

AIM 511-Machine Learning

Project Report

Loan Default Prediction

Team Name: Project SAS

Team P. Anirudh (IMT2022505)

Members: Saishree (IMT2022578)

 Sri Surya (IMT2022567)

[GitHub Link](#)

1. Introduction

This report tackles the Loan Default Prediction Challenge presented by Coursera.

1.1 Dataset

The dataset contains 255,347 records and 18 columns. Following are the features in the dataset and their descriptions:

	Column_name	Column_type	Data_type	Description
0	LoanID	Identifier	string	A unique identifier for each loan.
1	Age	Feature	integer	The age of the borrower.
2	Income	Feature	integer	The annual income of the borrower.
3	LoanAmount	Feature	integer	The amount of money being borrowed.
4	CreditScore	Feature	integer	The credit score of the borrower, indicating their creditworthiness.
5	MonthsEmployed	Feature	integer	The number of months the borrower has been employed.
6	NumCreditLines	Feature	integer	The number of credit lines the borrower has open.
7	InterestRate	Feature	float	The interest rate for the loan.
8	LoanTerm	Feature	integer	The term length of the loan in months.
9	DTIRatio	Feature	float	The Debt-to-Income ratio, indicating the borrower's debt compared to their income.
10	Education	Feature	string	The highest level of education attained by the borrower (PhD, Master's, Bachelor's, High School).
11	EmploymentType	Feature	string	The type of employment status of the borrower (Full-time, Part-time, Self-employed, Unemployed).
12	MaritalStatus	Feature	string	The marital status of the borrower (Single, Married, Divorced).
13	HasMortgage	Feature	string	Whether the borrower has a mortgage (Yes or No).
14	HasDependents	Feature	string	Whether the borrower has dependents (Yes or No).
15	LoanPurpose	Feature	string	The purpose of the loan (Home, Auto, Education, Business, Other).
16	HasCoSigner	Feature	string	Whether the loan has a co-signer (Yes or No).
17	Default	Target	integer	The binary target variable indicating whether the loan defaulted (1) or not (0).

Image credits: Image by vectorjuice on Freepik

2. Data Preprocessing and Cleaning

2.1 Cleaning:

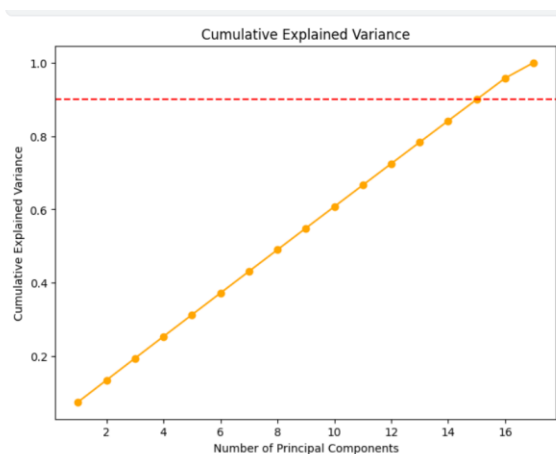
- Null Values:** An initial check for missing values revealed that the dataset contained no null values across any of the 18 columns. This eliminated the need for imputation or other methods to handle missing data, enabling a more streamlined preprocessing pipeline.
- Duplicate Rows:** A search for duplicate records confirmed that there were no duplicate rows in the dataset, ensuring that each entry represented a unique data point.
- Outliers:** To identify potential outliers, box plots were generated for each numerical feature. This visual examination showed that the dataset did not contain significant outliers across any of the primary features. As a result, no additional steps were necessary for outlier handling, allowing us to retain all data points for modeling.

2.2 Encoding:

The dataset includes categorical features like Education, EmploymentType, MaritalStatus, HasMortgage, HasDependents, LoanPurpose, and HasCoSigner. We used LabelEncoder to convert these features, assigning each category a unique integer, which ensured compatibility with machine learning models without increasing dimensionality.

2.3 Covariance Matrix and PCA:

Constructed a covariance matrix and took the covariance of each feature to the 'Default' or Target Feature. From it, we observed that InterestRate, LoanAmount, Age, Income, and MonthsEmployed have comparably the highest covariance out of all the features present. Noticing this, we wanted to observe if performing PCA on our dataset would reduce the dimensions. However, by plotting the cumulative variance plot, we observed the following:



Only with 14 out of 16 features present, we get 90% of the total variance. This explains that PCA is not suitable for our dataset.

3. Models

We have tested among several machine learning algorithms:

Part – 1:

3.1 K Nearest Neighbours

KNN is a algorithm that uses classification based on the majority class of the k nearest neighbors. We have optimized the hyperparameters such as k, distance metric, and weighting scheme by using a grid search. Cross-validation has also been used for robust evaluation. Final hyperparameters obtained were:

```
# Fit the model to the training data
grid_search_knn.fit(x_train, y_train)

print("Best Parameters for KNN:", grid_search_knn.best_params_)
print("Best Cross-Validation Accuracy for KNN:", grid_search_knn.best_score_)
✓ 2m 20.0s

Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Parameters for KNN: {'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'uniform'}
Best Cross-Validation Accuracy for KNN: 0.8823356596923272
```

3.2 Gaussian Naive Bayes

We used Gaussian Naive Bayes because it is an easy model suited to handle continuous values; we assumed that features are normally distributed. We trained this model using continuous variables such as Age, Income and InterestRate. It ran really fast with respect to computation of probabilities and predictions.

3.3 Decision Tree with Hyperparameter Tuning

We used a Decision Tree Classifier because it is interpretable and can represent complex patterns in the data. We used a grid search with 5-fold cross-validation to tune the hyperparameters `max_depth`, `min_samples_split`, `min_samples_leaf`, and `criterion`, which could be gini or entropy, fitting this kind of model best for accuracy and balancing complexity with generalization to avoid overfitting and to boost the capability of the model. Final hyperparameters obtained were:

```
# Train the model and find the best parameters
grid_search.fit(x_train, y_train)
print("Best Parameters:", grid_search.best_params_)
✓ 27.0s

Fitting 5 folds for each of 54 candidates, totalling 270 fits
Best Parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

3.4 Random Forest with Hyperparameter Tuning

We further tried to enhance the model performance by using a Random Forest Classifier. To deduce optimal hyperparameters of `n_estimators`, `max_features`, `max_depth`, `min_samples_split`, and `min_samples_leaf`, we used Grid search with 4-fold cross-validation. This approach balanced model diversity and generalization. Best hyperparameters obtained were:

```
print("Best parameters for Random Forest:", rf_grid_search.best_params_)
✓ 70m 12.4s

Fitting 4 folds for each of 243 candidates, totalling 972 fits
/Users/anirudhpathaneni/Desktop/Predict_Defaulters/env/lib/python3.13/site-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in divide
  _data = np.array(data, dtype=dtype, copy=copy,
Best parameters for Random Forest: {'max_depth': 6, 'max_features': None, 'min_samples_leaf': 6, 'min_samples_split': 2, 'n_estimators': 250}
```

3.5 XGBoost with Hyperparameter Tuning

Lastly, we used the XGBoost model for better prediction accuracy. This model iteratively builds decision trees, thereby efficiently managing complex, nonlinear patterns. On 4-fold cross-validation, Grid search tuned the vital hyperparameters like `n_estimators`, `learning_rate`, `subsample`, `max_depth`, `min_child_weight`, and `gamma`. Best parameters were:

```
print("Best parameters for Random Forest:", grid_search2.best_params_)
✓ 0.0s

Best parameters for Random Forest: {'gamma': 0, 'learning_rate': 0.15, 'max_depth': 3, 'min_child_weight': 5, 'n_estimators': 150, 'subsample': 0.7}
```

Part – 2:

3.6 Logistic Regression

Logistic Regression was implemented to model the relationship between input features and the probability of the target class. To optimize its performance, 5-fold cross-validation was employed

alongside a Grid search approach for hyperparameter tuning. Key hyper parameters such as C (regularization strength), penalty (L1 or L2 regularization), solver, and class_weight were fine-tuned to account for class imbalance and improve the model's generalization. Best hyperparameters obtained were:

```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Best Parameters: {'C': 0.1, 'class_weight': None, 'penalty': 'l1', 'solver': 'saga'}
Best F1 Score: 0.8851616361433903
```

3.7 Support Vector Machines

We utilized two types of Support Vector Machine (SVM) models to tackle the classification problem: the standard Support Vector Classification (SVC) and the Linear Support Vector Classification (LinearSVC). The SVC model was optimized using RandomizedSearchCV, which explored a range of hyperparameters, including regularization parameter C, kernel type (linear, poly, rbf, sigmoid), and gamma, employing a log-uniform distribution to cover a wide range of values. This approach, with 5-fold cross-validation and 100 iterations, helped identify the best configuration for the model. Additionally, we trained a LinearSVC model directly on the dataset to compare its performance with the more complex SVC model. The combination of both models provided valuable insights into their suitability for the dataset. Best hyperparameters were:

```
Best Parameters: {'C': np.float64(0.1767016940294795), 'gamma': np.float64(5.669849511478847), 'kernel': 'rbf'}
Best Cross-Validation Score: 0.889
```

3.8 Neural Networks

A neural network model, LoanNN, was implemented using PyTorch for binary classification. The model consists of three hidden layers with dropout for regularization. CrossEntropyLoss was used as the loss function, and the Adam optimizer was applied for training. To address class imbalance, various strategies were experimented with to improve model performance.

Another neural network model was implemented using Keras for binary classification. The architecture consists of multiple dense layers with LeakyReLU activations, BatchNormalization, and Dropout for regularization. The model was compiled with the Adam optimizer and binary cross-entropy loss. Performance was evaluated using accuracy, precision, and recall metrics. Additionally, SMOTE was experimented with to address class imbalance.

4. Performance Comparison

4.1 K Nearest Neighbours

Kaggle Score:



knn_predictions.csv

Complete · Anirudh Pathaneni · 1mo ago

0.88241

0.88241




4.2 Gaussian Naive Bayes

Kaggle Score:

 gb.csv Complete · Anirudh Pathaneni · 1mo ago	0.88488	0.88488	<input type="checkbox"/>
---	----------------	----------------	--------------------------


4.3 Decision Tree

Kaggle Score:

 decision.csv Complete · Anirudh Pathaneni · 1mo ago	0.88547	0.88547	<input type="checkbox"/>
---	----------------	----------------	--------------------------


4.4 Random Forest

Kaggle Score:

 rfc.csv Complete · Anirudh Pathaneni · 1mo ago	0.88584	0.88584	<input type="checkbox"/>
--	----------------	----------------	--------------------------

4.5 XGBoost

Kaggle Score:

 xgboosting.csv Complete · Anirudh Pathaneni · 1mo ago	0.88811	0.88811	<input type="checkbox"/>
---	----------------	----------------	--------------------------

4.6 Logistic Regression

Kaggle Score:

Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
 ⓘ LR_Predictions.csv Complete (after deadline) · Anirudh Pathaneni · 2m ago	0.88594	0.88594	<input type="checkbox"/>

4.7 Support Vector Machines

Kaggle Score:

SVC



 ⓘ svm.csv Complete (after deadline) · Anirudh Pathaneni · 7m ago	0.88447	0.88447	<input type="checkbox"/>
--	----------------	----------------	--------------------------

LinearSVC

 ⓘ linear_svm.csv Complete (after deadline) · Anirudh Pathaneni · 7m ago	0.88447	0.88447	<input type="checkbox"/>
---	----------------	----------------	--------------------------

4.8 Neural Networks

PyTorch

<div>  NN_Predictions.csv Complete (after deadline) · Xcaliber · 11h ago </div>	0.88652	0.88652	<input type="checkbox"/>
Keras			
<div>  keras_nn.csv Complete (after deadline) · Saishree Vardhan Kosuru · 8h ago </div>	0.88742	0.88742	<input type="checkbox"/>

5. Conclusion

KNN may have struggled due to high dimensionality, noisy or irrelevant features, sensitivity to feature scaling, difficulty handling non-linear decision boundaries.

After observing that the accuracy was low, we realized that the assumption of Gaussian distribution for the continuous features might not hold in this dataset. Features like income, loan amount, and credit score often exhibit skewed distributions which can lead to poor performance with Gaussian Naive Bayes. This violation of the normality assumption may have contributed to the suboptimal accuracy.

As for tree models, each of these models gives only a fractional change in accuracy, indicating that the underlying patterns in the data are relatively simple and well-represented by each of these models. Given the similar performance across all three models, it might be practical to prioritize XGBoost because it gave marginally better performance from the validation score. Or a simpler model (e.g., Decision Tree) can be chosen for interpretability and faster computation. The highest accuracy on test.csv was also from XGBoost model.

Logistic regression outperformed the SVM models likely due to its ability to handle linear decision boundaries more effectively in this case. The simplicity of logistic regression, coupled with regularization, made it more robust to the class imbalance in the dataset compared to the SVM models.

Both the SVM with RBF kernel and the LinearSVC model gave similar results (0.88447 accuracy) because the dataset may not contain complex non-linear patterns that the RBF kernel can leverage. As a result, both models were able to effectively separate the classes using linear decision boundaries.

The PyTorch-based neural network yielded an accuracy of 0.88652, slightly outperforming the previous models. Lastly, the Keras neural network achieved the accuracy of 0.88742, showcasing its ability to model complex patterns.