COMS W4995 Applied Machine Learning Final Report

Group members: *Meilin Guo (mg4578), Chiara Wollner (crw2160), Angela Mu (aym2122), Hussain Doriwala (hd2551), Nagavasavi Jeepalyam (nj2506)*

Project Goal: This project aims to use credit-related information and other banking details to predict and classify an individual's credit score bracket.

Data Exploration and Processing

We've transformed columns like 'Credit_History_Age' from strings to integers measured in total months. Custom functions were developed to correct formatting errors, substitute problematic values with NaNs in numerical columns, and fix negative values that likely represent data entry errors. Data is classified into numerical, categorical, or integer types for targeted processing. We also investigated categorical data for irregularities and thoroughly examined missing data. Basic visualization techniques were employed to review data distributions and handle outliers effectively. In our correlation analysis, we identify and remove highly correlated features such as 'Monthly_Inhand_Salary'.

Machine Learning Framework

Development-Test split

Given that our dataset was not extremely imbalanced, with a class distribution of 53% Standard; 29% Poor; 18% Good, and given that in the real world, data for the credit score scenario can be expected to be imbalanced as well, we chose to not use techniques like undersampling, oversampling, or SMOTE, etc. to balance out the classes. However, we experimented with using both a stratified and non-stratified data split in our model training to compare whether one method of data splitting would lead to better performance. In both cases, an 80/20 dev to test ratio was employed, following common industry practice.

Hyperparameter tuning

We applied grid search for hyperparameter tuning, setting specific parameter grids tailored to each model type (refer to the model details below). During our grid search, we evaluated models using both F1 score and accuracy as our primary scoring metrics. Moreover, we decided to utilize 5-fold cross-validation. Considering the size of our dataset, this method proved to be stable and effective compared to other model selection strategies.

Optimal model training

Decision Trees

Model Structure: This model was chosen due to its interpretability, ease of handling different types of data, and its ability to model non-linear relationships without assuming data distributions—attributes crucial for the complexity of financial data.

Initial model - Basic Decision Tree: We tested two variations: one using the Gini index and another using entropy for node splitting. The Gini model outperformed the entropy model across various metrics (F1, precision, recall). Given its superior performance and greater computational efficiency, we decided to continue using the Gini criterion. Our initial tests revealed that while the model achieved perfect training scores, it dropped to a test accuracy and F1 score of 0.743, indicating overfitting. This led us to implement grid search for hyperparameter tuning.

Model Hyperparameters: Max depth: [None, 5, 10, 30, 50], Min Samples Split: [2, 10, 30, 50], Min Samples Leaf: [1, 2, 4, 10], Criterion: ['gini', 'entropy']

Support Vector Machine

Model Structure: The Linear SVC model constructs a linear decision boundary to separate the classes in the feature space. It was chosen over regular SVC given the large dataset and feature space sizes.

Model Hyperparameters: Regularization parameter: [0.1, 1, 10, 100]

XGBoost

Model Structure: This model was chosen as it builds decision trees sequentially and uses gradient boosting, with each tree correcting errors made by the previous one, while still being efficient.
Model Hyperparameters: Learning Rate: [0.1, 0.01, 0.001], Max Depth: [3, 4, 5], # Estimators: [100, 200, 300]

Random Forest
Model Structure: A random forest is composed of a large number of individual decision trees, each making a prediction based on the input features by following decision paths from the root to a leaf node.
Model Hyperparameters: Number of estimators:[2,5,10],Criterion:[‘gini’,’entropy’], Maximum depth of trees:[None,5,10,30,50], Minimum samples split:[2,10,30,50], Minimum samples leaf:[1,2,4,10]

KNN (K-Nearest Neighbors)
Model Structure: K-Nearest Neighbors (KNN) classifier operates on the principle of similarity, assigning class labels based on the majority class among the k nearest neighbors in the feature space.
Model Hyperparameters: n_neighbors: [3, 5, 7, 9, 11]

Feed-Forward Neural Networks
Model Structure: The architecture includes densely connected layers with decreasing units from 256 to 32, interspersed with batch normalization to stabilize inputs, ReLU activation functions to introduce non-linearity, and dropout layers to reduce overfitting. The network culminates in a softmax output layer for classifying into three categories. It's compiled with the Adam optimizer, sparse categorical crossentropy for loss, and accuracy as the performance metric. Stratified K-Fold cross-validation with five splits ensures consistent class distributions across each fold during model evaluation.
Hyperparameters: number of neurons per layer (256, 128, 64, 32), dropout rates (0.3 for first two layers, 0.2 for last two layers), learning rate (0.0001), optimizer (Adam), loss function (sparse categorical crossentropy), number of training epochs (50).

Optimal Model Evaluation and Comparison:

The table below details the optimal parameters and performance metrics for each model we tested. The model that performed the best overall is highlighted.

Model	Accuracy	F1	Recall	Precision	Model Details
SVM	0.64	0.64	0.64	0.64	Not stratified split, Best params: {'C': 1}
XGBoost	0.77	0.77	0.77	0.78	Not stratified split, Best params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Decision Trees	0.75	0.75	0.75	0.75	Stratified split, Best params: {'criterion': 'gini', 'max_depth': 30, 'min_samples_leaf': 2, 'min_samples_split': 2}
Random Forest	**0.83**	**0.83**	**0.83**	**0.83**	**Not stratified split, Best params: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, ‘min_samples_leaf’:3, ‘max_depth’:10, ‘n_estimors’:2}**
KNN	0.77	0.77	0.77	0.77	Stratified split,Initializing KNN classifier with custom distance metric and weighting:{n_neighbors=5, metric='manhattan', weights='distance'}
Neural Network	0.70	0.69	0.71	0.68	Stratified Split, Learning Rate = 0.001, Loss Function = sparse categorical cross entropy

Comparison of Feature Importances Across Models:

Top 3 Feature Importances for each model:
Decision Tree: 1) Outstanding Debt (0.20), 2) Credit Mix (0.13), 3) Interest Rate (0.07)
XGboost: 1) Credit Mix (0.57), 2) Outstanding Debt (0.06), 3) Interest Rate (0.03)
Random Forest: 1) Outstanding Debt (0.10), 2) Interest Rate (0.07), 3) Credit Mix (0.07)

All 3 models consistently identify Outstanding Debt, Credit Mix, and Interest Rate as the most important features for predicting credit scores, which shows their significance in our classification task. Outstanding Debt is ranked as the most important feature in both the Decision Tree and Random Forest models, but it is less significant in XGBoost. XGBoost assigns over half importance on Credit Mix, and all other features are of less than 0.1 importances, which might contribute to overfitting. Interest Rate is consistently assigned with less than 0.1 importance scores, indicating it plays a supportive but less decisive role in these models.

Run Time Comparison: It takes 2 minutes to train Decision Trees and XGboosts, 5 minutes to train SVMs, KNNs and Neural Networks, and 2 hours to train Random Forests.

Prioritizing F1 over Accuracy:

1. **Data is slightly imbalanced:** Accuracy can be misleading in imbalanced datasets. F1 score harmonizes precision (accuracy of positive predictions) and recall (coverage of actual positives), essential for balanced performance in credit decisions.
2. **Consider financial risks:** Firms care more on reducing False Positives and False Negatives than simple accuracy: Minimizing instances where bad credit is predicted as good, thus avoiding potential defaults. Ensuring creditworthy individuals are not wrongly denied loans, thus avoiding loss of potential revenue.

Prioritizing Precision over Recall:
In the context of credit scoring, false positives (e.g. incorrectly classifying someone as having a good credit score when they are likely to default) are particularly problematic. Therefore, it is crucial to prioritize precision. This approach lowers financial risks by reducing false positives, since the costs of defaults are often much higher than the potential gains from issuing additional loans.

Best Model Found: The model with the best performance is Random Forest. This ensemble approach typically provides a higher performance than any individual tree could, especially in terms of robustness and error reduction. Given these aspects, the performance of the Random Forest could likely be attributed to its ability to maintain high performance across various dimensions of the evaluation metrics while effectively handling the inherent complexities and potential biases in the data.

Stratified vs Non-Stratified Train/Test Split: Looking at our best models, 3 used non-stratified splitting and 4 used stratified splitting. From an empirical perspective, the performance difference when using stratified and non-stratified split is very small (e.g. SVM F1 score on not stratified (0.64) v.s. stratified (0.63)). Thus, using a stratified split is not much more effective than using a non-stratified split.

Recommendations for Further Analysis:
Hyperparameter Tuning: Use random search for hyperparameter tuning instead of grid search, as it can explore the hyperparameter space more efficiently and often finds better parameters in less time.
Cross-Validation: Use repeated K-Fold Cross Validation, which may more accurately evaluate validation set performances by averaging results over several iterations.
Feature Selection: Leveraging techniques such as Principal Component Analysis for feature selection.
Model Refinement*:* Exploring different kernel choices (for SVM) or network architectures (for Neural Network) could yield better results.
Ensemble Techniques*:* Considering an ensemble of the models to leverage the strengths of each model, potentially leading to better overall performance.