Chapter 11: Credit Analytics – A Loan Prediction Model

The materials in the first part of this chapter are from a GitHub project *https://github.com/joymnyaga/CreditAnalytics-Loan-Prediction*. But there are a couple of similar projects using the same dataset online, for example, https://www.codespeedy.com/loan-prediction-project-using-machine-learning-in-python/ and https://medium.com/devcareers/loan-prediction-using-selected-machine-learning-algorithms-1bdc00717631.

For simplicity, we'll follow the GitHub project but you can also go over the other two websites and see other ways of doing things.

In the second part, you'll learn how to use deep neural networks to predict loan approvals.

In the third part, I'll discuss the potential problems in the analyses in the GitHub project, namely, the number of Ys and Ns are disproportional in the dataset. I'll discuss the concept of ROC curve and how to calculate the AUC-ROC score. This is important when you predict rare events such as credit card frauds, or predicting cancerous cells based on X-ray images and so on.

Part 1:

Credit-Analysis (Machine-Learning)

The following project aims to predict the eligibility of loan applicants for credit using several machine learning algorithms. The algorithms are assessed based on their accuracy score to select the best algorithm for building a predictive model

Algorithms Used

a) Logistic Regression

b) Decision Tree

c) Random Forest

d) Support Vector Machine

e) Naive Bayes

f) k - Nearest Neighbors

g) Gradient Boosting Machine

Data Attributes

1.Loan_ID

2.Gender

3.Married

4.Dependents

5.Education

6.Self_Employed

7.ApplicantIncome

8.CoApplicantIncome

9.Loan_Amount

10.Loan_Amount_Term

11.Credit_History

12.Property Area

13.Loan_Status

Loan data cleaning and processing

Save the program *loan_prediction.py* in your project folder. Save the two csv files associated with the project in the same folder and name them *train.csv* and *test.csv*.

We'll go over the program one part at a time.

Loading the data

Highlight the following part of the program and press the F9 key on your keyboard to run it.

```
#Import libraries
import pandas as pd
import numpy as np

#Load data (the two csv files must be in the same folder as this program)
train=pd.read_csv("train.csv")    #
test=pd.read_csv("test.csv")
print(train.info())
#List of column names
```

```
list(train)  #❷

#Sample of data
train.head(10)   #❸

#Types of data columns
train.dtypes   #❹

#Summary statistics
train.describe()   #❺
```

We first import the two modules *pandas* and *numpy*. We then use the `read_csv()` function form the *pandas* module to load the csv files ❶.

At ❷, we look at the basic information about the dataset *train* by using the `info()` method associated with pandas DataFrame. The result is shown below:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
None
```

Results show that the dataset *train* has 13 variables, including *Gener*, *Married*, *Education*, and so on. The dataset has 614 rows. Note that Python is using 0 indexing, so the rows are numbered 0 to 613. It also provides the data type of each variable.

At ❷, we list the variable names in the dataset *train* by using the `info()` method associated with pandas DataFrame. At ❸, we use the `head()` method to see the first ten observations of the data, which is shown below:

```
    Loan_ID Gender Married  ... Credit_History Property_Area Loan_Status
0  LP001002   Male      No  ...            1.0         Urban           Y
1  LP001003   Male     Yes  ...            1.0         Rural           N
2  LP001005   Male     Yes  ...            1.0         Urban           Y
3  LP001006   Male     Yes  ...            1.0         Urban           Y
4  LP001008   Male      No  ...            1.0         Urban           Y
5  LP001011   Male     Yes  ...            1.0         Urban           Y
6  LP001013   Male     Yes  ...            1.0         Urban           Y
7  LP001014   Male     Yes  ...            0.0     Semiurban           N
8  LP001018   Male     Yes  ...            1.0         Urban           Y
9  LP001020   Male     Yes  ...            1.0     Semiurban           N

[10 rows x 13 columns]
```

At ❹, we see the data types of the variables. At ❺, we use the `describe()` method to see the summary statistics of the numerical variables. The result is below:

```
       ApplicantIncome  CoapplicantIncome  ...  Loan_Amount_Term  Credit_History
count       614.000000         614.000000  ...         600.00000      564.000000
mean       5403.459283        1621.245798  ...         342.00000        0.842199
std        6109.041673        2926.248369  ...          65.12041        0.364878
min         150.000000           0.000000  ...          12.00000        0.000000
25%        2877.500000           0.000000  ...         360.00000        1.000000
50%        3812.500000        1188.500000  ...         360.00000        1.000000
75%        5795.000000        2297.250000  ...         360.00000        1.000000
max       81000.000000       41667.000000  ...         480.00000        1.000000

[8 rows x 5 columns]
```

The summary statistics include the total count, mean, standard deviation, minimum, and maximum of the variables. The 25, 50, and 75 percentile values are also provided. Note that the 50 percentile value is also known as the median value.

Handle missing values

As you can see from the summary statistics above, the count (that is, the number of observations) is different for different variables. This means some variables have missing values, and we must decide how to handle them.

Run the following block of code by highlighting them and pressing the F9 key.

```
#Find missing values
train.isnull().sum()
test.isnull().sum()

#Impute missing values with mean (numerical variables)
train.fillna(train.mean(),inplace=True)    #❶
train.isnull().sum()    #❷

#Test data
test.fillna(test.mean(),inplace=True)
test.isnull().sum()

#Impute missing values with mode (categorical variables)
train.Gender.fillna(train.Gender.mode()[0],inplace=True)   #❸
train.Married.fillna(train.Married.mode()[0],inplace=True)
train.Dependents.fillna(train.Dependents.mode()[0],inplace=True)
train.Self_Employed.fillna(train.Self_Employed.mode()[0],inplace=True)
train.isnull().sum()    #❹

#Test data
test.Gender.fillna(test.Gender.mode()[0],inplace=True)
test.Dependents.fillna(test.Dependents.mode()[0],inplace=True)
test.Self_Employed.fillna(test.Self_Employed.mode()[0],inplace=True)
test.isnull().sum()

#Treatment of outliers
train.Loan_Amount_Term=np.log(train.Loan_Amount_Term)   #❺
```

We first replace the missing values in the train data with the mean value of each variable ❶. If you check the number of missing values again ❷, you'll see the following:

```
Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
```

```
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      0
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

Not all variables have 0 missing values. This is because for categorical variables such as *Gender* or *Married*, this is no way to calculate the mean. We therefore replace the missing values in categorical variables with the mode of the variable (that is, the most frequent observation) ❸.

If you check the number of missing values now ❹, you'll see the following:

```
Loan_ID               0
Gender                0
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
```

Now all variables have 0 missing values. We do the same for the test data. Finally, we use the log value of the loan amount in the train data so that the impact of the outliers is not as much ❺.

Predictive modeling

Split train data for cross validation

The project provides you with train data and test data. However, in the real world, all you get is a large dataset and you'll have to split it into train and test yourself.

Below, we'll further split the train data into train and validation so that you know how to split the data, and also it provides another layer of validation. Further, when you have multiple methods, you can use the validation data to evaluation them and choose the best method to use on the test data.

Run the following block of code by highlighting them and pressing the F9 key.

```
#Remove Loan_ID variable - Irrelevant
train=train.drop('Loan_ID',axis=1)    #❶
test=test.drop('Loan_ID',axis=1)

#Create target variable
X=train.drop('Loan_Status',1)   #❷

y=train['Loan_Status']   #❸
#Build dummy variables for categorical variables
X=pd.get_dummies(X)    #❹
train=pd.get_dummies(train)
test=pd.get_dummies(test)

#Split train data for cross validation
from sklearn.model_selection import train_test_split
x_train,x_cv,y_train,y_cv = train_test_split(X,y,test_size=0.2)    #❺
```

We first drop the variable *Loan_ID* in both the train and test data because it's irrelevant ❶.

We then create the independent variables X and the dependent variable y so that we can run regressions. Everything except the variable *Loan_Status* in the train dataset are independent variables ❷. The dependent variable is *Loan_Status* ❸.

We use the `get_dummies()` method in *pandas* to create dummy variables for categorical data ❹. For example, the variable *Gender* now becomes two dummy variables *Gender_Female* and *Gender_Male*. Dummy variables take values of either 0 or 1. For all female applicants, the variable *Gender_Female* has a value of 1 and the variable *Gender_Male* has a value of 0. The reverse is true for male applicants.

At ❺, we split the datasets *X* and *y* into train and test using 80%--20% split.

Logistic regression

Next, we use the logistic regression to predict.

In statistics, a logistic model is used to model the probability of a certain event. A logistic regression (sometimes also called a logit regression) is a statistical model that uses a logistic function to model a binary dependent variable. A logistic function has the following functional form:

$$f(x) = \frac{e^x}{e^x + 1}$$

A binary dependent variable takes two values. In our example, the decision to approve a loan application or not is a binary dependent variable.

It's different from the linear regression we discussed in Chapter 9 in two aspects: first, the dependent variable is a binary variable (i.e., a dummy variable); second, the functional form of the model is a logistic function instead of a linear function.

Run the following block of code by highlighting them and pressing the F9 key.

```
#Fit model
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train,y_train)    #❶

#Predict values for cv data
pred_cv=model.predict(x_cv)    #❷

#Evaluate accuracy of model
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
accuracy_score(y_cv,pred_cv) #78.86%    #❸
matrix=confusion_matrix(y_cv,pred_cv)
print(matrix)
```

We first use the logistic regression to fit the train data ❶. We then use the regression results to predict the cross-validation data ❷. Based on the actual value of y and the predicted value of y in the cross-validation data, we can calculate the accuracy score and the confusion matrix ❸.

The accuracy score is 78.86%, and the confusion matrix is as follows

```
[[20 23]
 [ 1 79]]
```

The four values in the confusion matrix indicate that there are 79 cases of true positives (y_cv=Y and pred_cv=Y), 20 cases of true negatives (y_cv=N and pred_cv=N), 23 cases of false positives (y_cv=N and pred_cv=Y), and 1 cases of false negatives (y_cv=Y and pred_cv=N).

Decision trees

Next, we use decision trees to predict.

We have discussed decision tree in Chapter 9, but in the context of regressions. In machine learning, a decision tree is a flowchart like structure. The decision tree model goes from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called classification trees. We are using the classifications trees in this example. Tree models where the target variable can take continuous values are called regression trees. We have used regression trees in Chapter 9.

Run the following block of code by highlighting them and pressing the F9 key.

```
#Fit model
from sklearn import tree
dt=tree.DecisionTreeClassifier(criterion='gini')
dt.fit(x_train,y_train)

#Predict values for cv data
pred_cv1=dt.predict(x_cv)

#Evaluate accuracy of model
accuracy_score(y_cv,pred_cv1) #71.54%
matrix1=confusion_matrix(y_cv,pred_cv1)
print(matrix1)
```

The method is similar to what we did with logistic regression, except that we use decision tree to fit the model.

The accuracy score is 71.54%, and the confusion matrix is as follows

```
[[27 16]
 [ 22 58]]
```

Random forest

Next, we use the random forest to predict.

The decision tree model uses one deep tree. It's prone to over-fitting. To overcome this problem, we can use the random forest classifier. In this approach, we use a forest of relatively shallow trees. There comes the wisdom of the crowd with the collective opinion of the trees as opposed to a single tree. Random forest consists of multiple decision trees. It's less prone to over-fitting.

Run the following block of code by highlighting them and pressing the F9 key.

```
#Fit model
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(x_train,y_train)

#Predict values for cv data
pred_cv2=rf.predict(x_cv)

#Evaluate accuracy of model
accuracy_score(y_cv,pred_cv2) #77.23%
matrix2=confusion_matrix(y_cv,pred_cv2)
print(matrix2)
```

The method is similar to what we did with logistic regression and decision tree, except that we use random forest to fit the model.

The accuracy score is 77.23%, and the confusion matrix is as follows

```
[[21 22]
 [6 74]]
```

Support vector machine

Next, we use the support vector machine to predict.

Support Vector Machine (SVM) is one of the most popular models in Machine Learning, capable of conducting linear or nonlinear classification, regression, and outlier detection.

The SVR we discussed in Chapter 9 is one type of SVM, with the regression method. Here, we introduce another function SVC, which is a classifier.

Run the following block of code by highlighting them and pressing the F9 key.

```
from sklearn import svm
svm_model=svm.SVC()
svm_model.fit(x_train,y_train)

#Predict values for cv data
pred_cv3=svm_model.predict(x_cv)

#Evaluate accuracy of model
accuracy_score(y_cv,pred_cv3) #64.23%
matrix3=confusion_matrix(y_cv,pred_cv3)
print(matrix3)
```

The method is similar to before, except that we use Support Vector Classifier (SVC) function in SVM to fit the model.

The accuracy score is 64.23%, and the confusion matrix is as follows

```
[[0 43]
 [0 80]]
```

Naïve Bayes

Next, we use the naïve Bayes algorithm to predict.

Naive Bayes is a classification method based on Bayes rule, which states that

$$prob(A|B) = \frac{prob(A,B)}{prob(B)}$$

Where prob(A|B) is the probability of event A conditional on event B, prob(A, B) is the probability of events A and B happening simultaneously, and prob(B) is the probability of event B.

It is one of the simplest supervised learning algorithms. It assumes that the effect of a particular feature in a class is independent of other features. For example, in our case here, we evaluate loan applications based on applicant's gender, marital status, income, and so on. Even though these characteristics are correlated, the model treats these variables independently. This assumption simplifies computation, hence the name naïve Bayes.

Run the following block of code by highlighting them and pressing the F9 key.

```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(x_train,y_train)

#Predict values for cv data
pred_cv4=nb.predict(x_cv)

#Evaluate accuracy of model
accuracy_score(y_cv,pred_cv4) #80.49%
matrix4=confusion_matrix(y_cv,pred_cv4)
print(matrix4)
```

The method is similar to before, except that we use `GaussianNB()` function in the *naive_bayes* submodule of the *sklearn* module to fit the model.

The accuracy score is 80.49%, and the confusion matrix is as follows

```
[[21 22]
 [3 77]]
```

K-nearest neighbor (kNN) algorithm

Next, we use the k-nearest neighbor (kNN) algorithm to predict.

In kNN classification, an observation is classified by a plurality vote of its neighbors, with the objective of assigning the observation to the class most common among its k nearest neighbors.

Run the following block of code by highlighting them and pressing the F9 key.

```
from sklearn.neighbors import KNeighborsClassifier
kNN=KNeighborsClassifier()
kNN.fit(x_train,y_train)

#Predict values for cv data
pred_cv5=kNN.predict(x_cv)

#Evaluate accuracy of model
accuracy_score(y_cv,pred_cv5) #64.23%
matrix5=confusion_matrix(y_cv,pred_cv5)
print(matrix5)
```

The method is similar to before, except that we use `KNeighborsClassifier()` function in the *neighbors* submodule of the *sklearn* module to fit the model.

The accuracy score is 64.23%, and the confusion matrix is as follows

```
[[11 32]
 [20 60]]
```

Gradient boosting algorithm

Next, we use the gradient boosting algorithm to predict.

Gradient boosting is a machine learning technique that uses an ensemble of weak prediction models, typically decisions trees. It builds the model in a stage-wise manner. The idea is to leave those observations that the weak prediction models can handle and focus on developing new models to handle the remaining difficult observations.

Run the following block of code by highlighting them and pressing the F9 key.

```
from sklearn.ensemble import GradientBoostingClassifier
gbm=GradientBoostingClassifier()
gbm.fit(x_train,y_train)

#Predict values for cv data
pred_cv6=gbm.predict(x_cv)
```

```
#Evaluate accuracy of model
accuracy_score(y_cv,pred_cv6) #78.86%
matrix6=confusion_matrix(y_cv,pred_cv6)
print(matrix6)
```

The method is similar to before, except that we use `GradientBoostingClassifier()` function in the *ensemble* submodule of the *sklearn* module to fit the model.

The accuracy score is 78.86%, and the confusion matrix is as follows

```
[[20 23]
 [8 72]]
```

Select the best model

From the six algorithms, we select the best performing method, which is naïve Bayes algorithm.

Run the following lines of code by highlighting them and pressing the F9 key.

```
#Select best model in order of accuracy
#Naive Bayes - 80.49%
#Logistic Regression - 78.86%
#Gradient Boosting Machine -78.86%
#Random Forest - 77.23%
#Decision Tree - 71.54%
#Support Vector Machine - 64.23%
#k-Nearest Neighbors(kNN) - 64.23%

#Predict values using test data (Naive Bayes)
pred_test=nb.predict(test)
#Write test results in csv file
predictions=pd.DataFrame(pred_test,columns=['predictions']).to_csv('Credit_Pr
edictions.csv')
```

Just like that, you have your prediction for all the loan applications in the test dataset.

PART 2

Predict Loan Approval Using Deep Neural Networks

Deep neural networks (DNNs) are a powerful tool. It becomes more and more important in machine learning and data analytics. Therefore, it's useful to learn how to use DNNs to make binary classifications as well.

Run the program *loan_dnn.py* below (the program is also available on Canvas).

```
#(1)LOADING DATA
#Import libraries
import pandas as pd
import numpy as np

#Load data files
train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")

#(2)DATA CLEANING AND PREPROCESSING
#Find missing values
train.isnull().sum()
test.isnull().sum()

#Impute missing values with mean (numerical variables)
train.fillna(train.mean(),inplace=True)
train.isnull().sum()

#Test data
test.fillna(test.mean(),inplace=True)
test.isnull().sum()

#Impute missing values with mode (categorical variables)
train.Gender.fillna(train.Gender.mode()[0],inplace=True)
train.Married.fillna(train.Married.mode()[0],inplace=True)
train.Dependents.fillna(train.Dependents.mode()[0],inplace=True)
train.Self_Employed.fillna(train.Self_Employed.mode()[0],inplace=True)
train.isnull().sum()
```

```
#Test data
test.Gender.fillna(test.Gender.mode()[0],inplace=True)
test.Dependents.fillna(test.Dependents.mode()[0],inplace=True)
test.Self_Employed.fillna(test.Self_Employed.mode()[0],inplace=True)
test.isnull().sum()

#Treatment of outliers
train.Loan_Amount_Term=np.log(train.Loan_Amount_Term)

#(3)PREDICTIVE MODELLING
#Remove Loan_ID variable - Irrelevant
train=train.drop('Loan_ID',axis=1)
test=test.drop('Loan_ID',axis=1)

#Create target variable
X=train.drop('Loan_Status',1)

y=train['Loan_Status']
#Build dummy variables for categorical variables
X=pd.get_dummies(X)
train=pd.get_dummies(train)
test=pd.get_dummies(test)

#Split train data for cross validation
from sklearn.model_selection import train_test_split
x_train,x_cv,y_train,y_cv =
train_test_split(X,y,test_size=0.2,random_state=0)

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
y_train.replace(('Y', 'N'), (1, 0), inplace=True)    #❶
X_train=x_train.to_numpy().reshape((-1, 20))  #❷
print(X_train.shape)

Y_train=y_train.to_numpy().reshape((-1, 1))
print(Y_train.shape)
```

```
X_cv=x_cv.to_numpy().reshape((-1, 20))
print(X_cv.shape)

import tensorflow as tf
dnn = tf.keras.Sequential()          #❸
dnn.add(tf.keras.layers.Dense(128,activation="relu",input_shape=(20,)))
dnn.add(tf.keras.layers.Dense(64,activation="relu"))
dnn.add(tf.keras.layers.Dense(1, activation='sigmoid'))

dnn.compile(optimizer="adam", loss="binary_crossentropy")
dnn.fit(X_train, Y_train, epochs=50)

pred_dnn=dnn.predict(X_cv)

# Convert values between 0 and 1 to Y or N
pred_dnn_yn = np.where(pred_dnn > 0.5, 'Y', 'N')
#print (pred_dnn_yn)
# Convert back to pd DataFrame
pred_nn = pd.DataFrame(data=pred_dnn_yn)
# Evaluate accuracy of model
accu_nn=accuracy_score(y_cv,pred_nn)
matrix_nn=confusion_matrix(y_cv,pred_nn)
print(matrix_nn)
print("the accuracy for deep neural network algorithm is", accu_nn)
```

The first part of the program is to get the data ready, as we have done in the first part of this chapter.

At ❶, we convert Y and N to 1 and 0 so that *numpy* can understand it. We also convert the *pandas* DataFrames to *numpy* arrays so that the DNN models can process it ❷. We do this for the cross-validation data sets *X_cv* and *y_cv* as well.

At ❸, we start to build the DNN model with just one hidden layer. Since this is a binary classification problem, there is only one neuron at the output layer, with `sigmoid` as the activation function. The sigmoid function converts an input to a value between 0 and 1, which can be interpreted as the probability of approving the loan in this case.

If you run the program, you'll have the following output:

```
--snip--
Epoch 47/50
16/16 [==============================] - 0s 4ms/step - loss: 5.2104
Epoch 48/50
16/16 [==============================] - 0s 4ms/step - loss: 2.2114
Epoch 49/50
16/16 [==============================] - 0s 4ms/step - loss: 2.5446
Epoch 50/50
16/16 [==============================] - 0s 4ms/step - loss: 6.6159
[[ 1 32]
 [ 0 90]]
the accuracy for deep neural network algorithm is 0.7398373983739838
```

The accuracy of the DNN forecast is 73.98%.

PART 3

Predicting Rare Events

In part 1, we used the `predict()` method in all the machine learning models in the *sklearn* module. The method implicitly assumes that the dataset has 50% Ys (we can call them positives) and 50% Ns (we can call them negatives). But you may have noticed that in our training data, we have 422 positives and 192 negatives, not exactly a balanced sample.

In many real-world situations, the true positive events are extremely rare. That is, the data is highly imbalanced with much more negative samples than positive samples. In business, if the probability of fraud charges among all credit card transactions is only 0.5%, forecasting accuracy is not the right measure: you can classify all transactions as nonfraudulent, and you'll have an accuracy of 99.5%. Similarly, if the cancer rate among all patients is 1%, classifying all X-ray images as negative will have an accuracy of 99%, but doing so will not provide any useful information.

In such cases, it is better to use the receiver operating characteristic (ROC) curve to evaluate the performance of a prediction model.

What Is a ROC Curve?

A ROC curve plots the false positive rate against the true positive rate. You may ask, what is a false positive rate and what is a true positive rate?

Type I Error: false positive, mistakenly label a negative event as positive; e.g., someone without Covid is being tested positive.

Type II Error: false negative, mistakenly label a positive event as negative; e.g., someone with Covid is being tested negative.

Suppose a binary classifier makes a forecast, and we have TP observations of true positives, TN observations of true negatives, FP observations of false positives, and FN observations of false negatives. Then,

$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$

$$Flase\ Positive\ Rate = \frac{FP}{FP + TN}$$

The ROC curve shows that at various false positive rates (1%, 2%, …, all the way to 100%), what is the corresponding true positive rate.

The AUC-ROC score measures the area under the ROC curve. The score is between 0.5 and 1. The higher the score, the better the prediction, with 1 indicating a perfect prediction, and 0.5 indicating a random and useless prediction.

Below, we'll use the logistic regression and loan approval data as the example, to illustrate how to calculate the AUC-ROC score and how to create a ROC curve.

Run the program *logit_roc.py* below, and the program is also available on Canvas.

```
#Import libraries
import pandas as pd
import numpy as np

#Load data files
train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")

#Find missing values
```

```
train.isnull().sum()
test.isnull().sum()

#Impute missing values with mean (numerical variables)
train.fillna(train.mean(),inplace=True)
train.isnull().sum()

#Test data
test.fillna(test.mean(),inplace=True)
test.isnull().sum()

#Impute missing values with mode (categorical variables)
train.Gender.fillna(train.Gender.mode()[0],inplace=True)
train.Married.fillna(train.Married.mode()[0],inplace=True)
train.Dependents.fillna(train.Dependents.mode()[0],inplace=True)
train.Self_Employed.fillna(train.Self_Employed.mode()[0],inplace=True)
train.isnull().sum()

#Test data
test.Gender.fillna(test.Gender.mode()[0],inplace=True)
test.Dependents.fillna(test.Dependents.mode()[0],inplace=True)
test.Self_Employed.fillna(test.Self_Employed.mode()[0],inplace=True)
test.isnull().sum()

#Treatment of outliers
train.Loan_Amount_Term=np.log(train.Loan_Amount_Term)

#(3)PREDICTIVE MODELLING
#Remove Loan_ID variable - Irrelevant
train=train.drop('Loan_ID',axis=1)
test=test.drop('Loan_ID',axis=1)

#Create target variable
X=train.drop('Loan_Status',1)

y=train['Loan_Status']
#Build dummy variables for categorical variables
```

```
X=pd.get_dummies(X)
train=pd.get_dummies(train)
test=pd.get_dummies(test)

#Split train data for cross validation
from sklearn.model_selection import train_test_split
x_train,x_cv,y_train,y_cv = train_test_split(X,y,test_size=0.2)

#(a)LOGISTIC REGRESSION ALGORITHM
#Fit model
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train,y_train)

from sklearn.metrics import roc_auc_score #❶
roc_logit=roc_auc_score(y_cv, model.predict_proba(x_cv)[:,1]) #❷
print("the roc for random forest is", roc_logit)

# plot ROC curve
from sklearn.metrics import plot_roc_curve #❸
import matplotlib.pyplot as plt

logit_disp = plot_roc_curve(model, x_cv, y_cv) #❹
ax = plt.gca()
plt.show()
```

The first part is just getting the data ready.

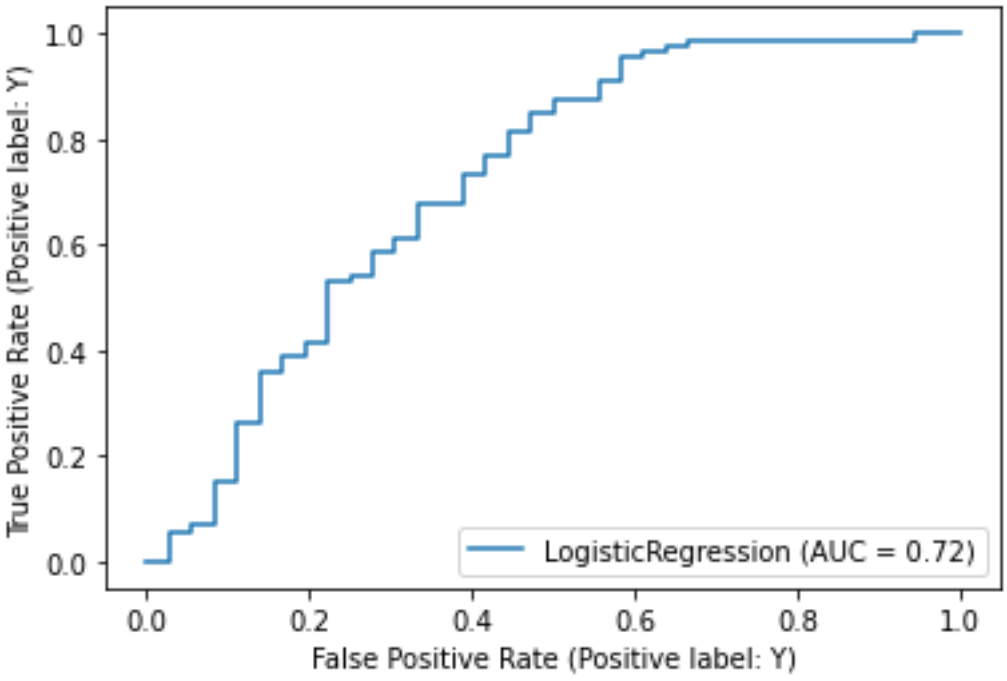
At ❶, we import the `roc_auc_score()` function from *sklearn*. The `predict_proba()` function from *sklearn* returns the probability of the observation being negative or positive. The `roc_auc_score()` function then compares the actual outcome (in this case *y_cv*) with the distribution of the probability and calculate the AUC-ROC score ❷.

At ❸, we import the `plot_roc_curve()` function from *sklearn*. It takes three arguments: the classification model, the X dataset, and the y dataset ❹. Finally, we use *matplotlib* to show the actual curve.

If you run the program, you'll have the following output:

```
the AUC-ROC score for logit is 0.7203065134099617
```

And the plot of the ROC curve is as follows:



End of Chapter Exercises

1: Modify the following block of code so that in the training data, you fill missing gender with "female" and missing marriage status with "No".

```
train.Gender.fillna(train.Gender.mode()[0],inplace=True)
train.Married.fillna(train.Married.mode()[0],inplace=True)
```

2: Modify the following block of code so that in the training data, you fill missing values in numerical variables with their median values.

```
train.fillna(train.mean(),inplace=True)
```

3: Modify the following block of code so that you split the data into train and cross validation at a 70%--30% ratio, with cross validation data being 30%.

```
#Split train data for cross validation
from sklearn.model_selection import train_test_split
x_train,x_cv,y_train,y_cv = train_test_split(X,y,test_size=0.2)
```

4: Suppose out of curiosity, you like to see the prediction outcomes using the kNN method. Modify the following block of code so that you predict using kNN. Save the output data as “knn_prediciton.csv”. Open the data and count how many Yes predictions and how many No predictions the method has made.

```
#Predict values using kNN
pred_test=nb.predict(test)
#Write test results in csv file
predictions=pd.DataFrame(pred_test,columns=['predictions']).to_csv('Credit_Pr
edictions.csv')
```

5: (Bonus question, 5 points; you’ll get 100% if you answered Q1-4 correctly) Find out the AUC-ROC score and plot the ROC curve by using the random forest classifier. You can modify based on *logit_roc.py*.