LAB 6

Exercise: Try logistic regression on BuyComputer dataset and set Random state=Your_RollNumber (last 3 digit of ID, incase if you don't have ID)

```
import numpy as np
import pandas as pd
import io
import matplotlib.pyplot as plt

data = pd.read_csv('/content/BuyComputer.csv')

data.drop(columns=['User ID',],axis=1,inplace=True)
data.head()
```

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

```
#Declare label as last column in the source file
y = data.iloc[:,-1].values
#Declaring X as all columns excluding last
X = data.iloc[:,:-1].values

#print(X)
#print(y)


# Splitting
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 = 142)

# Sacaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


y_pred = []
len_x = len(X_train[0])
w = []
b = 0.2
print(len_x)
```

```
2
```

```
entries = len(X_train[:,0])
print(entries)

for weights in range(len_x):
    w.append(0)
print(w)
```

```
300
[0, 0]
```

```
def sigmoid(z):
    return (1/(1+np.exp(-z)))

def predict(inputs):
    z = np.dot(w,inputs)+b
    a = sigmoid(z)
    return a

def loss_func(y,a):
    J = -(y*np.log(a) + (1-y)*np.log(1-a))
    return J


dw = []
db = 0
J = 0
alpha = 0.1
for x in range(len_x):
    dw.append(0)


#Repeating this process 1000 times
for iterations in range(1000):
    for i in range(entries):
        localx = X_train[i]
        a = predict(localx)
        dz = a - y_train[i]
        J += loss_func(y_train[i],a)
        for j in range(len_x):
            dw[j] = dw[j]+(localx[j]*dz)
        db += dz
    J = J/entries
    db = db/entries
    for x in range(len_x):
        dw[x]=dw[x]/entries
    for x in range(len_x):
        w[x] = w[x]-(alpha*dw[x])
    b = b-(alpha*db)
    J=0
```

```
print(w)
print(b)
```

```
[2.6355669936878874, 1.2284767892702433]
-1.0771011981687983
```

```
#predicting the label
for x in range(len(y_test)):
    y_pred.append(predict(X_test[x]))
```

```
#print actual and predicted values in a table
for x in range(len(y_pred)):
    print('Actual ',y_test[x],' Predicted ',y_pred[x])
    if y_pred[x]>=0.5:
        y_pred[x]=1
    else:
        y_pred[x]=0
```

```
Actual  0  Predicted  0.12259699039413076
Actual  0  Predicted  0.10084521739862946
Actual  0  Predicted  0.07501430631975593
Actual  1  Predicted  0.3791897941158939
Actual  1  Predicted  0.7693472356351628
Actual  1  Predicted  0.9987479851876083
Actual  1  Predicted  0.9674201820795184
Actual  0  Predicted  0.8724402111387103
Actual  0  Predicted  0.22293995766090136
Actual  0  Predicted  0.1425806276686564
Actual  0  Predicted  0.20403639016658362
Actual  1  Predicted  0.7577690001770833
Actual  0  Predicted  0.004554127016971811
Actual  0  Predicted  0.004860963624473335
Actual  0  Predicted  0.0009010484906925063
Actual  0  Predicted  0.05394707883221033
Actual  0  Predicted  0.43967481399833663
Actual  0  Predicted  0.013699564024998216
Actual  0  Predicted  0.02430952636651579
Actual  0  Predicted  0.055751446346558646
Actual  0  Predicted  0.12244981970421391
Actual  0  Predicted  0.0013335582809467835
Actual  0  Predicted  0.005087411114455209
Actual  1  Predicted  0.9801096298757461
Actual  0  Predicted  0.2909150456306235
Actual  1  Predicted  0.38047961245370066
Actual  1  Predicted  0.5546843444292209
Actual  1  Predicted  0.9940954156301087
Actual  1  Predicted  0.40307506943908533
Actual  0  Predicted  0.9912227813967265
Actual  0  Predicted  0.0006976938607882772
Actual  0  Predicted  0.0023668342295633557
Actual  0  Predicted  0.32194142036122786
Actual  0  Predicted  0.10772968853835205
Actual  1  Predicted  0.9451584479184649
Actual  0  Predicted  0.007200144220545985
Actual  0  Predicted  0.14723264189983243
Actual  0  Predicted  0.01897414587261145
Actual  1  Predicted  0.3075189655166428
Actual  0  Predicted  0.09773246866122626
Actual  1  Predicted  0.8226397825845715
Actual  1  Predicted  0.9187207240380215
Actual  0  Predicted  0.024881867985903545
Actual  1  Predicted  0.21673546913501845
Actual  1  Predicted  0.6919840372309739
Actual  1  Predicted  0.6163842450992859
Actual  1  Predicted  0.36010088799069345
Actual  1  Predicted  0.5833066512215003
Actual  0  Predicted  0.01828772561152434
Actual  0  Predicted  0.01889785491881847
Actual  1  Predicted  0.9866964996656186
Actual  1  Predicted  0.5866300198863112
Actual  0  Predicted  0.3880676884075185
Actual  0  Predicted  0.9834578322375367
Actual  0  Predicted  0.024051131134035827
Actual  0  Predicted  0.3237369610375424
Actual  1  Predicted  0.9810333906161756
Actual  0  Predicted  0.4138001946700178
```

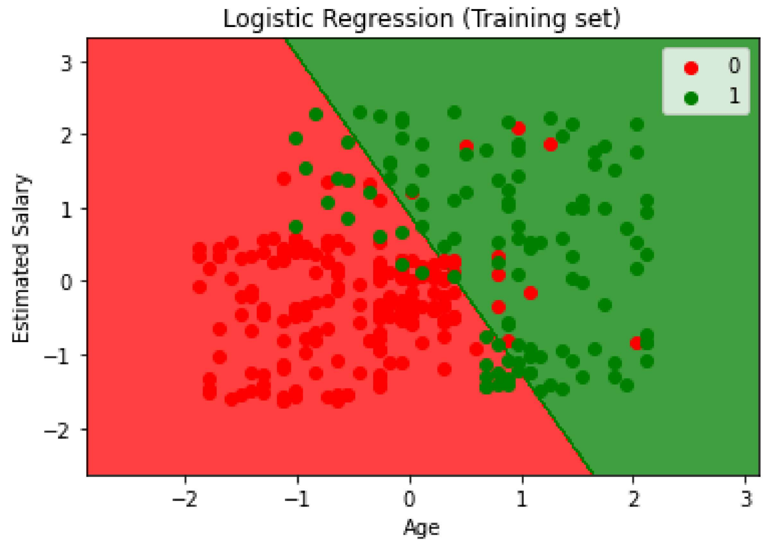
```
# Calculating accuracy
count = 0
for x in range(len(y_pred)):
    if(y_pred[x]==y_test[x]):
        count=count+1
print('Accuracy:',(count/(len(y_pred)))*100)
```

```
Accuracy: 83.0
```

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(random_state = 0)
LR.fit(X_train, y_train)

#predicting the test label with LR. Predict always takes X as input
y_pred=LR.predict(X_test)

#Accuracy Calculation:
#Compare accuracy of Sklearn with your previous output

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.0
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.0
plt.contourf(X1, X2, LR.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.0
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.0
plt.contourf(X1, X2, LR.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
```

```
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-m
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-m
```




```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-m
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-m
```