**Loan Approval Prediction**

For this 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 614 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, Random Forest and XGBoost to check the accuracy of each model

1. **INTRODUCTION**
   1. **Motivation and Objective**

Machine Learning has become increasingly popular tool in almost every industry. With Machine Learning we can achieve so much. We will be using machine learning algorithms along with some data analysis techniques in our project.

Our project focus is to use existing customer’s details and analyse it further by applying a few machine learning techniques and predict which future applicant can be approved the loan.

* 1. **Problem Statement**

The problem is 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 decided to automate the loan approval process using machine learning.

* 1. **Machine Learning**

Machine Learning is an idea to learn from examples and experience, without being explicitly programmed.

* + 1. **Examples of Machine Learning**

There are many examples of machine learning. Here are a few examples of classification problems where the goal is to categorize objects into a fixed set of categories.

Face detection: Identify faces in images (or indicate if a face is present).

Email filtering: Classify emails into spam and not-spam.

Medical diagnosis: Diagnose a patient as a sufferer or nonsufferer of some disease. Weather prediction: Predict, for instance, whether or not it will rain tomorrow.

* + 1. **Need of Machine Learning**

Machine Learning is a field which is raised out of Artificial Intelligence (AI). Applying AI, we wanted to build better and intelligent machines.

If big data and cloud computing are gaining importance for their contributions, machine learning as technology helps analyze those big chunks of data, easing the task of data scientists in an automated process and gaining equal importance and recognition.

* + 1. **Kinds of Machine Learning**

There are three kinds of Machine Learning Algorithms.

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

**1.3.3.1 Supervised Learning :**

A majority of practical machine learning uses supervised learning. In supervised learning, the system tries to learn from the previous examples that are given.

Supervised learning problems can be further divided into two parts, namely classification, and regression.

**Classification:** A classification problem is when the output variable is a category or a group, such as “black” or “white” or “spam” and “no spam”. **Regression:** A regression problem is when the output variable is a real value, such as “Rupees” or “height.”

**1.3.3.2 Unsupervised Learning**

Unsupervised learning problems can be further divided into association and clustering problems

**1.3.3.3 Reinforcement Learning**

A computer program will interact with a dynamic environment in which it must perform a particular goal

**2. PROCESS and SPECIFICATION REQUIREMENT**

**2.1 Process:**

• Hypothesis Generation

• Understanding the Data

• Exploratory Data Analysis (EDA)

♣ Univariate analysis

♣ Bivariate analysis

• Missing Value and Outlier Treatment

• Feature Engineering

• Model Building

♣ Logistic Regression

♣ Decision Tree

♣ Random Forest

♣ XGBoost

**2.2 Software Requirements:**

• Python

• Anaconda (JUPYTER Notebook)

**2.2.1 LIBRARIES USED:**

• NumPy

• SciKit Learn

• Pandas

• MatplotLib

• Seaborn

• XGBoost

**3. METHODOLOGY**

**3.1 Hypothesis Generation**

This is the first and foremost step which is performed even before looking at the data. It involves understanding the problem in detail by brainstorming as many factors as we can. Below are some of the factors which I think can affect the Loan Approval (dependent variable for this loan prediction problem):

**Independent Variables:**

1. Loan\_ID - This refer to the unique identifier of the applicant's affirmed purchases
2. Gender - This refers to either of the two main categories (male and female) into which applicants are divided on the basis of their reproductive functions
3. Married - This refers to applicant being in a state of matrimony
4. Dependents - This refres to persons who depends on the applicants for survival
5. Education - This refers to number of years in which applicant received systematic instruction, especially at a school or university
6. Self\_Employed - This refers to applicant working for oneself as a freelancer or the owner of a business rather than for an employer
7. Applicant Income - This refers to disposable income available for the applicant's use under State law.
8. CoapplicantIncome - This refers to disposable income available for the people that participate in the loan application process alongside the main applicant use under State law.
9. Loan\_Amount - This refers to the amount of money an applicant owe at any given time.
10. Loan\_Amount\_Term - This refers to the duaration in which the loan is availed to the applicant
11. Credit History - This refers to a record of applicant's ability to repay debts and demonstrated responsibility in repaying them.
12. Property\_Area - This refers to the total area within the boundaries of the property as set out in Schedule.
13. Loan\_Status - This refres to whether applicant is eligible to be availed the Loan requested.

**3.2 Reading the data**

Load data from the given data set

df=pd.read\_csv('https://raw.githubusercontent.com/dsrscientist/DSData/master/loan\_prediction.csv')

df.head()

**3.3 Understanding the Data**

Loan\_ID object

Gender object

Married object

Dependents object

Education object

Self\_Employed object

ApplicantIncome int64

CoapplicantIncome float64

LoanAmount float64

Loan\_Amount\_Term float64

Credit\_History float64

Property\_Area object

Loan\_Status object

We can see there are three format of data types:

• **object:** Object format means variables are categorical. Categorical variables in our dataset are: Loan\_ID, Gender, Married, Dependents, Education, Self\_Employed, Property\_Area, Loan\_Status

• **int64**: It represents the integer variables. ApplicantIncome is of this format.

• **float64**: It represents the variable which have some decimal values involved. They are also numerical variables. Numerical variables in our dataset are: CoapplicantIncome, LoanAmount, Loan\_Amount\_Term, and Credit\_History

**3.5 Bivariate Analysis**

After looking at every variable individually in univariate analysis, we will now explore them again with respect to the target variable.

Gender=pd.crosstab(df['Gender'],df['Loan\_Status'])

Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind="bar",

stacked=True, figsize=(4,4))

A graph of a person and person

Description automatically generated

It can be inferred that the proportion of male and female applicants is more or less same for both approved and unapproved loans.

**3.6 Missing value , Outlier Treatment and correlation**

**3.6.1 Missing value Imputation**

After exploring all the variables in our data, we can now impute the missing values and treat the outliers because missing data and outliers can have adverse effect on the model performance.

• For numerical variables: imputation using mean or median

• For categorical variables: imputation using mode

# Imputating Missing value with mode for categorical features

df['Credit\_History'].fillna(df['Credit\_History'].mode()[0],inplace=True)

df['Self\_Employed'].fillna(df['Self\_Employed'].mode()[0],inplace=True)

df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)

df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)

df['Married'].fillna(df['Married'].mode()[0],inplace=True)

# Imputation of Numerical features

df['Loan\_Amount\_Term'].fillna(df['Loan\_Amount\_Term'].mode()[0],inplace=True)

df['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)

**3.6.2 Outlier Treatment**

As we saw earlier in univariate analysis, LoanAmount contains outliers so we have to treat them as the presence of outliers affects the distribution of the data. Due to these outliers bulk of the data in the loan amount is at the left and the right tail is longer. This is called right skew-ness. One way to remove the skew-ness is by doing the log transformation. As we take the log transformation, it does not affect the smaller values much, but reduces the larger values. So, we get a distribution similar to normal distribution

df['LoanAmount\_log'] = np.log(df['LoanAmount'])

df['LoanAmount\_log'].hist(bins=20)

df['LoanAmount\_log'] = np.log(df['LoanAmount'])

A graph with blue squares

Description automatically generatedNow the distribution looks much closer to normal and effect of extreme values has been significantly subsided.

**3.6.3 Correlation**

df.corr()

plt.figure(figsize=(10,6))

sns.heatmap(df.corr(),linewidths=0.1,fmt=".1g",linecolor="black",annot=True,cmap="YlOrRd\_r")

plt.yticks(rotation=0);

plt.show()

A red and yellow squares with white text

Description automatically generated

plt.figure(figsize = (18,6))

df.corr()['Loan\_Status'].drop(['Loan\_Status']).sort\_values(ascending=False).plot(kind='bar',color = 'purple')

plt.xlabel('Features',fontsize=15)

plt.ylabel('Income',fontsize=15)

plt.title('Correlation of features with Target Variable Loan\_Status',fontsize = 18)

plt.show()

A graph with purple lines

Description automatically generated

Credit\_history highly correlated to target variable and other variable are moderate correlated to LoanStatus

ApplicantIncome highly positively correlated to LloanAmount

**3.7 Model Building**

After creating new features, we can continue the model building process. So we will start with logistic regression model and then move over to more complex models like Random Forest and XGBoost. We will build the following models.

• Logistic Regression

• Decision Tree

• Random Forest

• XGBoost

Let’s prepare the data for feeding into the models.

# Splitting data in target and dependent feature

x = df.drop(['Loan\_Status'], axis =1)

y = df['Loan\_Status']

**Machine Learning Model Building**

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report,roc\_curve

from sklearn.linear\_model import LogisticRegression

maxaccu=0

maxrs=0

for i in range(1,200):

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x1,y1,test\_size=.30,random\_state=i)

ls=LogisticRegression()

ls.fit(x\_train,y\_train)

pred=ls.predict(x\_test)

acc=accuracy\_score(y\_test,pred)

if acc>maxaccu:

maxaccu=acc

maxrs=i;

print("Best accuracy score is ",maxaccu,"at random\_state ",maxrs)

Best accuracy score is 0.8057851239669421 at random\_state 16

x\_train,x\_test,y\_train,y\_test**=**train\_test\_split(x,y,test\_size**=**0.30,random\_state**=**maxrs)

In Train Test Split validation we split the training data into training and validating data as to evaluate our model. Since we don’ have the outcome for Test data so we can’t check its accuracy after predicting the outcome as we don’t have original outcomes to compare with. Train Test Split is a validation technique used to solve the above issue and hence evaluate our model.

**3.7.1 Logistic Regression**

Here we build our model using the popular Logistic Regression. Logistic regression is an estimation of Logit function. Logit function is simply a log of odds in Favor of the event.

LOGISTIC REGRESSION

from sklearn.linear\_model import LogisticRegression

LR=LogisticRegression()

LR.fit(x\_train,y\_train)

predLR=LR.predict(x\_test)

print(accuracy\_score(y\_test,predLR))

print(confusion\_matrix(y\_test,predLR))

print(classification\_report(y\_test,predLR))

0.8352272727272727

[[ 28 28]

[ 1 119]]

precision recall f1-score support

0 0.97 0.50 0.66 56

1 0.81 0.99 0.89 120

accuracy 0.84 176

macro avg 0.89 0.75 0.78 176

weighted avg 0.86 0.84 0.82 176

We got an accuracy score as 0.8352272727272727 for this model.

**3.7.2 Decision Tree**

Decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables. Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable

#EXTRA TREE CLASSIFIER

from sklearn.ensemble import ExtraTreesClassifier

ET=ExtraTreesClassifier()

ET.fit(x\_train,y\_train)

predET=ET.predict(x\_test)

print(accuracy\_score(y\_test,predET))

print(confusion\_matrix(y\_test,predET))

print(classification\_report(y\_test,predET))

0.7897727272727273

[[ 30 26]

[ 11 109]]

precision recall f1-score support

0 0.73 0.54 0.62 56

1 0.81 0.91 0.85 120

accuracy 0.79 176

macro avg 0.77 0.72 0.74 176

weighted avg 0.78 0.79 0.78 176

We got an accuracy score of 0.7897727272727273 for this model

**3.7.3 Random Forest**

#RandomForestClassifier

RFC=RandomForestClassifier()

RFC.fit(x\_train,y\_train)

predRFC=RFC.predict(x\_test)

print(accuracy\_score(y\_test,predRFC))

print(confusion\_matrix(y\_test,predRFC))

print(classification\_report(y\_test,predRFC))

0.8181818181818182

[[ 30 26]

[ 6 114]]

precision recall f1-score support

0 0.83 0.54 0.65 56

1 0.81 0.95 0.88 120

accuracy 0.82 176

macro avg 0.82 0.74 0.76 176

weighted avg 0.82 0.82 0.81 176

We got an accuracy score of 0.8181818181818182 for this model

We observe that Logistic Regression score is the highest and hence we will predict our Test data using Logistic Regression

**3.8 Use Case Diagram**

Here we can see area under curve for each model used

from sklearn.metrics import RocCurveDisplay

disp = RocCurveDisplay.from\_estimator(f\_model,x\_test,y\_test)

plt.legend(prop={'size':11}, loc='lower right')

plt.figure(figsize=(10,10))

plt.show()

print('\033[1m'+'Auc Score :'+'\033[0m\n',acc)

A green line graph with a white background

Description automatically generated

**3.9 save the model**

We have to save the model using ,pkl for further usage

# Saving the model

import joblib

joblib.dump(f\_model,"Loan\_Application\_Status\_project.pkl")

**3.9 Predict the saved model**

#predicting the saved model

model=joblib.load("Loan\_Application\_Status\_project.pkl")

a=pd.DataFrame([model.predict(x\_test)[:],y\_test[:]],index=["predicted","original"])

a

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **...** | **166** | **167** | **168** | **169** | **170** | **171** | **172** | **173** | **174** | **175** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **predicted** | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | ... | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| **original** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | ... | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

1. rows × 176 columns

**4 Conclusion**

1. Out of all the classification algorithms used on the dataset, Logistic Regression algorithm gives the best overall prediction accuracy.

2. Credit History, Balance Income, EMI, Property Area are the most important factors for predicting the class of the loan applicant (whether the applicant would be ‘approved’ or ‘not’).

3. 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