**Loan Prediction with Machine Learning**

Dissertation submitted in fulfillment of the requirements for the Degree of **BACHELOR OF TECHNOLOGY**

in

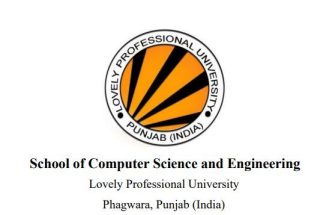
**COMPUTER SCIENCE AND ENGINEERING**

***With Specialization in Data Science (AI & ML*)**

**By**

**P. OMKAR**

**Reg.No.: 12018074**

**** GITHUB LINK: https://github.com/omkar3334/loanprediction

LOAN PREDICTION USING MACHINE LEARNING.

**0.0.1 Abstract: -**

**0.0.2 Loans are the core business of banks. The main profit comes directly from the loan’s interest. The loan companies grant a loan after an intensive process of verification and validation.However, they still don’t have assurance if the applicant is able to repay the loan with nodifficulties.In this tutorial,**

**we’ll build a predictive model to predict if an applicant is able to repay the lending company or not. We will prepare the data using Jupyter Notebook and use various models to predict the**

**target variable.**

**0.0.3 Introduction:-**

**0.0.4 Loans are the core business of banks. The main profit comes directly from the loan’s interest. The loan companies grant a loan after an intensive process of verification and validation.However, they still don’t have assurance if the applicant is able to repay the loan with nodifficulties.In this tutorial,**

**we’ll build a predictive model to predict if an applicant is able to repay the lending company or not. We will prepare the data using Jupyter Notebook and use various models to predict the**

**target variable.**

**0.0.5 Problem Statement & Dataset Information:-**

**0.0.6 Dream Housing Finance company deals in all home loans. They have presence across all urban, semi urban and rural areas. Customer first apply for home loan after that company validates the customer eligibility for loan. Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details areGender, Marital Status, Education,**

**Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers**

**0.0.7 segments, those are eligible for loan amount so that they can specifically target these customers.This is a standard supervised classification task.A classification problem where we have to predict whether a loan would be approved or not. Below is the dataset attributes with description.**

**0.0.8 Variable Description**

**0.0.9 Loan\_ID Unique Loan ID**

**0.0.10 Gender Male/ Female**

**0.0.11 Married Applicant married (Y/N)**

**0.0.12 Dependents Number of dependents**

**0.0.13 Education Applicant Education (Graduate/ Under Graduate)**

**0.0.14 Self\_Employed Self employed (Y/N)**

**0.0.15 ApplicantIncome Applicant income**

**0.0.16 CoapplicantIncome Coapplicant income 0.0.17 LoanAmount Loan amount in thousands 0.0.18 Loan\_Amount\_Term Term of loan**2 **in months**

**0.0.19 Credit\_History credit history meets guidelines 0.0.20 Property\_Area Urban/ Semi Urban/ Rural**

[1]:

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**from matplotlib import** pyplot **as** plt

**import matplotlib**

%matplotlib inline

**import warnings**

warnings.filterwarnings('ignore')

**0.0.23 Loading the dataset**

[2]:

df=pd.read\_csv(r"C:\Users\P.omkar\OneDrive\Desktop\LoanApprovalPrediction.csv")

**0.0.24 it will show top 5 rows of a dataset**

df.head()

[3]:

[3]: Loan\_ID Gender Married Dependents Education Self\_Employed \ 0 LP001002 Male No 0.0 Graduate No 1 LP001003 Male Yes 1.0 Graduate No 2 LP001005 Male Yes 0.0 Graduate Yes 3 LP001006 Male Yes 0.0 Not Graduate No 4 LP001008 Male No 0.0 Graduate No

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \ 0 5849 0.0 NaN 360.0 1 4583 1508.0 128.0 360.0 2 3000 0.0 66.0 360.0 3 2583 2358.0 120.0 360.0 4 6000 0.0 141.0 360.0

Credit\_History Property\_Area Loan\_Status

0 1.0 Urban Y

1 1.0 Rural N

2 1.0 Urban Y

3 1.0 Urban Y

4 1.0 Urban Y

[4]:

df.tail()

[4]: Loan\_ID Gender Married Dependents Education Self\_Employed \ 593 LP002978 Female No 0.0 Graduate No 594 LP002979 Male Yes 3.0 Graduate No 595 LP002983 Male Yes 1.0 Graduate No 596 LP002984 Male Yes 2.0 Graduate No 597 LP002990 Female No 0.0 Graduate Yes

3

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \ 593 2900 0.0 71.0 360.0 594 4106 0.0 40.0 180.0 595 8072 240.0 253.0 360.0 596 7583 0.0 187.0 360.0 597 4583 0.0 133.0 360.0

Credit\_History Property\_Area Loan\_Status

593 1.0 Rural Y

594 1.0 Rural Y

595 1.0 Urban Y

596 1.0 Urban Y

597 0.0 Semiurban N

[5]:

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 598 entries, 0 to 597

Data columns (total 13 columns):

# Column Non-Null Count Dtype

0 Loan\_ID 598 non-null object

1 Gender 598 non-null object

2 Married 598 non-null object

3 Dependents 586 non-null float64

4 Education 598 non-null object

5 Self\_Employed 598 non-null object

6 ApplicantIncome 598 non-null int64

7 CoapplicantIncome 598 non-null float64

8 LoanAmount 577 non-null float64

9 Loan\_Amount\_Term 584 non-null float64

10 Credit\_History 549 non-null float64

11 Property\_Area 598 non-null object

12 Loan\_Status 598 non-null object

dtypes: float64(5), int64(1), object(7)

memory usage: 60.9+ KB

df.describe()

[6]:

[6]: Dependents ApplicantIncome CoapplicantIncome LoanAmount \ count 586.000000 598.000000 598.000000 577.000000 mean 0.755973 5292.252508 1631.499866 144.968804 std 1.007751 5807.265364 2953.315785 82.704182 min 0.000000 150.000000 0.000000 9.000000 25% 0.000000 2877.500000 0.000000 100.000000 50% 0.000000 3806.000000 1211.500000 127.000000

4

75% 1.750000 5746.000000 2324.000000 167.000000 max 3.000000 81000.000000 41667.000000 650.000000

Loan\_Amount\_Term Credit\_History

count 584.000000 549.000000

mean 341.917808 0.843352

std 65.205994 0.363800

min 12.000000 0.000000

25% 360.000000 1.000000

50% 360.000000 1.000000

75% 360.000000 1.000000

max 480.000000 1.000000

**0.0.25 from below we can say there are 598 rows and 13 coloumns in a dataset** df.shape

[7]:

[7]: (598, 13)

**0.0.26 checking the null values from the dataset and removing the null values**

[8]:

df.isna().sum()

[8]: Loan\_ID 0 Gender 0 Married 0 Dependents 12 Education 0 Self\_Employed 0 ApplicantIncome 0 CoapplicantIncome 0 LoanAmount 21 Loan\_Amount\_Term 14 Credit\_History 49 Property\_Area 0 Loan\_Status 0 dtype: int64

[9]: [10]:

df=df.dropna()

**0.0.27 removed all the null values** df.isna().sum()

[10]: Loan\_ID 0 Gender 0

5

[11]:

Married 0 Dependents 0 Education 0 Self\_Employed 0 ApplicantIncome 0 CoapplicantIncome 0 LoanAmount 0 Loan\_Amount\_Term 0 Credit\_History 0 Property\_Area 0 Loan\_Status 0 dtype: int64

df

[11]: Loan\_ID Gender Married Dependents Education Self\_Employed \ 1 LP001003 Male Yes 1.0 Graduate No 2 LP001005 Male Yes 0.0 Graduate Yes 3 LP001006 Male Yes 0.0 Not Graduate No 4 LP001008 Male No 0.0 Graduate No 5 LP001011 Male Yes 2.0 Graduate Yes .. … … … … … …

593 LP002978 Female No 0.0 Graduate No 594 LP002979 Male Yes 3.0 Graduate No 595 LP002983 Male Yes 1.0 Graduate No 596 LP002984 Male Yes 2.0 Graduate No 597 LP002990 Female No 0.0 Graduate Yes

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \ 1 4583 1508.0 128.0 360.0 2 3000 0.0 66.0 360.0 3 2583 2358.0 120.0 360.0 4 6000 0.0 141.0 360.0

5

..

593

5417

…

2900

4196.0

…

0.0

267.0

…

71.0

360.0

…

360.0

594 4106 0.0 40.0 180.0 595 8072 240.0 253.0 360.0 596 7583 0.0 187.0 360.0 597 4583 0.0 133.0 360.0

Credit\_History Property\_Area Loan\_Status

1 1.0 Rural N

2 1.0 Urban Y

3 1.0 Urban Y

4 1.0 Urban Y

5 1.0 Urban Y

6

[12]:

.. … … … 593 1.0 Rural Y 594 1.0 Rural Y 595 1.0 Urban Y 596 1.0 Urban Y 597 0.0 Semiurban N

[505 rows x 13 columns]

**0.0.28 finding the unique values in a dependent** df['Dependents'].unique()

[12]: array([1., 0., 2., 3.])

[13]:

df['Dependents'].value\_counts()

[13]: 0.0 289

2.0 90

1.0 84

3.0 42

Name: Dependents, dtype: int64 **0.0.29 exploratory data analysis**

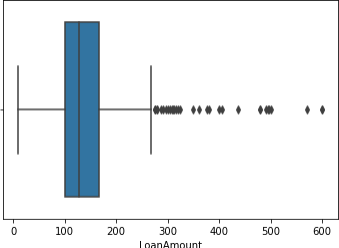
[14]:

sns.boxplot(df['LoanAmount'])

[14]: <AxesSubplot:xlabel='LoanAmount'>

7

[15]: [16]:



print(np.where(df["LoanAmount"]>280))

(array([ 8, 18, 28, 43, 55, 105, 125, 144, 212, 230, 255, 264, 265, 272, 285, 300, 303, 311, 336, 356, 401, 422, 430, 431, 440, 461, 496], dtype=int64),)

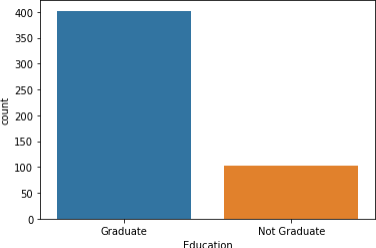
*#visualizing eduaction*

sns.countplot(df['Education'])

[16] : <AxesSubplot:xlabel='Education', ylabel='count'>

8

[17]:



**0.0.30 From the above most of the educated person are applied from the loan**

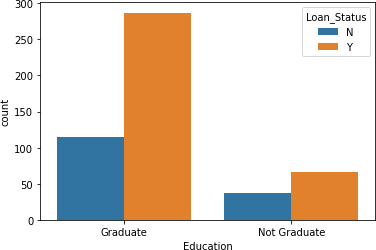
*#combining educatin status with loan status*

sns.countplot(x = 'Education',hue = 'Loan\_Status',data=df)

[17] : <AxesSubplot:xlabel='Education', ylabel='count'>

9

[18]:

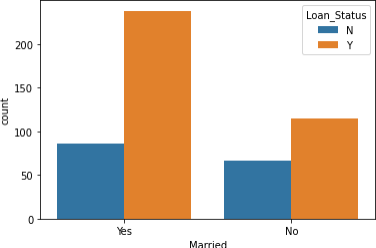


**0.0.31 from the above figure loan status was approved for the educated persons** sns.countplot(x = 'Married',hue = 'Loan\_Status',data=df)

[18] : <AxesSubplot:xlabel='Married', ylabel='count'>

10

[19]:



**0.0.32 from the above figure loan status was approved for the married persons 0.0.33 finidng correlation between the variables using heatmap**

*#finidng correlation between the variables using heatmap*

corr = df.corr()

plt.figure(figsize=(15,10))

sns.heatmap(corr, annot = **True**, cmap="BuPu")

[19] : <AxesSubplot:>

11

[20]:

df['Married'].unique()

[20] : array(['Yes', 'No'], dtype=object) Encoding the categories values

[21]:

df.head()

[21] : Loan\_ID Gender Married Dependents Education Self\_Employed \ 1 LP001003 Male Yes 1.0 Graduate No 2 LP001005 Male Yes 0.0 Graduate Yes 3 LP001006 Male Yes 0.0 Not Graduate No 4 LP001008 Male No 0.0 Graduate No 5 LP001011 Male Yes 2.0 Graduate Yes

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \ 1 4583 1508.0 128.0 360.0 2 3000 0.0 66.0 360.0 3 2583 2358.0 120.0 360.0 4 6000 0.0 141.0 360.0 5 5417 4196.0 267.0 360.0

12

[22]: [23]:

Credit\_History Property\_Area Loan\_Status

1 1.0 Rural N

2 1.0 Urban Y

3 1.0 Urban Y

4 1.0 Urban Y

5 1.0 Urban Y

**0.0.34 replace the categorical coloumns of married,gender ,education,self em ployed,property\_area into 1 or 0 for the convience of predictions**

df.replace({'Married':{'Yes':1,'No':0},'Gender':{'Male':1,'Female': ↪0},'Education':{'Graduate':1,'Not Graduate':0},'Self\_Employed':{'Yes':1,'No': ↪0},'Property\_Area':{'Rural':1,'Urban':0,'Semiurban':0}},inplace=**True**)

df.head()

[23]: Loan\_ID Gender Married Dependents Education Self\_Employed \ 1 LP001003 1 1 1.0 1 0 2 LP001005 1 1 0.0 1 1 3 LP001006 1 1 0.0 0 0 4 LP001008 1 0 0.0 1 0 5 LP001011 1 1 2.0 1 1

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \ 1 4583 1508.0 128.0 360.0 2 3000 0.0 66.0 360.0 3 2583 2358.0 120.0 360.0 4 6000 0.0 141.0 360.0 5 5417 4196.0 267.0 360.0

Credit\_History Property\_Area Loan\_Status

1 1.0 1 N

2 1.0 0 Y

3 1.0 0 Y

4 1.0 0 Y

5 1.0 0 Y

**0.0.35 converting the data type of dependents of float into int**

[53]: [54]: [55]:

df['Dependents']=df['Dependents'].astype('int') X=df.iloc[: ,2 :-1].values

X[0]

13

[55]: array([1.000e+00, 1.000e+00, 1.000e+00, 0.000e+00, 4.583e+03, 1.508e+03, 1.280e+02, 3.600e+02, 1.000e+00, 1.000e+00])

[56]: [57]: [58]:

df.replace({'Loan\_Status':{'Y':1,'N':0}},inplace=**True**) Y=df.iloc[: ,-1].values

Y

[58]: array([0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],

dtype=int64)

[59]:

X

[59]: array([[ 1., 1., 1., …, 360., 1., 1.], [ 1., 0., 1., …, 360., 1., 0.],

[ 1., 0., 0., …, 360., 1., 0.],

…,

[ 1., 1., 1., …, 360., 1., 0.],

[ 1., 2., 1., …, 360., 1., 0.],

[ 0., 0., 1., …, 360., 0., 0.]])

14

[31]: [32]:

**0.0.36 Train-Test Split**

**from sklearn.model\_selection import** train\_test\_split x\_train,x\_test,y\_train,y\_test=train\_test\_split(X,Y,test\_size=0. ↪25,random\_state=42)

x\_train.shape

[32]: (378, 10)

[33]:

x\_test.shape

[33]: (127, 10)

**0.0.37 Model Training**

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

[34]:

**from sklearn.linear\_model import** LogisticRegression log\_classifier=LogisticRegression()

log\_classifier.fit(x\_train ,y\_train)

[34]: LogisticRegression()

[35]: [36]:

log\_y\_pred=log\_classifier.predict(x\_test)

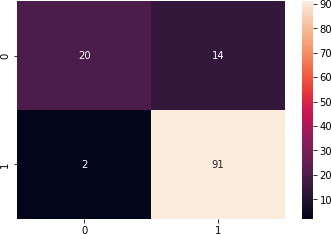
**from sklearn.metrics import** confusion\_matrix cm=confusion\_matrix(y\_test,log\_y\_pred) cm

sns.heatmap(cm,annot=**True**)

[36]: <AxesSubplot:>

15

[37]:



**from sklearn.metrics import** accuracy\_score

accuracy\_score(y\_test,log\_y\_pred)

[37]: 0.8740157480314961

[38]:

**from sklearn.neighbors import** KNeighborsClassifier k\_classifier=KNeighborsClassifier()

k\_classifier.fit(x\_train,y\_train)

[38]: KNeighborsClassifier()

[39]: [40]:

k\_y\_pred=k\_classifier.predict(x\_test)

sns.heatmap(confusion\_matrix(y\_test,k\_y\_pred),annot=**True**)

[40]: <AxesSubplot:>

16

[41]:

accuracy\_score(y\_test,k\_y\_pred)

[41]: 0.7244094488188977

[42]:

**from sklearn.svm import** SVC

s\_classifier=SVC(kernel='rbf',random\_state=42) s\_classifier.fit(x\_train,y\_train)

[42]: SVC(random\_state=42)

[43]: [44]:

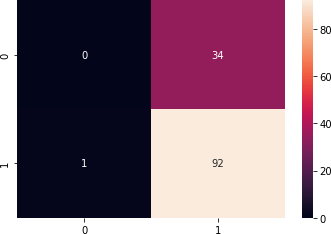
s\_y\_pred=s\_classifier.predict(x\_test)

sns.heatmap(confusion\_matrix(y\_test,s\_y\_pred),annot=**True**)

[44]: <AxesSubplot:>

17

[46]:

accuracy\_score(y\_test,s\_y\_pred)

[46]: 0.7244094488188977

**0.0.39 RandomForest Classifier:-**

**0.0.40 RandomForest is a tree-based bootstrapping algorithm wherein a certain no. of weak learners (decision trees) are combined to make a powerful prediction model.For every individual learner, a random sample of rows and a few ran domly chosen variables are used to build a decision tree model.Final prediction can be a function of all the predictions made by the individual learners.**

[47]:

**from sklearn.ensemble import** RandomForestClassifier

classifier=RandomForestClassifier(n\_estimators=25,criterion='entropy') classifier.fit(x\_train,y\_train)

[47]: RandomForestClassifier(criterion='entropy', n\_estimators=25)

[48]:

y\_pred=classifier.predict(x\_test)

18

[49]:

Confusion Matrix

A confusion matrix is a summary of prediction results on a classification prob lem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that

are being made.

sns.heatmap(confusion\_matrix(y\_test,y\_pred),annot=**True**)

[49]: <AxesSubplot:>



[61]:

accuracy\_score(y\_test,y\_pred)

[61]: 0.8503937007874016

**Conclusion**

We have built our classification model and prediction,

we notice that Logistic Regression algorithm gives the :

best results for our dataset, the accuracy results are around 87% and 85% with Random forest classifier

19

20