Loan Prediction using Machine Learning

* Project Statement

The idea behind this project is to build a model that will classify how much loan the user can take. It is based on the user’s marital status, education, number of dependents, and employments.

* Approach

I have used K-Nearest Neighbors (KNN) to predict the loan status of the applicant and linear model to estimate how much the user can given the user's details.

Algorithm used:

* K-Nearest Neighbors (KNN) Algorithm

K-Nearest Neighbors (KNN) is one of the simplest algorithms used in Machine Learning for regression and classification problem. KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbors. The data is assigned to the class which has the nearest neighbors. As you increase the number of nearest neighbors, the value of k, accuracy might increase.

* Now, let us understand the implementation of K-Nearest Neighbors (KNN) in Python

### Import the Libraries

We will start by importing the necessary libraries required to implement the KNN Algorithm in Python. We will import the \*\*numpy\*\* libraries for scientific calculation. Next, we will import the \*\*matplotlib.pyplot\*\* library for plotting the graph. We will import two machine learning libraries KNeighborsClassifier from sklearn.neighbors to implement the k-nearest neighbors vote and accuracy score from\*\*sklearn.metrics\*\* for accuracy classification score. \*\*Seaborn\*\* is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. \*\*pandas\*\* is a Python package providing fast, flexible, and expressive data structures.

**Various steps involved in below program:**

1.loading data sets

2.data analyzing and preprocessing

3.data visualization

4.training model using independent features

5.prediction depended values

**PROGRAMMING CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

`

1.loading datasets:

Importing datasets(loan\_train) by using pd.read\_csv() function and store those data in “df” data frame

### Read the data

`

df=pd.read\_csv("loan\_train.csv")

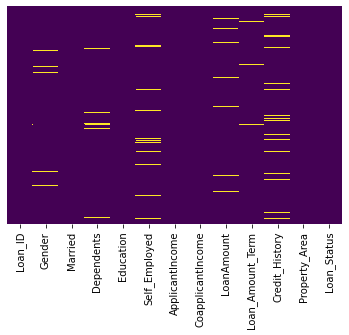
df.head()

#Heat map showing values which are not available or null

`

sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')

`



2.data analyzing and preprocessing:

Here we analyzing the data and finding duplicate values using get\_dummies() functions and drop those values by using df.drop() function.and also eliminate the null valuesby using the dropna() function

We have to eliminate these null values

The get\_dummies() function is used to convert categorical variable into dummy/indicator variables (0's and 1's).

`

sex = pd.get\_dummies(df['Gender'],drop\_first=True)

married = pd.get\_dummies(df['Married'],drop\_first=True)

education = pd.get\_dummies(df['Education'],drop\_first=True)

self\_emp = pd.get\_dummies(df['Self\_Employed'],drop\_first=True)

loanStatus = pd.get\_dummies(df['Loan\_Status'],drop\_first=True)

PropArea = pd.get\_dummies(df['Property\_Area'],drop\_first=True)

df.drop(['Gender','Loan\_ID','Self\_Employed','Married','Education','Loan\_Status','Property\_Area'],axis=1,inplace=True)

df = pd.concat([df,married,education,sex,self\_emp,loanStatus,PropArea],axis=1)

df.head()

`

#Rename the columns after data manipulation.

`

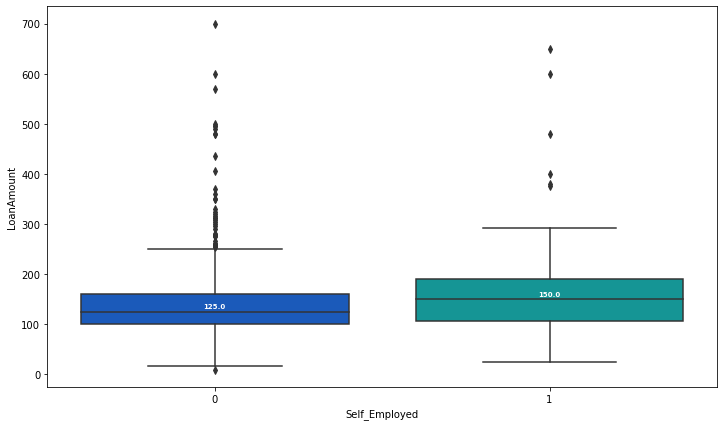
df=df.rename(columns={'Yes':'Married','Male':'Gender','Yes':'Self\_Employed'})

```

#Heat map after introducing dummy variables.



* We can see that the attribute LoanAmount contains null values. We identify that the Loan Amount is related to whether the user is self employed. This is represented by the following Box-Plot



### We write the impute\_LoanAmount function to replace the null Loan Amount values with the values calculated by the function

```

def impute\_LoanAmt(cols):

Loan = cols[0]

selfemp = cols[1]

if pd.isnull(Loan):

if selfemp == 1:

return 150

else:

return 125

else:

return Loan

``

3.Data visualization:

Using matplotlib and visualize realtion between dependent and independent features

And also analyze the data by using data visualization

#Apply the impute\_LoanAmt function on the dataset

`

df['LoanAmount'] = df[['LoanAmount','Self\_Employed']].apply(impute\_LoanAmt,axis=1)

sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')

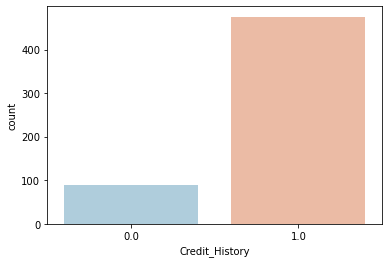
`

### From the below count plot we can see that most of the credit history values are 1.0. So we can replace the null values by 1.0.

`

sns.countplot(x='Credit\_History',data=df,palette='RdBu\_r')

`



`

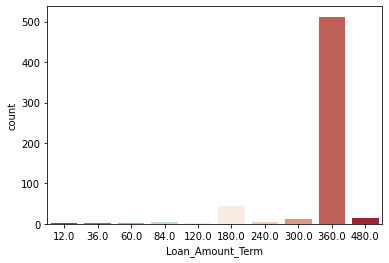
df['Credit\_History'].fillna(1.0,inplace=True)

`

### From the below count plot we can see that most of the Loan amount term values are 360. So we can replace the null values by 360.

`

sns.countplot(x='Loan\_Amount\_Term',data=df,palette='RdBu\_r')



`

df['Loan\_Amount\_Term'].fillna(360,inplace=True)

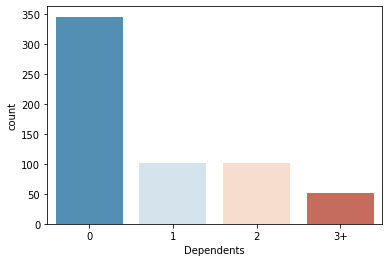
`

### From the below count plot we can see that most of the 'Dependents' values are 0. So we can replace the null values by 0.

`

sns.countplot(x='Dependents',data=df,palette='RdBu\_r')

```

```

df['Dependents'].fillna(360,inplace=True)

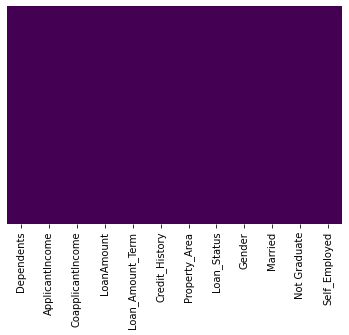
```

### Heatmap aftet eliminating all the null values

`

sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')

`



* StandardScaler will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.Assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

`

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

train=pd.DataFrame(df.drop('Loan\_Status',axis=1))

scaler.fit(train)

scaled\_features = scaler.transform(train)

df\_feat = pd.DataFrame(scaled\_features,columns=train.columns)

`4.training model

by using train test split function here test train split the loan dataset can de divide into 70 %training data and 30 %testing data

### Train/Test Split

The data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) in order to test our model’s prediction on this subset.

\_#Here we split the dataset 70/30....70% training dataset, 30% testing dataset.\_

`

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(scaled\_features,df['Loan\_Status'],test\_size=0.30)

`

### Now let predict the loan status using KNN algorithm.

`

from sklearn.neighbors import KNeighborsClassifier

`

* Let fit training dataset into the model. Let the initial value of k be 3.

`

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X\_train,y\_train)

`

#Predict the values using test dataset.

`

pred = knn.predict(X\_test)

`

#Print the classification reprot and confusion matrix

`

from sklearn.metrics import classification\_report,confusion\_matrix

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

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

`

* Confusion matrix

[[ 30 34]

[ 14 107]]

* Classification report

```

precision recall f1-score support

0 0.68 0.47 0.56 64

1 0.76 0.88 0.82 121

accuracy 0.74 185

macro avg 0.72 0.68 0.69 185

weighted avg 0.73 0.74 0.73 185

`

* To find a optimum value of k we plot a graph of error rate vs k value ranging from 0 to 40.

`

error\_rate = []

for i in range(1,40):

knn = KNeighborsClassifier(n\_neighbors=i)

knn.fit(X\_train,y\_train)

pred\_i = knn.predict(X\_test)

error\_rate.append(np.mean(pred\_i != y\_test))

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

plt.plot(range(1,40),error\_rate,color='blue', linestyle='dashed', marker='o',

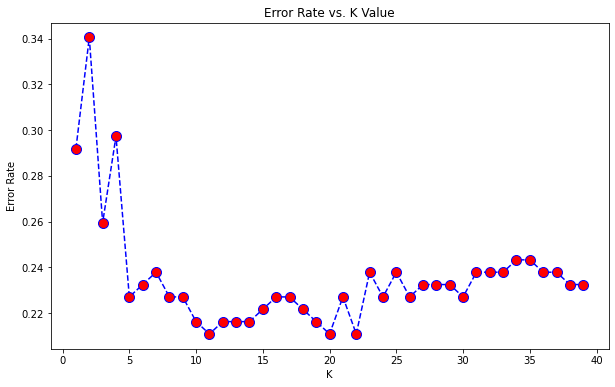
markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

`



* So from the graph wecan deduce that k=10 has least error rate. Hence we re\_apply the algorithm for k=10.

`

knn = KNeighborsClassifier(n\_neighbors=10)

knn.fit(X\_train,y\_train)

`

### Print the accuracy score

`

from sklearn.metrics import accuracy\_score

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

`

### We can see that the accuracy score has increased from 0.74 to 0.79.

# Linear regression

Linear regression is probably one of the most important and widely used regression techniques. It’s among the simplest regression methods. One of its main advantages is the ease of interpreting results.

* Now we use linear regression to predict the loan amount based on user's details.

#Assign the train and test datasets.

`

x=df[['Married', 'Not Graduate', 'Dependents',

'Self\_Employed','ApplicantIncome','CoapplicantIncome','Semiurban','Urban','Loan\_Amount\_Term']]

y=df['LoanAmount']

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=101)

`

`

from sklearn.linear\_model import LinearRegression

`

#Train the model to predict the values.

`

lm=LinearRegression()

lm.fit(X\_train,Y\_train)

`

Now we find the intercept

`

print(lm.intercept\_)

`

>> 53.85642282094781

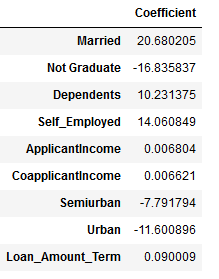
* Now we print the values of coefficients. These coefficients can be used to find the value of loan amount that the user can take given the user's details

`

coeff=pd.DataFrame(lm.coef\_,x.columns,columns=['Coefficient'])

coeff

```



* Now let us predict the loan amount for a random set of values.

`

lm.predict([[1,0,3,1,4000,3000,0,1,360]])

```

>> 187.17168163

* Bayesian Ridge regression

Bayesian regression allows a natural mechanism to survive insufficient data or poorly distributed data by formulating linear regression using probability distributors rather than point estimates. The output or response ‘y’ is assumed to drawn from a probability distribution rather than estimated as a single value.

`

from sklearn import linear\_model

bm=linear\_model.BayesianRidge()

bm.fit(X\_train,Y\_train)

BayesianRidge(alpha\_1=1e-06, alpha\_2=1e-06, alpha\_init=None,

compute\_score=False, copy\_X=True, fit\_intercept=True,

lambda\_1=1e-06, lambda\_2=1e-06, lambda\_init=None, n\_iter=300,

normalize=False, tol=0.001, verbose=False)

`

* Now let us predict the loan amount for a random set of values.

`

bm.predict([[1,0,3,1,4000,3000,0,1,360,1.0]])

`

>> 143.98938586

* XGB regression

Regression predictive modeling problems involve predicting a numerical value such as a dollar amount or a height. XGBoost can be used directly for regression predictive modeling.

```

from xgboost import XGBRegressor

model=XGBRegressor()

model.fit(X\_train,Y\_train)

XGBRegressor(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,

colsample\_bynode=1, colsample\_bytree=1, gamma=0,

importance\_type='gain', learning\_rate=0.1, max\_delta\_step=0,

max\_depth=3, min\_child\_weight=1, missing=None, n\_estimators=100,

n\_jobs=1, nthread=None, objective='reg:linear', random\_state=0,

reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, seed=None,

silent=None, subsample=1, verbosity=1)

`

* Now let us predict the loan amount for a random set of values.

`

yhat = model.predict([[1,0,3,1,4000,3000,0,1,360,1.0]])

`

>> 155.877

# Random forest regressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max\_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

```

from sklearn.ensemble import RandomForestRegressor

regr = RandomForestRegressor(max\_depth=2, random\_state=0)

regr.fit(X\_train,Y\_train)

RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0, criterion='mse',

max\_depth=2, max\_features='auto', max\_leaf\_nodes=None,

max\_samples=None, min\_impurity\_decrease=0.0,

min\_impurity\_split=None, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=100, n\_jobs=None, oob\_score=False,

random\_state=0, verbose=0, warm\_start=False)

5.Testing the data:

Here test the model by using remaining 30%data and predict the loan\_status by using dependent features like gender , self-employed ,property…..

Identify the model score by using the function modle.score().if the score is nearly ”one” the model predicted accuracy values and the given model is correct

* Now let us predict the loan amount for a random set of values.

`

regr.predict([[1,0,3,1,4000,3000,0,1,360,1.0]])

```

>> 123.46250175

## Model performance comparision

* Max\_error

The max\_error function computes the maximum residual error , a metric that captures the worst case error between the predicted value and the true value. In a perfectly fitted single output regression model, max\_error would be 0 on the training set and though this would be highly unlikely in the real world, this metric shows the extent of error that the model had when it was fitted.

Variance score

The explained variance score explains the dispersion of errors of a given dataset. Scores close to 1.0 are highly desired, indicating better squares of standard deviations of errors.

