Import dataset¶

In [1]:

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

Out[1]:

	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
4	0	137	40	35	168	43.1	2.288	33	1

In [2]:

```
print(dataset.shape)
```

```
(768, 9)
```

Divide dataset into inputs and output¶

In [3]:

```python
import numpy as np
x=np.array(dataset.iloc[:,0:8])
y=np.array(dataset[["Outcome"]])
print(x.shape)
print(y.shape)
```

```
(768, 8)
(768, 1)
```

Standardize the dataset¶

In [4]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_norm=sc.fit_transform(x)
print(x_norm.shape)
```

```
(768, 8)
```

Split dataset into test and train¶

In [5]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_norm,y,test_size=0.2)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
(614, 1)
(154, 1)
```

implement binary classification for single layer from scratch¶

In [6]:

```
def initialize_parameters(x_train):
    w1=np.random.randn(5,x_train.shape[1])
    w2=np.random.randn(1,5)
    b1=np.random.randn(5,1)
    b2=np.random.randn(1,1)
    return w1,w2,b1,b2
def sigmoid_act(x):
    s=1/(1+np.exp(-x))
    return s
def forward_path(x,w1,b1,w2,b2):
    #Hidden layer 1
    z1=np.reshape(np.matmul(w1,x.T),b1.shape)+b1
    A1=sigmoid_act(z1)
    z2=np.matmul(w2,A1)+b2
    A2=sigmoid_act(z2)
    return A2,A1
def compute_cost(y_pred,y_true):

cost=(y_true*np.log(y_pred+(10**-7))+(1-y_true)*np.log(1-y_pred+(10**-7)))
    return -cost
def backward_path(A2,A1,w2,y,x):
    #output layer
    dA2=(A2-y)/(A2*(1-A2))
    dz2=(A2-y)
    dw2=np.matmul(dz2,A1.T)
    db2=dz2.mean(axis=1)

    # Hidden layer 1
```

```python
    dA1=np.matmul(w2.T,dz2)
    dz1=np.multiply(dA1,A1*(1-A1))
    dw1=dz1*x
    db1=dz1.mean(axis=1)
    return dw2,dw1,db2,db1
def update_parameters(w,b,dw,db,learning_rate):
    w=w-learning_rate*dw
    db=np.reshape(db,b.shape)
    b=b-learning_rate*db
    return w,b
```

Training using Stocastic Gradient Descent (SGD)¶

In [7]:

```python
num_iter=500
learning_rate=0.001
his1=[]
w1,w2,b1,b2=initialize_parameters(x_train)
for i in range(num_iter):
    cost=0
    for j in range(len(x_train)):
        A2,A1=forward_path(x_train[j],w1,b1,w2,b2)
        y_pred=A2
        cost+=compute_cost(y_pred,y_train[j])
        dw2,dw1,db2,db1=backward_path(A2,A1,w2,y_train[j],x_train[j])
        w2,b2=update_parameters(w2,b2,dw2,db2,learning_rate)
        w1,b1=update_parameters(w1,b1,dw1,db1,learning_rate)
    his1.append(cost/len(x_train))
```

In [17]:

```
import matplotlib.pyplot as plt
his=np.reshape(np.array(his1),(len(his1)))
plt.plot(his)
plt.show()
```

In [18]:

```
print(w1)
print(b1)
```

```
[[ 0.32385187  1.1900291  -0.76254125  0.01752983 -0.84480722  0.43757137
   0.09531382  2.62984223]
 [ 0.44286016  1.31953953 -0.18590058 -0.93723997 -0.05738639  1.4594579
   0.76503163 -0.20043728]
 [ 0.96985735  2.09322267 -0.22260825  0.33845303  0.30026756  0.38490425
   0.11366201 -1.43321973]
 [-0.08168839  0.0523091  -2.21631094 -3.4799177   0.10566627  1.0868113
   0.03936299  0.5617054 ]
 [ 0.88682649 -1.31567447  0.30141611  0.43905836 -0.59329018 -0.62608331
  -1.18582117 -0.22705802]]
[[ 0.64027671]
 [-0.26862568]
 [ 0.36797966]
 [ 1.11078951]
 [-0.23536245]]
```

In [19]:

```
print(w2)
print(b2)
```

```
[[ 2.4190079   1.68142103  1.04165345 -0.85635738 -0.92270507]]
[[-2.59541766]]
```

Prediction on test dataset¶

In [20]:

```
y_pred=np.zeros(y_test.shape)
for i in range(len(x_test)):
    A2,A1=forward_path(x_test[i],w1,b1,w2,b2)
    A2=np.where(A2>0.5,1,0)  #thresholding-a2 is 1 id greater then 0.5 else
0
    y_pred[i]=A2
print(y_pred)
print(y_test)
```

```
[[0.]
```

```
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
```

```
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
```

```
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[1.]
[0.]
[1.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
```

```
 [1.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]]
[[0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
```

[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[1]
[0]

[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]

[0]
[1]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[0]

```
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]]
```

In [21]:

```
print("No of wrong prediction:",sum(y_pred!=y_test))
```

```
No of wrong prediction: [31]
```

Evaluating Quality metrics of model on the test set¶

In [22]:

```
TP=sum(np.logical_and(y_test==1,y_pred==1))
TN=sum(np.logical_and(y_test==0,y_pred==0))
FP=sum(np.logical_and(y_test==0,y_pred==1))
FN=sum(np.logical_and(y_test==1,y_pred==0))
Accuracy=sum(y_pred==y_test)/len(y_test)
print("Accuracy",Accuracy)
precision=TP/(TP+FP)
```

```
print("Precision",precision)
recall=TP/(TP+FN)
print("Recall",recall)
f1=(2*precision*recall)/(precision+recall)
print("F1-score",f1)
```

```
Accuracy [0.7987013]
Precision [0.80851064]
Recall [0.63333333]
F1-score [0.71028037]
```

In [24]:

```
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
cm=np.array(confusion_matrix(y_test,y_pred,labels=[0,1]))
print(cm)
confusion=pd.DataFrame(cm,index=["Diabetics","No
Diabetics"],columns=["Diabetics","No Diabetics"])
print(confusion)
print(classification_report(y_test,y_pred,labels=[0,1]))
acc=accuracy_score(y_test,y_pred)
print(acc)
```

```
[[85  9]
 [22 38]]
              Diabetics  No Diabetics
Diabetics            85             9
No Diabetics         22            38
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.85        94
           1       0.81      0.63      0.71        60

    accuracy                           0.80       154
   macro avg       0.80      0.77      0.78       154
weighted avg       0.80      0.80      0.79       154

0.7987012987012987
```

In [ ]: