ANN for Classification Task from scratch¶

In [ ]:

```
# Dataset Loading
```

In [68]:

```
import pandas as pd
path="C:\\Users\\TANUJA HARISH\\Desktop\\ML and DL Summer Internship\\diabetes.csv"
data=pd.read_csv(path)
data.head(10)
```

Out[68]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

In [69]:

```
print(data.shape)
```

```
(768, 9)
```

In [70]:

```
data.describe()
```

Out[70]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [71]:

```
print(data.dtypes)
```

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
```

```
Outcome                        int64
dtype: object
```

In [72]:

```
 data['Outcome'].value_counts()
```

Out[72]:

```
0    500
1    268
Name: Outcome, dtype: int64
```

In [73]:

```
 data["Outcome"].value_counts().plot(kind="bar",color=["red","blue"])
```

Out[73]:

```
<AxesSubplot:>
```

In [74]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
df=data
#scatter with positive example
plt.scatter(df.Age[df.Outcome==1],df.Pregnancies[df.Outcome==1],color="red"
)
#scatter with negative examples
plt.scatter(df.Age[df.Outcome==0],df.Pregnancies[df.Outcome==0],color="blue
")
plt.title("Diabetics in Pregnant women")
plt.xlabel("Age")
plt.ylabel("Pregnancies")
plt.legend(["Diabetics","No Diabetics"])
```

Out[74]:

```
<matplotlib.legend.Legend at 0x151f9cc92b0>
```

Dataset Preparation¶

In [75]:

```
from sklearn.model_selection import train_test_split
x=data.drop("Outcome",axis=1)
y=data[["Outcome"]]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_st
ate=7)
```

In [76]:

```
import numpy as np
x_train=np.array(x_train)
x_test=np.array(x_test)
y_train=np.array(y_train)
y_test=np.array(y_test)
print("x_train",x_train.shape)
print("x_test",x_test.shape)
print("y_train",y_train.shape)
print("y_test",y_test.shape)
```

```
x_train (537, 8)
x_test (231, 8)
y_train (537, 1)
y_test (231, 1)
```

In [77]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
print("x_train",x_train.shape)
print("x_test",x_test.shape)
print(x_train)
print(x_test)
```

```
x_train (537, 8)
x_test (231, 8)
[[ 2.21124332 -0.25635346  0.80035334 ...  1.85869458  1.38157439
   1.06697501]
 [-0.53604176  0.05182162  0.0681732  ...  0.87654509  1.26457779
  -0.84406847]
 [ 0.07446603 -0.62616354 -0.03642397 ...  0.07296823 -0.95835763
   0.02458766]
 ...
 [-1.14654956 -1.92049886 -0.45481262 ... -1.34285767  0.81159095
   2.97801849]
 [-0.84129566 -0.4412585  -0.55940978 ... -1.01122277 -0.83236129
  -1.01779969]
 [ 1.29548163  1.83923705  0.17277036 ...  0.06021304  0.76359234
   0.2851845 ]]
[[-0.84129566 -0.90352111 -0.35021546 ... -0.64132231  0.34660446
  -0.75720285]
 [ 0.99022773  1.90087206  0.80035334 ...  0.46837906  0.36460393
   1.58816869]
 [ 2.82175112  1.00716435  1.11414484 ... -0.69234307  0.7995913
   0.89324379]
 ...
 [-1.14654956  1.31533942  0.38196469 ...  2.67502663  0.88358886
  -0.67033724]
 [-0.23078786 -1.11924366 -0.55940978 ...  0.26429605 -0.38537428
  -0.67033724]
 [ 0.68497383 -0.07144841  1.42793633 ... -0.44999449 -0.92235868
  -0.23600918]]
```

In [78]:

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

In [79]:

```
def cost_compute(y,y_hat):
    cost=-np.mean(y*(np.log(y_hat))-(1-y)*np.log(1-y_hat))
    return cost
```

In [80]:

```
def stochastic(x,y):
    w=np.zeros(shape=(1,x.shape[1]))
    b=0
    learning_rate=0.01
    m=len(y)
    iterations=40
    cost_list=[]
    for i in range(iterations):
        for j in range(m):
            #forward propagation
            prediction=sigmoid(np.dot(w,x[j]+b))
            #computing Loss value
            loss_value=cost_compute(y[j],prediction)
            #gradient Calculations
            w_gradient=-x[j]*(y[j]-(prediction))
            b_gradient=-(y[j]-prediction)
            #parameter Updation
            w=w-learning_rate*(w_gradient)
            b=b-learning_rate*(b_gradient)
            cost_list.append(loss_value)
    return w,b,cost_list
```

In [81]:

```
def predict(x,w,b):
    y_pred=[]
    for i in range(len(x)):
        y=np.asscalar(np.dot(w,x[i]+b))
        if sigmoid(y)<0.5:
            y_pred.append(0)
        else:
            y_pred.append(1)
    return np.array(y_pred)
```

In [82]:

```
w,b,cost_list=stochastic(x_train,y_train)
print(w)
print(b)
print(len(cost_list))
```

```
[[ 0.43003082  1.06352446 -0.32176709  0.07631241 -0.1297943   0.71120916
   0.4311711   0.15901056]]
[-0.34842537]
21480
```

In [83]:

```
y_pred=predict(x_test,w,b)
```

```
 print("Predicted Classes")
 print(y_pred)
 print("original Classes")
 print(y_test.T)
```

```
Predicted Classes
[0 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 0 1 1 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0
 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 0 1 0 0 0 0
 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1
 1 0 0 1 0 0 1 0 0]
original Classes
[[0 1 1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0
  1 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 1 1 1
  0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0
  0 1 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 1
  0 0 0 0 1 0 0 1 1 0 0 1 1 1 1 0 1 0 0 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 1 0
  0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0
  0 1 0 1 0 0 1 0 1 0 1 0 1 0 0]]
```

```
C:\Users\TANUJA HARISH\AppData\Local\Temp\ipykernel_11832\3211749504.py:4:
DeprecationWarning: np.asscalar(a) is deprecated since NumPy v1.16, use
a.item() instead
  y=np.asscalar(np.dot(w,x[i]+b))
```

In [84]:

```
 #Confusion Matrix
 def cm(x,y):
     tp=0
     tn=0
     fp=0
     fn=0
     for i in range(len(x)):
         if (x[i]==1 and y[i]==1):
             tp=tp+1
         elif (x[i]==1 and y[i]==0):
             fn=fn+1
         elif (x[i]==0 and y[i]==0):
             tn=tn+1
         else:
             fp=fp+1
     return tp,tn,fp,fn
 print("confusion Matrix")
 print(cm(y_pred,y_test))
```

```
confusion Matrix
(52, 129, 32, 18)
```

In [85]:

```
def acc(x,y):
    tp,tn,fp,fn=cm(x,y)
    acc_score=((tp+tn)/(tp+fp+tn+fn))
    return acc_score
print("Average Accuracy")
print(acc(y_pred,y_test))
```

```
Average Accuracy
0.7835497835497836
```

In [86]:

```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.88      0.80      0.84       161
           1       0.62      0.74      0.68        70

    accuracy                           0.78       231
   macro avg       0.75      0.77      0.76       231
weighted avg       0.80      0.78      0.79       231
```

In [ ]:

In [ ]: