Logistic Regression using pytorch library

In [1]:

```
#import libraries
import torch
import numpy as np
import pandas as pd
import io
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
```

In [2]:

```
#Load Data
candidates = {'gmat': [780,750,690,710,680,730,690,720,740,690,610,690,710,680
,770,610,580,650,540,590,620,600,550,550,570,670,660,580,650,660,640,620,660,6
60,680,650,670,580,590,690],
              'gpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.
3,3,2.7,3.7,2.7,2.3,3.3,2,2.3,2.7,3,3.3,3.7,2.3,3.7,3.3,3,2.7,4,3.3,3.3,2.3,2.
7,3.3,1.7,3.7],
              'work_experience': [3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,
4,1,2,6,4,2,6,5,1,2,4,6,5,1,2,1,4,5],
              'admitted': [1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0,0,0,1
,1,0,1,1,0,0,1,1,1,0,0,0,0,1]
              }
data = pd.DataFrame(candidates,columns=['gmat','gpa','work_experience','admitt
ed'])
```

In [3]:

```
#define X and y (input and targets)
X=data.iloc[:,:-1].values
y=data.iloc[:,-1].values
inputs = torch.tensor(X,dtype=torch.float32)
targets = torch.tensor(y,dtype=torch.float32)
targets.resize_(targets.shape[0],1)
m=targets.shape[0]
print(inputs.shape)
print(targets.shape)
```

```
torch.Size([40, 3])
torch.Size([40, 1])
```

In [4]:

```
#Add bias
bias = torch.ones(targets.shape[0],dtype=torch.float32)
bias.resize_(1,targets.shape[0])
new_input = torch.cat((inputs,bias.t()),1)
print(new_input[0:5])
```

```
tensor([[780.0000,   4.0000,   3.0000,   1.0000],
        [750.0000,   3.9000,   4.0000,   1.0000],
        [690.0000,   3.3000,   3.0000,   1.0000],
        [710.0000,   3.7000,   5.0000,   1.0000],
        [680.0000,   3.9000,   4.0000,   1.0000]])
```

In [5]:

```
#Assign weight to random values
weight = torch.rand((new_input.shape[1],1),dtype=torch.float32)
weight.resize_(new_input.shape[1],1)
print(weight)
print(weight.shape)
```

```
tensor([[0.0411],
        [0.6801],
        [0.8380],
        [0.1359]])
torch.Size([4, 1])
```

In [6]:

```
#Define All Functions
def gradientDescent(x,y,alpha,num_of_epochs,weight):
  for i in range(0,num_of_epochs):
    weight = weight - (alpha)*torch.mm(x.t(),(sigmoid(x,weight)-y))
  return weight

def sigmoid(input,weight):
  z=torch.mm(input,weight)
  return 1/(1+torch.exp(-z))

def predict(prob):
  if prob>=0.5:
    return 1
  else:
    return 0

def cross_entropy(y_pred,y):
  return -torch.sum(y*torch.log(y_pred)+(1-y)*torch.log(1-y_pred))
```

In [7]:

```
#Define alpha and num_of_epochs
alpha = 1e-6
num_of_epochs = 1000000
```

In [8]:

```
#model execution for num_of_epochs
final_weight = gradientDescent(new_input,targets,alpha,num_of_epochs,weight)
```

In [9]:

```
#Final weight
print(final_weight)
```

```
tensor([[-0.0156],
        [ 1.8616],
        [ 1.2115],
        [-0.6801]])
```

In [10]:

```
#predict probability
y_prob=torch.zeros(m,1)
y_prob=sigmoid(new_input,final_weight)
print(y_prob[0:5])
```

```
tensor([[0.1443],
        [0.4289],
        [0.1574],
        [0.7645],
        [0.6914]])
```

In [11]:

```
#find loss
loss=cross_entropy(y_prob,targets)
print(loss)
```

```
tensor(16.4898)
```

In [12]:

```
#Predict class using probabily with given thresold=0.5
for i,prob in enumerate(y_prob):
  y_pred = predict(prob)
  print("Probability : ",prob,"Predicted class : ",y_pred,"Actual class: ",targets[i])
```

```
Probability :  tensor([0.1443]) Predicted class :  0 Actual class:  tensor
([1.])
Probability :  tensor([0.4289]) Predicted class :  0 Actual class:  tensor
([1.])
Probability :  tensor([0.1574]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.7645]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.6914]) Predicted class :  1 Actual class:  tensor
([0.])
Probability :  tensor([0.8886]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.0026]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.2820]) Predicted class :  0 Actual class:  tensor
([1.])
Probability :  tensor([0.4911]) Predicted class :  0 Actual class:  tensor
([1.])
Probability :  tensor([0.0008]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.1758]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.8161]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.9160]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.4231]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.0508]) Predicted class :  0 Actual class:  tensor
([1.])
Probability :  tensor([0.0320]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.5336]) Predicted class :  1 Actual class:  tensor
([0.])
Probability :  tensor([0.9653]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.1593]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.1216]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.1423]) Predicted class :  0 Actual class:  tensor
([1.])
Probability :  tensor([0.0060]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.4647]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.0460]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.1717]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.9063]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.6785]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.0460]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.9653]) Predicted class :  1 Actual class:  tensor
```

```
([1.])
Probability :  tensor([0.7710]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.0203]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.0515]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.7867]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.9187]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.7112]) Predicted class :  1 Actual class:  tensor
([1.])
Probability :  tensor([0.0048]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.0243]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.0845]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.1320]) Predicted class :  0 Actual class:  tensor
([0.])
Probability :  tensor([0.8161]) Predicted class :  1 Actual class:  tensor
([1.])
```