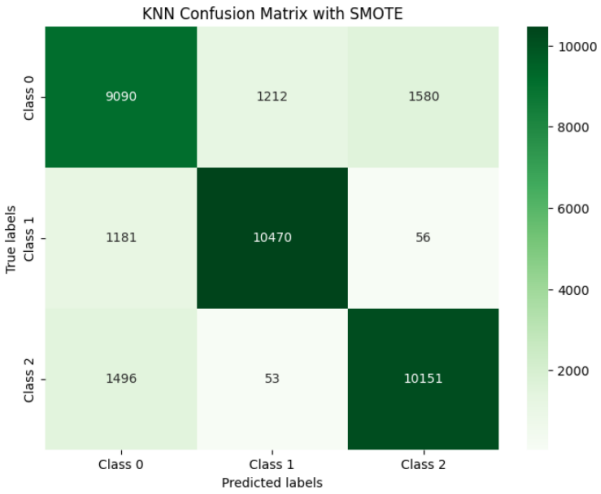

Fig. 13 KNN Confusion Matrix with outliers, with SMOTE.

The performance metrics for the K-Nearest Neighbor Classifier without outliers and with SMOTE are shown in table 14. The confusion matrix generated is shown in figure 14.

Table 14: KNN Report without outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.78	0.77	0.78
1	0.90	0.90	0.90
2	0.87	0.87	0.87
Accuracy			0.85
Macro Avg	0.85	0.85	0.85
Weighted Avg	0.85	0.85	0.85

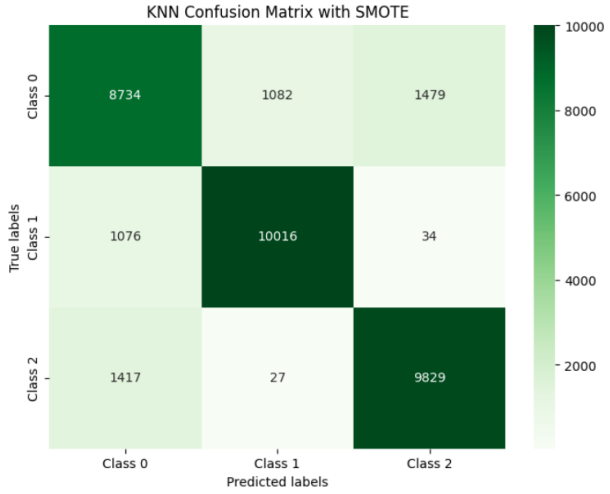

Fig. 14 KNN Confusion Matrix without outliers, with SMOTE

4.4. XGBoost Classifier

For the dataset with SMOTE and outliers, the best parameters were n estimators=200, max _depth=10, min child weight=5.

For the dataset with SMOTE and without outliers, the best parameters were n estimators=200, max depth=10, min child weight=3.

The performance metrics for the XGBoost Classifier with outliers and without SMOTE are shown in table 15. The confusion matrix generated is shown in figure 15.

Table 15: XGB Report with outliers, without SMOTE

Class	Precision	Recall	F1-Score
0	0.84	0.76	0.80
1	0.72	0.81	0.76
2	0.76	0.84	0.80
Accuracy			0.79
Macro Avg	0.78	0.80	0.79
Weighted Avg	0.80	0.79	0.79

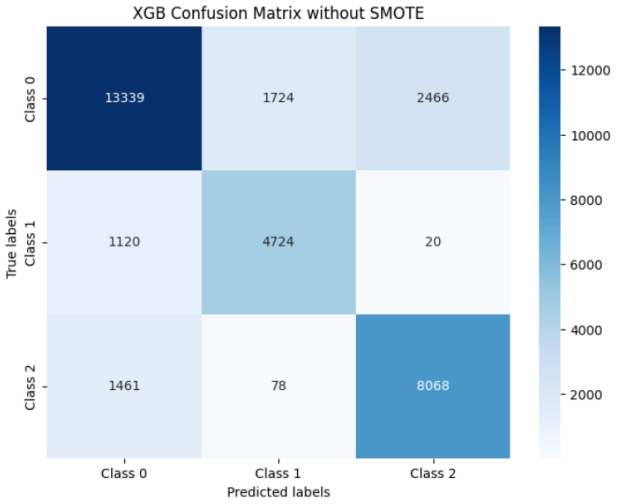

Fig. 15 XGB Confusion Matrix with outliers, without SMOTE.

The performance metrics for the XGBoost Classifier without outliers and without SMOTE are shown in table 16. The confusion matrix generated is shown in figure 16.

Table 16: XGB Report without outliers, without SMOTE

Class	Precision	Recall	F1-Score
0	0.83	0.76	0.80
1	0.71	0.80	0.75
2	0.76	0.83	0.80
Accuracy			0.79
Macro Avg	0.77	0.80	0.78
Weighted Avg	0.79	0.79	0.79

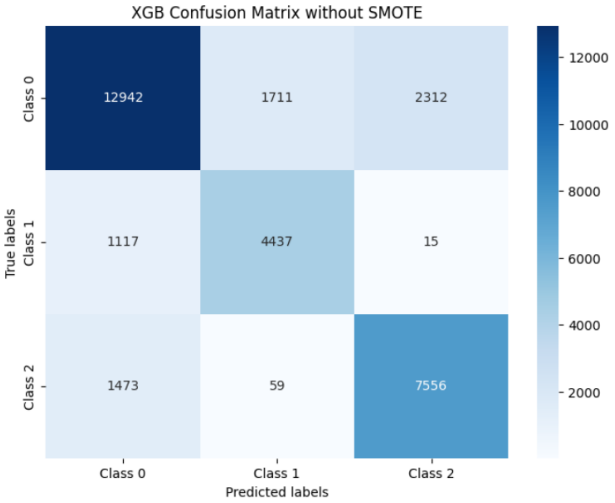

Fig. 16 XGB Confusion Matrix without outliers, without SMOTE.

The performance metrics for the XGBoost Classifier with outliers and with SMOTE are shown in table 17. The confusion matrix generated is shown in figure 17.

Table 17: XGB Report with outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.84	0.76	0.80
1	0.88	0.93	0.90
2	0.86	0.89	0.88
Accuracy			0.86
Macro Avg	0.86	0.86	0.86
Weighted Avg	0.86	0.86	0.86

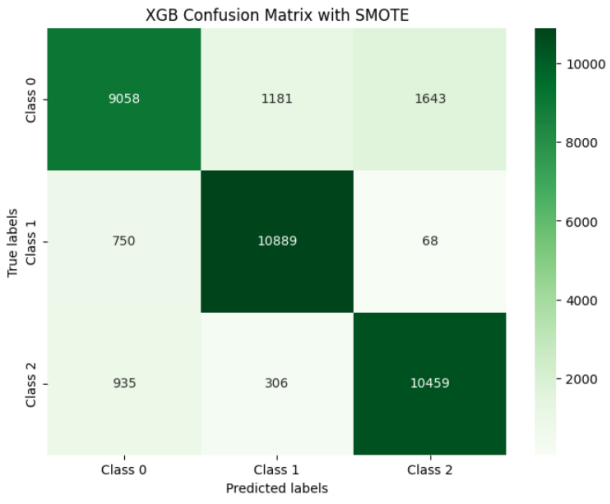

Fig. 17 XGB Confusion Matrix with outliers, with SMOTE

The performance metrics for the XGBoost Classifier without outliers and with SMOTE are shown in table 18. The confusion matrix generated is shown in figure 18.

Table 18: XGB Report without outliers, with SMOTE

Class	Precision	Recall	F1-Score
0	0.85	0.76	0.80
1	0.88	0.94	0.91
2	0.86	0.90	0.88
Accuracy			0.86
Macro Avg	0.86	0.86	0.86
Weighted Avg	0.86	0.86	0.86

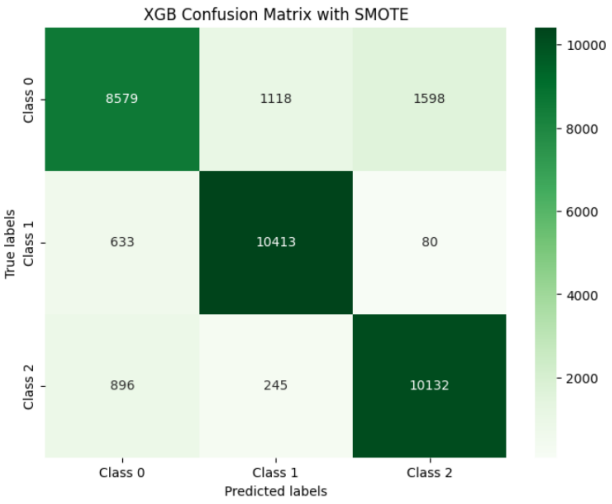

Fig. 18 XGB Confusion Matrix without outliers, with SMOTE

5. RELATED WORKS

In a study titled "Experimental analysis of machine learning methods for credit score classification" [5], the authors used various classification algorithms. They applied feature selection and classification. Random forests and time delay neural networks yielded the highest accuracy. However, this study used credit scoring datasets from 7 different countries. Specific credit score datasets from countries yielded higher accuracy with different methods. A limitation of this study is that there may be different standards for credit scoring across the globe, which affects the accuracy and methods when investigating credit score classification for 7 countries. There is 1 dataset from the 7 that includes jobs of the credit score owner, which is a Portuguese dataset. They found that two models performed the best for the Portuguese dataset. The Portuguese dataset was connected to marketing campaigns of a Portuguese bank. A lot of features related to how often the credit score owner engaged with the marketing campaigns. The findings on the credit score owner from this Portuguese dataset may not transfer well to our chosen dataset because they might have differed credit score standards

than the standards in our chosen dataset. Additionally, our dataset has no apparent marketing affiliations as some of the datasets in this study.

Another paper titled "A Comparative Performance Assessment of Ensemble Learning for Credit Scoring" [3], found random forests performed very well in terms of accuracy when compared to multiple other algorithms. The researchers used bagging, LightBGM, Adaboost, and stacking among other techniques. However, they discussed the importance of models that have parameters that can be finetuned. Random forests were fine-tuned, but no other methods were fine-tuned in their study. This study focuses on finding the best models for average companies to be able to run credit score classification models, instead of finding the best models for credit score classification in general. This paper has a business-oriented approach, without considering the impact of classification on the credit score owner. The dataset from this study does not include the same information as in our chosen dataset.

A study titled "Credit card score prediction using machine learning models: A new dataset" aimed to reduce the occurrence of false positives to protect banks [1]. This study used a dataset for an American bank. They chose the recall metric because their purpose was to protect the bank from false positives. They used methods like logistic regression, XGBoost, LightGBM, decision trees, random forest, and MLP neural networks. They found that MLP yielded the highest accuracy. Additionally, the dataset measures accuracy but does not use this measure to determine the best model for their dataset. They rely on the recall measure alone due to their purpose of reducing false positive classifications.

6. CONCLUSION: COMPARISON OF PERFORMANCES

A comparison of experiments regarding outliers' removal and SMOTE is significant to understand the costs and benefits of applying these techniques. Applying SMOTE improved the classification models performances in all experiments. Applying SMOTE and removing outliers improved model performance, however, financial institutions can consider the costs and benefits to balancing classes and removing outliers in terms of time, resources, and classification accuracy for their respective institutions. Institutions may adjust the training data to hyperparameter tune models and investigate the ideal models for streamlining credit score classification.

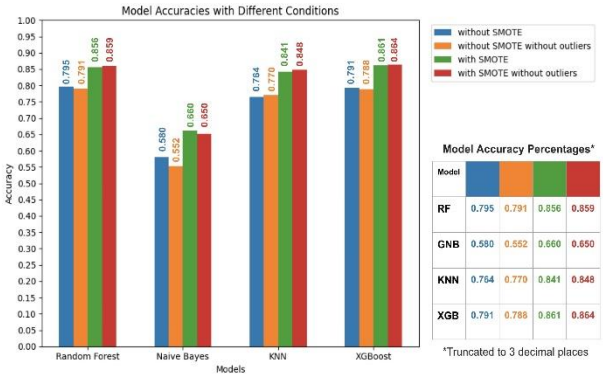

Fig. 1: Model Accuracies per Experiment

A comparison of model accuracy and the different experiments with each model is shown in figure 1. A comparison of the weighted averages for precision, recall, and f1-score with each model's different experiments are shown in figure 2. We compare the weighted averages due to some experiments not containing balanced datasets, although the unweighted metrics differ for every experiment as seen in tables 3-18.

Considering the imbalanced dataset and the model performance results, the best hyperparameter tuned model for credit score classification was XGBoost with outliers removed and SMOTE applied. This experiment had comparable performance compared to random forest. XGBoost and random forest had the best metrics overall. In figure 1, the accuracy of XGBoost with SMOTE applied is higher than random forest's accuracy with SMOTE applied by 0.5%. When SMOTE is not applied to both, XGBoost performs worse by 0.3% and 0.4%. In figure 2, the weighted average of precision, recall, and f1-score are the same for XGBoost and random forest where SMOTE was applied, except random forest has a lower f1-score than XGBoost by .01 for the experiment with SMOTE and with outliers. When outliers were not removed and SMOTE was not applied, XGBoost performed worse in recall and f1-score by .01 when compared to random forest. When outliers were removed and SMOTE was not applied, XGBoost performed worse in precision by .01 when compared to random forest.

XGBoost with SMOTE and without outliers yielded the highest accuracy from all the experiments and has the best metrics for precision, recall, and f1score when compared to random forest. We deem the experiment of XGBoost with SMOTE and without outliers as the best for streamlining credit score classification as per a financial institution's credit classification standards. We advise financial institutions to balance their dataset, remove outliers, and apply XGBoost for credit score classification. Considering the metrics, we suggest random forest with outliers removed and SMOTE applied as the next best method for credit score classification.

Limitations of this project include the feature selection methods restricted to the Mutual Information score and selecting the 14 best columns. Other feature selection methods include Chi-squared to compare results or choosing different numbers of columns to select. The findings are specific to the chosen dataset and will change depending on the training and testing data provided to the models.

Model Weighted Averages for Precision, Recall, and F1-Score, per Experiment

Model	Weighted Averages	without SMOTE	without SMOTE without outliers	with SMOTE	with SMOTE without outliers
RF	Precision	0.80	0.80	0.86	0.86
	Recall	0.80	0.79	0.86	0.86
	F1-Score	0.80	0.79	0.85	0.86
GNB	Precision	0.67	0.67	0.67	0.67
	Recall	0.58	0.55	0.66	0.65
	F1-Score	0.58	0.55	0.65	0.63
KNN	Precision	0.77	0.77	0.84	0.85
	Recall	0.76	0.77	0.84	0.85
	F1-Score	0.77	0.77	0.84	0.85
XGB	Precision	0.80	0.79	0.86	0.86
	Recall	0.79	0.79	0.86	0.86
	F1-Score	0.79	0.79	0.86	0.86

Fig 2: Model Weighted Averages of Precision, Recall, and F1-Score, for Each Experiment

7. DIVISION OF WORK

For the code, Vishal and Kaartikeya performed the data pre-processing, normalization, and dataset preparation. Rinki performed feature engineering, feature selection, debugging, fine-tuning the models for the dataset including outliers, displayed the respective model results, and created tables. Sharanya performed outlier removal, fine-tuning the models for the dataset excluding outliers, displayed the respective model results, and generated diagrams comparing the model accuracies.

For the paper, Rinki drafted the paper. Vishal, Kaartikeya, and Sharanya, reviewed their sections, proofread, and facilitated submission.

For the presentation, all members completed their slides and made edits.

Acknowledgements – We want to acknowledge Professor Keren Zhou for introducing us to the fundamental concepts allowing us to conduct this project.

REFERENCES

[1] Anas Arram, Masri Ayob, Musatafa Abbas Abbood Albadr, Alaa Sulaiman, and Dheeb Albashish. Credit card score prediction using machine learning models: A new dataset. *ArXiv*, abs/2310.02956, 2023.

[2] Aman Kharwal. Credit score classification: Case study, Dec 2022.

[3] Yiheng Li and Weidong Chen. A comparative performance assessment of ensemble learning for credit scoring, Oct 2020.

[4] Rohan Paris. Credit score classification, Jun 2022.

[5] Diwakar Tripathi, Damodar Reddy Edla, Annushree Bablani, Alok Kumar Shukla, and B. Ramachandra Reddy. Experimental analysis of machine learning methods for credit score classification. *Progress in Artificial Intelligence*, 10(3):217–243, Mar 2021.