THE iSCHOOL
Syracuse University

IST 707 – APPLIED MACHINE LEARNING

LOAN ELIGIBILITY PREDICTION

GROUP 8

FINAL PROJECT REPORT SPRING 2023

Submitted By

Shrish Kiran Vaidya

Vedant Devidas Patil

Tejas Gawade

Jainish Savaliya

Table of contents

1. INTRODUCTION

Loan approval processes are critical for financial institutions as they determine the eligibility of borrowers to receive loans. However, these traditional processes are often time-consuming, costly, and prone to errors, leading to delays and higher expenses for financial institutions. To address these challenges, this project aims to build a machine learning model that accurately predicts loan applicant eligibility based on various factors such as gender, marital status, education, income, employment status, credit history, and other demographic information.

The use of machine learning algorithms in this project will improve the accuracy and speed of loan approval processes, benefiting both financial institutions and their customers. The model will provide an efficient and cost-effective solution to traditional loan approval processes, reducing delays and expenses while increasing accuracy

2. PROBLEM STATEMENT

The loan approval process is a crucial aspect of any financial institution, as it determines whether a borrower is eligible to receive a loan or not. However, traditional loan approval processes can be time-consuming, expensive, and prone to errors, leading to delays in loan processing and higher costs for financial institutions.

Therefore, we are aiming to a machine learning model that can accurately predict whether a loan applicant is eligible for a loan or not based on various factors such as their gender, married, education, applicant income, employment status, credit history, and other demographic information.

This project has two use cases: Firstly, banks may utilize it to identify client categories that are suitable for loan amounts so that they can precisely target these consumers. Second, it may be used by a loan eligibility predictors application, in which individuals can enter their information and the application will tell them if they will be accepted for a loan or not.

3. DATA DESCRIPTION

The loan application dataset utilized in this project is comprised of 614 entries, each with 13 features. The dataset contains information on loan applicants, including their gender, marital status, education level, applicant income, co-applicant income, loan amount, loan amount term, credit history, property area, loan status, number of dependents, and a unique loan ID.

The dataset required some pre-processing due to missing data, which was handled by imputing the mean for numerical features and the mode for categorical features. This approach was chosen to preserve the overall characteristics of the dataset and ensure that there were no erroneous values.

Overall, the dataset provides valuable insight into the loan application process, allowing us to analyze factors such as credit history, income, and other demographic information that may impact the loan approval process. By leveraging this dataset and performing machine learning techniques, we aim to build a model that can accurately predict loan eligibility and improve the loan approval process for financial institutions.

4. DATA PREPARATION

The first step in our data analysis project involved importing the dataset and checking its structure. We used the pandas library to read in the CSV file containing the loan application data and displayed the first few rows using the head () function.

Next, we inspected the structure of the dataset using the info() function to check for any missing values or incorrect data types. We found that the dataset contained a total of 614 entries and 13 features, including loan ID, gender, marital status, education level, applicant income, co-applicant income, loan amount, loan amount term, credit history, property area, loan status, and the number of dependents.

However, we also discovered that some of the features contained missing values, which could potentially impact our analysis. Therefore, we took the necessary steps to remove or impute these missing values.

We first used the isnull().sum() function to identify the number of null values present in each feature. We then dropped the rows containing null values in the Gender, Dependents, Self_Employed, and Credit_History features using the dropna() function. For the LoanAmount and Loan_Amount_Term features, which had numerical values, we used the interpolate() function to impute the missing values with the mean.

After removing or imputing the missing values, we checked the dataset again using the isnull().sum() function and found that all features were now free of missing values. However, we noticed that one row still contained missing values, so we used the iloc() function to locate and drop this row from the dataset.

Finally, we confirmed that our dataset was now clean and ready for exploratory data analysis by using the info() function to display the updated structure of the dataset.

5. EXPLORATORY DATA ANALYSIS AND PROCESSING

We first started by analyzing the dataset to gain a better understanding of its distribution. We began by looking at the distribution of the "Married" column, using the value_counts() method to count the number of married and unmarried applicants in the dataset. We then created a pie chart to visualize the distribution, using the plot() method. This allowed us to see that approximately 65% of the applicants in the dataset were married.

Next, we repeated this process for the "Gender" column, again using value_counts() to count the number of male and female applicants, and creating a pie chart to visualize the distribution. We found that the dataset was heavily skewed towards male applicants, with more than 80% of applicants being male.

We then moved on to analyse the distribution of the "Dependents" and "Education" columns, using the same methods as before. We found that most applicants did not have any dependents and were also highly educated, with more than 75% of applicants holding a graduate degree.

Moving on to the "Self_Employed" column, we again used value_counts() to count the number of self-employed and non-self-employed applicants, finding that only around 14% of applicants were self-employed.

We then saw the distribution of the "ApplicantIncome" column, using the distplot() method from the seaborn library. The resulting plot showed that the data was heavily skewed towards the lower end, with a few outliers on the higher end. To address this, we applied a logarithmic transformation to the data, reducing the skewness and bringing the data more in line with a normal distribution.

We repeated this process for the "LoanAmount" column, again using the distplot() method to plot the histograms and the boxplot() method to visualize the outliers. In this case of the "LoanAmount" column, we applied a logarithmic transformation to the data to reduce the skewness and address the outliers.

Next, we plotted a histogram of the "Loan_Amount_Term" column, finding that the data was approximately normally distributed. We also plotted a pie chart of the "Credit_History" which is showing the number of applicants with and without a credit history.

Moving on to data visualization, we created a cross-tabulation table of the "Gender" and "Loan_Status" columns, using the crosstab() method to count the number of male and female applicants who were approved or rejected for a loan. We then created a stacked bar chart to visualize the data, showing that there was no significant difference between the approval rates for male and female applicants.

Finally, we used the corr() method to calculate the correlation matrix between the different columns in the dataset, visualizing the results using a heatmap. We also get to know that from the heatmap that Credit History has strongest correlation with Loan status. We also converted all categorical features to numerical values using the LabelEncoder() method, allowing us to use them in our machine learning models.

To prepare for modeling, we split our preprocessed data into training and testing sets using the train_test_split function from the scikit-learn library. We set aside 20% of the data as our test set, stratified by the target variable 'Loan_Status', and use a random state of 42 for reproducibility.

Next, we split our training data into features (X) and target variable (y). We then split X and y into training and validation sets using train_test_split with a test size of 20% and random state of 42. The resulting data is now ready for modeling.

6. DECISION TREE

First, we defined the parameter grid to search over. A param_grid variable is a dictionary that holds the hyperparameters and their corresponding values that we want to test for the Decision Tree Classifier model. In this case, we are searching over ccp_alpha and min_samples_leaf hyperparameters. ccp_alpha is the complexity parameter used for post-pruning decision trees, and min_samples_leaf is the minimum number of samples required to be at a leaf node.

We then created an instance of the Decision Tree Classifier and defined the grid search object. The "GridSearchCV" function performs a grid search over the specified parameter values and returns the best combination of hyperparameters that yield the highest accuracy score. We set cv=5, which means we used 5-fold cross-validation for model selection.

Next, we fit the grid search object to our training data. This process trains the Decision Tree Classifier model on our data and tests different hyperparameters to find the best model. The fit() function returns the best model with the hyperparameters that yielded the highest accuracy score.

After fitting the model, we printed the best hyperparameters and the corresponding accuracy score using the best_params_ and best_score_ attributes of the grid search object. We also used the "predict()" function to predict the target variable of our test data.

To evaluate the performance of the model, we generated a confusion matrix using the confusion_matrix() function from the sklearn.metrics module. The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives. We got the accuracy score of 82.92% using the accuracy_score() function from the same module.

7. NAÏVE BAYES & LOGISTIC REGRESSION

Firstly, we import the required libraries - pandas, GaussianNB from sklearn.naive_bayes, GridSearchCV from sklearn.model_selection and LogisticRegression from sklearn.linear_model.

Next, we define the parameter grid for both models which are the values of the hyperparameters that we want to test for our models. For the Naive Bayes classifier, we define a parameter grid with values for 'var_smoothing' ranging from 1e-9 to 1e-5. For the Logistic Regression model, we define a parameter grid with values for 'C' ranging from 0.01 to 100 and values for 'penalty' as 'l1' and 'l2'.

We then define the Naive Bayes and Logistic Regression classifiers with their respective default parameters.

Next, we define the grid search object with the defined parameter grid, classifier and cross-validation value. The grid search object is then fit to the training data to find the best hyperparameters using cross-validation. This is done using the fit() method of the GridSearchCV object.

We then print the best parameters and best score obtained by the grid search object. The predicted classes are obtained using the predict() method of the fitted model and the confusion matrix is generated for the predictions using the confusion_matrix() method from sklearn.metrics. The accuracy score for the predictions is calculated using the accuracy_score() method from sklearn.metrics.

Finally, we print the predicted classes, confusion matrix and the accuracy score on the test data.

This process is repeated for both Naive Bayes and Logistic Regression models. We got accuracy of 80.41% for Naïve Bayes and for Logarithmic Regression we got accuracy of 82.92%.

8. RANDOM FOREST

First, we import the necessary modules, including the RandomForestClassifier class from sklearn.ensemble. We then define the random forest model, specifying the number of trees to be used (n_estimators) and the number of features to consider when looking for the best split (max_features).

We then define the parameter grid to search over, in this case, just the max_features parameter with two values to choose from: sqrt and log2.

Next, we create a GridSearchCV object, which will perform a search over the specified parameter grid. We fit the grid search object to the training data (X_train and y_train) and save the resulting model in the model_random_forest variable.

We then print the best parameters found by the grid search and the corresponding training score. We use the model to make predictions on the test set (X_test) and print the predicted classes.

We generate the confusion matrix for the predictions and print it out. Finally, we calculate the accuracy score of the model on the test set and print it out as well. We got the accuracy of 87.80%.

This method of hyperparameter tuning using grid search allows us to systematically evaluate multiple combinations of hyperparameters and select the best one based on the cross-validated performance.

From Random Forest, the important feature plot show that Credit History, LoanAmount, and ApplicantIncome are the most important features that help in classifying weather the person gets loan or not.

9. SUPPORT VECTOR MACHINE(SVM)

Firstly, we load the required libraries like tidyverse, caret, e1071, and kernlab. We also define a grid of values for the tuning parameter 'C' for the SVM algorithm. 'C' controls the balance between achieving a low training error and a low testing error.

Next, we create a train control object for cross-validation. We are using the repeated k-fold cross-validation method with 3 folds repeated 10 times.

Then, we train our SVM model using the train function from the caret package. We use the svmLinear method for our first model and pass our grid of tuning parameters and train control object to the function.

After training, we predict the loan status of our test data using our trained SVM model and compute the confusion matrix to evaluate the model's performance.

Next, we tune the SVM model with a polynomial kernel by creating a new grid of tuning parameters and retraining the model using the svmPoly method.

Again, we predict the loan status of our test data using our trained SVM model and compute the confusion matrix to evaluate the model's performance.

Finally, we perform cross-validation with the radial kernel by creating a new grid of tuning parameters and training the model using the svmRadial method and got the accuracy based on testing data based on this model. We got the Poly and Linear model as best models with accuracy of 81.76%.

10. RESULTS

To evaluate the performance of our models, we used accuracy as our evaluation metric. The accuracy metric gives us an understanding of the proportion of correctly classified instances in our test data. The higher the accuracy, the better our model's performance.

After training and testing our models on final testing dataset, we found the Decision Tree model, Random Forest and Logistic Regression had the highest accuracy of 84%, Naïve Bayes with 82%, and Random Forest with 82%, and SVM with 81%.

We also used different techniques to fine-tune our models, such as hyperparameter tuning and cross-validation, to increase the accuracy of our models.

Overall, based on the accuracy of our models, we can conclude that the Decision Tree and Logistic Regression models performed the best among all the models for predicting the loan eligibility criteria.