Loan Approval Prediction

By- Ameya Prashant Bahadure

1 Abstract-

For our project we have decided to experiment, design, and implement a loan prediction problem. We will use the loan prediction dataset to automate the loan eligibility process (real time) based on customer details provided like Education, Loan amount, Credit history, Income etc. Since we have to classify whether the loan will get approved or not so this is a Classification problem which comes under Supervised Machine Learning. The dataset we will be working on has 615 rows & 13 columns. We will need to preprocess the data, perform data cleaning & feature engineering and finally will be implementing models like - Logistic

Regression, Decision tree, Support Vector Machine to check the accuracy of each model. Our results showed that Support Vector Machine gave the most accurate predictions, which suggests it could be useful for real-world loan approval processes.

2. Introduction-

The problem statement goes like this - A Finance Company deals in all home loans. Its customers first apply for home loan after that company validates the customer eligibility for loan. Since the process is time taking and tedious, the company decide to automate the loan approval process using machine learning.

Our project focus is to use existing customer's details and analyze it further by applying machine learning techniques and algoritms and predict which future applicant can be approved for the loan.

Since process of approving loans is a critical part for any finance company as it directly affects their profitability. Approving loans for high-risk applicants can lead to financial losses, while rejecting loans for low-risk applicants can reduce potential profits. In this project, we aim to develop a predictive model for loan approvals to help streamline decision-making and reduce risks associated with lending.

The input to our algorithm consists of various attributes from applicants' profiles, such as income, credit score, marriage status, occupation etc By training three distinct machine learning models which are Support Vector Machine (SVM), Logistic Regression, and Decision Tree—we aim to output a prediction of whether each applicant's loan will likely be approved or denied. This will help the loan officer to sort the applicants based on the prediction in initial stage and help it to further evaluate them in later process.

---

3 Related work-

1. Loan Eligibility Prediction Using Logistics Regression Algorithm

This paper explores the potential of machine learning techniques, specifically Logistic Regression, to automate the loan eligibility prediction process. The goal is to streamline the loan application process, reduce manual effort, and improve decision-making accuracy.

Subsequently, a Logistic Regression model is trained on this data to predict loan eligibility. The model's performance is evaluated using metrics such as accuracy, precision, recall, and AUC-ROC.

Strenghts-

Practical Approach: The use of a real-world dataset and a common machine learning algorithm makes the research practical and applicable to industry settings.

Comprehensive Evaluation: The paper employs multiple evaluation metrics to assess the model's performance, providing a well-rounded assessment.

Detailed Explanation: The steps involved in data preparation, model training, and evaluation are explained in a clear and concise manner.

Weakness-

Limited Model Exploration: The paper focuses solely on Logistic Regression. Exploring other algorithms like Decision Trees, Random Forest, or XGBoost could provide insights into their relative performance.

Lack of Feature Importance Analysis: Identifying the most influential features could help understand the model's decision-making process and potentially improve its accuracy.

Absence of Hyperparameter Tuning: Optimizing hyperparameters can significantly impact model performance. The paper could discuss techniques like grid search or randomized search for hyperparameter tuning.

Link-: https://www.researchgate.net/publication/369143834

2. A Machine Learning Method for Predicting Loan Approval by Comparing the Random Forest and Decision Tree Algorithms.

This paper study compares Random Forest and Decision Tree algorithms for predicting loan approval, aiming to determine the more effective machine learning approach. Utilizing a loan prediction dataset from Kaggle, the research samples 20 cases split evenly between the two algorithms. Results show that the Random Forest algorithm achieves a precision of 79.45% and a loss rate of 21.03%, outperforming the Decision Tree's precision of 67.29% and loss of 32.71%. The study underscores the importance of reliable predictive models in reducing loan default risks for financial institutions.

Strengths-

Comparison of Algorithms: The research directly compares two popular machine learning algorithms, providing insights into their relative effectiveness for loan approval prediction.

Statistical Analysis: The study employs statistical tests to assess the significance of the observed differences in accuracy between the algorithms.

Real-World Application: The research explores a practical problem with significant implications for the financial sector.

Weaknesses of the Content-

Limited Dataset Exploration: The paper provides minimal details about the dataset used, including its size, features, and potential limitations.

Lack of Model Explainability: The research focuses on accuracy but doesn't explore how the algorithms arrive at their predictions. Understanding the underlying decision-making process could be valuable.

Limited Future Work Discussion: While the paper mentions the possibility of expanding the research with more advanced techniques, it could benefit from a more detailed exploration of potential future directions.

Link- https://sifisheriessciences.com/journal/index.php/journal/article/view/414/397

3. PREDICTING ACCEPTANCE OF THE BANK LOAN BY USING SUPPORT VECTOR MACHINES

This study investigates the prediction of customer acceptance of bank loan offers using the Support Vector Machine (SVM) algorithm. By evaluating four different SVM kernels and employing grid search for optimization and cross-validation for reliability, the research identifies the polynomial kernel as the most effective, achieving a high accuracy of 97.2%. In contrast, the sigmoid kernel performed the worst with an accuracy of 83.3%. The study highlights the challenge of an unbalanced dataset, with a ratio of 1 positive to 9 negative instances, which adversely affected precision (0.108) and recall (0.008) metrics. The findings underline the potential of SVM in enhancing loan acceptance predictions, thereby improving profitability for banks by enabling them to better identify trustworthy customers for personal loan offers. Ultimately, the study recommends integrating SVM into banking systems to enhance decision-making related to loan offers.

Strengths-

Exploration of Different SVM Techniques: The study investigates the performance of SVM with various kernel functions, providing insights into their effectiveness for this specific task.

Emphasis on Model Optimization: The paper highlights the use of grid search and cross-validation for optimizing the SVM model, which strengthens the research methodology.

Weakness-

Limited Dataset Discussion: The paper lacks details about the dataset used, including its size, features, and the potential presence of imbalances.

Lack of Evaluation Metrics: While accuracy is mentioned, a broader discussion on other relevant evaluation metrics like precision, recall, and F1-score is missing.

Limited Discussion on Unbalanced Data: The paper acknowledges the dataset imbalance but doesn't delve into its potential impact on model performance or explore potential mitigation strategies.

Link-
https://www.proquest.com/openview/f6b25e02ddf430fbbd0b1f6c994142d9/1?pqorigsite=gscholar&cbl=2028922

4. . Study comparing classification algorithms for loan approval predictability (Logistic Regression, XG boost, Random Forest, Decision Tree)- This study compares various machine learning algorithms for predicting loan approval, aiming to identify the most effective model for accuracy. Using a loan prediction dataset, the researchers tested Logistic Regression, Decision Tree, Random Forest, and XGBoost models. Results indicated that Logistic Regression achieved the highest accuracy (83.24%), while Decision Tree had the lowest (70.34%). The study found statistically significant differences ($p<0.05$) among the algorithms' performance, with Logistic Regression outperforming others for loan approval prediction. The research highlights the instability of Decision Trees, as minor changes in data can greatly affect model structure. Machine learning can automate loan approval processes, helping banks reduce risks and losses. The study addresses a research gap by demonstrating that more robust algorithms like Logistic Regression and Random Forest may be better suited for loan prediction than traditional Decision Trees, especially under conditions requiring high accuracy and model stability.

Strengths-

Comparison of Algorithms: The research directly compares four popular machine learning algorithms, offering insights into their relative effectiveness for loan approval prediction.

Methodology Details: The paper describes the research setting, hardware/software configuration, and the dataset source (Kaggle).

Weakness-

Limited Discussion on Dataset: While the paper mentions obtaining the dataset from Kaggle, it lacks details about the dataset size, features, and potential imbalances.

Lack of Evaluation Metrics: The focus seems to be solely on accuracy. A broader discussion on other relevant evaluation metrics like precision, recall, and F1-score is missing.

Limited Explanation of Model Selection: The paper doesn't delve into the rationale behind choosing the specific machine learning algorithms for comparison.

Link- https://sifisheriessciences.com/journal/index.php/journal/article/view/475/458

Comparison to Our Work-

Our project focus on using the classification algorithm logistic Regression, SVM and Decision tree to predict wether the loan will be approved or not. Our approach is similar to previous studies that have demonstrated the effectiveness of ensemble and hybrid methods. However, our focus on employing and comparing multiple machine learning algorithms on a real-world dataset distinguishes our work from prior research. We also have put focus more on EDA by preprocessing the data , dealing with null values and also scaling the dataset for optimizing the model.

---

4. Dataset and Features-

The dataset was taken from Kaggle the link is dataset

The dataset is split into training and testing set with training set having 40% or (245,12) and testing set having 60% or (369,12) ie X_train and X_test respectively.

Before splitting the dataset we have done preprocessing on it such as -finding the shape of the data, using the **describe()** function to get mean mode etc information on each features ,

```
[ ] loan_dataset.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

,finding the null values by using **isnull()**

function and then replacing it by **mean** for integer column and with **mode** for categorical columns by using **fillna()**

```
#now again using isnull() function to see if there is any null value left
loan_dataset.isnull().sum()
```

	0
Loan_ID	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	0
Property_Area	0
Loan_Status	0

dtype: int64

using encoding to convert all the categorical columns into integer type as model can be only feed integer or float value by using one-hot encoding, then removing the feature like **Loan_ID** as it has no use.

We also introduced a new feature called **Income_to_Loan_Ratio** which is a type of domain-specific feature creation technique.

After splitting the dataset we also perform the normalisation so that all the values are in same range.

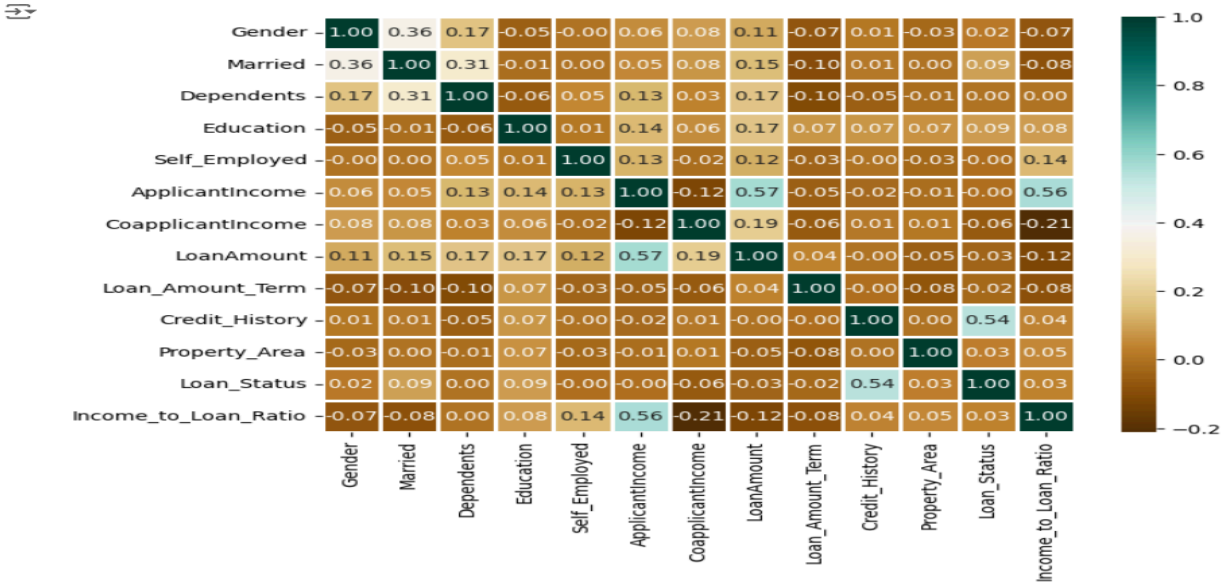
```
[59] # Initialize StandardScaler
     scaler = StandardScaler()

     # Fit and Transform the Features
     X_scaled = scaler.fit_transform(X)

     # Convert Scaled Features back to DataFrame for easier handling
     X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

     # Verify Scaling
     print(X_scaled.head())
```

```
     Gender   Married  Dependents  Education  Self_Employed  ApplicantIncome  \
0  0.472343 -1.372089   -0.682729   0.528362      -0.392601         0.072991
1  0.472343  0.728816    0.142459   0.528362      -0.392601        -0.134412
2  0.472343  0.728816   -0.682729   0.528362       2.547117        -0.393747
3  0.472343  0.728816   -0.682729  -1.892641      -0.392601        -0.462062
4  0.472343 -1.372089   -0.682729   0.528362      -0.392601         0.097728

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0          -0.554487   -0.211241          0.273231        0.411733
1          -0.038732   -0.211241          0.273231        0.411733
2          -0.554487   -0.948996          0.273231        0.411733
3           0.251980   -0.306435          0.273231        0.411733
4          -0.554487   -0.056551          0.273231        0.411733

   Property_Area  Income_to_Loan_Ratio
0       1.223298              0.187892
1      -1.318513             -0.104585
2       1.223298              0.171055
3       1.223298             -0.527179
4       1.223298              0.095876
```

The features used are Gender, Married, Dependents, Education, Self_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term, Credit history, Property_Area, Income_to_Loan_Ratio.

Out of all the features the most important feature was the **Credit history** which is directly proportional to loan approval.

Heat map –



---

5 Methods

In this section we will see how each ML algorithms that is used in this project works along with equations given to help you understand them more properly.

The algorithms used are Support Vector Machine (SVM) , Logistic Regression, Decision Tree.

Support Vector Machine-

Support Vector Machine (SVM) is a supervised learning algorithm primarily used for classification tasks. The core objective of SVM is to find the optimal hyperplane that best separates the data into distinct classes. Mathematically, SVM seeks to maximize the margin between the two classes, defined as the distance between the hyperplane and the nearest data points from each class, known as **support vectors**. The optimization problem can be formulated as:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad \forall i$$

where w is the weight vector, b is the bias term, xi represents the feature vectors, and yi are the class labels (+1 or −1). For cases where the data is not linearly separable, SVM employs kernel functions (e.g., radial basis function) to project the data into a higher-dimensional space, enabling the creation of a separating hyperplane in that space. This flexibility makes SVM a powerful tool for both linear and non-linear classification problems.

Here is the simple analogy: Imagine you have two groups of items on a table, and you want to draw a line between them so they're separated as clearly as possible. SVM is like an algorithm that finds the best line (or boundary) to divide these groups. It looks for the line that leaves the biggest gap, or "margin," between the closest items from each group. If the items are hard to separate, SVM can add extra "dimensions" to make it easier to find a clear dividing line. This approach helps SVM make reliable predictions when sorting things into two categories.

Logistic Regression-

Logistic Regression is a fundamental supervised learning algorithm used for binary classification tasks. Unlike linear regression, which predicts continuous outcomes, Logistic Regression estimates the probability that a given input belongs to a particular class. The model applies the **logistic (sigmoid) function** to a linear combination of input features to constrain the output between 0 and 1:

$$P(y=1|\mathbf{x}) = \sigma(\mathbf{w}\cdot\mathbf{x}+b) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

where N is the number of samples, yi are the true labels, and y^i are the predicted probabilities. Logistic Regression is valued for its simplicity, interpretability, and effectiveness, especially when the relationship between the features and the target is approximately linear.

Analogy- Logistic Regression is used to figure out if something belongs to one of two groups. It takes different characteristics (or features) of an item, like age or income for a loan applicant, and calculates a "score" for it. Then it uses a formula to convert this score into a probability—a number between 0 and 1. If the probability is above a certain threshold (like 0.5), the algorithm classifies it into one group; if it's below, it goes into the other. It's popular because it's straightforward and works well when the data can be separated by a line.

Decision tree-

Decision Tree is a versatile supervised learning algorithm capable of performing both classification and regression tasks. It operates by recursively partitioning the feature space into subsets based on feature values, creating a tree-like structure of decisions. Each internal node represents a **decision rule** on a specific feature, while each leaf node corresponds to a class label or a continuous value. The algorithm selects the feature and threshold that result in the highest **information gain** or the greatest reduction in **impurity**, measured by metrics such as **Gini impurity** or **entropy**.

For example, the Gini impurity for a node is defined as:

$$Gini = 1 - \sum_{k=1}^{K} p_k^2$$

where pk is the probability of class k in the node. The tree-building process continues until a stopping criterion is met, such as a maximum depth or a minimum number of samples per leaf. Decision Trees are highly interpretable, as the decision paths can be visualized and easily

understood, making them suitable for scenarios where model transparency is crucial. However, they can be prone to overfitting, which is often mitigated through techniques like pruning or by using ensemble methods.

Analogy- A Decision Tree is like a flowchart for making decisions. Starting from the top, it asks simple yes-or-no questions about each item, like "Is the income above $50,000?" or "Is the credit score high?" Each question splits the data further until each path leads to a final decision, like "Approve" or "Deny" for a loan application. The tree is easy to understand because it follows logical steps, and each question brings you closer to a decision. Decision Trees are helpful for understanding how different factors lead to different outcomes.

---

6 Experiments/Results/Discussion

In this section we will see the result of each model about its accuracy, and other evaluation metrics to come to a conclusion that which model is best for Loan Prediction.

AUC curves were generated for each model to evaluate their performance across different classification thresholds. illustrating the trade-off between the true positive rate and false positive rate. Higher AUC values signify better performance

The primary metrics for evaluation are accuracy,recall score,precision score,F1 score and AUC to get better understanding of the model.

Accuracy: This metric represents the proportion of correctly predicted instances (both true positives and true negatives) among the total number of observations. Accuracy is calculated using the formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where:

- TP = True Positives (the number of correct positive predictions)
- TN = True Negatives (the number of correct negative predictions)
- FP = False Positives (the number of incorrect positive predictions)
- FN = False Negatives (the number of incorrect negative predictions)

High accuracy indicates that the model correctly predicts a large proportion of the total cases.

Precision: This metric measures the accuracy of the positive predictions made by the model. It is calculated as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

A high precision value indicates that a significant proportion of the predicted positive instances are actually positive, which is critical in scenarios where false positives can lead to unnecessary actions or costs.

Recall (Sensitivity): Recall measures the ability of the model to identify all relevant instances. It is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

A high recall value indicates that the model successfully identifies a large number of actual positive cases, which is especially important in customer churn scenarios where retaining customers is essential.

F1 Score: This is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as follows:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 score is particularly useful when dealing with imbalanced datasets, as it accounts for both false positives and false negatives.

Area Under the Receiver Operating Characteristic Curve (AUC-ROC): This metric evaluates the model's ability to distinguish between classes across different threshold settings. A higher AUC indicates a better capability of the model to classify positive and negative instances correctly.

Support Vector Machine (SVM)-

The parameters choose for the model were the default parameters as it provided best accuracy after multiple try with different parameters.

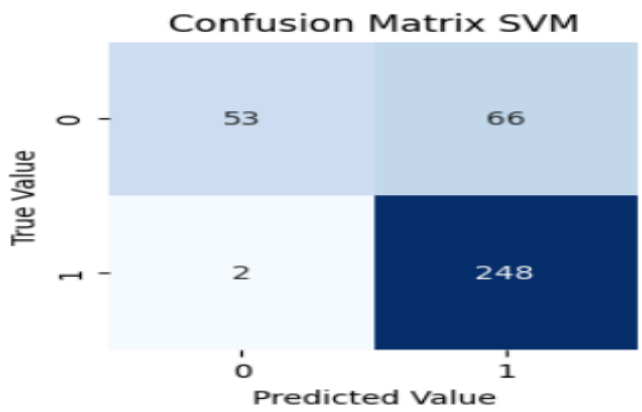
```
#train the model using the training dataset
# Initialize SVM with default parameters
svm_classifier = SVC(kernel='rbf', random_state=42)
svm_classifier.fit(X_train, y_train)
```

```
SVC
SVC(random_state=42)
```

The RBF(Radial Basis Function) kernel allows SVM to create flexible, curved decision boundaries in the original feature space as the dataset was non-linear and complex. This is particularly beneficial for data that isn't easily separable by a straight line, as it can help to achieve better classification performance.

By fixing the random_state your results become reproducible. This is especially important when we want to compare model performances across different algorithms or parameters, as consistent randomness allows for a fair comparison. The number 42 provided a decent help to have a better accuracy. **No hyperparameter tuning** was done for SVM as the accuracy was degraded.

```
#plotting the confusion matrix
cm = confusion_matrix(y_test, svm_classifier.predict(X_test))

# Plot the confusion matrix
plt.figure(figsize=(3, 3))
sb.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix SVM')
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



TP=248, TN=53, FP=2, FN=66

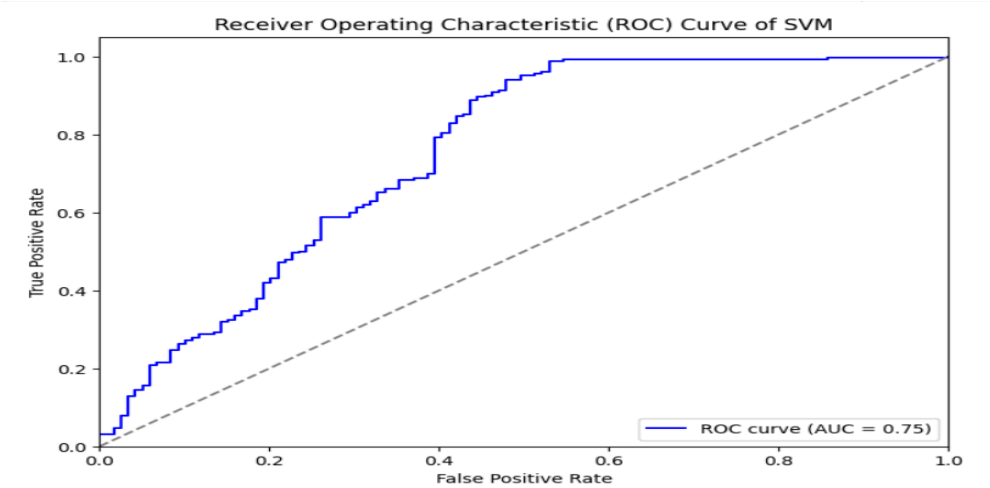
```
#calculating the accuracy,recall,precision and F1 score by using the score()method
from sklearn.metrics import accuracy_score, recall_score, precision_score,f1_score,classification_report, confusion_matrix
print("The initial performance is:")
print("the accuracy is:", accuracy_score(y_test,y_pred))
print("the recall score is:", recall_score(y_test,y_pred))
print("the precision score is:", precision_score(y_test,y_pred))
print("the F1 score is:", f1_score(y_test,y_pred))
```

```
The initial performance is:
the accuracy is: 0.8157181571815718
the recall score is: 0.992
the precision score is: 0.7898089171974523
the F1 score is: 0.8794326241134752
```

Accuracy = 81.57%

ploting the ROC and AUC using the **roc_curve** and **auc functions** from sklearn.metrics

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Generate predicted probabilities for the positive class (class 1)
Y_pred_prob = svm_classifier.decision_function(X_test)  # For SVM, use decision_function

# Calculate FPR, TPR, and threshold values
fpr, tpr, thresholds = roc_curve(y_test, Y_pred_prob)

# Calculate AUC score
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line (no skill)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of SVM')
plt.legend(loc="lower right")
plt.show()
```



An AUC score of 0.75 indicates that the model performs moderately well in distinguishing between the approved and non-approved classes.

An AUC above 0.75 suggests that the model has some predictive capability but is not highly accurate or optimal.

This score suggests that the model is better than random guessing but still has room for improvement. ROC Curve Shape:

The ROC curve's shape shows a gradual rise, indicating that the model has an acceptable balance between sensitivity (True Positive Rate) and specificity (False Positive Rate) but could be further optimized to improve performance.

The model does not undergoes overfitting and underfitting which can be seen based on the train and test score

now predicting the score and evaluating it in initial

```
[76] #score of training dataset
     svm_classifier.score(X_train,y_train)

     0.8285714285714286
```

the accuracy of training dataset is 82.85%

```
[77] #score of testing dataset
     svm_classifier.score(X_test,y_test)

     0.8157181571815718
```

the accuracy of testing dataset is 81.57%

Logistic Regression-

The parameters used for the logistic regression model are the default ones as it give us the good accuracy

penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='deprecated', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

penalty='l2':

This applies L2 regularization, which helps the model avoid overfitting by adding a penalty for larger coefficients.

dual=False:

This affects how the model solves the optimization problem. dual=False is typically best for situations where the number of samples is greater than the number of features.

tol=0.0001:

This is the tolerance for stopping the optimization. The model will stop trying to improve once it gets this close to the best solution, saving time if improvements become negligible.

C=1.0:

This is the inverse of regularization strength. A lower C means more regularization, while a higher C means less. C=1.0 is the default balance, avoiding too much regularization without allowing overfitting.

fit_intercept=True:

This tells the model to add an intercept term, which adjusts the decision boundary rather than forcing it through the origin.

intercept_scaling=1:

This scales the intercept, which only has an effect if using liblinear as the solver. Since you're using lbfgs, it doesn't affect your model.

class_weight=None:

This lets each class have equal weight in the model. If set to 'balanced', it would adjust the weights based on class frequency, which is useful for imbalanced datasets.

random_state=None:

This keeps the randomness consistent if a specific integer is used (e.g., random_state=42). With None, the randomness changes with each run, so results may vary slightly each time.

solver='lbfgs':

This is the algorithm used to optimize the model's parameters. 'lbfgs' is efficient for smaller datasets or models with many classes.

max_iter=100:

Sets the maximum number of iterations the solver will use to try and improve the model. If it reaches 100 without finding an optimal solution, it stops.

multi_class='deprecated':

This setting used to control how multi-class problems were handled. It is now deprecated, and for models with more than two classes, auto selects the best method.

verbose=0:

Controls the amount of logging/output. 0 means silent, while higher numbers show more details as it trains.

l1_ratio=None:

This parameter is used only if penalty='elasticnet' to balance L1 and L2 regularization. Since you're using l2, it has no effect here.

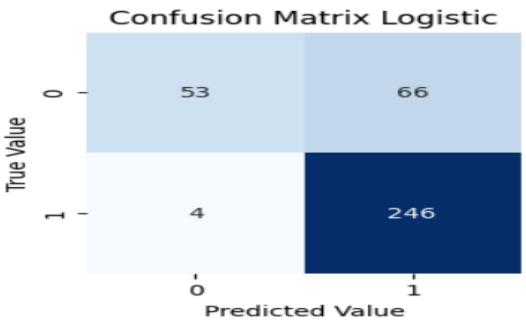
```
#importing the nessccesary modules
from sklearn.linear_model import LogisticRegression
```

```
#since dataset is already split into traning and testing

#train the model using the training dataset

# Initialize LogisticRegression with default parametrs
logistic_classifier = LogisticRegression(penalty='l2', dual=False, tol=0.0001,
                                         C=1.0, fit_intercept=True, intercept_scaling=1,
                                         class_weight=None, random_state=None, solver='lbfgs', max_iter=100,
                                         multi_class='deprecated', verbose=0, warm_start=False, n_jobs=None,
                                         l1_ratio=None)

# Train the model
logistic_classifier.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

No hyperparameter tuning was done.

```python
#plotting the confusion matrix
cm1 = confusion_matrix(y_test, logistic_classifier.predict(X_test))

# Plot the confusion matrix
plt.figure(figsize=(3, 3))
sb.heatmap(cm1, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix Logistic')
plt.xlabel('Predicted Value')
plt.ylabel('True Value')
plt.show()
```



TP=246, TN=53, FP= 66, FN=4

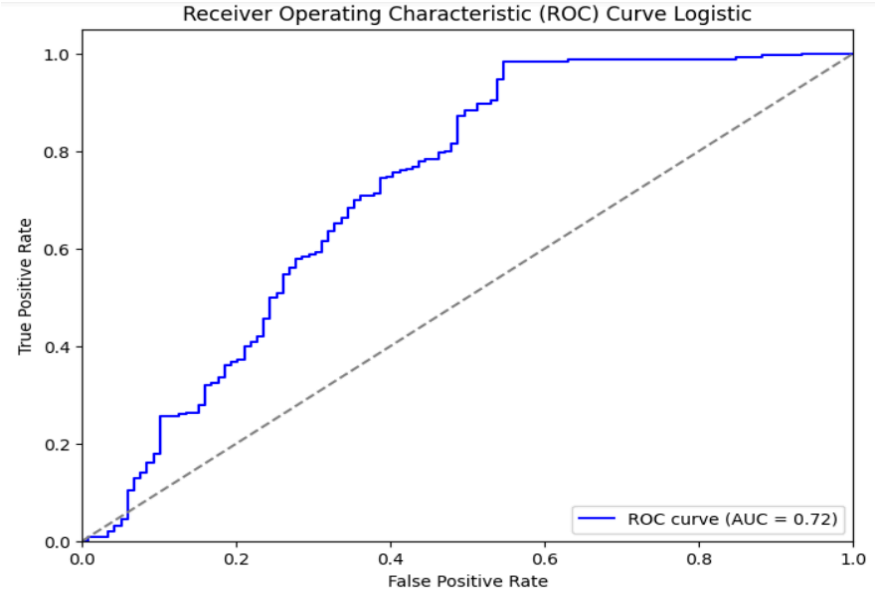
```python
#calculating the accuracy,recall,precision and F1 score by using the score()method
from sklearn.metrics import accuracy_score, recall_score, precision_score,f1_score,classification_report, confusion_matrix
print("The initial performance is:")
print("the accuracy is:", accuracy_score(y_test,y_pred1))
print("the recall score is:", recall_score(y_test,y_pred1))
print("the precision score is:", precision_score(y_test,y_pred1))
print("the F1 score is:", f1_score(y_test,y_pred1))
```

```
The initial performance is:
the accuracy is: 0.8102981029810298
the recall score is: 0.984
the precision score is: 0.7884615384615384
the F1 score is: 0.8754448398576512
```

Accuracy is 81.03%

as we can see the accuracy and other parametrs are improved

```python
#plotting AUC
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Generate predicted probabilities for the positive class (class 1)
Y_pred_prob1 = logistic_classifier.predict_proba(X_test)[:, 1]  # Select only the second column# For SVM, use decision_function

# Calculate FPR, TPR, and threshold values
fpr, tpr, thresholds = roc_curve(y_test, Y_pred_prob1)

# Calculate AUC score
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line (no skill)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



The model does not undergoes overfit or underfit

now predicting the score and evaluating it

```
[50] logistic_classifier.score(X_train,y_train)
0s
0.8040816326530612
```

the accuracy of training data is 80.40%

```
logistic_classifier.score(X_test,y_test)
0s
0.8102981029810298
```

the accuracy of testing data is 81.03%

Decision Tree-

```
# Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
                                       min_samples_leaf=1, min_weight_fraction_leaf=0.0,
                                       max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)

# Train the Model
dt_classifier.fit(X_train1, y_train1)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

criterion ('gini'):

- This is the method used to decide how the tree splits data at each step.

- 'gini' helps make splits that reduce the chance of errors in classification. 'entropy' is another option that tries to maximize information gained with each split.

splitter ('best'):

- This determines how the algorithm chooses splits at each decision point.
- 'best' means it finds the most optimal split, while 'random' picks splits randomly, which can be faster but less accurate.

max_depth (None):

- Sets how deep the tree is allowed to grow.
- By default, it grows until each group is pure or too small to split, which can lead to overfitting. Limiting depth keeps the tree simpler and less likely to overfit.

min_samples_split (2):

- The minimum number of samples needed to split a node.
- If you increase this, nodes won't split if they don't have enough samples, which helps prevent overfitting.

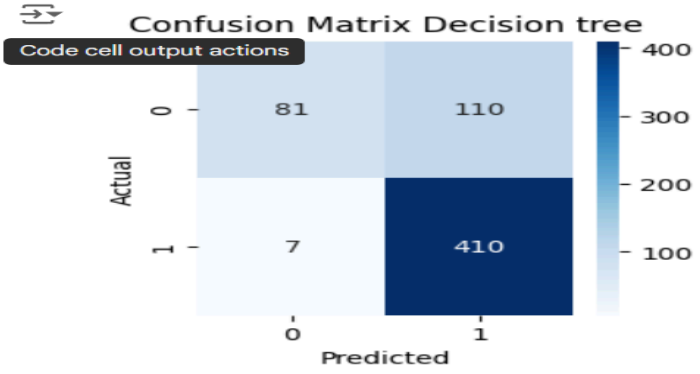
min_samples_leaf (1):

- The minimum number of samples needed to create a new branch in the tree.
- Higher values mean that each branch will be based on more data, which can make the tree less sensitive to noise.

No hyperparameter tuning was done.

```
cm3 = confusion_matrix(y_test1, y_pred2)

# Visualize Confusion Matrix
plt.figure(figsize=(3, 3))
sb.heatmap(cm3, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix Decision tree')
plt.show()
```

Code cell output actions

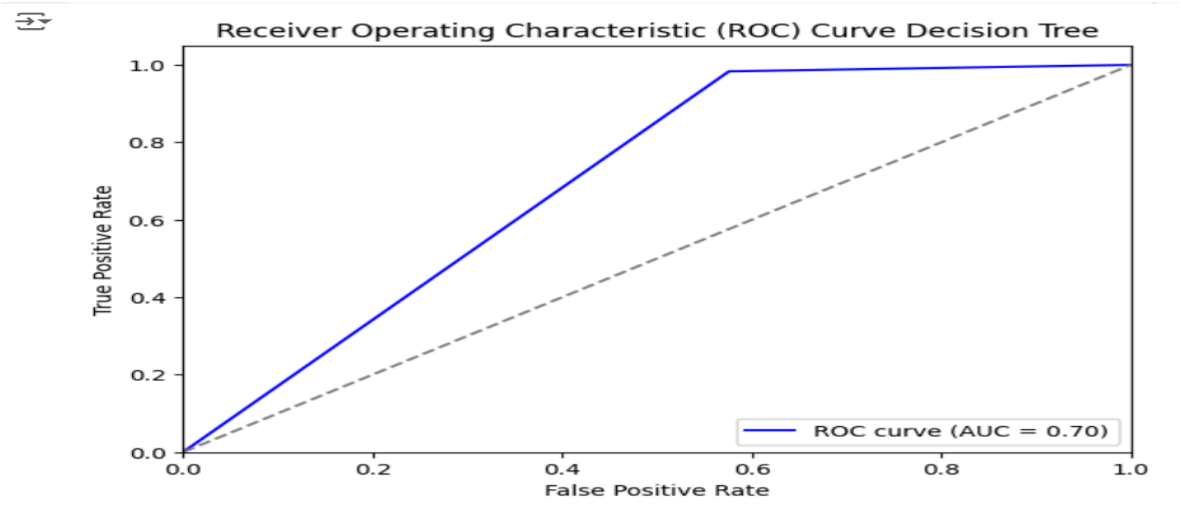


```
[361] #calculating the accuracy,recall,precision and F1 score by using the score()method
      from sklearn.metrics import accuracy_score, recall_score, precision_score,f1_score,classification_report, confusion_matrix

      print("The initial performance is:")
      print("the accuracy is:", accuracy_score(y_test1,y_pred2))
      print("the recall score is:", recall_score(y_test1,y_pred2))
      print("the precision score is:", precision_score(y_test1,y_pred2))
      print("the F1 score is:", f1_score(y_test1,y_pred2))
```

```
The initial performance is:
the accuracy is: 0.8075657894736842
the recall score is: 0.9832134292565947
the precision score is: 0.7884615384615384
the F1 score is: 0.8751334044823906
```

Accuracy = 80.75%

```
#plotting AUC
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Generate predicted probabilities for the positive class (class 1)
Y_pred_prob2 = dt_classifier.predict_proba(X_test1)[:, 1]  # Select only the second column as it needs only 1D array

# Calculate FPR, TPR, and threshold values
fpr, tpr, thresholds = roc_curve(y_test1, Y_pred_prob2)

# Calculate AUC score
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line (no skill)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve Decision Tree')
plt.legend(loc="lower right")
plt.show()
```



An AUC of 0.70 tells us that the model has a good, though not exceptional, ability to distinguish between approved and rejected loans.A curve closer to the top-left corner indicates

a better model.Here, the curve does not hug the top-left corner closely, indicating some room for improvement in model performance.

Interpretation of the Curve Shape:

The model's performance is decent, indicating a reasonably good classifier. However, it may not be fully optimized and could benefit from further improvement, such as feature selection, hyperparameter tuning, or possibly using a different model to achieve a higher AUC score.

The model does not undergoes overfit or underfit

```
#testing data score
dt_classifier.score(X_test1,y_test1)
```

```
0.8075657894736842
```

```
[379] #training data score
dt_classifier.score(X_train1,y_train1)
```

```
1.0
```

Model	Accuracy (%)	Recall	Precision	F1 Score	AUC Score
SVM	81.57	0.992	0.7898	0.8794	0.75
Logistic Regression	81.03	0.984	0.7885	0.8754	0.72
Decision Tree	80.76	0.9832	0.7885	0.8751	0.7

7. Conclusion/Future Work

Based on accuracy, F1 score, and AUC score, **SVM** is the best-performing model overall. It has the highest scores in most metrics, making it the top choice among the three. Credit History,Education,Applicant Income and Co-applicant Income, Property Area are the most important factors for predicting the class of the loan applicant (whether the applicant would be‘approved’ or ‘not’). In near future this module of prediction can be integrated with the module of automated processing system. The system is trained on old training dataset, in future software can be made such that new testing data can be used after certain time

Future Scope

Time Series Analysis can be done using the Loan data of several years, for prediction of the approximate time, when the client can default. Future analysis can be done on predicting the approximate Interest rates that the loan applicant is expected to be charged as per his profile, if his loan is approved. This can be useful for loan applicants, since some banks approve loans,

but give very high interest rates to the customer. An app with proper UI can be built, which will take various inputs from the user like name, address, loan amount, loan duration, etc. and give a prediction of whether their loan application can be approved by the banks or not based on their inputs along with an approximate interest rates.

References/Bibliography

Kaggle dataset for the project-

https://www.kaggle.com/datasets/ninzaami/loan-predication?resource=download

journals links-

'Loan Eligibility Prediction Using Logistics Regression Algorithm' by Cornelius Mellino Sarungu

link: https://www.researchgate.net/publication/369143834

P. Bhargav, K. Sashirekha report on 'A Machine Learning Method for Predicting Loan Approval by Comparing the Random Forest and Decision Tree Algorithms'

link: https://sifisheriessciences.com/journal/index.php/journal/article/view/414/397

'PREDICTING ACCEPTANCE OF THE BANK LOAN BY USING SUPPORT VECTOR MACHINES' by Ali, Syed Imran , Talhai, Shaik , Yawer, Shaik , Reddy, Nagi

Link -https://www.proquest.com/openview/f6b25e02ddf430fbbd0b1f6c994142d9/1?pqorigsite=gscholar&cbl=2028922

Study comparing classification algorithms for loan approval predictability (Logistic Regression, XG boost, Random Forest, Decision Tree)

Link- https://sifisheriessciences.com/journal/index.php/journal/article/view/475/458

Libraries used-

Scikit-learn developers, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011. [Online]. Available: http://scikit-learn.org/

Seaborn library: https://seaborn.pydata.org/

Matplot library: https://matplotlib.org/stable/plot_types/stats/boxplot_plot.html

Pandas library: https://pandas.pydata.org/

Numpy library : https://numpy.org/

Imbalanced learn library: https://imbalanced-learn.org/stable/