LOAN PREDICTION MODEL IN PYTHON

Mainly, we need to use the Supervised Machine Learning Algorithms of Classification and regression for Loan Prediction.

Algorithms Can be Used, For Ex., –

1. ***Decision Tree Classification(Low Accuracy)*** – create Tree with different conditions in **Parent Node** and gives output in **Leaf Node** as YES or NO i.e. Is he/she able to approve the loan or not.

Ex. Credit History > 0

Annual Income > 4Lakhs

Then Only, Approve Loan Otherwise Reject.

2. ***Naive Bayes Classification(Gives better Accuracy) –*** Probabilistic Machine Learning Algorithm based on the **Bayes theorem.**

Formulae- $P(A|B) = (P(B|A) P(A)) / P(B)$

$P(A|B)$ – Likelihood of occurrence of A,

$P(B|A)$ – Likelihood of occurrence of B, $P(A)$ – probability of Occurrence of A, $P(B)$ – probability of Occurrence of B.

Python Libraries used in this Project –

1. Pandas-

Powerful, fast and flexible library for **Data Analysis, manipulation and filtering.**
Read and write the CSV File and store in the Python object.
It is also used for viewing and selecting the Data.

2. NumPy-

Used to perform **mathematical and logical operations** on Arrays.
Contains Multi-dimensional Arrays and Matrices.

3. Matplotlib-

Comprehensive library for creating **static, interactive and animated visualizations.**
Creating interactive **Charts, map and Graphs.**

4. Scikit-learn-

Most Powerful Library with efficient tools for **ML and statistical modeling** including range of supervised and unsupervised learning algorithms.

Steps need to follow while making model –

1. Read CSV file using **PANDAS** function- **read_csv()**, and store in the Python Object.

2. Then, we are able to find the **head(), shape, info(), describe()** of our dataset using that Python Object.

3. See how Credit_History **affects** Loan_Status –
   By using **crosstab()** function in **PANDAS** library.

   pd.crosstab(data['Credit_History'] , data['Loan_Status'] , margin=True)

4. We are able to use different functions for **Visualization**-
   i. **data.boxplot(column="")** – Used for Plotting.
   ii. **data['column_name'].hist(bins=20)**- Used to draw **Histogram**.
   iii. **Data.boxplot(column="", by="")** – Used to check if that one column values will really related to the other column values.

5. If we want to **normalize** the column which has numeric values, then we use the **log()** function from **NumPy** Library.

   data['column_name_log'] = np.log(data['column_name'])

6. Then, **check** is there any **missing values** in all columns using function **isnull()** and to add total no. of each column we use **sum()**.

print(data.isnull().sum())

7. **Handle these missing values** by filling them- using **fillna()**

i. Using **mode()** – If our column store the value in some **categories** like, **gender is in male or female, Married is in yes/no.** Then, we use this function to fill missing place with obtained mode value. We need to fill one by one column which has missing values.

data['column_name'].fillna(data['column_name'].mode()[0],inplace=True)

ii. Using **mean()** – If our column store the value in **numeric value** like, **LoanAmount.** Then, we use this function to fill missing place with mean value. We need to fill one by one column which has missing values.

data.Column_Name = data.Column_Name.fillna(data.Column_Name.mean())

Successfully Handled Missing Values.

AFTER ALL HANDLING THESE MISSING VALUES, THERE IS NO LEFT MISSING VALUE IN ANY COLUMN, IT GIVES RESULT '0' WHEN WE CHECK FOR NULL USING data.isnull().sum() function.

8. Add **“ApplicantIncome”** and **“CoapplicantIncome”** columns in single column named **“TotalIncome”**.

   data[‘TotalIncome’] = data[‘ApplicantIncome’] + data[‘CoapplicantIncome’]

   This will create Another column named - TotalIncome

9. Create **log** of **“TotalIncome”** column values by using **NumPy** Library which create another column named- **“TotalIncome_log”**.

   data[‘TotalIncome_log’] = np.log(data[‘TotalIncome’]);

10. **Separate Independent variables and Dependant variable** in different Python Objects (x & y).

    Independent Variable – Those Variables which are not affected by any other column values (Ex. **Gender, married, LoanAmount, TotalIncome**..,etc.)

    X = data.iloc(:, np.r_[1:5, 9:11, 13:15]).values
    1,5,9,11,13,15 are column indices_1:5 gives values of 1 To 5 indexed columns.

    Dependant Variable – Those Variables which are affected by other column values (Ex. **Loan_Status** which is depend on Credit_History, TotalIncome,,..etc.)

    Y = data.iloc(:,12).values # 12th column values ONLY.

11. **Split** the **Train & Test DataSet** by using **train_test_split** from **sklearn.model_selection -**

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=0)

 # test_size=0.2 for the train:test ratio – (80:20)%

12. Convert **Text Data into Numeric data** using **LabelEncoder** from **sklearn.preprocessing** because, machine understand only numeric Data.

i. Create instance of LabelEncoder

Label_encoder_X = LabelEncoder() – created to encode X_train text data.

ii. **Select column** which has text data and then convert it into numeric as –

X_train[:, 7] = Label_encoder_X.fit_transform(X_train[:, 7)

 # we can use for-loop for multiple columns present sequencially.

iii. Same for Y_train, X_test, Y_test Datasets, convert all text data into numeric one using **LabelEncoder.**

13. Now, we need to **scale our training and testing dataset(X_train and Y_train)** using **StandardScaler** from **sklearn.preprocessing** library.

```
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.fit_transform(X_test)
```

14. **NOW, FINALLY WE COME TO APPLY THE CLASSIFICATION ALGORITHMS ON TRAINED DATASET.**

A. 1st Algorithm:DecisionTreeClassifier–from **sklearn.tree**

```
# first create the instance of DecisionTreeClassifier
DTClassifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
```

 # Now, we are providing trained dataset to the Algorithm that means, for independent_variables which are X_train, we need dependant_variable as Y_train

```
DTClassifier.fit(X_train,Y_train)
```

 # PREDICTION OF CORRECT OUTPUT - 'Y' BY GIVING X_test DATASET TO PREDICT

```
Y_pred = DTClassifier.predict(X_test);
```

 # TO CHECK THE ACCURACY OF THE ALGORITHM PREDICTION – USE metrics from **sklearn**. - **GIVES 70% ACCURACY**

```
metrics.accuracy_score(Y_pred, Y_test)  # check similarity between these.
```

B. 2nd Algorithm: Naïve bayes Algorithm –

Library - GaussianNB from sklearn.naive_bayes

CREATE instance of **GaussianNB-**

NBclassifier = GaussianNB()

 # Now, we are providing trained dataset to the Algorithm that means, for independent_variables which are X_train, we need dependant_variable as Y_train

NBClassifier.fit(X_train,Y_train)

 # PREDICTION OF CORRECT OUTPUT - 'Y' BY GIVING X_test DATASET TO PREDICT **Y_pred_2 = NBClassifier.predict(X_test);**

 # TO CHECK THE ACCURACY OF THE ALGORITHM PREDICTION – USE metrics from **sklearn**. – **GIVES 83% ACCURACY**

metrics.accuracy_score(Y_pred_2, Y_test) # check similarity between these.

15. FINALLY, we come on the test_data to check whether our algorithm works fine with unknown dataset, REPEAT ONE BY ONE ALL PROCESS FROM READING CSV_FILE TO APPLYING ALGORITHM. Following is the Short Overview –

In TEST_DATASET, we have only INDEPENDENT_VARIABLES and our target is to find the CORRECT DEPENDANT_VARIABLE for Particular row.

i. **Read_csv file** of test_dataset
ii. **Check for Null Values using isnull().sum()**
iii. **Fill the null values by using fillna(mode() / mean())**
iv. **Where necessary, mainly for numeric values of column apply log on that column and create another column named column_name_log**

Like, Here, for Loan_Amount, TotalIncome, etc.

v. Diving independent & dependant variable is **only while training** and **for testing** we **separate ONLY needed independent columns** in single object by using **dataset.iloc[]**
vi. **Convert Text data into Numeric data** from the above retrieved columns using **LabelEncoder.**
vii. **Scale the dataset** using **StandardScaler.**
viii. **THEN, FINALLY PREDICT FOR Y using NBClassifier or DTClassifier.predict(test), but NBClassifier is BEST between these two.**