Machine Learning

Lab Sheet 5

Logistic Regression

S Abhishek

AM.EN.U4CSE19147

Colab Link

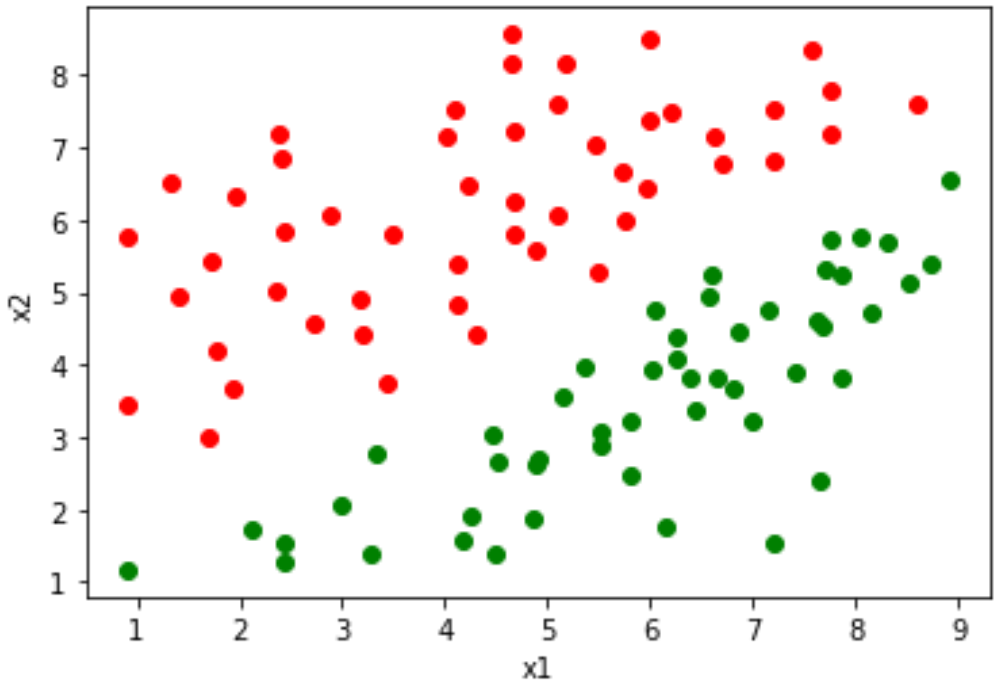
Part A

1 - Plot the attached dataset data1.csvusing scatter plot. There is a target feature with discrete values 0,1. If the target feature is 1, the samples should be shown as red circle. If the target feature is 0, the samples should be shown as green x

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

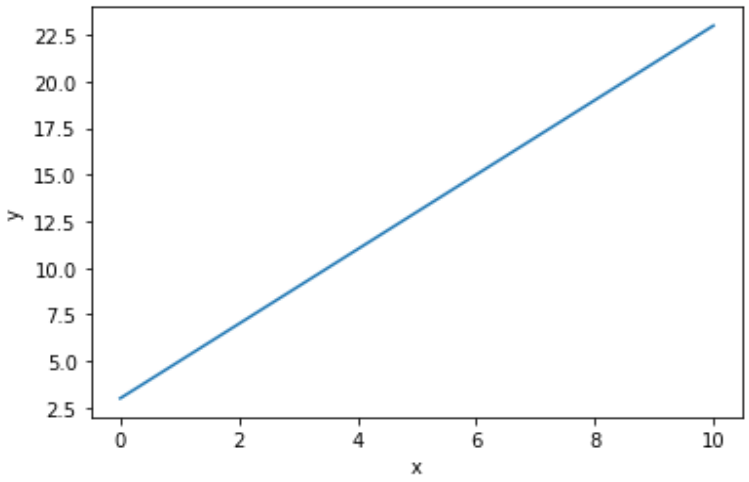
```
df = pd.read_csv('/content/data1.csv');
X = np.array(df['x1'] + df['x2']);
y = np.array(df['y']);
plt.scatter(df[y == 0]['x1'], df[y == 0]['x2'], label = 'Class1', c='red');
plt.scatter(df[y == 1]['x1'], df[y == 1]['x2'], label = 'Class2', c='green');

plt.xlabel('x1');
plt.ylabel('x2');
```



2 - Plot a line y = (2x+3)

```
x = np.linspace(0, 10, 100);
plt.plot(x, 2*x+3, linestyle='-');
plt.xlabel('x');
plt.ylabel('y');
```



3 - Define a function sigmoid(z) that takes one parameter z and computes 1/(1+e^-z). Create a vector V with 10 values randomly in the range [-1000,1000]. Transform V to V' that consists of respective sigmoid values using the defined function. Observe the range of output values in V'.

```
def sigmoid(z):
  if z.all() < 0 :
    return np.exp(z)/(1 + np.exp(z))
  else:
    return 1 / (1 + np.exp(-z))
V = np.random.randint(-1000, 1000, size=10)
V1 = sigmoid(V)
V1
```

```
array([1.00000000e+000, 1.00000000e+000, 3.44440613e-285, 2.11631627e-290,
       1.00000000e+000, 1.00000000e+000, 1.00000000e+000, 1.00000000e+000,
       0.00000000e+000, 1.00000000e+000])
```

4 - Define a function hypothesis(theta, X) that takes two vectors as parameters, theta and X. If sigmoid(theta.X)>= 0.5, output 0 else output 1

```
def hypothesis(theta, X):
  if sigmoid(np.dot(theta, X)) > 0.5 : print(0)
  else: print(1)
```

5 - Define a function cost(theta,X,y) to compute the error Error = 1/m*Σ-yilog(hθ(xi))-(1-yi)log(1-hθ(xi)) where xi is the ith sample and yi is the ith label, hθ(xi) is the hypothesis(theta,xi)

```
def cost(theta, X, y):
  return(-y * np.log(hypothesis(theta, X)) - (1-y) * np.log(1-hypothesis(theta, X)))
```

Part B

6 - Implement gradient descent algorithm for logistic regressionin data set loan_data.csv

```
df = pd.read_csv('/content/loan_data.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Loan_ID           614 non-null    object 
 1   Gender            601 non-null    object 
 2   Married           611 non-null    object 
 3   Dependents        599 non-null    object 
 4   Education         614 non-null    object 
 5   Self_Employed     582 non-null    object 
 6   ApplicantIncome   614 non-null    int64  
 7   CoapplicantIncome 614 non-null    float64
 8   LoanAmount        592 non-null    float64
 9   Loan_Amount_Term  600 non-null    float64
 10  Credit_History    564 non-null    float64
```

```
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

```
df.shape
```

```
(614, 13)
```

```
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
```

```
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History']).median()
```

```
 df.isnull().sum()
```

```
Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term    14
Credit_History       0
Property_Area        0
```

```
Loan_Status           0
dtype: int64
```

```
df.dropna(inplace=True)
```

```
df.isnull().sum()
```

```
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
```

```
df.shape
```

```
(542, 13)
```

```
df['Loan_Status'].replace('Y',1,inplace=True)
df['Loan_Status'].replace('N',0,inplace=True)
df['Loan_Status'].value_counts()
```

```
1    376
0    166
Name: Loan_Status, dtype: int64
```

```
df.Gender = df.Gender.map({'Male':1,'Female':0})
df['Gender'].value_counts()
```

```
1    444
0     98
Name: Gender, dtype: int64
```

```
df.Married = df.Married.map({'Yes':1,'No':0})
df['Married'].value_counts()
```

```
1    355
0    187
Name: Married, dtype: int64
```

```
df.Dependents=df.Dependents.map({'0':0,'1':1,'2':2,'3+':3})
df['Dependents'].value_counts()
```

```
0    309
2     94
1     94
```

```
3     45
Name: Dependents, dtype: int64
```

```
df.Education=df.Education.map({'Graduate':1,'Not Graduate':0})
df['Education'].value_counts()
```

```
1    425
0    117
Name: Education, dtype: int64
```

```
df.Self_Employed=df.Self_Employed.map({'Yes':1,'No':0})
df['Self_Employed'].value_counts()
```

```
0    467
1     75
Name: Self_Employed, dtype: int64
```

```
df.Property_Area=df.Property_Area.map({'Urban':2,'Rural':0,'Semiurban':1})
df['Property_Area'].value_counts()
```

```
1    209
2    174
0    159
Name: Property_Area, dtype: int64
```

```
df['LoanAmount'].value_counts()
```

```
146.412162    19
120.000000    15
```

```
100.000000    14
110.000000    13
187.000000    12
              ..
53.000000      1
65.000000      1
109.000000     1
156.000000     1
89.000000      1
Name: LoanAmount, Length: 195, dtype: int64
```

```
df['Loan_Amount_Term'].value_counts()
```

```
360.0    464
180.0     38
480.0     13
300.0     12
84.0       4
240.0      3
120.0      3
36.0       2
60.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

```
df['Credit_History'].value_counts()
```

1.0 542

Name: Credit_History, dtype: int64

df.head()

	Loan_ID	Gender	Married	...	Credit_History	Property_Area	Loan_Status
0	LP001002	1	0	...	1.0	2	1
1	LP001003	1	1	...	1.0	0	0
2	LP001005	1	1	...	1.0	2	1
3	LP001006	1	1	...	1.0	2	1
4	LP001008	1	0	...	1.0	2	1

[5 rows x 13 columns]

7 - Use sklearn built in function to find the model

```
X = df.iloc[1:542,1:12].values
Y = df.iloc[1:542,12].values

import numpy as np
class LogisticRegression:
  def __init__(self,x,y):
    self.intercept = np.ones((x.shape[0], 1))
    self.x = np.concatenate((self.intercept, x), axis=1)
    self.weight = np.zeros(self.x.shape[1])
    self.y = y
```

```
def sigmoid(self, x, weight):
  z = np.dot(x, weight)
  return 1 / (1 + np.exp(-z))
def loss(self, h, y):
  return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
def gradient_descent(self, X, h, y):
  return np.dot(X.T, (h - y)) / y.shape[0]
def fit(self, lr , iterations):
  for i in range(iterations):
    sigma = self.sigmoid(self.x, self.weight)
    loss = self.loss(sigma,self.y)
    dW = self.gradient_descent(self.x , sigma, self.y)
    self.weight -= lr * dW
def predict(self, x_new , treshold):
  x_new = np.concatenate((self.intercept, x_new), axis=1)
  result = self.sigmoid(x_new, self.weight)
  result = result >= treshold
  y_pred = np.zeros(result.shape[0])
  for i in range(len(y_pred)):
    if result[i] == True:
      y_pred[i] = 1
    else:
      continue
 return y_pred
```

```
regressor = LogisticRegression(X, Y)
regressor.fit(0.1 , 5000)
y_pred = regressor.predict(X, 0.5)
y_imp = y_pred
accuracy_imp = sum(y_pred == Y) / Y.shape[0]
print('Accuracy -> {}'.format(sum(y_pred == Y) / Y.shape[0]))
```

```
Accuracy -> 0.6931608133086876
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')


df = pd.read_csv('/content/loan_data.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Loan_ID           614 non-null    object
```

```
 1   Gender            601 non-null    object
 2   Married           611 non-null    object
 3   Dependents        599 non-null    object
 4   Education         614 non-null    object
 5   Self_Employed     582 non-null    object
 6   ApplicantIncome   614 non-null    int64
 7   CoapplicantIncome 614 non-null    float64
 8   LoanAmount        592 non-null    float64
 9   Loan_Amount_Term  600 non-null    float64
 10  Credit_History    564 non-null    float64
 11  Property_Area     614 non-null    object
 12  Loan_Status       614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

```
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].median()
df.dropna(inplace=True)
df['Loan_Status'].replace('Y',1,inplace=True)
df['Loan_Status'].replace('N',0,inplace=True)
df.Gender = df.Gender.map({'Male':1,'Female':0})
df['Gender'].value_counts()
df.Married=df.Married.map({'Yes':1,'No':0})
df['Married'].value_counts()
```

```
df.Dependents = df.Dependents.map({'0':0,'1':1,'2':2,'3+':3})
df['Dependents'].value_counts()
df.Education = df.Education.map({'Graduate':1,'Not Graduate':0})
df['Education'].value_counts()
df.Self_Employed = df.Self_Employed.map({'Yes':1,'No':0})
df['Self_Employed'].value_counts()
df.Property_Area = df.Property_Area.map({'Urban':2,'Rural':0,'Semiurban':1})
df['Property_Area'].value_counts()
```

```
1    209
2    174
0    159
Name: Property_Area, dtype: int64
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

X = df.iloc[1:542,1:12].values
y = df.iloc[1:542,12].values

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
random_state = 2)
```

```
model = LogisticRegression()
model.fit(X_train,y_train)
lr_prediction = model.predict(X_test)
```

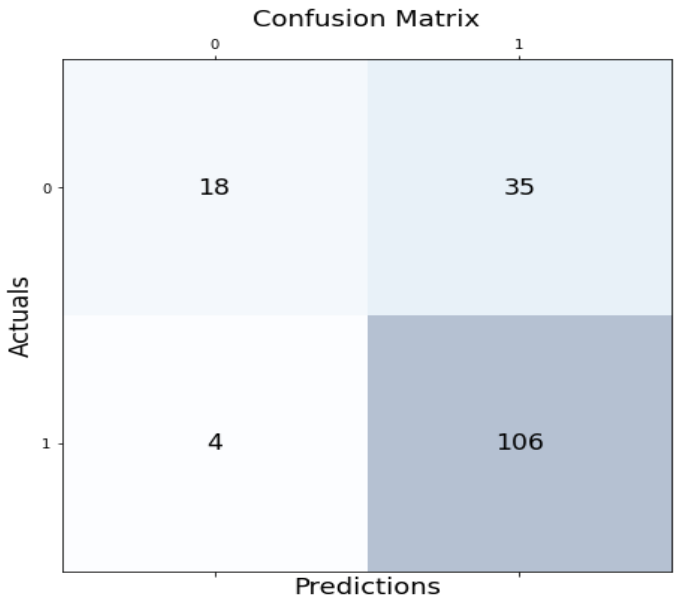
Part C

8 - Compute confusion matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,lr_prediction)
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)

for i in range(cm.shape[0]):
  for j in range(cm.shape[1]):
    ax.text(x=j, y=i,s=cm[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



9 - Compute the accuracy score

metrics.accuracy_score(lr_prediction,y_test)

accuracy_inb = metrics.accuracy_score(lr_prediction,y_test)

10 - Print a classification report using the following sklearn function

from sklearn.metrics import classification_report

print(classification_report(y_test,lr_prediction))

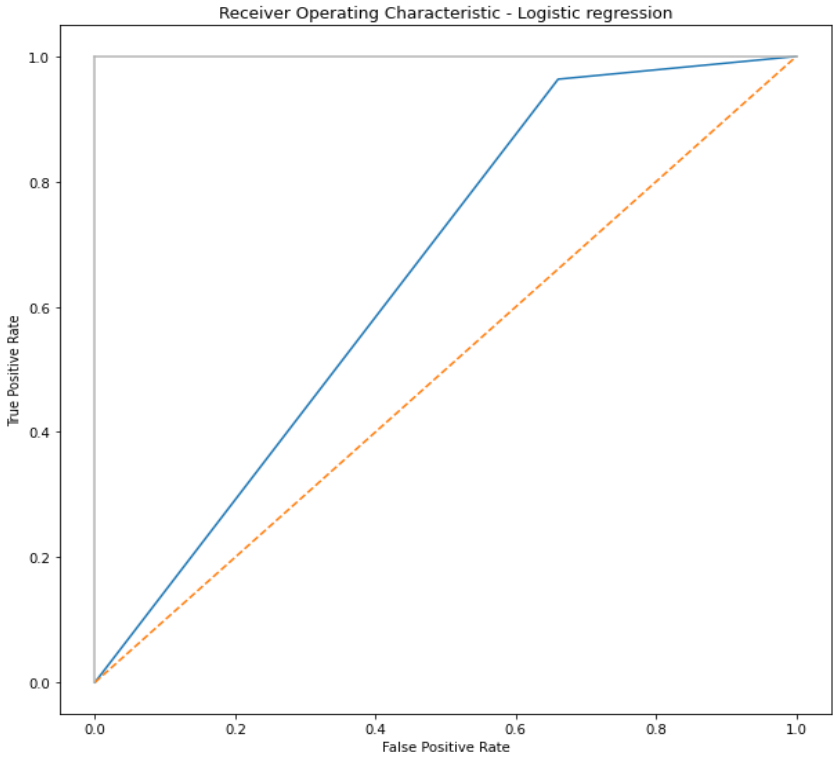
	precision	recall	f1-score	support
0	0.82	0.34	0.48	53
1	0.75	0.96	0.84	110

accuracy			0.76	163
macro avg	0.78	0.65	0.66	163
weighted avg	0.77	0.76	0.73	163

11 - Plot ROC curve for loan status

```
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test,
lr_prediction)

plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic - Logistic regression')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



12 - Compare the performance of classifiers obtained in 6 and 7

```
print('Accuracy of implemented model: {}'.format(accuracy_imp))
print('Accuracy of model implemented using In - Built functions: {}'.format(accuracy_inb))
```

Accuracy of implemented model: 0.6931608133086876

Accuracy of model implemented using In - Built functions: 0.7607361963190185

Thankyou!!