Machine Learning: Project Report

Rudra Pathak
Rudra.Pathak@iiitb.ac.in
IMT2022081

Kandarp Dave
Dave.Kandarp@iiitb.ac.in
IMT2022115

Soham Pawar
Soham.Pawar127@iiitb.ac.in
IMT2022127

1 Introduction

In this project, we study the Lend or Lose dataset. The target is to predict whether a person would default on their loan or not. The dataset contains 255,347 rows and 18 columns in total. This is essentially a classification tasks, and various machine learning models were created to predict the target label *Default*. The github repository can be found here.

2 Procedure Followed

1. The original train and test data are stored in the data folder.
2. The *data_preprocessing.ipynb* file is used to preprocess the data and perform exploratory data analysis (EDA). The training data is split into training data and validation data. These files are stored in the *cleaned_data* folder.
3. Various classification models are used to predict the target label for the test data, which is in the *data* folder. The code and predictions of each model can be found in the respective folders. To decide the hyperparameters of each model, *GridSearchCV*, *RandomizedSearchCV*, and *BayesSearchCV* were used. Also, to identify the misclassifications, confusion matrices were drawn for each model.

3 Preprocessing and EDA

To start off we checked if there were any null values or duplicates present in the dataset and there were none. We then checked for the presence of any outliers. We also drew various boxplots to identify any outliers in our dataset but there were no outliers. A key observation that we made was that the number of 0's

in the **Default** column was way higher than the number of 1's. As it was a classification task we did not scale the data.

We also added various plots and visualization in our notebook.

After this, we look at the distribution of the columns *Income* and *Loan Amount*. For both columns, the values are distributed almost uniformly. On applying the *log* transformation, a skewed distribution is observed, which does not resemble a Gaussian curve. Hence, no transformation was applied on these columns.

We one-hot encoded any non-numerical columns that were present and performed train test split to complete our data pre-processing.

4 Models Used

4.1 XGBoost

In this project, we used XGBoost for loan default prediction. Hyperparameters such as the number of estimators, maximum depth, and learning rate were optimized using RandomizedSearchCV with 8-fold cross-validation. As the model is quite complex it took a fair bit of time to fine tune the hyperparameters. The model achieved a test accuracy of 0.885500, and a confusion matrix was used to evaluate performance and identify misclassification. As XGBoost is an ensemble model which uses multiple weak learners for final prediction we got our best accuracy score using it.

The best model was refitted on the entire test dataset for generating final predictions, which were submitted to Kaggle, achieving a leaderboard score of 0.88776.

4.2 AdaBoost

In the AdaBoost algorithm, the weights of training samples are adjusted in each iteration to account for previous misclassification. Then, the classifier is trained again on the same training data. In this project, we used a random forest as the estimator. This gave a better accuracy when compared to using a single decision tree as the estimator. The best hyperparameters were decided by *BayesSearchCV*. *BayesSearchCV* was preferred over *RandomizedSearchCV* as AdaBoost took high time in training and thus lower number of iterations had to be performed during hyperparameter tuning. It was found that a lower learning rate and higher number of iterations of AdaBoost overfitted the training data. It was also seen that a better result was achieved on dividing the values of *LoanTerm* by 12. A best score of 0.88768 was achieved on Kaggle using AdaBoost.

4.3 Logistic Regression

In this project, logistic regression was used to predict loan defaults. The model was trained on a cleaned dataset but with no hyperparameter tuning as there weren't any significant hyperparameters to tune and for the same reason training was very fast however we did get a warning that the model needed more iterations to converge and max iterations limit was already exceeded. After training, the model achieved an accuracy of 0.8830526 on the test set. A confusion matrix was used to evaluate the model's performance further and identify specific misclassifications. The trained model was then refitted on the entire test dataset before generating predictions on the final test data. Predictions, along with LoanID, were saved to LR_predictions.csv. The results achieved a leaderboard score of 0.88455 on Kaggle.

4.4 Decision Tree

Along with the other models used in the project, one of the elementary models used is the Decision Tree Classifier. The Decision Tree Classifier is a tree-based supervised algorithm. It works by recursively splitting the dataset into subsets based on feature values. Instead of directly using the features originally present in the dataset, we made a few changes:

1. Introduced a new feature called the 'AmountRatio_I_La' which is the ratio of the features 'Income' and 'LoanAmount'.
2. Divided the 'LoanTerm' feature by 12 since the data values were a multiple of 12.

The motivation behind these changes is:

1. A new determining feature which signifies a relation between the earlier mentioned features. This feature is assumed to bring a strong determining feature for the classification task for better results.
2. This change is assumed to make the decision-making process more focused for the tree for this feature as the decision boundary in the nodes of the tree would have a smaller possible range of values to consider.

The model was trained on multiple hyperparameters and different combinations of features among 'Income' (I), 'LoanAmount' (LA) and 'AmountRatio_I_La' (Ratio). We use 5 cases:

1. Case A - (I) present, (LA) present, (Ratio) not present in training dataset
2. Case B - (I) present, (LA) not present, (Ratio) present in training dataset
3. Case C - (I) not present, (LA) present, (Ratio) present in training dataset
4. Case D - (I) not present, (LA) not present, (Ratio) present in training dataset

5. Case E - (I) present, (LA) present, (Ratio) present in training dataset

For the DecisionTree, the hyperparameter tuning was performed using the RandomSearchCV and GridSearchCV algorithms. For this model, the best precision obtained was 0.88519 for case B of the feature combination from the GridSearchCV algorithm.

4.5 Random Forest

Random Forest is also a supervised algorithm which combines multiple decision trees to improve the accuracy and robustness of classification tasks. It builds several decision trees during training and makes predictions by aggregating the results.

The pre-processing and the addition of the new features is the same as the process used for the Decision Tree algorithm. From this training and comparison process for all the earlier mentioned cases, it was found that the best result obtained was 0.88686 using the RandomSearchCV algorithm for Case C.

4.6 Bayes Classifier

In this project, Gaussian Naive Bayes was employed for loan default prediction. The model was trained on cleaned datasets, and hyperparameter tuning was performed using GridSearchCV to optimize the var_smoothing parameter. An 8-fold crossvalidation was used for evaluation, with accuracy as the scoring metric. The best model achieved an accuracy of 0.884300 on the test set. A confusion matrix was plotted to analyze the model's performance further.As we tuned only a single hyperparameter the training was very quick.

The best model was retrained on the complete test dataset and used to predict outcomes for the final test data. Predictions, along with LoanID, were saved to gaussianNB_predictions.csv. The submission achieved a leaderboard score of 0.88545 on Kaggle.

4.7 K Nearest Neighbors

K Nearest Neighbors algorithm uses the labels of the nearest K neighbors in the training dataset to predict the label of a new data point. We explored all K values in the range of 1 to 30 in this project. The KNeighborsClassifier provides 3 different algorithms to compute the nearest neighbors:

1. *Brute*

   This algorithm calculates the distances from each data point in the training dataset. Since the dataset consists of a large number of data points, this method would take a lot of time and hence was not used.

2. *Ball Tree*

The Ball Tree represents data points as nodes in a tree, where each node is associated with a hypersphere (ball) that contains a subset of data points. This method was found to be the fastest among all the three methods.

3. *KD Tree*

   KDTree is a data structure that is used to partition K-dimensional data points. This makes range search easier as compared to brute force. However, in our dataset, the number of columns is very high due to one-hot encoding of categorical columns. Thus, the number of levels in KDTree increases, and hence the speed of the algorithm decreases.

Additionally, we also tuned the weights associated with each data point. Uniform weight means equal weight for all points. The weights can also be decided to be inversely proportional to the distance from the new data point. A best score of 0.88384 was achieved on Kaggle using KNN. Since there are a large number of categorical columns, and lesser numerical data, KNN performed bad on the given dataset.

4.8 ANN : Artificial Neural Network

Artificial Neural Network or NN for short, is a biology-inspired graphical structure which mimics the functioning of the brain. The MO of this architecture is that it has the so-called layers which are composed of multiple perceptrons or neurons. These neurons can be connected in multiple ways with other neurons. For the purpose of this example, the neurons in any given layer are not interconnected, whereas the neurons between two consecutive layers are connected in the **Dense** fashion, as described by the **pytorch** library. Fairly advanced concepts like **Dropout** is also implemented, borrowing from the effect of this feature on other benchmark models. The structure of the neural network implemented in the project looks like the following:

In this Neural Network, the output of the model is considered to be binary, and thus two neurons are implemented; one for representing **Yes** and the other for **No**. As a result while training, the class to be predicted: **Default** is converted into the binary type into one-hot-encoded type vectors. The activation function used is **ReLU**: Rectified Linear Unit.

Another pre-processing applied in the implementation was taking the log values of the first eight classes as the data range of most of these eight classes is unbounded. As a result, we chose to go with logarithmic scaling instead of doing min-max or normalized standardization, to prevent scaling to the local values.

After training this model for **30** epochs, with the help of the **Adam** optimizer and **Cross Entropy Loss**, the accuracy obtained was **0.88447**.

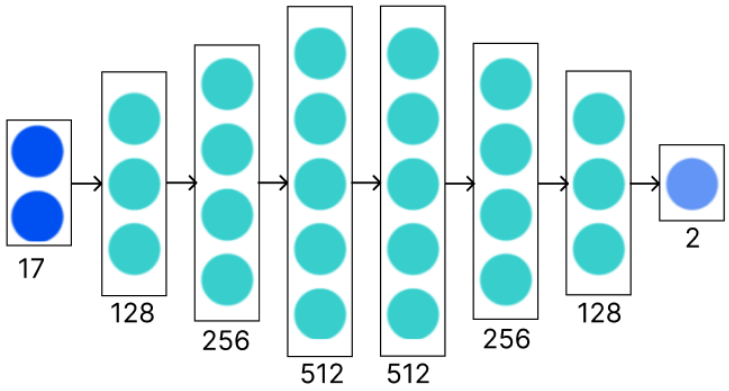


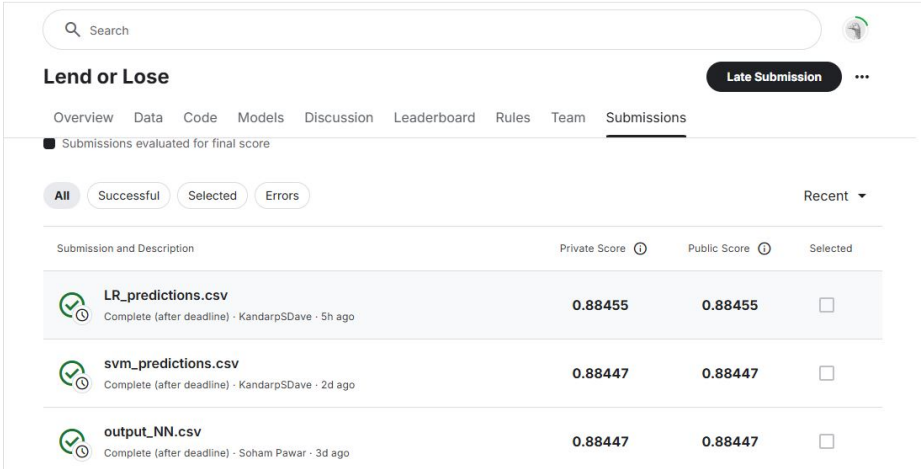
Figure 1: Illustration of the NN implemented in the project with the text below each layer representing the number of neurons in that layer.

4.9 Support Vector Machine (SVM)

For applying the SVM model, we first needed to standardize the numerical columns of the dataset, such as Age, LoanAmount, Income, etc. After this, we did hyperparameter tuning to decide the best kernel. Out of the kernels 'rbf' and 'poly', 'poly' gave the best result. One observation was that the SVM predicted only 0s. This might be because of the large number of 0s in the dataset. Also, SVM took significantly more time to train than the rest of the models. It gave a score of **0.88447** on Kaggle.

4.10 Part 2 Submissions

For part 2 of the project, the scores obtained are:



5 Conclusion

It was seen that all the models gave an accuracy higher than 0.88. However, models like XGBoost and AdaBoost outperformed the other models as they combine multiple weak predictors to create a strong predictor. Also, the dataset was inherently biased as it contained more number of 0s in the *Default* column, as compared to 1s. The confusion matrices for each model show that the models wrongly classified a significant number of 1s as 0s. XGBoost achieved the best accuracy of 0.88776 on Kaggle.