Credit Risk Modeling Using Machine Learning Classifiers

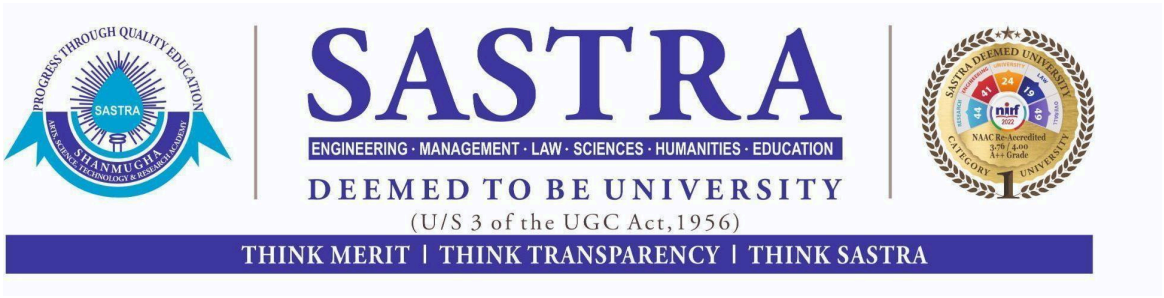
Report submitted to SASTRA Deemed to be University As per the requirement for the course

CSE425 : MACHINE LEARNING ESSENTIALS

Submitted by

Rimmalapudi Sridhar
(Reg No: 125018065, B. Tech Computer Science and Business Systems)

OCTOBER- 2024



SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Table of Contents

ABSTRACT

This project focuses on credit risk classification, an essential task in the financial sector aimed at predicting whether a borrower is likely to default on a loan. The project applies machine learning techniques using various models to predict loan default based on factors such as age, income, home ownership, employment length, loan amount, and credit history. The models are compared based on evaluation metrics, such as accuracy, precision, recall, F1 score, and ROC AUC, to identify the best performing model.

INTRODUCTION

Importance of Dataset

In the world of finance, the ability to accurately assess credit risk is crucial for minimizing defaults and ensuring the financial stability of lending institutions. The dataset used in this project provides valuable insights into a borrower's financial history and loan characteristics, which are essential for predicting the likelihood of default. Features such as `person_age`, `person_income`, `loan_amnt`, and `loan_grade` allow us to analyze the financial behavior of individuals and classify them as either low-risk or high-risk borrowers. By leveraging this dataset, we aim to build a reliable credit risk classification model that helps financial institutions make informed lending decisions.

Project Objective

The primary objective of this project is to develop and compare multiple machine learning models to predict whether a person will default on their loan. By analyzing the dataset, we aim to create a robust predictive model that accurately classifies borrowers based on their likelihood of default. The

focus is on improving model performance through various machine learning algorithms, ensuring that the final model can assist in real-world credit risk evaluation.

Problem Statement

The main challenge in this project is to accurately predict loan default status based on historical data of borrowers. Given the complexity and variability in financial data, it is difficult to distinguish high-risk from low-risk borrowers without the use of advanced machine learning techniques. Additionally, the dataset contains missing values and imbalances, making it critical to preprocess and handle the data effectively. The problem becomes further complicated by the need to balance model accuracy with interpretability, especially when dealing with financial decisions that impact lending institutions and borrowers alike.

Approach

To tackle this problem, we employ a variety of machine learning algorithms, including traditional models like Logistic Regression and Decision Trees, as well as more advanced ensemble methods like Random Forest, XGBoost, and LightGBM. The workflow consists of several steps:

- Data Preprocessing: Handling missing values, encoding categorical variables, and normalizing the dataset to ensure compatibility with machine learning algorithms.

- Model Training and Evaluation: Training multiple models and evaluating them based on key metrics such as Accuracy, Precision, Recall, F1 Score, and ROC-AUC to identify the best-performing model.

- Comparison and Results: Comparing the performance of all models to select the most suitable one for credit risk classification, with a focus on balancing predictive power and interpretability.

By following this approach, we aim to build a reliable and efficient model that can be used for credit risk classification, aiding financial institutions in making better-informed decisions regarding loan approval and risk management.

Related Work

References:

Dataset: https://www.kaggle.com/code/jacevu/credit-risk-analysis/input

https://www.w3schools.com/python/python_ml_logistic_regression.asp

https://www.datacamp.com/tutorial/decision-tree-classification-python

https://www.datacamp.com/tutorial/xgboost-in-python

https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/

https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python/

https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/

https://www.datacamp.com/tutorial/adaboost-classifier-python

https://www.geeksforgeeks.org/ml-bagging-classifier/

Background

- **Models**:

Model	Type	Working Principle	Advantages (Faced in Project)	Disadvantages (Faced in Project)
K-Nearest Neighbors	Non-Parametric	Finds the K nearest neighbors to a data point based on a distance metric (e.g., Euclidean distance).	Easy to understand and implement.	Slower prediction time with large datasets, sensitive to feature scaling.
K-Means Clustering	Unsupervised Learning	Partitions data into K clusters by iteratively minimizing the distance between points and centroids.	Efficient for clustering and easy to interpret.	Sensitive to outliers, requires predefining the number of clusters (K).
Support Vector Machine	Supervised (Classification)	Finds an optimal hyperplane that separates data with the maximum margin.	Works well for high-dimensional data, good with outliers.	Time-consuming to tune hyperparameters like kernel and C value.
Decision Tree	Supervised (Classification)	Splits data recursively based on feature values to form a tree of decisions.	Easy to interpret, visualizable.	Prone to overfitting, especially without pruning, struggled with noisy data.
Random Forest	Ensemble (Bagging)	Constructs multiple decision trees on bootstrapped samples and aggregates results.	Reduced overfitting, good performance on various datasets.	Training took longer, high memory consumption during this project.
Logistic Regression	Supervised (Classification)	Calculates the probability of class membership using a linear combination of features and a sigmoid function.	Interpretable model with coefficients, fast training.	Performed poorly on non-linearly separable data.

Gradient Boosting	Ensemble (Boosting)	Sequentially builds models that correct the errors of previous models using residuals.	High accuracy, especially with hyperparameter tuning.	Prone to overfitting if not properly regularized, longer training time.
XGBoost	Ensemble (Boosting)	Uses gradient boosting with optimizations such as regularization and parallelization.	Very high performance and scalability, handles large datasets efficiently.	Longer time for hyperparameter tuning during experimentation.
LightGBM	Ensemble (Boosting)	Leaf-wise gradient boosting with optimized speed and memory usage, grows trees asymmetrically.	Very fast training and low memory usage, ideal for large datasets.	Required careful tuning to avoid overfitting, especially with small datasets.
AdaBoost	Ensemble (Boosting)	Combines weak learners sequentially by focusing more on the misclassified points from previous rounds.	Improved accuracy by emphasizing difficult examples.	Struggled with noisy data and misclassifications, required balancing weights.

DATASET DESCRIPTION

The dataset used in this project consists of 123418 instances.This includes a range of features that are likely useful for age prediction, including physical measurements and weight-related metrics.

Dataset Features:

The dataset used for this project consists of 32,581 entries with the following columns:
- **person_age:** The age of the individual (int).
- **person_income:** The annual income of the person (int).
- **person_home_ownership:** Home ownership status (object).
- **person_emp_length:** Employment length in years (float).
- **loan_intent:** The purpose for which the loan was taken (object).
- **loan_grade:** The grade assigned to the loan (object).

- **loan_amnt:** The amount of the loan (int).
- **loan_int_rate:** The interest rate of the loan (float).
- **loan_status:** The default status of the loan (1: default, 0: no default) (int).
- **loan_percent_income:** Loan amount as a percentage of annual income (float).
- **cb_person_default_on_file:** Whether the person has a history of default (object).
- **cb_person_cred_hist_length:** Length of the credit history (int).

Missing values in the dataset were handled through imputation techniques for some columns (e.g., loan interest rate) while others like employment length had missing values filled with the median value.

Data Sample

person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate
22	59000	RENT	123.0	PERSONAL	D	35000	16.02
21	9600	OWN	5.0	EDUCATION	B	1000	11.14
25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87
23	65500	RENT	4.0	MEDICAL	C	35000	15.23
24	54400	RENT	8.0	MEDICAL	C	35000	14.27

- **Preprocessing**:

Preprocessing is a crucial step in preparing the data for machine learning algorithms. It ensures that the dataset is clean, complete, and suitable for training models. The following preprocessing steps were undertaken for the credit risk classification project:

1. Handling Missing Values

The dataset contained missing values, particularly in the `person_emp_length` and `loan_int_rate` columns. To address this:

- For numerical columns like `loan_int_rate` and `person_emp_length`, we used mean imputation to replace the missing values with the average value of the column. This ensures that missing data does not affect model performance.

2. Encoding Categorical Variables

Several columns, such as `person_home_ownership`, `loan_intent`, and `cb_person_default_on_file`, contained categorical values that needed to be converted into a numerical format for machine learning models:

- We applied One-Hot Encoding to transform categorical columns into binary columns, ensuring compatibility with algorithms like Logistic Regression and SVM, which require numerical input.

3. Feature Scaling

To ensure that features with large numerical ranges do not dominate models that rely on distance metrics (e.g., K-Nearest Neighbors and SVM), we performed feature scaling:

- Standardization was applied to scale features like `person_income`, `loan_amnt`, and `loan_percent_income`, transforming them to a standard normal distribution (mean = 0, standard deviation = 1).

4. Handling Imbalanced Data

Credit risk datasets often have an imbalance between classes (default vs. non-default), which can bias models:

- To mitigate this issue, we explored methods like class weighting in models such as Logistic Regression and Random Forest. Additionally, Resampling Techniques (such as oversampling the minority class) were considered to ensure a balanced training set.

5. Feature Selection

To improve model performance and reduce computational

complexity, we carefully selected relevant features:

- Based on domain knowledge and exploratory data analysis, we retained key features like `person_age`, `person_income`, `loan_grade`, and `cb_person_cred_hist_length` that are highly correlated with loan default prediction.

- Features with low variance or high correlation with other features were either dropped or combined to avoid multicollinearity.

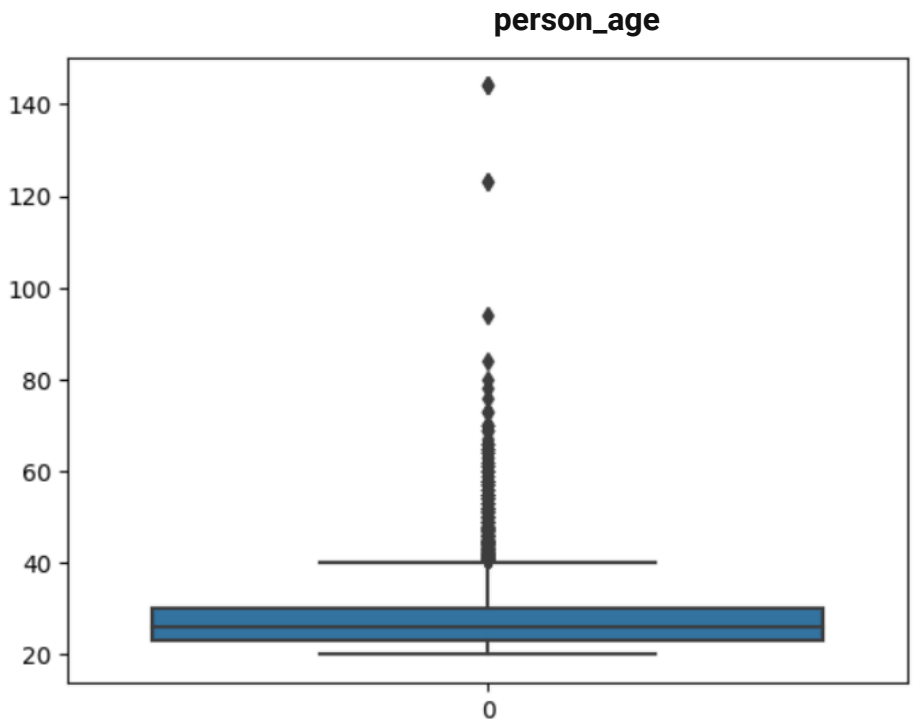
6. Splitting Data into Training and Testing Sets

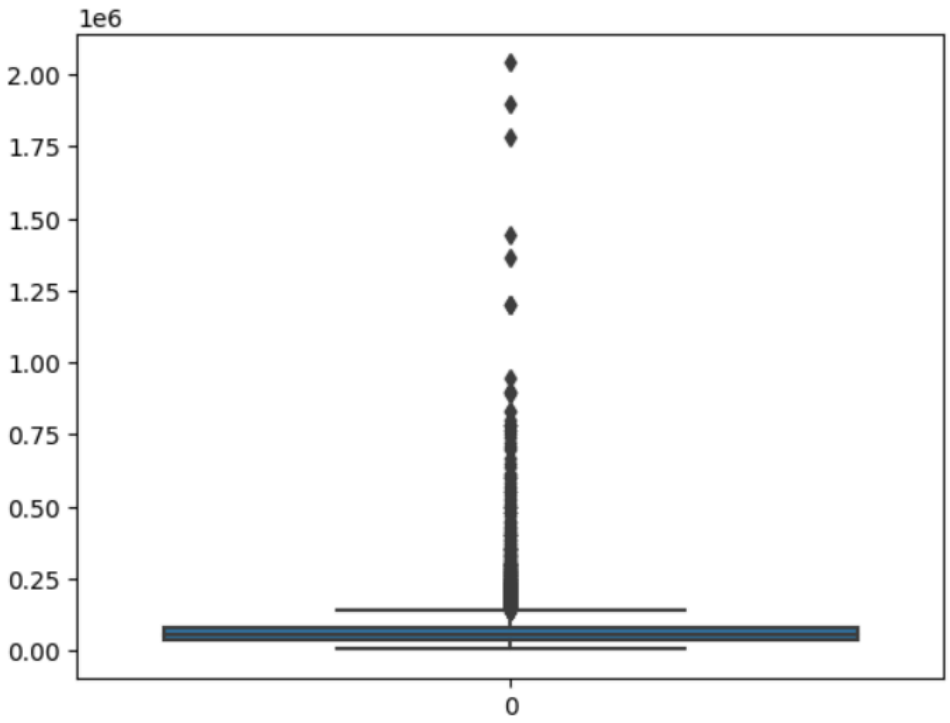
To assess model performance, we split the dataset into training and testing sets:

- We used an 80-20 split, where 80% of the data was used for training the models and 20% for evaluating their performance. This step ensures that models are tested on unseen data, simulating real-world scenarios.

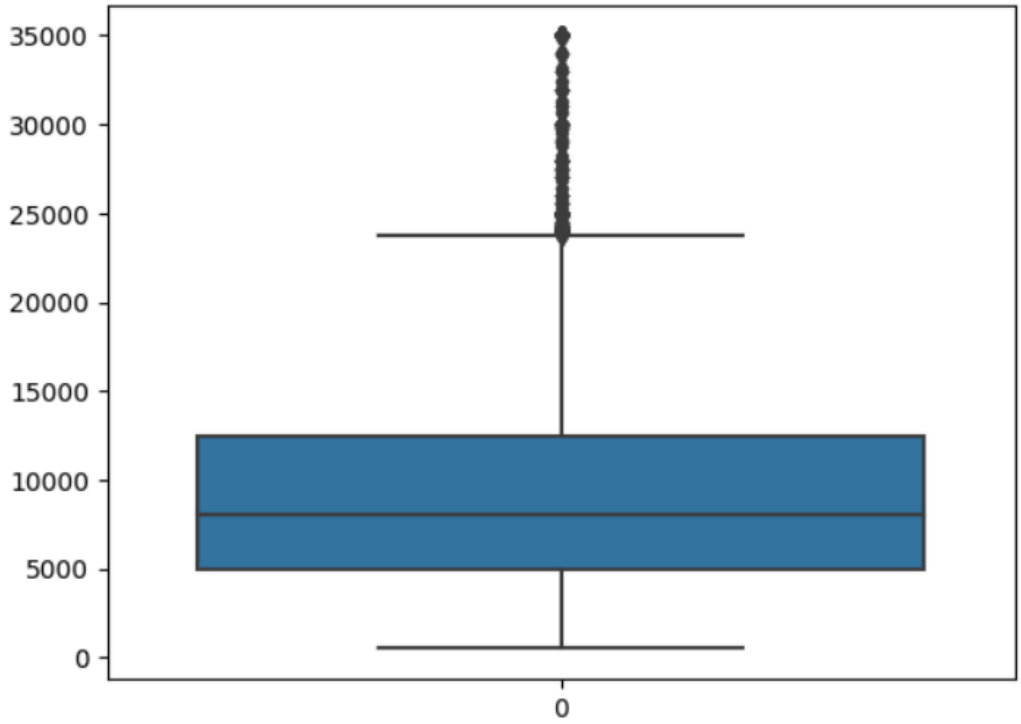
These preprocessing steps ensure that the dataset is clean, balanced, and ready for model training, helping to improve the accuracy and reliability of the credit risk classification models.

OUTLIER DETECTION

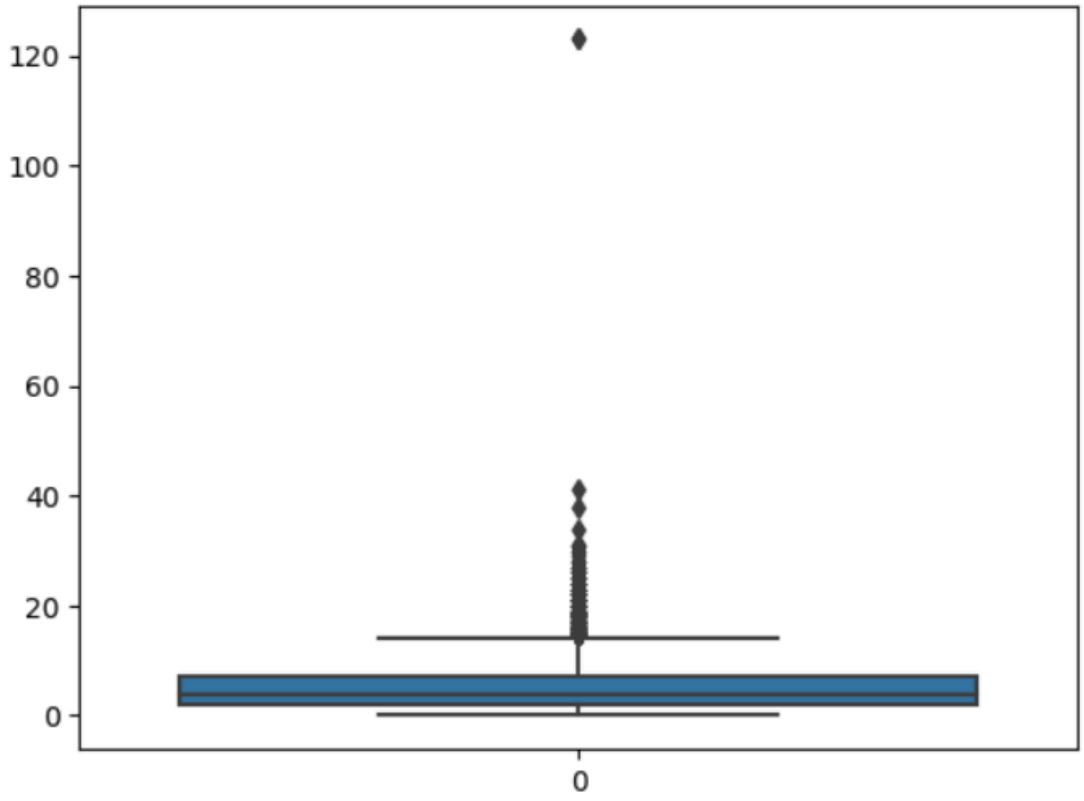




person_income



loan_amnt



person_emp_length

Methodology

Experimental Design:

The experimental design includes several steps, from data preparation to model evaluation, ensuring a structured approach to finding the optimal machine learning model.

Data Preprocessing

As detailed in the preprocessing steps section, the dataset was cleaned, encoded, and scaled to ensure compatibility with various machine learning algorithms. Missing values were imputed, and categorical variables were converted into numerical formats using One-Hot Encoding.

Model Selection

A wide variety of machine learning models were selected to compare their performances in solving the credit risk classification problem. These models

include:

Support Vector Machine (SVM)
K-Nearest Neighbors (KNN)
Decision Tree
Logistic Regression
Random Forest
AdaBoost
Bagging Classifier
Gradient Boosting
XGBoost
LightGBM

These models represent a combination of:

Linear Models (e.g., Logistic Regression, SVM)
Non-Linear Models (e.g., Decision Tree, Random Forest)
Ensemble Models (e.g., Gradient Boosting, XGBoost, LightGBM)

Model Training and Tuning

Train-Test Split: The dataset was split into training (80%) and testing (20%) sets to evaluate model performance on unseen data.

Cross-Validation: A 5-fold cross-validation approach was applied to ensure the model's generalization capabilities and avoid overfitting. Cross-validation helped fine-tune model hyperparameters while balancing bias and variance.

Hyperparameter Tuning: Using GridSearchCV, we fine-tuned hyperparameters for each model, such as the number of neighbors for KNN,

depth for Decision Trees, and learning rate for boosting models like XGBoost and LightGBM.

Environment and Tools: Python, Scikit-learn, XGBoost, CatBoost, LightGBM and Google Colab.

Model Parameters Explanation

1. K-Nearest Neighbors (KNN)

- K Value (Number of Neighbors):

Determines how many nearest neighbors to consider when making a prediction. A smaller K value may make the model more sensitive to noise, while a larger K smooths predictions but may miss finer details.

- Distance Metric:

The method used to calculate the distance between points (e.g., Euclidean, Manhattan). The choice of distance metric impacts the model's ability to identify similar points.

2. K-Means Clustering

- K Value (Number of Clusters):

Specifies the number of clusters the algorithm will attempt to form. Selecting the right K value is crucial for optimal clustering, often done using methods like the Elbow Method.

- Distance Metric (Euclidean Distance):

Used to assign points to the nearest cluster center. K-Means minimizes the sum of squared distances between data points and their corresponding cluster centroid.

3. Support Vector Machine (SVM)

- Kernel Type:

Defines how the algorithm transforms the input data into higher dimensions to make it linearly separable. Common kernels include linear, polynomial, and radial basis function (RBF).

- C Parameter:

Controls the trade-off between maximizing the margin and correctly classifying training points. A smaller C makes the margin larger but allows more misclassifications; a larger C prioritizes correct classification with a smaller margin.

- Gamma (in RBF Kernel):

Determines the influence of a single training example. A high gamma means closer points are considered, leading to a more complex model.

4. Decision Tree

- Max Depth:

The maximum depth of the tree, controlling how deep the tree can grow. Limiting the depth prevents overfitting.

- Min Samples Split:

The minimum number of samples required to split an internal node, which prevents the tree from growing too large and complex.

- Criterion (Gini or Entropy):

The metric used to determine the quality of a split. Gini impurity or information gain (entropy) helps decide how to split data at each node.

5. Random Forest

- Number of Estimators:

The number of trees in the forest. A larger number of trees increases stability but also computational cost.

- Max Features:

The number of features considered when splitting a node. Controlling this parameter reduces variance and helps avoid overfitting.

6. Logistic Regression

- Penalty (L1/L2 Regularization):

L1 (Lasso) or L2 (Ridge) regularization is used to penalize large coefficients, reducing overfitting and improving generalization.

- C Parameter:

Inversely controls the strength of regularization. A smaller C applies stronger regularization, simplifying the model but potentially underfitting.

7. Gradient Boosting/XGBoost

- Learning Rate:

Determines the contribution of each tree to the overall model. A lower learning rate makes the model more robust but slower to converge.

- Number of Estimators:

The number of boosting rounds, or trees, to fit. More estimators can improve accuracy but risk overfitting if not controlled.

- Max Depth:

Limits the depth of each tree to prevent overfitting, making the model less complex.

8. LightGBM

- Num Leaves:

Controls the complexity of the tree, with a higher number of leaves leading to a more accurate model but a greater risk of overfitting.

- Min Data in Leaf:

The minimum number of data points in one leaf, which prevents small leaves and controls overfitting.

Each model's parameters directly influence its performance and behavior. Tuning them helps optimize model accuracy and generalization, balancing complexity and computational cost.

Model Architecture Explanation

1. K-Nearest Neighbors (KNN)

- Architecture:

KNN is a non-parametric and lazy learning algorithm. It does not explicitly learn a model but stores the entire dataset during training. When making predictions, it calculates the distance between the input and every training point and identifies the K nearest neighbors to make a classification or regression decision. The model architecture is simple since it directly relies on data comparison, with no training phase but a computationally expensive inference phase.

2. K-Means Clustering

- Architecture:

K-Means follows an iterative architecture to partition the data into K clusters. It starts by initializing K centroids, then alternates between assigning each data point to the nearest centroid and recalculating the centroids based on the average of assigned points. This process repeats until centroids no longer change significantly. The architecture is unsupervised and relies on distance-based clustering.

3. Support Vector Machine (SVM)

- Architecture:

SVM works by finding an optimal hyperplane in high-dimensional space that separates data points of different classes with the maximum margin. It can use various kernels to map data into higher dimensions if linear separation isn’t possible in the original space. The

architecture is supervised and focuses on maximizing the margin between different classes. Support vectors are the critical data points that define the hyperplane.

4. Decision Tree

- Architecture:

Decision Tree has a hierarchical, tree-like architecture, where each node represents a decision based on a feature. Splits are made recursively to partition the data into homogeneous groups. The leaves of the tree represent the final classification or prediction. The architecture is supervised and greedy, as it tries to reduce impurity (e.g., Gini or Entropy) at each split.

5. Random Forest

- Architecture:

Random Forest is an ensemble of decision trees. It builds multiple decision trees using bootstrapped samples of the dataset and random subsets of features for splitting. Each tree makes its own prediction, and the final prediction is based on the majority vote (classification) or average (regression). This architecture enhances the performance of individual trees by reducing variance and overfitting, thanks to randomness.

6. Logistic Regression

- Architecture:

Logistic Regression is a linear model with a sigmoid activation function. It calculates the weighted sum of input features and applies the logistic (sigmoid) function to map the output to a probability between 0 and 1, allowing binary classification. The architecture is

supervised and assumes a linear relationship between input features and the log-odds of the target variable.

7. Gradient Boosting

- Architecture:

Gradient Boosting is an additive model that builds weak learners (typically decision trees) sequentially. Each new tree focuses on correcting the errors made by the previous trees by fitting the residuals (errors). The architecture uses a combination of models to reduce the overall prediction error and is driven by the principle of gradient descent optimization.

8. XGBoost

- Architecture:

XGBoost is an extension of Gradient Boosting with optimizations such as regularization to prevent overfitting, parallelized tree construction, and sparse matrix handling for high performance. It uses a similar architecture to Gradient Boosting but with improvements in speed and accuracy through advanced engineering optimizations, making it more scalable.

9. LightGBM

- Architecture:

LightGBM is a highly efficient gradient boosting algorithm. Its architecture focuses on leaf-wise growth rather than the traditional level-wise approach, which allows it to grow deeper trees with fewer splits. LightGBM is designed to handle large datasets and complex models efficiently, using techniques like histogram-based decision tree learning and reduced precision to save memory and improve speed.

10. AdaBoost

- Architecture:

AdaBoost combines multiple weak learners (often decision stumps) in an iterative boosting process. Each learner is trained to focus more on the data points misclassified by the previous one by assigning higher weights to those points. The final model is a weighted sum of all weak learners, where each contributes based on its accuracy. The architecture is supervised and aims to improve accuracy by emphasizing difficult-to-classify instances.

Summary:
- Lazy vs. Active Learners: KNN is a lazy learner that doesn't build a model until prediction, while others actively learn during training.
- Linear vs. Non-Linear Models: Logistic Regression and SVM (with linear kernel) are linear models, while Decision Trees, Random Forests, and boosting models capture non-linear relationships.
- Ensemble Methods: Random Forest, Gradient Boosting, AdaBoost, and XGBoost use ensembles of trees to improve accuracy and reduce overfitting.
- Boosting vs. Bagging: AdaBoost, Gradient Boosting, XGBoost, and LightGBM are boosting techniques focusing on correcting errors, while Random Forest is a bagging method reducing variance by aggregating independent trees.

Results

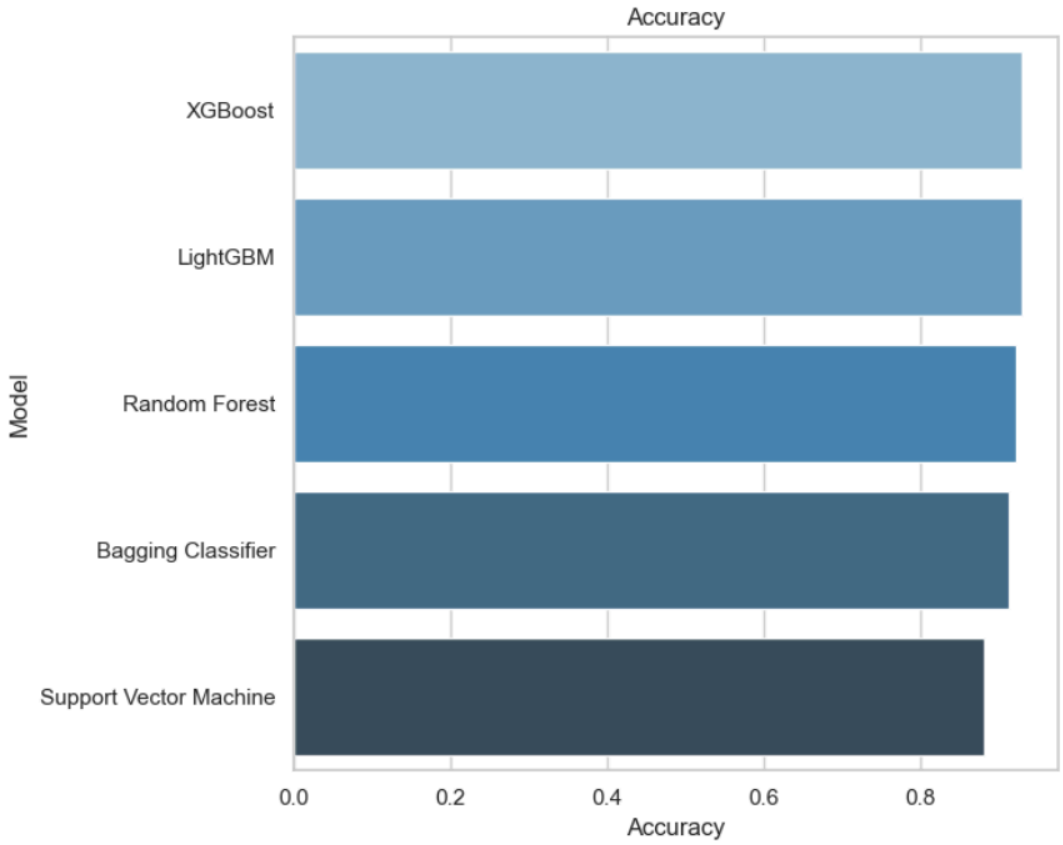
Evaluation: Summary of model performance:

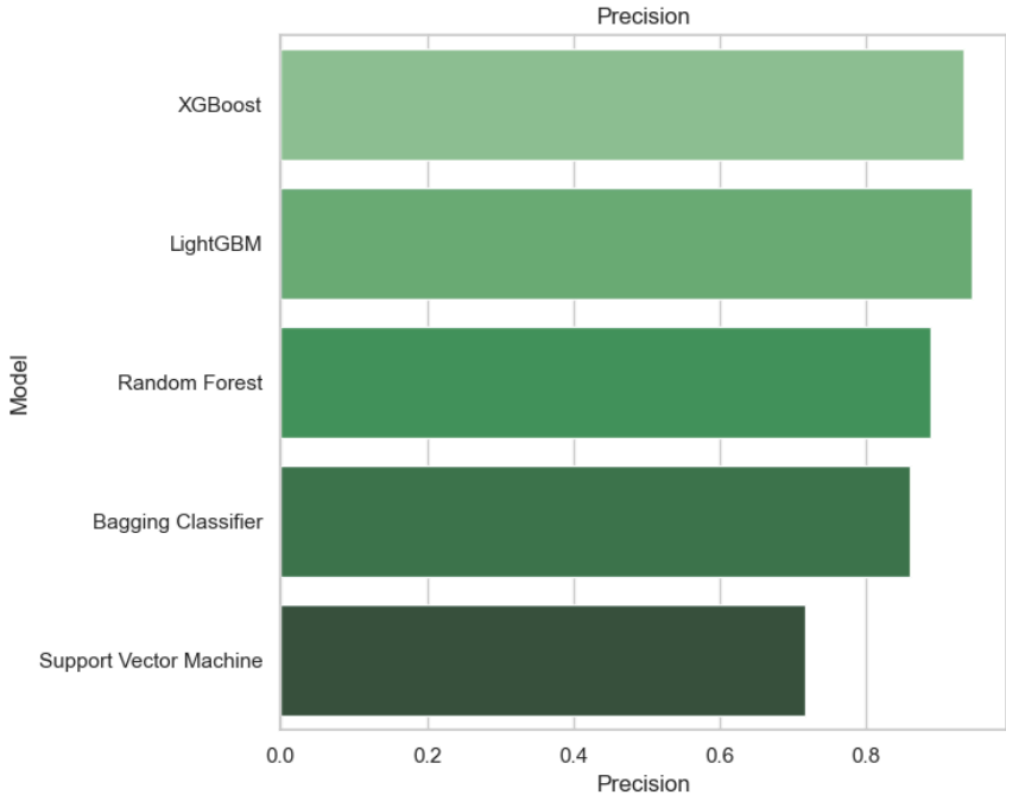
Evaluation Metrics:

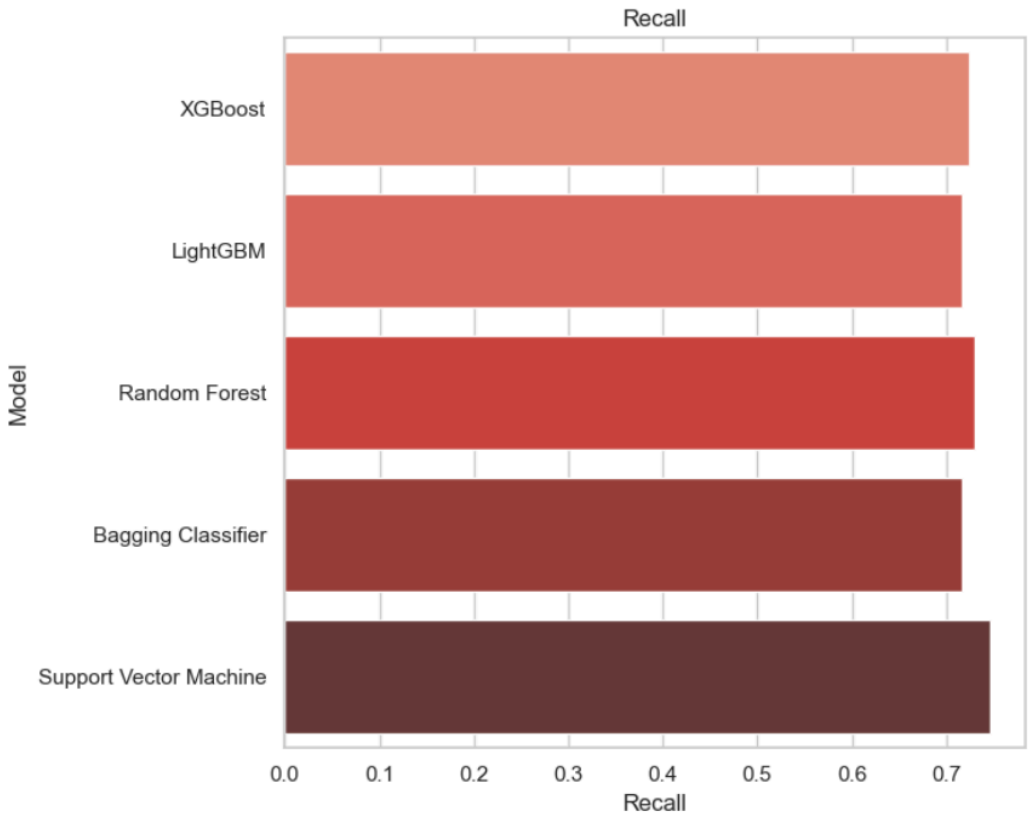
- **Accuracy:** The percentage of correctly predicted instances.
- **Precision:** The proportion of positive predictions that were correct.
- **Recall:** The proportion of actual positives that were identified correctly.
- **F1 Score:** The harmonic mean of precision and recall.
- **ROC AUC:** The area under the receiver operating characteristic curve.

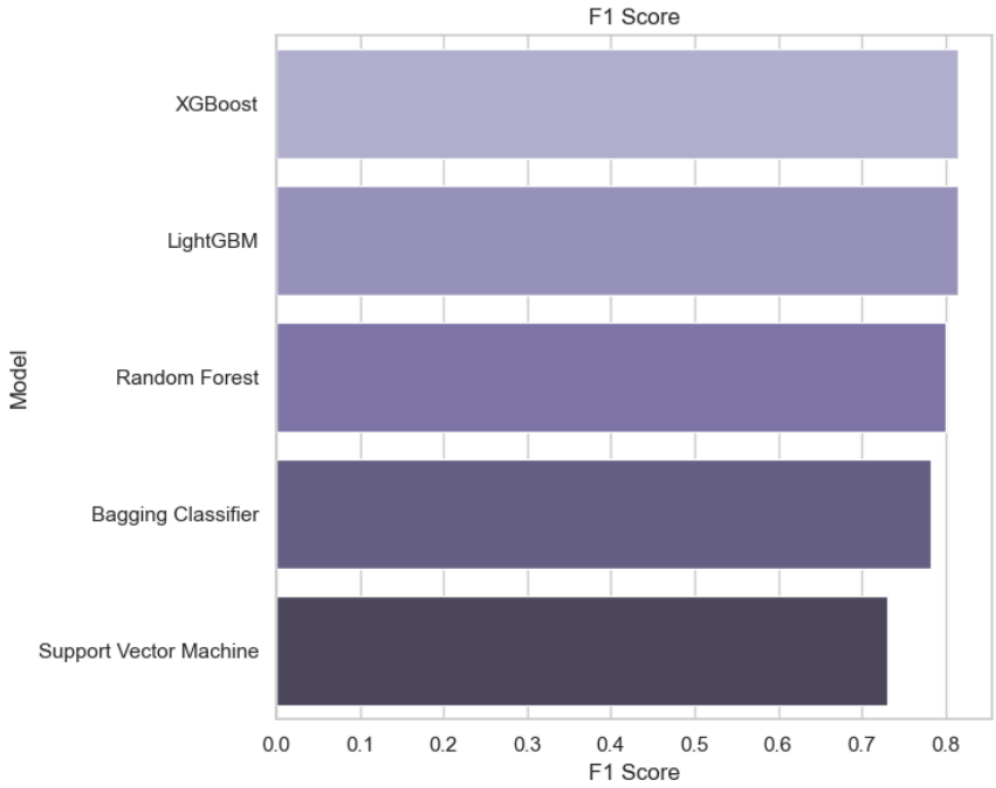
Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
Support Vector Machine	0.882	0.716	0.746	0.730	0.903
K-Nearest Neighbors	0.816	0.551	0.754	0.637	0.850
Decision Tree	0.867	0.670	0.750	0.708	0.825
Logistic Regression	0.813	0.545	0.760	0.635	0.862
Random Forest	0.922	0.888	0.730	0.801	0.923
AdaBoost	0.850	0.624	0.755	0.683	0.883
Bagging Classifier	0.914	0.861	0.717	0.782	0.901
Gradient Boosting	0.898	0.771	0.745	0.758	0.913
XGBoost	0.930	0.932	0.724	0.815	0.937
LightGBM	0.930	0.944	0.717	0.815	0.931

Result Visualization of top 5 models:


Accuracy
XGBoost
LightGBM
Random Forest
Bagging Classifier
Support Vector Machine
Model
0.0
0.2
0.4
0.6
0.8
Accuracy


Precision
XGBoost
LightGBM
Random Forest
Bagging Classifier
Support Vector Machine
Model
0.0
0.2
0.4
0.6
0.8
Precision


Recall
XGBoost
LightGBM
Random Forest
Bagging Classifier
Support Vector Machine
Model
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Recall



Discussion

In this project, we evaluated multiple machine learning models for credit risk classification. Each model demonstrated different strengths across performance metrics. XGBoost and LightGBM achieved the highest accuracy and ROC AUC scores, indicating superior predictive power. Random Forest and Bagging Classifier also performed well, offering balanced results in terms of precision and recall. In contrast, models like Logistic Regression and K-Nearest Neighbors showed lower precision, though their recall was relatively high.

The results highlight the importance of model selection based on specific use cases—where high recall is critical, certain models may outperform others despite lower precision. Overall, boosting algorithms like XGBoost and LightGBM consistently provided the best performance, making them suitable for this classification task.

Learning Outcome

1. Importance of Data Preprocessing

Preprocessing steps, such as handling missing values and encoding categorical features, significantly impact the performance of machine learning models.

2. Model Comparison and Evaluation

We learned that advanced models like XGBoost and LightGBM outperformed simpler models such as Logistic Regression and K-Nearest Neighbors in accuracy and overall predictive performance.

3. Impact of Model Selection on Business Problems

Different models prioritize precision or recall, which directly affects business decisions, such as minimizing false positives or false negatives in credit risk classification.

4. Model Performance Metrics

Evaluating models using metrics like accuracy, precision, recall, F1-score, and ROC AUC helped understand model performance from various perspectives, providing a comprehensive view of their strengths and weaknesses.

5. Boosting Algorithms

Boosting algorithms (XGBoost and LightGBM) demonstrated superior learning capabilities, making them ideal for classification tasks with complex patterns.

Conclusion

In this project, multiple machine learning models were compared to classify credit risk. The ensemble-based methods, specifically XGBoost and LightGBM, demonstrated superior performance in both accuracy and ROC AUC, making them the recommended models for this task. Future work can include hyperparameter tuning and feature engineering to further improve the performance of these models.

Accomplishments:

1. Successful Data Preparation and Cleaning

We handled missing values, converted categorical data, and prepared the dataset for machine learning, ensuring data consistency and reliability.

2. Implementation of Multiple Machine Learning Models

Successfully implemented and evaluated various machine learning models including SVM, Decision Tree, Random Forest, XGBoost, LightGBM, and others, providing a broad comparison of different approaches to credit risk classification.

3. Comprehensive Model Evaluation

We compared models using key metrics such as accuracy, precision, recall, F1-score, and ROC AUC,

gaining insights into their strengths and areas for improvement.

4. Identification of Best-Performing Models

XGBoost and LightGBM emerged as top performers, showcasing the value of boosting algorithms for high accuracy and predictive power in credit risk classification tasks.

5. Visualization of Model Performance

Generated clear visualizations to compare model performance across different metrics, enabling a better understanding of the results and aiding in decision-making for model selection.

.

Advantages/Limitations:

Advantages

1. Improved Prediction Accuracy

Advanced models like XGBoost and LightGBM provided high accuracy, offering reliable predictions for credit risk classification.

2. Diverse Model Evaluation

By evaluating multiple models, we gained insights into their varying strengths, allowing for informed model selection based on specific business needs.

3. Effective Use of Boosting Techniques

Boosting algorithms significantly improved performance by correcting the errors of weaker models, making them ideal for complex tasks.

4. Comprehensive Performance Metrics

Using multiple metrics (accuracy, precision, recall, etc.) provided a well-rounded assessment of model performance.

Disadvantages

1. High Computational Cost

Models like XGBoost and LightGBM are computationally intensive, requiring more resources and longer training times compared to simpler models.

2. Complexity in Tuning

Advanced models often require careful hyperparameter tuning to achieve optimal performance, which can be time-consuming.

3. Overfitting Risk

Some models, particularly Decision Trees and Random Forest, may overfit the data if not properly regularized, reducing their generalization capability.

Google collablink

https://colab.research.google.com/drive/1vpp9_oEJO3dQRSw5yNu69qknGpIv7pQt?usp=sharing

Github link

https://github.com/sridhar98765/ml-project