CS361: Loan Approval Predictor

Team Name: India
Piyush Tayal(210101076) Pratham Goyal(210101080) Priyanshu Raj(210101083) Nitya Parikh(210101073)

Abstract

This project employs Logistic Regression, Gaussian Naive Bayes, Decision Tree and Random Forest algorithms to predict loan approval status based on diverse applicant attributes. The dataset encompasses features like age, marital status, income, and job experience. The objective is to optimize loan approval processes, providing a tool for efficient decision-making while minimizing biases. Thorough data exploration, preprocessing, and model training, this approach contributes to the evolution of data-driven lending practices, enhancing the precision and robustness of loan approval predictions.

1. Introduction

The target problem of this project is to develop a robust loan approval prediction system. Leveraging machine learning algorithms, we aim to analyze a dataset containing various applicant attributes to predict whether a loan application is likely to be approved or denied. This predictive model holds significant value for financial institutions by streamlining the decision-making process and improving efficiency in assessing loan applications.
The dataset utilized in this project comprises 252,000 instances and encompasses 13 columns, each representing different features pertinent to loan approval decisions.

1.1. Dataset Features

- **Id**: Unique identifier for each record.
- **Income**: The income of the loan applicant.
- **Age**: Age of the loan applicant.
- **Experience**: Years of professional experience.
- **Married/Single**: Marital status of the applicant.
- **House_Ownership**: Indicates whether the applicant owns a house or not.
- **Car_Ownership**: Indicates whether the applicant owns a car or not.
- **Profession**: Occupation or profession of the applicant.
- **CITY**: City of residence of the applicant.
- **STATE**: State of residence of the applicant.
- **CURRENT_JOB_YRS**: Years of employment in the current job.
- **CURRENT_HOUSE_YRS**: Years of residency in the current house.
- **Risk_Flag**: Binary indicator (0 or 1) denoting whether the applicant is considered high-risk (1) or not (0) for loan approval.

1.2. Data Type Summary

The dataset consists predominantly of integer and object data types, with the 'Risk_Flag' column representing the target variable for prediction. The categorical variables such as 'Married/Single', 'House_Ownership', 'Car_Ownership', 'Profession', 'CITY', and 'STATE' will require appropriate encoding or transformation for integration into the machine learning model.

1.3. Challenges and Considerations

1.3.1. Data Quality and Availability

The quality and completeness of the dataset can significantly impact the performance of the predictive model. Challenges may arise from missing values, outliers, or inaccuracies in the data. Additionally, the availability of historical loan data for training the model could be limited, especially for newer financial products or markets.

1.3.2. Feature Selection and Engineering

Identifying relevant features for predicting loan approval can be challenging, especially when dealing with a large number of potential variables. Careful feature selection and engineering are necessary to ensure that the model captures meaningful patterns in the data while avoiding overfitting.

1.3.3. Imbalanced Data

Imbalanced datasets, where one class (e.g., approved loans) is significantly more prevalent than the other class (e.g., denied loans), can pose challenges for training machine learning models. Strategies such as resampling techniques, class weights, or specialized algorithms may be needed to address class imbalance and prevent bias in the predictions.

2. Methods

In this project we are planning to use different supervised learning methods to predict loan approval. All these methods are classification methods.

2.1. Algorithms

2.1.1. Logistic Regression

Logistic regression is a statistical method used for binary classification, which models the probability of a binary outcome (such as approval or rejection of a loan application) based on one or more predictor variables, applying a logistic function to predict the probability of the outcome.Training logistic regression models is computationally efficient, making it suitable for large datasets.

2.1.2. Decision Tree

The decision tree algorithm recursively splits the dataset based on features to create a tree structure where leaf nodes represent the final class labels or regression values.Decision trees can capture nonlinear relationships between features and the target variable, making them suitable for complex classification tasks.

2.1.3. Gaussian Naive Bayes

Gaussian Naive Bayes assumes that features within each class are normally distributed and predicts the class label based on the highest posterior probability calculated using Gaussian probability density functions.Naive Bayes works well with categorical input variables and is particularly effective for text classification tasks.

2.1.4. Random Forest

Random Forest is an ensemble learning technique that constructs multiple decision trees during training. Each tree is trained on a random subset of the data and features, reducing overfitting. Predictions are made by aggregating the outputs of individual trees. Random forests generally have high accuracy compared to individual decision trees, as they reduce overfitting by averaging the predictions of multiple trees.

3. Algorithms

Algorithm 1 Naive Bayes Algorithm

```
Input: Training data $(X_{train}, y_{train})$, Test data $X_{test}$
Output: Predicted labels for test data $y_{pred}$
Initialize class probabilities $P(C)$ and feature probabilities $P(X_i|C)$ for each class $C$
Training:
Calculate class probabilities $P(C)$ based on the frequency of each class in $y_{train}$
for each class $C$ do
    Calculate mean and standard deviation for each feature per class: $\mu_{C,i}$ and $\sigma_{C,i}$
end for
Testing:
for each sample $x_{test}$ in $X_{test}$ do
    for each class $C$ do
        Calculate class probability $P(C|x_{test})$ using Bayes' theorem
        Calculate feature probabilities $P(x_{test}|C)$ using Gaussian distribution
        Compute class score $P(C|x_{test}) \cdot \prod_{i=1}^{n} P(x_{test,i}|C)$
    end for
    Assign the class with the highest score to $y_{pred}$
end for
```

Algorithm 2 Logistic Regression Algorithm

```
Input: Training data $(X_{train}, y_{train})$, Test data $X_{test}$
Output: Predicted labels for test data $y_{pred}$
Initialize parameters: learning rate $\alpha$, number of iterations $n_{iters}$
Initialize weights $\mathbf{w}$ and bias $b$ to zeros
Initialize list to store losses
Training:
for $i = 1$ to $n_{iters}$ do
    Compute forward pass: $\mathbf{z} = \mathbf{X} \cdot \mathbf{w} + b$, $\mathbf{A} = \text{sigmoid}(\mathbf{z})$
    Compute loss: $loss = -\frac{1}{m} \sum_{i=1}^{m} (y_i \log(\mathbf{A}) + (1 - y_i) \log(1 - \mathbf{A}))$
    Compute backward pass: $\mathbf{dz} = \mathbf{A} - \mathbf{y}$, $\mathbf{dw} = \frac{1}{m} \mathbf{X}^T \cdot \mathbf{dz}$, $db = \frac{1}{m} \sum_{i=1}^{m} \mathbf{dz}$
    Update parameters: $\mathbf{w} = \mathbf{w} - \alpha \cdot \mathbf{dw}$, $b = b - \alpha \cdot db$
    Append loss to list
end for
Testing:
Compute predictions: $\hat{\mathbf{y}} = \text{sigmoid}(\mathbf{X_{test}} \cdot \mathbf{w} + b)$
Threshold predictions: $y_{pred} = 1$ if $\hat{y}_i > 0.5$ else 0
```

Algorithm 3 Decision Tree Classifier

```
class DecisionTreeClassifier:
function __init__(min_samples_split=2, max_depth=2):
set self.root to None
set self.min_samples_split to min_samples_split
set self.max_depth to max_depth
function fit(X, Y):
dataset = concatenate X and Y
self.root = build_tree(dataset)
function build_tree(dataset, curr_depth=0):
X, Y = extract features and labels from dataset
num_samples, num_features = shape of X
if num_samples $\geq$ self.min_samples_split and curr_depth $\leq$ self.max_depth: then
    best_split = get_best_split(dataset, num_samples, num_features)
    if best_split["info_gain"] $>$ 0: then
        left_subtree = build_tree(best_split["dataset_left"], curr_depth + 1)
        right_subtree = build_tree(best_split["dataset_right"], curr_depth + 1) Node(best_split["feature_index"], best_split["threshold"], left_subtree, right_subtree, best_split["info_gain"])
    end if
end if
leaf_value = calculate_leaf_value(Y) Node(value=leaf_value)
function get_best_split(dataset, num_samples, num_features):
best_split = {}
max_info_gain = -inf
for feature_index in range(num_features): do
    feature_values = get values of feature_index from dataset
    possible_thresholds = unique values of feature_values
    for threshold in possible_thresholds: do
        dataset_left, dataset_right = split(dataset, feature_index, threshold)
        if len(dataset_left) $>$ 0 and len(dataset_right) $>$ 0: then
            y, left_y, right_y = labels from dataset and splits
            curr_info_gain = information_gain(y, left_y, right_y, "gini")
            if curr_info_gain $>$ max_info_gain: then
                update best_split and max_info_gain
            end if
        end if
    end for
end for
best_split
function split(dataset, feature_index, threshold): dataset_left = rows in dataset where feature_index $\leq$ threshold dataset_right = rows in dataset where feature_index $>$ threshold dataset_left, dataset_right
function information_gain(parent, l_child, r_child, mode="entropy"): calculate weights for left and right child mode == "gini": gain = gini_index(parent) - (weighted sum of gini indices for left and right child) : gain = entropy(parent) - (weighted sum of entropies for left and right child) gain
function entropy(y): calculate entropy for labels y entropy
function gini_index(y): calculate gini index for labels y gini index
function calculate_leaf_value(Y): value with maximum frequency in Y
function predict(X): predictions for each data point in X
function make_prediction(x, tree): tree.value is not None: tree.value : feature_val = x[tree.feature_index] feature_val $\leq$ tree.threshold: make_prediction(x, tree.left) : make_prediction(x, tree.right)
```

Algorithm 4 Random Forest Classifier

```
class RandomForestClassifier:
function __init__(n_estimators=100, min_samples_split=3, max_depth=5):
initialize self.trees as empty list
set self.n_estimators to n_estimators
set self.min_samples_split to min_samples_split
set self.max_depth to max_depth
function fit(X, Y):
for _ in range(self.n_estimators): do
    indices = random indices with replacement from 0 to X.shape[0]
    X_bootstrap = X[indices]
    Y_bootstrap = Y[indices]
    tree = DecisionTreeClassifier(min_samples_split=self.min_samples_split, max_depth=self.max_depth)
    tree.fit(X_bootstrap, Y_bootstrap)
    add tree to self.trees
end for
function predict(X):
predictions = []
for tree in self.trees: do
    predictions.append(tree.predict(X))
end for
predictions = convert predictions to numpy array
y_pred = compute mean of predictions along axis 0, round, and convert to int y_pred
```

4. Feature Selection

4.1. Label Encoding

Label encoding is a technique used to convert categorical variables into numerical format. Each unique category in a feature is assigned a unique integer label. We applied label encoding to the 'Married/Single' and 'Car_Ownership' features using the LabelEncoder from scikit-learn.

4.2. One-Hot Encoding

One-hot encoding is another method for encoding categorical variables, particularly when the categories do not have an ordinal relationship. It creates binary columns for each category, representing its presence or absence. We applied one-hot encoding to the 'House_Ownership' feature using the OneHotEncoder from scikit-learn.

4.3. High Cardinality Features Encoding

High cardinality categorical features such as 'Profession', 'CITY', and 'STATE' pose challenges for traditional encoding techniques. To address this, we used count encoding, which replaces each category with the count of occurrences in the dataset.

4.4. Data Imbalance

The original dataset exhibited a significant class imbalance, with a large majority of instances belonging to the '0' class (no risk) compared to the '1' class (high risk). The class distribution before resampling was as follows:

```
Train class distribution:
Risk_Flag
0    176803
1     24797
```

4.5. Resampling Technique: Random UnderSampling

To mitigate the effects of data imbalance, we employed the Random UnderSampling technique. This technique involves randomly selecting a subset of instances from the majority class (no risk) to match the number of instances in the minority class (high risk), thus achieving a balanced distribution.

```
Resampled Train class distribution:
Risk_Flag
0    61992
1    30996
```

By balancing the class distribution using Random UnderSampling, we ensure that the predictive model is trained on a more representative dataset, leading to more accurate and reliable predictions, especially for the minority class instances.

4.6. Final Dataset

After feature encoding and selection, the dataset consists of features suitable for training the loan approval predictor model. The target variable 'Risk_Flag' is separated from the features for model training and evaluation.

5. Challenges

5.1. Naive Bayes

The challenges we faced while implementing the Naive Bayes algorithm are:

- Handling Imbalanced Data: The dataset we chose was imbalanced, with one class (0) dominating the other (1). This imbalance leads to biased predictions and affects the model's performance. The model accuracy was given 0.87, but the confusion matrix told the story that the model predicted zero for all the test data. So, we have to do an under-sampling of the data so that the model can be trained over a balanced dataset.

- Applying PCA: As the accuracy was less than 0.5, after some examining, we found that As the dataset had a large dimension, and the larger the dimension, the larger the dataset required, so we have to apply PCA to this to lower the dimension which leads to increase in accuracy.

5.2. Logistic Regression

- Hyperparameter tuning: changing value of learning rate and n_iters to find most optimal solution

- Handling Imbalanced Data: The dataset we chose was imbalanced, with one class (0) dominating the other (1). This imbalance leads to biased predictions and affects the model's performance. The model accuracy was given 0.87, but the confusion matrix told the story that the model mostly predicted 0. So, we have to do an under-sampling of the data so that the model can be trained over a balanced dataset. This gave worse accuracy but all other scores were increase

5.3. Decision tree

The challenges we faced while implementing the Decision tree algorithm are:

- Implementation of decision tree was a big challenge as due to long training time of data modifying and optimizing code was tough.

- Hyperparameter tuning: changing value of max_depth and other fields to find most optimal solution
- Handling Imbalanced Data: The dataset we chose was imbalanced, with one class (0) dominating the other (1). This imbalance leads to biased predictions and affects the model's performance. The model accuracy was given 0.87, but the confusion matrix told the story that the model mostly predicted 0. So, we have to do an under-sampling of the data so that the model can be trained over a balanced dataset. This gave worse accuracy but all other scores were increased.

5.4. Random Forest

The challenges we faced while implementing the Random Forest are:

- Huge Complexity: Random Forest involves creating multiple decision trees and aggregating their predictions, which lead to higher complexity leading to huge training time.
- Hyperparameter tuning: changing value of max_depth and other fields to find most optimal solution
- Handling Imbalanced Data: The dataset we chose was imbalanced, with one class (0) dominating the other (1). This imbalance leads to biased predictions and affects the model's performance. The model accuracy was given 0.88, but the confusion matrix told the story that the model mostly predicted 0. So, we have to do an under-sampling of the data so that the model can be trained over a balanced dataset.

6. Results

6.1. Naive Bayes

Performance Metrics:

- **Accuracy Score:** 0.57
- **Recall Score:** 0.43
- **Precision Score:** 0.13
- **F1 Score:** 0.20
- **Confusion Matrix:**

	Predicted Negative	Predicted Positive
Actual Negative	25904	18297
Actual Positive	3537	2662

Discussion:

- The Naive Bayes model achieved an accuracy of 0.57 on the test dataset.
- It showed a relatively high recall score, indicating its ability to correctly identify positive cases, but with a low precision score, suggesting a high number of false positives.
- The F1 score reflects the trade-off between precision and recall, providing a balanced measure of the model's performance.
- The confusion matrix illustrates the distribution of true positives, false positives, true negatives, and false negatives.

6.2. Logistic Regression(UnderSampled to 10000)

Performance Metrics:

- **Test Accuracy:** 0.67
- **Recall Score:** 0.004
- **Precision Score:** 0.41
- **F1 Score:** 0.008
- **Confusion Matrix:**

	Predicted Negative	Predicted Positive
Actual Negative	6627	46
Actual Positive	3288	39

Discussion:

- The logistic regression model achieved an accuracy of 0.67 on the test dataset.
- It struggled with correctly identifying positive cases, as evidenced by the low number of True Positives (39).
- Further analysis is required to understand the factors contributing to the misclassification of positive cases.

6.3. Decision Tree

Performance Metrics:

- **Accuracy Score:** 0.87
- **Recall Score:** 0.0078
- **Precision Score:** 0.29
- **F1 Score:** 0.0015
- **Confusion Matrix:**

	Predicted Negative	Predicted Positive
Actual Negative	8695	25
Actual Positive	1270	10

Discussion:

- The decision tree model achieved an accuracy of 0.87 on the test dataset.
- It showed a very low recall score and precision score, indicating poor performance in correctly identifying positive cases.
- The F1 score reflects the model's overall poor performance in both precision and recall.

6.4. Decision Tree (UnderSampled to 1000)

Performance Metrics:

- **Accuracy Score:** 0.65
- **Recall Score:** 0.28
- **Precision Score:** 0.40
- **F1 Score:** 0.33
- **Confusion Matrix:**

	Predicted Negative	Predicted Positive
Actual Negative	565	129
Actual Positive	221	85

Discussion:

- The decision tree model trained on undersampled data achieved an accuracy of 0.65 on the test dataset.
- It showed improved recall, precision, and F1 score compared to the model trained on the original data, indicating better performance in correctly identifying positive cases.
- However, the precision score is still relatively low, indicating a significant number of false positives.

6.5. Random Forest (UnderSampled to 1000)

Performance Metrics:

- **Accuracy Score:** 0.69
- **Recall Score:** 0.026
- **Precision Score:** 0.33
- **F1 Score:** 0.048
- **Confusion Matrix:**

	Predicted Negative	Predicted Positive
Actual Negative	678	16
Actual Positive	298	8

Discussion:

- The random forest model achieved an accuracy of 0.69 on the test dataset.
- It showed a low recall score, indicating a high number of false negatives.
- The precision score indicates that among the predicted positive instances, only 33.3% were actually positive.
- The F1 score is also low, reflecting the model's poor performance in correctly identifying positive cases.

7. Future scope

- Scalability and Efficiency: Optimize the implementation of the algorithms to ensure scalability and efficiency, especially when dealing with large-scale datasets. Consider techniques such as mini-batch training, parallel processing, or distributed computing to improve performance.
- Class Imbalance Handling: Further explore techniques to address class imbalance issues, such as synthetic data generation (e.g., SMOTE) or different resampling strategies, to improve the model's ability to generalize to minority class instances.
- Feature Importance Analysis: Perform feature importance analysis for each algorithm to understand which features have the most significant impact on predictions. This analysis can provide insights into the underlying relationships between features and the target variable, guiding feature selection and engineering efforts.
- Ensemble Learning: Explore ensemble learning techniques such as model stacking, where predictions from multiple models (e.g., logistic regression, random forest, decision trees, Naive Bayes) are combined to make final predictions. This approach often leads to improved performance and robustness compared to individual models.

8. Citations and References

Prediction of Loan Approval in Banks using Machine Learning Approach
Loan Approval Prediction using Machine Learning: A Review
Implementation of algorithms
Dataset:
Loan Defaulters