Comparing Algorithms to Determine Credit Score

Nicolas Jorquera

***Abstract*—In this project, I aimed to classify credit scores into three categories: Good, Poor, and Standard. I employed various machine learning algorithms, including Decision Tree, Random Forest, Gradient Boosting Classifier, and Neural Networks, to solve the classification problem. Through hyperparameter tuning with RandomizedSearchCV, I optimized the performance of these algorithms and compared their results using classification reports and visualizations. The experiments demonstrated that the Random Forest and Gradient Boosting Classifier generally outperformed the Decision Tree algorithm in terms of f1-score and accuracy. The major contribution of this work lies in the comprehensive comparison of different machine learning techniques for credit score classification.**

I. Introduction

In the world of finance, credit scores play a crucial role in determining an individual's creditworthiness. Accurate credit score classification is essential for financial institutions to make informed decisions regarding loans, interest rates, and credit limits. In this project, I aim to develop a machine learning model that can effectively classify credit scores into three categories: Good, Poor, and Standard. By leveraging various machine learning algorithms, I seek to identify the most suitable approach for credit score classification and provide valuable insights for financial institutions.

The data set used for this project consists of 100,000 records with 28 features, including demographic information, financial behavior, and credit history. The target variable is the credit score, which has three possible values: Good, Poor, and Standard. The objective is to build a machine learning model that can predict the credit score category based on the given features.

To achieve this goal, I employ a range of machine learning algorithms, including Decision Tree, Random Forest, Gradient Boosting Classifier, and Neural Networks. I start by implementing these algorithms with their default parameters and evaluate their performance using classification reports, which provide metrics such as precision, recall, f1-score, and accuracy. Next, I perform hyperparameter tuning using RandomizedSearchCV to optimize the performance of these algorithms further. The experimental results are then compared and analyzed to identify the best-performing model.

The experiments demonstrate that the Random Forest and Gradient Boosting Classifier generally outperform the Decision Tree algorithm in terms of f1-score and accuracy. Moreover, I implement a Neural Network algorithm as an alternative approach to classify credit scores, discussing its pros and cons and potential directions for further improvement.

The major contribution of this work lies in the comprehensive comparison of different machine learning techniques for credit score classification. By identifying the strengths and weaknesses of each algorithm, I provide valuable insights into their advantages over existing solutions. Furthermore, the project serves as a foundation for future work in the domain of credit score prediction, exploring more advanced techniques, data augmentation, feature engineering, and alternative problem formulations, such as regression models, to enhance the performance further.

II. Related Work

Over the years, numerous studies and solutions have been proposed to tackle credit scoring and credit risk assessment problems. In the early 2000's; statistical approaches were used for credit scoring, such as linear discriminant analysis (LDA). These methods are widely used due to their simplicity and ease of implementation. However, they may not always capture complex relationships and patterns in the data, which could lead to sub optimal classification performance.

In the coming years machine learning approaches overcame the limitations of statistical techniques. Various machine learning algorithms have been applied to credit scoring problems, such as Decision Trees, Support Vector Machines (SVM), Neural Networks, Random Forests, and Gradient Boosting. These algorithms are capable of capturing non-linear relationships and can handle large datasets with numerous features. Despite their advantages, machine learning approaches might be prone to overfitting and may require careful feature selection and hyperparameter tuning for optimal performance.

The work builds upon these existing solutions by providing a comprehensive comparison of different machine learning techniques for credit score classification and exploring potential improvements over existing methods.

III. Solution

A. Description of Dataset

The dataset used in this project is a clean and well-organized credit scoring dataset consisting of both numerical and categorical features. The dataset did not require much feature engineering, as numerical data was already in numerical format and categorical data was already in categorical format.

Upon initial examination, I found that there were several repeating $customer_ID$, which could potentially skew the data if a single customer appeared too frequently. To address this, I analyzed how many times a customer's credit score changed and found that at most, a customer's credit score changed six times, with more than 4,305 customers having their credit score change more than twice. I also added a new column representing the number of months a client has had an account for.

To better understand the relationships between different features and credit scores, I created a series of visualizations. I grouped the data into various categories based on age, annual income, monthly balance, outstanding debt, number of credit inquiries, number of bank accounts, number of loans, and number of delayed payments. The resulting count plots revealed some interesting insights: *All plots can be found in the code*

- The distribution of credit scores for most categories appeared to be normal.
- Poor credit scores were more common among certain age groups, those with higher outstanding debt, higher number of loans and high delayed payments. These factors intuitively make sense since young people tend to have less financial stability, and loans / delayed payments have a direct impact on credit score.
- High credit scores were common among high income individuals with higher income, and these individuals have minimal to no loans, credit inquiries and outstanding debt.

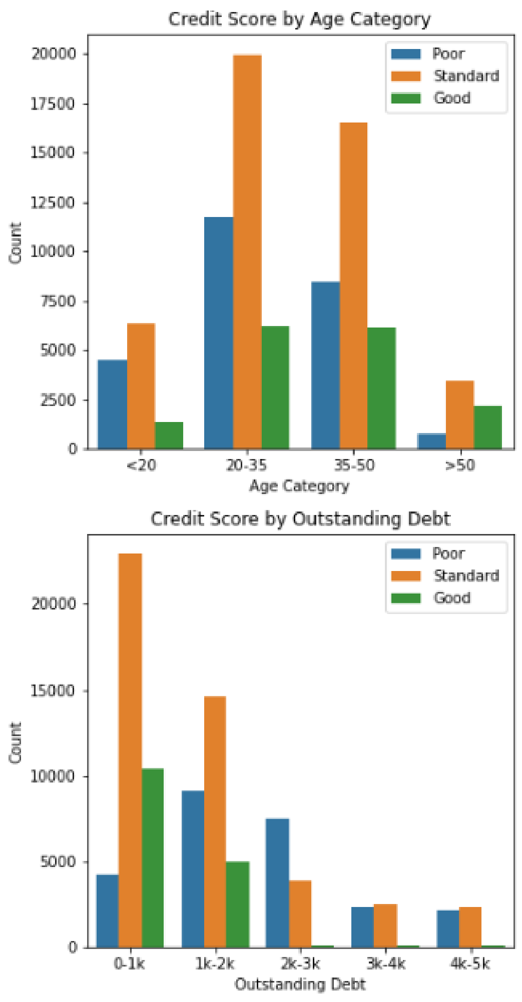


Fig. 1. Comparison of Features of Credit Score

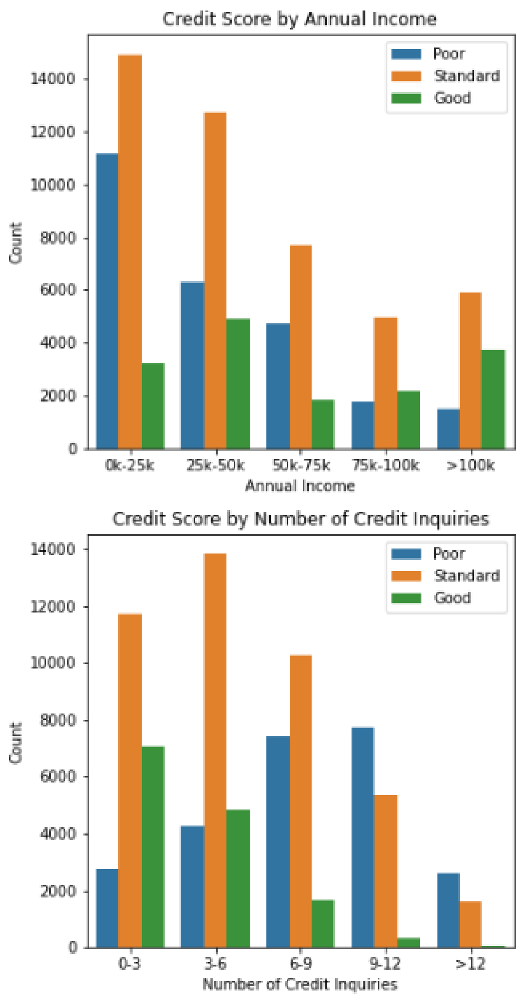


Fig. 2. Comparison of Features of Credit Score

B. Machine Learning Algorithms

A **decision tree classifier** is a tree-like model where each internal node represents a decision based on a certain feature, and each leaf node represents a class label. The decision tree algorithm splits the data into sub-groups based on the most significant features, with the goal of creating homogeneous groups that have the same class label. This process continues recursively until a stopping criterion is met. Once the decision tree is built, it can be used to classify new data by traversing the tree and making decisions based on the feature values.

A **random forest classifier** is an ensemble learning method that constructs multiple decision trees and combines them to make a final prediction. The algorithm creates multiple decision trees, each with a random subset of the features and a random subset of the training data. These trees are then combined through a voting mechanism to make the final prediction. The main idea behind the random forest algorithm is that the combined decision of multiple trees will be more accurate than the decision of any individual tree.

A **gradient boosting classifier** is an ensemble learning method that works by combining multiple weak learners (decision trees) to create a strong learner. It builds the model in a stage-wise fashion, where each subsequent model focuses on the errors made by the previous models. Gradient Boosting Classifier can perform well in cases where the data has high variance or noise, as it can reduce overfitting by combining multiple weak models.

C. Implementation Details

Data Preprocessing: Before implementing the machine learning models, I performed data preprocessing to ensure that the dataset was clean and suitable for training. I handled missing values, encoded categorical features into numerical values. Because of the nature of the problem, I determined it was unnecessary to normalize the numerical features as needed. This is because decision trees, random forests, and gradient boosting models, are based on splitting the data into partitions based on the values of the features. These models are not sensitive to the scale of the data and thus normalization is not necessary.

Train-Test Split: We split the dataset into training and testing sets to evaluate the performance of each model. A typical split ratio of 80

Model Training and Validation: I trained each machine learning algorithm on the preprocessed training dataset and evaluated their performance using the test dataset. The performance was assessed using various evaluation metrics such as accuracy, precision, recall, and F1 score. These metrics provided insights into the effectiveness of each model in classifying credit scores.

Hyperparameter Tuning: To improve the performance of the models, I performed hyperparameter tuning using RandomizedSearchCV. This technique allowed us to search for the best combination of hyperparameters by randomly sampling from a given search space. Compared to GridSearchCV, RandomizedSearchCV can be more efficient and faster in finding the optimal hyperparameters.

Model Selection: After tuning the hyperparameters, I compared the performance of each model using the evaluation metrics mentioned earlier. The model with the best performance was selected as the most suitable algorithm for classifying credit scores in this project.

IV. Comparison

Compared to Random Forest, Gradient Boosting Classifier builds trees sequentially, focusing on the errors made by previous trees, while Random Forest builds multiple trees independently and combines their results through averaging or majority voting. Gradient Boosting can sometimes outperform Random Forest when the data has high variance, but it might be more prone to overfitting if not tuned properly. Random forests have several advantages over decision trees. They are less prone to overfitting and can handle a larger number of features. Additionally, they provide a measure of feature importance, which can be useful in feature selection. However, they can be slower to train than decision trees due to the creation of multiple trees. Now I will explore how these algorithms differ, based on the data. Below I will explore the pro / cons of each algorithm, and how they differ in regards to overfitting, noise and class distribution.

Decision Trees:

- Overfitting: Decision Trees are prone to overfitting, especially when they are deep, because they can perfectly fit the training data by creating overly complex decision boundaries. Overfitting leads to poor generalization performance on unseen data.
- Noise: Decision Trees can be sensitive to noise in the data, as even small changes in the input data might lead to completely different tree structures. This sensitivity makes them less robust in handling noisy data.
- Class Distributions: Decision Trees can handle imbalanced class distributions reasonably well, as they can create decision boundaries that separate minority classes from majority classes. However, in the presence of severe class imbalance, the model might still be biased towards the majority class.

Random Forests:

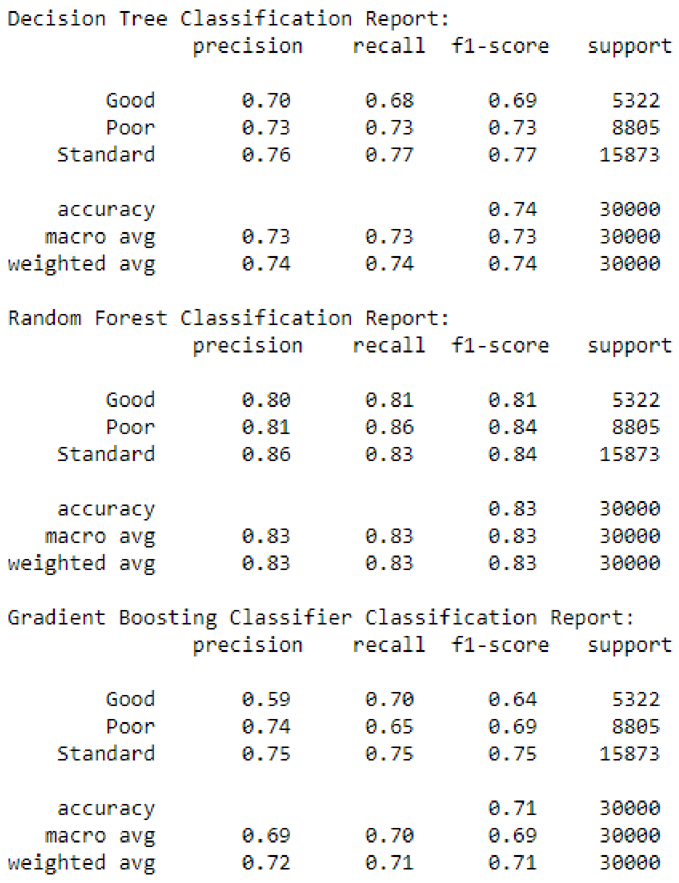
- Overfitting: Random Forests help mitigate overfitting by averaging the predictions of multiple trees. The diverse set of trees, each trained on a different subset of data and features, reduces the overall model's complexity and provides a more generalized decision boundary.
- Noise: Random Forests are more robust to noise than single decision trees, as the ensemble's averaging effect reduces the impact of noise present in individual trees. However, Random Forests can still be maffected by noise when the noise level is high.
- Class Distributions: Random Forests can handle imbalanced class distributions by adjusting the class weights or using techniques like bootstrapping, which can create a more balanced representation of the classes during the training process. However, severe class imbalance might still lead to biased predictions towards the majority class.

Gradient Boosting Classifier:

- Overfitting: Gradient Boosting Classifier can control overfitting by using shallow trees as weak learners and by limiting the number of boosting iterations. However, if the model is not well-tuned, overfitting can still occur, especially when the learning rate is high, or the number of boosting iterations is excessive.
- Noise: Gradient Boosting Classifier can handle noisy data to some extent, as it focuses on the errors made by previous models during the boosting process. However, if the noise level is high, the algorithm might end up fitting the noise, resulting in overfitting and poor generalization performance.
- Class Distributions: Gradient Boosting Classifier can handle imbalanced class distributions by adjusting the loss function or using techniques like sample weighting or cost-sensitive learning. However, it might still struggle with severe class imbalance, which can lead to biased predictions.

As mentioned above, the following will be used to evaluate performance:

- Precision: Measures the proportion of correctly predicted positive instances out of all instances predicted as positive.
- Recall: It is the ratio of true positive predictions to the sum of true positive and false negative predictions. It measures the proportion of correctly predicted positive instances out of all actual positive instances.
- F1-score: The F1-score balances both precision and recall, giving equal importance to both. The F1-score ranges from 0 (worst) to 1 (best).



```
Decision Tree Classification Report:
              precision    recall  f1-score   support

        Good       0.70      0.68      0.69      5322
        Poor       0.73      0.73      0.73      8805
    Standard       0.76      0.77      0.77     15873

    accuracy                           0.74     30000
   macro avg       0.73      0.73      0.73     30000
weighted avg       0.74      0.74      0.74     30000

Random Forest Classification Report:
              precision    recall  f1-score   support

        Good       0.80      0.81      0.81      5322
        Poor       0.81      0.86      0.84      8805
    Standard       0.86      0.83      0.84     15873

    accuracy                           0.83     30000
   macro avg       0.83      0.83      0.83     30000
weighted avg       0.83      0.83      0.83     30000

Gradient Boosting Classifier Classification Report:
              precision    recall  f1-score   support

        Good       0.59      0.70      0.64      5322
        Poor       0.74      0.65      0.69      8805
    Standard       0.75      0.75      0.75     15873

    accuracy                           0.71     30000
   macro avg       0.69      0.70      0.69     30000
weighted avg       0.72      0.71      0.71     30000
```

Fig. 3. Classification Report Comparison

Random Forests perform the best in this problem because they effectively mitigate overfitting and handle noise better by averaging the predictions of multiple decision trees, leading to a more accurate and stable model. Gradient Boosting Classifier can also perform well when tuned correctly, but it might be more sensitive to noise and overfitting depending on the learning rate and number of boosting iterations. This is proved as both Gradient Boosting Classifier and Random Forest perform significantly better than decision trees. Decision Trees, while simple and easy to understand, tend to suffer from overfitting and are less robust to noise compared to the ensemble methods. Now I will explore how to improve these models using hyperparameter tuning:

Decision Trees:

- max_depth: The maximum depth of the tree. Deeper trees can lead to overfitting, while shallow trees might result in underfitting.
- $min_samples_split$: The minimum number of samples required to split an internal node. Larger values result in a more regularized model.
- $min_samples_leaf$: The minimum number of samples required to be at a leaf node. Larger values can help prevent overfitting.

Random Forests:

- $n_estimators$: The number of trees in the forest. Increasing this value generally improves the model's performance but also increases the computational cost.
- $n_estimators$: The number of trees in the forest. Increasing this value generally improves the model's performance but also increases the computational cost.
- $min_samples_split$ and $min_samples_leaf$: These parameters have the same meaning as in Decision Trees and help in regularizing the model.

Gradient Boosting Classifier:

- $n_estimators$: The number of boosting stages to perform. A larger number of estimators can improve performance but may lead to overfitting.
- $learning_rate$: The rate at which the model learns from the errors of previous iterations. A smaller learning rate requires more iterations but can result in a more accurate model.
- max_depth: The maximum depth of the individual trees. Similar to Decision Trees, controlling the depth helps in balancing bias and variance.
- $subsample$: The fraction of samples to be used for fitting the individual base learners. Using a smaller subsample value introduces randomness and can help in reducing overfitting.

Both GridSearchCV and RandomizedSearchCV are techniques used in hyperparameter tuning of machine learning models. The key difference between them is the way in which they search the hyperparameter space.

- **GridSearchCV** performs an exhaustive search over a predefined set of hyperparameters. It generates all possible combinations of hyperparameters and evaluates the model for each combination. This approach can be effective for small hyperparameter spaces, but can quickly become computationally expensive for large hyperparameter spaces.
- **RandomizedSearchCV**, on the other hand, performs a randomized search over the hyperparameter space. It generates a fixed number of random combinations of hyperparameters and evaluates the model for each combination. This approach can be more efficient for large hyperparameter spaces and can often find good hyperparameter combinations faster than GridSearchCV.

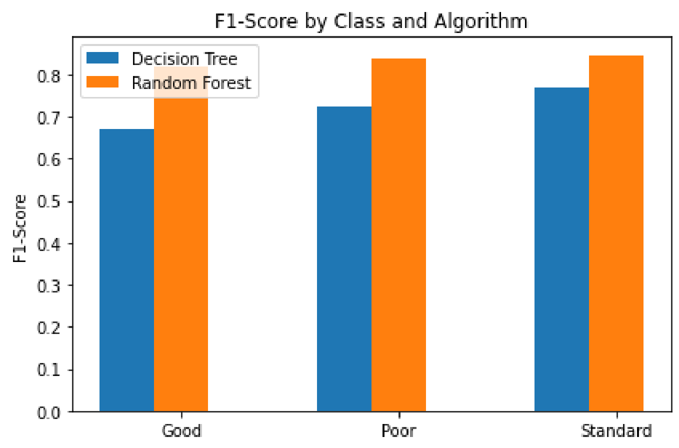


Fig. 4. Comparison of F1 scores

As expected, Random Forest improved slightly, however not as significant as Decision Trees. This is most likely because the Random Forest default parameters were most likely close to the parameters found using RandomSearchCV; which signifies the model was almost optimized. To find a more optimal solution I can use GridSearchCV but I would not expect to see significant improvements in performance. Alternatively, Decison Tree was not optimized therefore the accuracy improved by almost 20 percent.

V. Future Directions

Neural Networks

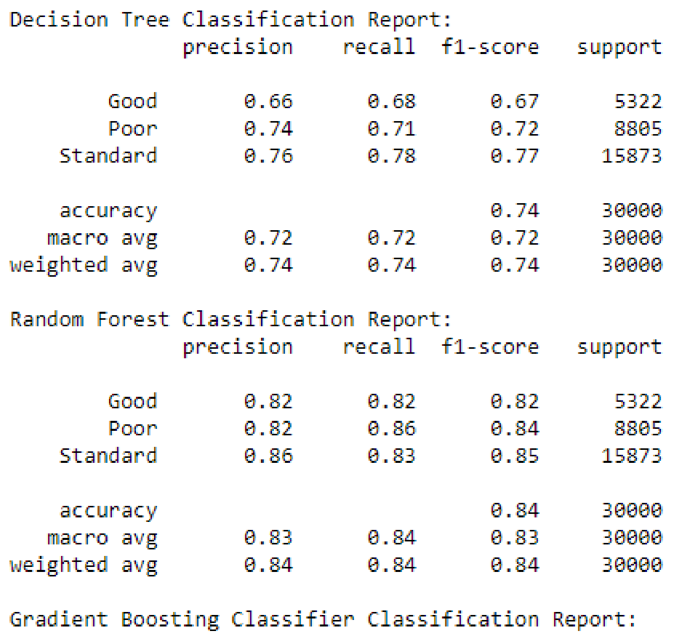
The pros of Neural Networks are that Neural networks can capture complex patterns in the data and are especially good at handling non-linear relationships, they can automatically learn and model feature interactions without the need for explicit feature engineering, and they can be highly accurate if properly tuned and trained on large datasets.

The cons of Neural Networks are that Neural networks can be computationally expensive and may take a long time to train, especially on large datasets, They are more sensitive to the quality and quantity of data, often requiring large amounts of labeled data to perform well, They can be prone to overfitting, especially when the model architecture is too complex, and they are often considered "black boxes" due to the difficulty of interpreting and understanding the internal workings of the model. In this project I did not explore Neural Networks because not sufficient data was given to fully take advantage. However in the future I would like to source more data from more individuals so that the accuracy of the model can improve.

Professional Expertise

Leveraging domain knowledge to prioritize the most informative samples for annotation and model training. This would mean using a financial advisor to determine which features are more important; or what feature are missing and should be added to algorithm. Particularly, this would improve the feature engineering process which in turn should improve the model.

Regression model


```
Decision Tree Classification Report:
              precision    recall  f1-score   support

        Good       0.66      0.68      0.67      5322
        Poor       0.74      0.71      0.72      8805
    Standard       0.76      0.78      0.77     15873

    accuracy                           0.74     30000
   macro avg       0.72      0.72      0.72     30000
weighted avg       0.74      0.74      0.74     30000

Random Forest Classification Report:
              precision    recall  f1-score   support

        Good       0.82      0.82      0.82      5322
        Poor       0.82      0.86      0.84      8805
    Standard       0.86      0.83      0.85     15873

    accuracy                           0.84     30000
   macro avg       0.83      0.84      0.83     30000
weighted avg       0.84      0.84      0.84     30000

Gradient Boosting Classifier Classification Report:
```

Fig. 5. Classification Report Comparison - Utilizing RandomSearchCV

Unfortunately, the data did not have the credit score in a numerical form. Instead, it's a classification system that sorts individuals into [Poor, Standard, Good]. Using a regression model to predict the numerical credit score, and then classifying the credit score can be an interesting approach. By predicting the exact credit score value first, you might capture more information about the underlying distribution and relationships in the data. After obtaining the predicted credit score, you can set thresholds to classify it into the categories [Poor, Standard, Good]. This approach could potentially improve the performance of the model, especially if the relationships between the features and the credit score are more linear in nature.

VI. Conclusion

In this project, I aimed to classify credit scores into three categories: Good, Poor, and Standard. I started by implementing three classification algorithms: Decision Tree, Random Forest, and Gradient Boosting Classifier. I evaluated their performance using classification reports, which provided precision, recall, f1-score, and accuracy metrics for each algorithm. Although Random Forest performed fairly well; I can improve it. Next, I used RandomizedSearchCV to perform hyperparameter tuning on these three algorithms, aiming to improve their performance. I compared their performance after tuning and observed improvements in the metrics for each algorithm compared to their default parameters.

In the future, I can use more advanced algorithms, however to fully utilize their benefits I would need more data to improve the performance of these models. I implemented a Neural Network algorithm as an alternative approach to classify credit scores. I discussed its pros and cons and potential directions to improve the performance of this project further.

Overall, the Random Forest and Gradient Boosting Classifier generally showed better performance than the Decision

Tree algorithm. The choice of the most suitable technique depends on the specific requirements and constraints of the project, such as computational resources, prediction speed, and interpretability. However, as mentioned their is a lot more to improve, such as feature engineering, amount of data utilized, and complexity of algorithms. To find the best performing model I should also utilize GridSearchCV if their was enough computational power and time.

References

- Dataset - https://www.kaggle.com/code/clkmuhammed/credit-score-classification-part-1-data-cleaning/notebook?scriptVersionId=108316213
- Credit Score utilizing Statistics - https://pages.ucsd.edu/ aronatas/project/academic/A20survey - A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers