Transparent Credit Scoring Using Explainable Machine Learning

CHONG Tin Tak[a]

[a]*Department of Industrial Engineering and Decision Analytics, The Hong Kong University of Science and Technology*

Prof. Jiashuo JIANG

Abstract—Credit scoring is fundamental in the financial technology (FinTech) sector for assessing loan applicant risk. However, traditional models often lack transparency, hindering trust and regulatory compliance. This project addresses this gap by developing a transparent credit scoring model using explainable machine learning (XAI) techniques. Leveraging a public dataset from Kaggle containing features like age, income, outstanding debt, and payment behaviour, we aim to predict creditworthiness, categorised into 'Good', 'Standard', and 'Poor'. We evaluate three distinct machine learning models: Logistic Regression, Random Forest, and XGBoost. Preprocessing steps, including handling missing values, feature engineering (e.g., debt-to-income ratio), encoding categorical variables, and scaling, were applied. Model performance was assessed using accuracy, precision, recall, F1-score, and ROC-AUC. The Random Forest model emerged as the optimal choice, achieving a validation accuracy of 80.13% and the highest AUC of 0.9228, demonstrating a strong balance between predictive power and interpretability. SHAP (SHapley Additive exPlanations) values were employed to explain the Random Forest model's predictions, identifying key factors influencing credit scores, thus enhancing transparency. Challenges related to SHAP runtime, potential overfitting, and feature availability were addressed through techniques like data sampling and robust validation. This work underscores the potential of XAI in building fair, accurate, and understandable credit scoring systems in FinTech.

Keywords—*credit scoring, explainable machine learning, XAI, SHAP, Random Forest, XGBoost, FinTech, transparency, classification*

Contents

1. Introduction

1.1. Background

Credit scoring plays a pivotal role in the modern financial landscape, particularly within the rapidly evolving FinTech industry. It involves the statistical analysis of a borrower's credit-related information to generate a score representing their likelihood of repaying debt. This score is critical for lenders in making informed decisions about loan approvals, interest rates, and credit limits.

While traditional scoring methods have existed for decades, the advent of machine learning (ML) has introduced more sophisticated and potentially more accurate models. However, many advanced ML models operate as "black boxes", making it difficult to understand the reasoning behind their predictions. This lack of transparency poses significant challenges, including potential biases, difficulties in regulatory compliance (e.g., GDPR's right to explanation), and erosion of customer trust.

1.2. Problem Statement & Objectives

The primary challenge addressed in this project is the inherent opacity of many high-performance ML models used for credit scoring. There is a critical need for models that are not only accurate but also transparent and fair.

The main objectives of this project are:

- To develop and evaluate ML models for predicting creditworthiness based on a multi-class target variable ('Good', 'Standard', 'Poor').
- To compare the performance of interpretable (Logistic Regression) and complex (Random Forest, XGBoost) models.
- To employ explainable AI (XAI) techniques, specifically SHAP (SHapley Additive exPlanations), to interpret the predictions of the best-performing model.
- To build a credit scoring system that is both accurate and transparent, facilitating fairer and more understandable lending decisions.

1.3. Contributions

This project contributes to the field by:

- Demonstrating a practical workflow for building an explainable credit scoring model using contemporary ML techniques.
- Providing a comparative analysis of Logistic Regression, Random Forest, and XGBoost in a multi-class credit scoring context.
- Applying SHAP to a tree-based ensemble model (Random Forest) to gain insights into feature contributions towards credit score predictions, thereby enhancing model transparency.
- Highlighting the importance and feasibility of integrating explainability into FinTech applications.

2. Literature Review

Credit scoring has been extensively studied, with early models often based on statistical techniques like Linear Discriminant Analysis and

Logistic Regression [4]. Logistic Regression remains popular due to its simplicity and inherent interpretability. However, its performance can be limited by its linearity assumption.

More recent research has focused on applying advanced ML algorithms, including Support Vector Machines (SVM), Neural Networks, and ensemble methods like Random Forest [1] and Gradient Boosting machines (like XGBoost) [2], which often yield higher predictive accuracy by capturing complex, non-linear relationships in the data.

The increasing use of complex models has spurred the development of XAI techniques. SHAP, introduced by Lundberg and Lee (2017) [3], provides a unified framework based on cooperative game theory (Shapley values) to explain the output of any machine learning model. It offers both global interpretability (overall feature importance) and local interpretability (explaining individual predictions). While XAI application in finance is growing, this project specifically focuses on leveraging SHAP with tree-based ensembles for transparent multi-class credit scoring.

3. Data Description & Preprocessing

3.1. Dataset

The dataset utilised for this project was sourced from Kaggle, specifically the "Credit Score Classification" dataset [5]. It contains **100,000 records** and **27 features** describing customer financial attributes and behaviour. The dataset can be accessed at: https://www.kaggle.com/datasets/parisrohan/credit-score-classification/data.

It comprises customer financial data with features potentially including:

- **Demographics:** Age
- **Financial Status:** Annual Income, Monthly Inhand Salary, Bank Accounts, Credit Cards
- **Debt Information:** Outstanding Debt, Interest Rate, Debt-to-Income ratio (potentially engineered)
- **Credit History:** Credit History Age, Payment Behaviour, Minimum Amount Payments, Delays from Due Date
- **Other:** Credit Mix, Number of Loans, Credit Inquiries

Key features include variables like age, income, outstanding debt, payment history, credit utilisation (`Credit_Mix`), and interest rates. The target variable for prediction is `Credit_Score`, a categorical feature with three classes: 'Good', 'Standard', and 'Poor'.

Figure 1 illustrates the distribution of these target classes within the dataset.

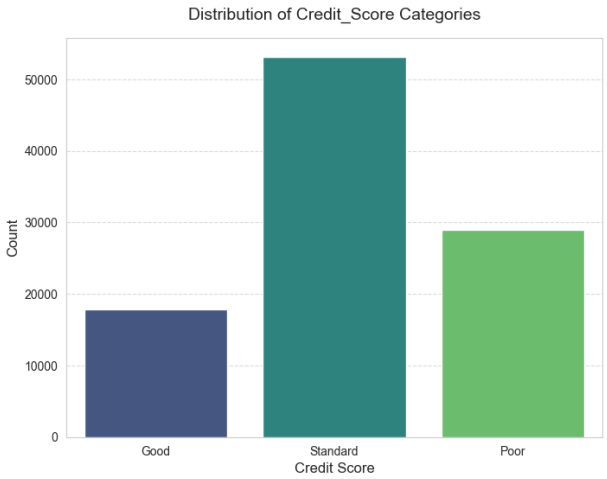

Figure 1. Distribution of the Target Variable (`Credit_Score`) Categories.

As observed in Figure 1, the dataset is imbalanced, with a significantly larger number of instances belonging to the 'Standard' credit score category compared to 'Good' and 'Poor'. This characteristic is important to consider during model training and evaluation, motivating the use of stratified splitting and metrics like macro-averaged F1-score or AUC.

3.2. Data Cleaning and Preprocessing Steps

Prior to model training, the raw data obtained from Kaggle underwent a rigorous cleaning and preprocessing pipeline to ensure data quality and suitability for machine learning algorithms. This involved several key stages:

1. **Initial Data Loading and Column Removal:** The training (`train.csv`) and testing (`test.csv`) datasets were loaded. Identifier columns deemed irrelevant for credit scoring prediction, namely `ID`, `Customer_ID`, `Name`, and `SSN`, were removed from both datasets using `df.drop()`.
2. **Numerical Feature Sanitisation:** Several columns intended to be numerical contained non-numeric characters or specific string formatting. This affected columns such as `Age`, `Annual Income`, `Monthly Inhand Salary`, `Num of Loan`, `Num of Delayed Payment`, `Changed Credit Limit`, `Num Credit Inquiries`, `Outstanding Debt`, `Total EMI per month`, `Amount Invested Monthly`, and `Monthly Balance`. These were cleaned using regular expressions to retain only digits, decimal points, or minus signs. The cleaned strings were then converted to numeric types using `pd.to_numeric(errors='coerce')`, ensuring that any values failing conversion became Not-a-Number (`NaN`).
3. **Specific Feature Handling:**
   - `Age`: Values were validated to be positive. Any non-positive or `NaN` entries were treated as missing values (`NaN`).
   - `Credit History Age`: This feature, initially a string (e.g., "X Years and Y Months"), was parsed using regular expressions (`re.search`) to calculate the total duration in months. Entries not matching the format were converted to `NaN`.
   - `Month`: The categorical `Month` feature was mapped to its numerical representation (1-12).
4. **Missing Value Imputation:** Missing values were handled as follows:
   - *Categorical Features:* For columns like `Occupation`, `Type of Loan`, `Credit Mix`, `Payment of Min Amount`, `Payment Behaviour`, and initially `Month`, missing values were imputed using the **mode** (`df[col].fillna(df[col].mode()[0])`).
   - *Numerical Features:* For the sanitised numerical columns and the converted `Credit History Age`, missing values were imputed using the **median** (`df[col].fillna(df[col].median())`), chosen for robustness to outliers.
5. **Categorical Feature Encoding:** Categorical features (`Occupation`, `Type of Loan`, `Credit Mix`, `Payment of Min Amount`, `Payment Behaviour`) were converted into numerical representations using **Label Encoding** (`LabelEncoder`). The target variable, `Credit_Score`, was also label encoded. The same encoder fitted on the training data was used for the test data.
6. **Feature Scaling:** Numerical features were scaled using **Standardisation** (`StandardScaler`) to have zero mean and unit variance. The scaler was fitted on the training data and applied to both training and validation/test sets.

3.3. Feature Engineering

To potentially capture more nuanced relationships within the data and enhance predictive power, several new features were derived from existing ones. A small epsilon ($\epsilon = 10^{-5}$) was added to denominators

to prevent potential division-by-zero errors where the denominator might otherwise be zero. The engineered features are defined as follows:

- **Debt-to-Income Ratio (*DTI*)**: This ratio relates the total outstanding debt to the annual income.

$$DTI = \frac{\text{Outstanding Debt}}{\text{Annual Income} + \epsilon}$$

- **Loan-to-Bank Account Ratio (*LBR*)**: This compares the number of loans held to the number of bank accounts.

$$LBR = \frac{\text{Num of Loans}}{\text{Num Bank Accounts} + \epsilon}$$

- **EMI-to-Salary Ratio (*ETS*)**: This calculates the proportion of monthly salary consumed by loan instalments.

$$ETS = \frac{\text{Total EMI per Month}}{\text{Monthly Inhand Salary} + \epsilon}$$

- **Credit Inquiry per Loan Ratio (*CIPL*)**: This relates the frequency of credit enquiries to the number of existing loans.

$$CIPL = \frac{\text{Num Credit Inquiries}}{\text{Num of Loans} + \epsilon}$$

These engineered features were added to both the training and test datasets before the feature scaling step.

4. Methodology

Three distinct classification models were selected for comparison, representing a spectrum from interpretable linear models to complex ensemble methods:

4.1. Model Choices

- **Logistic Regression**: A linear model widely used as a baseline in classification tasks. It is highly interpretable, fast to train, and performs well when the relationship between features and the target log-odds is approximately linear. Its main limitation is the inability to capture complex non-linear patterns.
- **Random Forest**: An ensemble learning method based on decision trees. It builds multiple decision trees on different subsets of data and features and aggregates their predictions (e.g., by averaging or voting). It handles non-linearity effectively, is robust to outliers, and implicitly performs feature selection. However, it is less directly interpretable than Logistic Regression.
- **XGBoost (Extreme Gradient Boosting)**: A highly efficient and powerful implementation of gradient boosted decision trees. It sequentially builds trees that correct the errors of previous ones, often achieving state-of-the-art performance on structured data. It includes built-in regularisation to prevent overfitting and can handle missing data internally. Like Random Forest, its complexity makes direct interpretation challenging without XAI techniques.

4.2. Hyperparameters

For this initial phase of the project, the models were trained using their default hyperparameter settings as provided by the scikit-learn and XGBoost libraries. While default settings often provide reasonable performance, optimising hyperparameters using techniques like Grid Search or Randomised Search Cross-Validation could potentially lead to further performance improvements. This is identified as an area for future work.

5. Experimental Setup

The experiments were conducted using Python (version 3.11 or similar) within a standard data science environment (e.g., Jupyter Notebook, Google Colab). Key libraries included:

- `pandas` and `numpy` for data manipulation.
- `scikit-learn` (version 1.5 or similar) for preprocessing, modelling (Logistic Regression, Random Forest), and evaluation metrics.
- `xgboost` (version 1.7 or similar) for the XGBoost model.
- `shap` for model explainability.
- `matplotlib` and `seaborn` for plotting.

Model performance was evaluated on the held-out validation set using standard classification metrics suitable for multi-class problems:

- **Accuracy**: Overall percentage of correct predictions.
- **Precision (Macro)**: Average precision across classes, measuring the accuracy of positive predictions.
- **Recall (Macro)**: Average recall across classes, measuring the ability to identify all positive instances.
- **F1-Score (Macro)**: Harmonic mean of macro-precision and macro-recall, providing a balanced measure.
- **ROC-AUC Score (OvR/OvO, Macro Average)**: Area Under the Receiver Operating Characteristic Curve, measuring the model's ability to distinguish between classes across different thresholds. Macro-averaging computes the metric independently for each class and then averages.

Macro-averaging was chosen for precision, recall, and F1-score to give equal importance to each credit score class, regardless of potential imbalance. While 5-fold cross-validation was mentioned as a validation method, the presented results appear based on the single train/validation split.

6. Results

The performance of the three trained models on the validation set is summarized below.

6.1. Performance Metrics

Table 1 shows the key performance metrics for each model.

Table 1. Model Performance Comparison on Validation Set

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1 (Macro)
Logistic Regression	0.6049	0.59	0.52	0.54
Random Forest	**0.8013**	**0.79**	**0.79**	**0.79**
XGBoost	0.7599	0.75	0.74	0.74

As observed, the **Random Forest** model achieved the highest scores across all metrics, notably reaching an accuracy of 80.13% and a balanced F1-score of 0.79. Logistic Regression performed significantly worse, likely due to its inability to capture non-linearities. XGBoost performed well, but slightly below Random Forest with the default parameters used.

6.2. ROC Curves and AUC

The Receiver Operating Characteristic (ROC) curves provide a visual comparison of the models' discriminative ability across different thresholds for each class. The Area Under the Curve (AUC) quantifies this ability.

Figure 2 displays the ROC curves for each class for all three models. The AUC scores (macro-averaged across classes) were:

- Logistic Regression: 0.7661
- **Random Forest**: **0.9228**
- XGBoost: 0.9008

Consistent with other metrics, Random Forest exhibits the highest AUC score (0.9228), indicating superior overall class separability compared to XGBoost (0.9008) and Logistic Regression (0.7661).

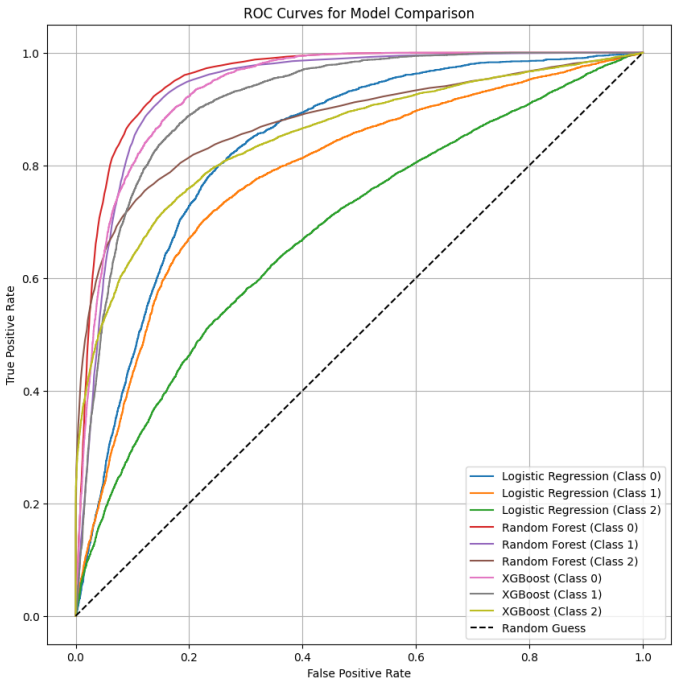

Figure 2. ROC Curves for Multi-Class Classification (One-vs-Rest shown for each model)

6.3. Confusion Matrices

Confusion matrices provide a detailed breakdown of classification performance per class, showing correct classifications (diagonal elements) and misclassifications (off-diagonal elements).

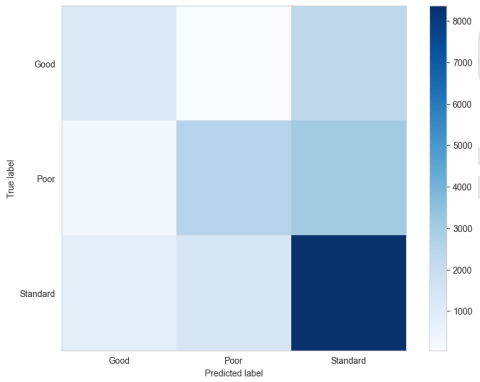

Figure 3. Confusion Matrix for Logistic Regression Model. Axes show predicted vs. true labels for credit score categories.

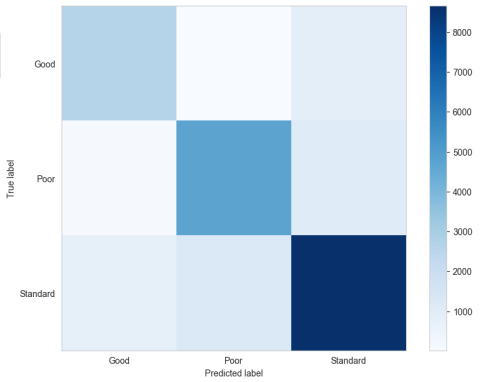

Figure 4. Confusion Matrix for Random Forest Model. Axes show predicted vs. true labels for credit score categories.

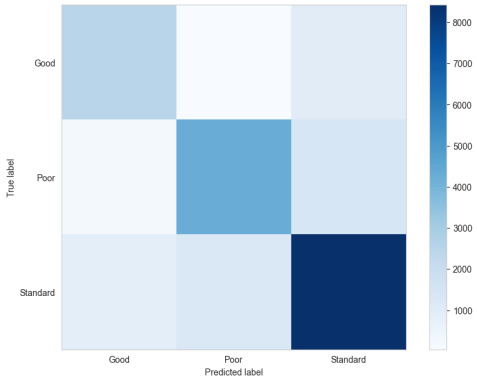

Figure 5. Confusion Matrix for XGBoost Model. Axes show predicted vs. true labels for credit score categories.

Analysis of the confusion matrices (Figures 3, 4, 5) reveals the specific types of errors made by each model. Typically, models might struggle more with distinguishing between adjacent classes (e.g., 'Standard' vs. 'Poor' or 'Standard' vs. 'Good'). The Random Forest matrix (Figure 4) shows stronger diagonal dominance compared to the others, reflecting its higher accuracy and balanced performance across classes. The RF model showed fewer misclassifications between 'Poor' and 'Standard' compared to Logistic Regression.

7. Model Explainability using SHAP

Given its superior performance, the Random Forest model was selected for further analysis using SHAP (SHapley Additive exPlanations) to understand its predictions and ensure transparency.

7.1. SHAP Summary Plot

SHAP values quantify the contribution of each feature to the prediction for each instance. The summary plot aggregates these values, providing a global view of feature importance and impact.

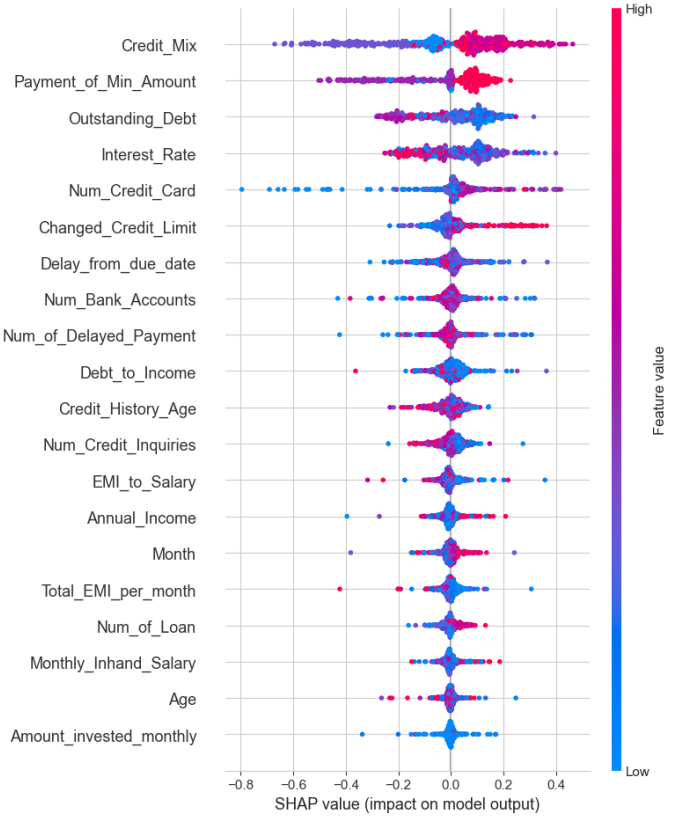

Figure 6. SHAP Summary Plot for Random Forest Model

Figure 6 illustrates the SHAP summary plot for the Random Forest model. Key observations include:

- **Feature Importance:** Features are ranked by the sum of their absolute SHAP values across all samples. The plot indicates that features like `Credit Mix`, `Payment of Min Amount`, `Outstanding Debt`, and `Interest Rate`, are among the most influential predictors for the Random Forest model.
- **Feature Impact:** The horizontal axis represents the SHAP value; positive values push the prediction towards a higher class (e.g., 'Good'), while negative values push it towards a lower class (e.g., 'Poor').
- **Value Correlation:** The color indicates the original value of the feature (high or low). For example, the plot might show that high values of `Outstanding_Debt` (red dots) typically have negative SHAP values, pushing predictions towards 'Poor' or 'Standard', while low values (blue dots) have positive SHAP values. Conversely, factors like timely payments or longer credit history might show positive SHAP values associated with higher feature values.

This analysis provides valuable insights into why the model makes certain predictions, moving beyond simple accuracy metrics towards a more transparent and trustworthy system. SHAP can also be used for local explanations (interpreting individual predictions), which is crucial for explaining decisions to customers or regulators.

7.2. Model-Specific Feature Importances

In addition to SHAP values, which provide a model-agnostic approach to explainability, we can also examine the feature importances directly derived from the models themselves. For tree-based ensembles like Random Forest and XGBoost, this is typically based on how much each feature contributes to reducing impurity (e.g., Gini impurity) across all trees. For Logistic Regression, the magnitude of the scaled coefficients can serve as an indicator of feature importance.

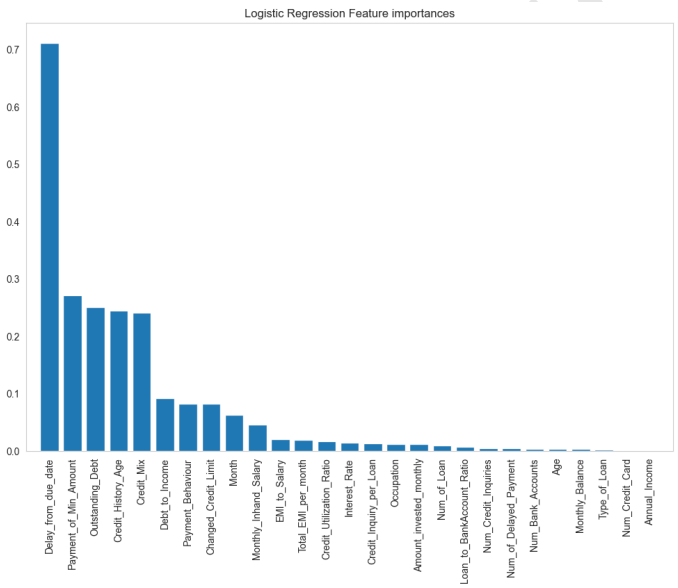

Figure 7. Feature Importances derived from the Logistic Regression Model.

Figures 7, 8, and 9 display these model-specific feature importances. For the Logistic Regression model (Figure 7), `Delay from due date` and `Payment of Min Amount` appear as highly influential. In the Random Forest model (Figure 8), `Outstanding Debt` and `Interest Rate` are among the top features, which aligns well with financial intuition and also shows some consistency with SHAP findings. The XGBoost model (Figure 9) highlights `Payment of Min Amount` and `Credit Mix` as particularly important.

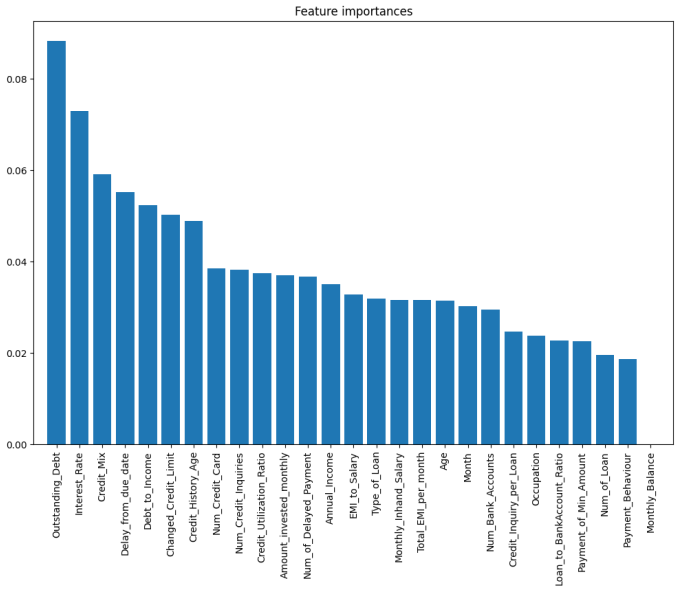

Figure 8. Feature Importances derived from the Random Forest Model.

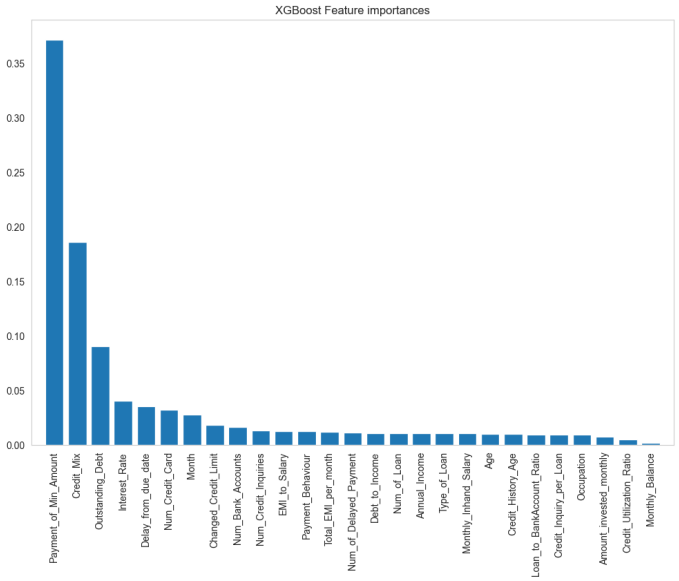

Figure 9. Feature Importances derived from the XGBoost Model.

Comparing these importances with SHAP values can provide a more holistic understanding of feature contributions. While direct model importances are useful, SHAP values offer the advantage of showing not only the magnitude but also the direction of a feature's impact on individual predictions.

8. Discussion

8.1. Key Findings

The experimental results demonstrate that ensemble methods, particularly Random Forest, significantly outperform traditional Logistic Regression for this multi-class credit scoring task. The Random Forest model achieved the best balance of predictive performance (80.13% accuracy, 0.9228 AUC) and, through the application of SHAP, offered substantial interpretability.

The SHAP analysis highlighted the key drivers of creditworthiness within the dataset, such as debt levels, payment history indicators (`Delay from due date`, `Payment of Min Amount`), credit utilisation (`Credit Mix`), and borrowing cost (`Interest Rate`). These findings align with established financial principles, lending credibil-

ity to the model's decision-making process. The ability of Random Forest to capture non-linear interactions between these features likely contributed to its superior performance over Logistic Regression.

8.2. Practical Implications

The developed Random Forest model, coupled with SHAP explanations, offers significant practical value for FinTech applications:

- **Improved Decision-Making:** Lenders can leverage the model's accuracy for better risk assessment.
- **Enhanced Transparency:** SHAP explanations allow lenders to understand the factors driving individual credit scores, facilitating internal audits and model validation.
- **Regulatory Compliance:** Explainability helps meet regulatory requirements (e.g., providing reasons for adverse credit decisions).
- **Customer Trust:** Providing clear explanations for credit decisions can improve customer understanding and trust.
- **Fairness Assessment:** While not a full fairness audit, SHAP can help identify if certain features are impacting predictions in potentially unfair ways, paving the way for bias detection and mitigation.

8.3. Limitations

Despite promising results, this study has limitations:

- **Data Scope and Bias:** The analysis is based on a single Kaggle dataset, which may not fully represent real-world lending scenarios or diverse populations. The dataset might also contain inherent biases.
- **Hyperparameter Optimisation:** Models were trained with default parameters; tuning could potentially improve performance further.
- **Feature Engineering:** Only basic feature engineering was performed; more sophisticated features could enhance predictive power.
- **SHAP Runtime:** Calculating SHAP values, especially for large datasets or complex models, can be computationally intensive, requiring workarounds like sampling (as noted in Challenges).
- **Validation Strategy:** Reliance on a single train-validation split is less robust than cross-validation for estimating generalization performance.

9. Challenges & Solutions

Several challenges were encountered during the project, along with the strategies used to address them:

- **SHAP Runtime:** Calculating SHAP values for the entire validation set was time-consuming. *Solution:* A representative subset (e.g., 500 samples) of the validation data was used for generating SHAP summary plots, balancing computational feasibility with informative insights.
- **Model Explainability Interpretation:** Understanding how to apply and interpret SHAP results effectively for different model types required learning. *Solution:* Studied SHAP documentation and examples, focusing on summary plots and the meaning of SHAP values for classification tasks.
- **Overfitting Risk:** Complex models like Random Forest and XGBoost are prone to overfitting the training data. *Solution:* Performance was strictly evaluated on the held-out validation set. Metrics like macro-F1 were monitored, which balance precision and recall, providing a good measure of generalisation, especially with potential class imbalance. Default regularisation parameters in RF and XGBoost also help mitigate overfitting.
- **Feature Availability and Quality:** Real-world data often has missing values or inconsistencies. *Solution:* Implemented robust preprocessing steps, including median/mode imputation for missing values. Error handling could be added for production systems.
- **Class Imbalance (Potential):** The distribution of 'Good', 'Standard', and 'Poor' classes might be uneven. *Solution:* Used stratified sampling during the train-validation split to preserve class proportions. Evaluated models using macro-averaged metrics (Precision, Recall, F1) which treat all classes equally.

10. Conclusion & Future Work

This project successfully demonstrated the development of a transparent and high-performing credit scoring model using explainable machine learning. By comparing Logistic Regression, Random Forest, and XGBoost, we identified Random Forest as the most suitable model for this dataset, achieving approximately 80% accuracy and a high AUC of 0.92. The integration of SHAP provided crucial interpretability, revealing the key factors influencing the model's predictions and aligning with financial intuition.

This approach addresses the critical need for transparency in FinTech, fostering trust and facilitating compliance. The findings indicate that it is feasible to achieve both high accuracy and explainability in credit scoring.

Future work could explore several avenues for enhancement:

- **Hyperparameter Tuning:** Employ systematic methods like Grid Search or Bayesian Optimization to fine-tune the Random Forest model for potentially higher performance.
- **Advanced Feature Engineering:** Create more sophisticated features (e.g., interaction terms, time-based features if applicable) to capture deeper insights.
- **Explore Alternative Models:** Investigate other advanced models like LightGBM, CatBoost, or deep learning approaches (e.g., TabNet) combined with appropriate XAI techniques (LIME, Integrated Gradients).
- **Formal Fairness Analysis:** Conduct a rigorous fairness audit using specialised toolkits (e.g., AIF360, Fairlearn) to assess and mitigate potential biases related to sensitive attributes (if available).
- **Dataset Expansion:** Validate the model and findings on larger, more diverse, and potentially proprietary datasets to ensure real-world applicability.
- **Deployment Considerations:** Investigate aspects of deploying such a model in a real-time system, including MLOps practices and monitoring for concept drift.

By pursuing these directions, we can further advance the development of fair, accurate, and transparent credit scoring systems that benefit both financial institutions and consumers.

References

[1] S. Lessmann, B. Baesens, H.-V. Seow, and L. C. Thomas, "Benchmarking classification models for credit scoring: A systematic literature review", *European Journal of Operational Research*, vol. 247, no. 1, pp. 124–139, 2015.

[2] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[3] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions", in *Advances in neural information processing systems (NIPS)*, 2017, pp. 4765–4774.

[4] L. C. Thomas, D. B. Edelman, and J. N. Crook, *Credit scoring and its applications*. SIAM, 2017.

[5] Parishon, *Credit score classification dataset*, 2023. [Online]. Available: https://www.kaggle.com/datasets/parishon/credit-score-classification/data.

A. Appendix: Selected Code Snippets

This appendix contains selected Python code snippets demonstrating key parts of the project workflow, including data cleaning, feature engineering, model training, and SHAP value calculation.

A.1. Data Cleaning Function

The following function was used to perform initial cleaning on the raw dataset, including dropping irrelevant columns, sanitizing numerical features, and converting specific columns like `Credit_History_Age`.

```python
def clean_dataset(df):
    # Drop unwanted columns
    df = df.drop(columns=['ID', 'Customer_ID', 'Name', 'SSN'],
      ↪ errors='ignore')

    # Numeric columns to clean
    num_cols = ['Age', 'Annual_Income', 'Monthly_Inhand_Salary',
      ↪  # ... (shorten if too long)
                'Num_of_Loan', 'Num_of_Delayed_Payment', '
      ↪ Changed_Credit_Limit',
                'Num_Credit_Inquiries', 'Outstanding_Debt', # ...
      ↪  (shorten if too long)
                'Total_EMI_per_month', 'Amount_invested_monthly',
      ↪  'Monthly_Balance']

    for col in num_cols:
        df[col] = pd.to_numeric(df[col].replace(r'[^0-9.-]', '',
      ↪  regex=True),
                                errors='coerce')

    # Clean Age
    df['Age'] = df['Age'].apply(lambda x: x if pd.notnull(x) and
      ↪  x > 0 else np.nan)

    # Convert Credit_History_Age
    def age_to_months(val):
        if isinstance(val, str):
            m = re.search(r'(\d+)\s+Years?\s+and\s+(\d+)\s+
      ↪ Months?', val)
            if m:
                return int(m.group(1))*12 + int(m.group(2))
        return np.nan

    df['Credit_History_Age'] = df['Credit_History_Age'].apply(
      ↪ age_to_months)

    # Impute categorical (example, full list in main text)
    cat_cols = ['Occupation', 'Type_of_Loan', 'Credit_Mix', # ...
      ↪  (shorten)
                'Payment_of_Min_Amount', 'Payment_Behaviour', '
      ↪ Month']
    for col in cat_cols:
        if col in df.columns:
            df[col] = df[col].fillna(df[col].mode()[0])

    # Impute numeric (example, full list in main text)
    for col in num_cols + ['Credit_History_Age']:
        df[col] = df[col].fillna(df[col].median())

    return df
```

Code 1. Python function for cleaning the dataset.

Note: Some longer lists of column names in the listing above have been truncated for brevity. Refer to the main text for complete lists.

A.2. Feature Engineering

This snippet shows the creation of derived features such as Debt-to-Income ratio.

```python
# Month mapping (assuming month_map is defined earlier)
# for df in [train, test]:
#     df['Month'] = df['Month'].map(month_map)

# Derived features
for df in [train, test]: # Applied to both train and test sets
    df['Debt_to_Income'] = df['Outstanding_Debt'] / (df['
      ↪ Annual_Income'] + 1e-5)
    df['Loan_to_BankAccount_Ratio'] = df['Num_of_Loan'] / (df['
      ↪ Num_Bank_Accounts'] + 1e-5)
    df['EMI_to_Salary'] = df['Total_EMI_per_month'] / (df['
      ↪ Monthly_Inhand_Salary'] + 1e-5)
    df['Credit_Inquiry_per_Loan'] = df['Num_Credit_Inquiries'] /
      ↪  (df['Num_of_Loan'] + 1e-5)
```

Code 2. Python code for feature engineering.

A.3. Categorical Encoding and Scaling

The following code demonstrates label encoding for categorical features and standard scaling for numerical features.

```python
# Encode categorical features (example, full list in main text)
cat_cols = ['Occupation','Type_of_Loan','Credit_Mix', # ... (
      ↪ shorten)
            'Payment_of_Min_Amount','Payment_Behaviour']
encoders = {}
for col in cat_cols:
    le = LabelEncoder()
    train[col] = le.fit_transform(train[col])
    test[col] = le.transform(test[col]) # Apply same encoding to
      ↪  test
    encoders[col] = le

# Encode target
target_le = LabelEncoder()
train['Credit_Score'] = target_le.fit_transform(train['
      ↪ Credit_Score'])

# Scale numeric features
scale_cols = train.select_dtypes(include=[np.number]).columns.
      ↪ tolist()
scale_cols.remove('Credit_Score') # Exclude target variable
scaler = StandardScaler()
train[scale_cols] = scaler.fit_transform(train[scale_cols])
test[scale_cols] = scaler.transform(test[scale_cols]) # Apply
      ↪ same scaling to test
```

Code 3. Python code for encoding and scaling.

A.4. Model Training (Example: Random Forest)

This shows the initialisation and training of the Random Forest classifier. Similar steps were followed for Logistic Regression and XGBoost.

```python
# Define features (X) and target (y)
features = [c for c in train.columns if c != 'Credit_Score']
X = train[features]
y = train['Credit_Score']

# Split data
X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  test_size=0.2,
                                                  random_state=42,
                                                  stratify=y)

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

Code 4. Example of Random Forest model training.

Note: Stratification was added to the `train_test_split` in the listing above for best practice, assuming `y` is the pre-encoded target.

A.5. SHAP Value Calculation (Example: Random Forest)

This snippet demonstrates how SHAP values were calculated for the Random Forest model to explain its predictions.

```python
# SHAP for Random Forest
# Sample validation data for faster SHAP computation if dataset
      ↪ is large
sampled_X_val = X_val.sample(500, random_state=42)

# For tree-based models like Random Forest, explainer can take
      ↪ the model directly
explainer_rf = shap.Explainer(rf, sampled_X_val) # Pass
      ↪ background data
shap_values_rf = explainer_rf(sampled_X_val) # Calculate SHAP
      ↪ for sampled validation

# To plot (example, assuming multi-class SHAP values)
# shap.summary_plot(shap_values_rf, sampled_X_val, class_names=
      ↪ target_le.classes_)
```

```
# Or for a specific class, e.g., class 1:
# shap.summary_plot(shap_values_rf[:, :, 1], sampled_X_val)
```

Code 5. SHAP value calculation for Random Forest.

Note: The SHAP explainer for tree models can often take the background dataset directly. The `predict_proba` or `predict` method might be needed for other SHAP explainers depending on the model type.

A.6. Cross-Validation Setup

The following shows the setup for performing 5-fold cross-validation.

```
from sklearn.model_selection import cross_val_score

# Define features and target (as before)
# X = train[features]
# y = train['Credit_Score']

# Initialize models (example for Random Forest)
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100,
     ↪ random_state=42)
    # ... other models
}

cv_results = {}
for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5, scoring='
     ↪ accuracy')
    cv_results[name] = scores
    print(f"Mean Accuracy for {name}: {scores.mean():.4f} (+/- {
     ↪ scores.std():.4f})")
```

Code 6. Cross-validation setup for model evaluation.