Cover page for answers.pdf CSE512 Fall 2019 - Machine Learning - Homework 3

Your Name: Shubham S Bagi

Solar ID: 112672171

NetID : sbagi

email address: shubham.bagi@stonybrook.edu

Names of people whom you discussed the homework with: Gurusangama

Machine Learning

HW-3

1.1 1.1.1 ⇒ False negative = 1 False positive = α. ($\alpha > 1$)

and $\eta(x)$ = probability that 'x' is (+)ve.

Risk $r(x)$

$$r(x) = \eta(x)(1-\eta(z)) + (1-\eta(x))\eta(z) \quad \text{(A)}$$

Considering data point x and $\eta(x)$ the probability of 'x' being (+)ve.

$$r^*(x) = \min\{\eta(x), 1-\eta(x)\}.$$

Here $r(x) = \eta(x) \times 1$ (For false negative) —(B)

$1-\eta(x) = (1-\eta(x))\alpha$ (For false (+)ve) —(C)

Hence when we take (A), (B) & (C).

$\Rightarrow r^*(x) = \min\{\eta(x) \times 1, (1-\eta(x))\alpha\}$.

1.1.2 ⇒ Here $r(x) = \eta(x)(1-\eta(z)) + (1-\eta(x))\eta(z)$

Here as we know $\eta(x)$ represents the probability that 'x' is positive, 'z' is the nearest data point in the training data.

So, the ~~probability~~ cost that 'x' is False negative. is 1

and the cost that it is False positive is 'α'

We use this in the risk for x.

Hence the equation becomes,

$$\Rightarrow r(x) = 1 \times \eta(x) * (1-\eta(z)) + \alpha(1-\eta(x))\eta(z) \;//$$

1.1.3>. To prove.

$$r(x) \leq (1+\alpha) r^*(x) (1-r^*(x)).$$

$$r(x) = \eta(x)(1-\eta(z)) + (1-\eta(x))(\eta(z))\alpha.$$

as n goes to infinity, z becomes 'x'.

$$r(x) = \eta(x)(1-\eta(x)) + (1-\eta(x))(\eta(x))\alpha.$$

$$r(x) = (1+\alpha)(\eta(x))(1-\eta(x)). \quad \text{---(1)}$$

The optimal Bayes Risk for 'x' is.

$$\Rightarrow r^*(x) = \min(\eta(x), 1-\eta(x)).$$

as we know $\alpha \geq 1$

$$(\alpha+1) > 1$$

Hence we can say (From [illegible])

$$r^*(x) = \eta(x) \quad \text{---(A)}$$

put (A) in (1)

$$r(x) \leq (1+\alpha) r^*(x) (1-r^*(x))$$

1.1.4>. To $r(x) \leq (1+\alpha) r^*(x) (1-r^*(x))$

Taking expectation.

$$E[r(x)] \leq E[(1+\alpha) r^*(x)(1-r^*(x))].$$

$$R(x) \leq (1+\alpha) R^*(x)(1-R^*(x))$$

Thus, $R \leq (1+\alpha) R^*(1-R^*)$. //

1.2) Consider the ~~neigh~~ nearest neighbour equation,

$$r(n) = (1-\eta(z))\,\eta(n) + (1-\eta(x))\,\eta(z) \quad \text{— (A)}$$

Here z is the nearest data point, so now let us consider $\frac{k+1}{2}$ neighbours of the total 'k' neighbour.

Hence if the probability of find $\frac{k+1}{2}$ neighbours is give by $g(n,k)$.

We now substitute $g(n,k)$ in the place of $\eta(z)$.

Hence $r(n) = (1-g(n,k))\,\eta(n) + g(n,k)\,(1-\eta(n))$.

$$r(n) = \eta(n) - g(n,k)\,\eta(n) + g(n,k) - g(n,k)\,\eta(n)$$

$$\Rightarrow r(n) = \eta(n) + (1-2\eta(n))\,g(n,k) \quad //$$

1.2.2) As $n \to \infty$, x turns to 'z'

$$r^*(n) = \min(\eta(n), (1-\eta(x)))$$

when we consider $r^*(n) = \eta(n)$ as $\frac{k+1}{2}$ neighbours are 4)ve; $g(n,k)$ – depends on 'n'

hence

$$r(n) = [1-g(n,k)]\,\eta(n) + (1-\eta(x))\,g(n,k)) \text{ becomes}$$

$$\Rightarrow r(n) = [1-g(n,k)]\,r^*(n) + (1-r^*(n))\,g(n,k)) \quad //$$

1.2.3> Hoeffding's inequality

$$P(H(n) \le k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

when $k = (p-\varepsilon)$ for $\varepsilon > 0$

$$P(H(n)) \le (p-\varepsilon)n) \le \exp(-2\varepsilon^2 (\tfrac{k+1}{2}))$$

here $n = \frac{k+1}{2}$.

$$P(H(\tfrac{k+1}{2})) \le \exp(-2\varepsilon^2 (\tfrac{k+1}{2}))$$

we have $\varepsilon = \frac{(1-2r^*(x))}{2}$.

$$g(r^*(n), k) \le \exp(-2(\tfrac{1}{2} - r^*(n))^2 k.$$

$$\Rightarrow g(r^+(n), k)) \le \exp(-2(0.5 - r^6(n))^2 k)$$

1.2.4> $r(n) \le r^*(n) + \frac{1}{\sqrt{2k}}$.

$$g(r^+(n)) \le \exp(-2(0.5 - r^*(n))^2 k).$$

$$\Rightarrow (g(r^*(n), k)) \le \frac{1}{\exp(2(0.5 - r^*(x)^2, k))}$$

2.1> To prove – $\frac{\partial}{\partial \theta_c} \log(P(Y^i|x^i;\theta)) = \delta(c=y^i) - P(c|x^i;\theta)\bar{x}^i$

⇒. Here lets consider $L(\theta) = \sum_{i=1}^{n} \log(P(Y^i|x^i,\theta))$

$$L(\theta) = \sum_{i=1}^{n} y_i \log(P(Y=1|x^i,\theta)) + (1-y_i)\log(P(Y=0|x^i,\theta))$$

$$= \sum_{i=1}^{n} y^i(\theta^T x^i) - \log\{1+\exp(\theta^T x^i)\}$$

Applying partial differentiation w.r.t θ.

$$\frac{\partial L}{\partial \theta} = \sum_{i=1}^{n} x^i y^i - \frac{\exp(\theta^T x^i)}{1+\exp(\theta^T x^i)} \cdot x^i$$

$$= \sum_{i=1}^{n} \left[y^i - \frac{\exp(\theta^T x^i)}{1+\exp(\theta^T x^i)} \right] x^i$$

y^i ↓ 1 or 0.

$$= \sum_{i=1}^{n} [\,]$$ Here as y^i as this is hot-encoded, it can hold only value '1', hence we can say that it is a delta function. Hence $\delta(c=y^i)$

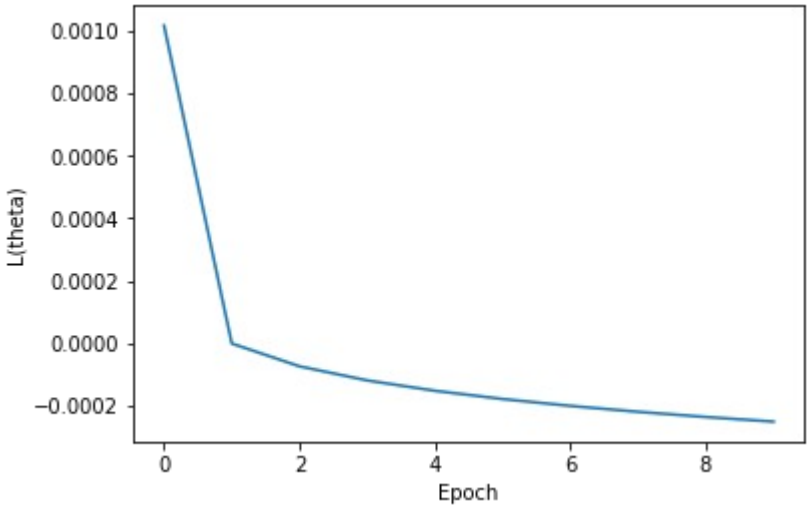
$$\frac{\partial L}{\partial \theta_c} = \sum_{i=1}^{n} \left[\delta(c=y^i) - P(y^i|x^i;\theta) \right] \bar{x}^i$$

hence proved.

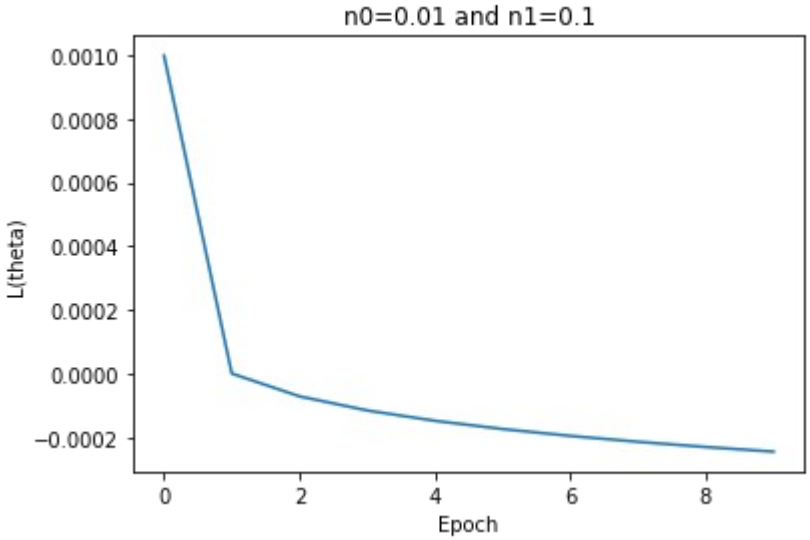
For 2.3

1.

For the above values mentioned I have got the following graph.
Number of epoch - **8**
Loss Function - **0.0009997871619652104**



2. For values
n0 - 0.01 and n1 = 0.1
Epoch - 8
Loss Function value – 0.0009998314425902813



3.

This is the combined graph for both validation and test data.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id

Enter your authorization code:
..........
Mounted at /content/gdrive
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split, cross_val_score
from statistics import mean
import pickle
from sklearn.preprocessing import OneHotEncoder
from numpy import linalg as LA
```

```
trainlabel= pd.read_csv("/content/gdrive/My Drive/ML/HW3/Train_Labels.csv")
vallabel = pd.read_csv("/content/gdrive/My Drive/ML/HW3/Val_Labels.csv")

with open("/content/gdrive/My Drive/ML/HW3/Train_Features.pkl", 'rb') as f:
    train_pickle = pickle.load(f, encoding="latin1")
    train_data = pd.DataFrame(train_pickle).T
with open("/content/gdrive/My Drive/ML/HW3/Test_Features.pkl", 'rb') as f:
    test_pickle = pickle.load(f, encoding="latin1")
    test_data = pd.DataFrame(test_pickle).T
with open("/content/gdrive/My Drive/ML/HW3/Val_Features.pkl", 'rb') as f:
    val_pickle = pickle.load(f, encoding="latin1")
    val_data = pd.DataFrame(val_pickle).T
train_data.index.name = 'Id'
train_data.reset_index(inplace=True)
test_data.index.name = 'Id'
test_data.reset_index(inplace=True)
val_data.index.name = 'Id'
val_data.reset_index(inplace=True)
train = pd.merge(train_data,trainlabel,left_on = 'Id',right_on = 'Id', how = 'inner')
test = pd.merge(test_data,trainlabel,left_on = 'Id',right_on = 'Id', how = 'inner')
val = pd.merge(val_data,vallabel,left_on = 'Id',right_on = 'Id', how = 'inner')
valuetest = test_data.iloc[:,1]
train_full =train
val_full =val
test_full = test
```

```
test_data_1=test_data.iloc[:,1:]
```

```
columns = ['Category', 'Id']
train_full_data = train_full.drop(columns, axis=1)
val_full_data = val_full.drop(columns,axis=1)
train_label = train[['Category']]
val_label = val[['Category']]
columns1 = ['Id']
train_full = train_full.drop(columns1,axis=1)
testdata  = test_full.drop(columns, axis=1)
```

```
test_data.shape
```

```
(2000, 513)
```

```
train_norm = LA.norm(np.transpose(train_full_data), 1)
train_full_data_norm = np.transpose(train_full_data)/train_norm
val_norm = LA.norm(np.transpose(val_full_data), 1)
val_full_data_norm = np.transpose(val_full_data)/val_norm
test_norm = LA.norm(np.transpose(testdata), 1)
test_full_data_norm = np.transpose(testdata)/test_norm
test_norm1 = LA.norm(np.transpose(test_data_1), 1)
test_full_data_norm = np.transpose(test_data_1)/test_norm1
```

```
onehotencoder = OneHotEncoder(categorical_features = [0])
train_label_encoded = onehotencoder.fit_transform(train_label).toarray()
train_label_dataframe = pd.DataFrame(train_label_encoded)
train_label_dataframe
```

```
import numpy as np
import math
from random import randrange

def Derivative(X,Y,wtrans,batchvalue):
  (d,N) = X.shape
  X2 = np.ones(N)
  Xbar = np.vstack([X,X2])
  Xbartran = np.transpose(Xbar)
  i = 0
  val = 0
  pr=[]
  if(wtrans is None):
    wtrans = np.ones([3,d+1])
  for i in batchvalue:
    #rav gets a column of X values with d dimnensions
    rav=Xbar[...,i].ravel()
    lis2 = []
    #pr is thetha values * the X value of 1 with d dimensions
    pr=wtrans.dot(rav)
    Y_value =Y[i][0:3]
    lis2.append(sigmoid(pr[0],pr))
    lis2.append(sigmoid(pr[1],pr))
    lis2.append(sigmoid(pr[2],pr))
    som = np.subtract(Y_value,lis2)
    sigarray = np.array(som)[np.newaxis]
    Xvalue = np.array(rav)[np.newaxis]
    val = val+ (np.transpose(sigarray)).dot(Xvalue)
  val = 0-((1/16.0)*val)
  return val

#This Function is to create the Batch in the Stochastic Gradient of batch m
def batch(N,m):
  x = np.random.rand(0,N)
  list1 = []
  biglist = []
  lis1 = []
  u =0
  for i in range(0,N):
    lis1.append(randrange(N))
    u = u+1
    if(u==m):
      biglist.append(lis1)
      lis1 = []
      u=0
  biglist.append(lis1)
  return biglist
```

```
#This sigmoid Function is used to for k-1 coefficients
def sigmoid(e,z):
  e_z = np.exp(z - np.max(z))
  return (e /(1+ e_z.sum(axis=0)))

#This sigmoid Function is used to for k coefficients
def sigmoid_kvalue(z):
  e_z = np.exp(z - np.max(z))
  return (1 /(1+ e_z.sum(axis=0)))

def prob(wtrans,Xbar):
  val = max(wtrans).dot(Xbar)
  return val

def LossFunction(wtrans,X,Y):
  (d,N) = X.shape
  X2 = np.ones(N)
  Xbar = np.vstack([X,X2])
  for i in range(N):
    rav=Xbar[...,i].ravel()
    value = 0
    lval= []
    for j in range(3):
      expval = np.log(1 + np.sum(np.exp(sigmoid(wtrans.dot(rav),wtrans.dot(rav)))))
      sumval = sigmoid(wtrans[j].dot(rav),wtrans.dot(rav)) - expval
      value =  value+ sumval
  finalvalue = 0-value*(1/N)
  print(finalvalue)
  return finalvalue

def Log(X,Y,m,n0,n1,max_epoch,delta):
  wtrans=None
  Loss = None
  lval = []
  for epoch in range(max_epoch):
    (k,N) = X.shape
    batchvalue = batch(N,m)
    n = n0/(n1+epoch)
    for i in range(len(batchvalue)):
      if(wtrans is None):
        wtrans  = Derivative(X,Y,wtrans,batchvalue[i])
      else:
        wtrans = np.subtract(wtrans, n*Derivative(X,Y,wtrans,batchvalue[i]))
    if(Loss is None):
      Loss =  LossFunction(wtrans,X,Y)
      lval.append((epoch,Loss))
    else:
      if(LossFunction(wtrans,X,Y)>(1-delta)*Loss):
        lval.append((epoch,(1-epoch**0.1)*Loss))
        continue
      else:
        lval.append((i,Loss))
        Loss =  LossFunction(wtrans,X,Y)

  return (wtrans,lval)



def LogisticRegression(Xtest,X,Y,m,n0,n1,max_epoch,delta):
  (d,N) = Xtest.shape
  X2 = np.ones(N)

  Xbar = np.vstack((Xtest,X2))
  Xbartran = np.transpose(Xbar)
  wtrans,lval = Log(X,Y,m,n0,n1,max_epoch,delta)
  Z=[]
  for i in range(N):
```

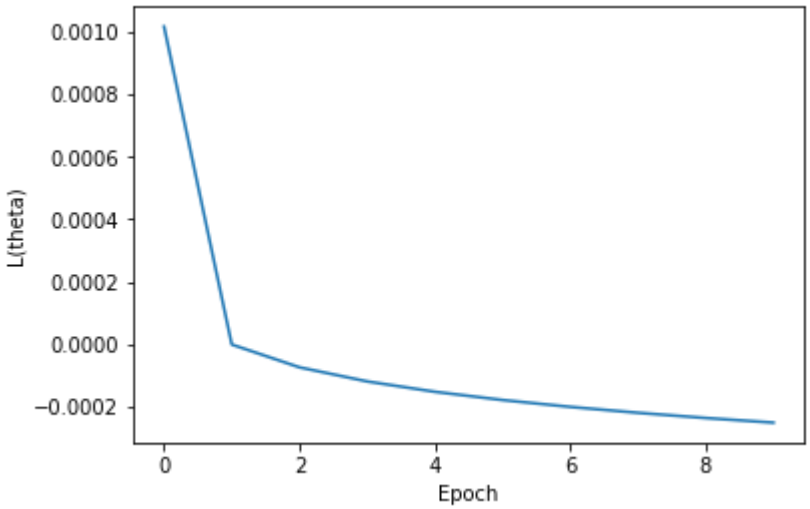
```
    Xval=Xbar[...,i].ravel()
    Xarrayvalue = np.array(Xval)[np.newaxis]
    Value = wtrans.dot(np.transpose(Xarrayvalue))
    y=[]
    for j in range(3):
      y.append(sigmoid(Value[j],Value))
    y.append(sigmoid_kvalue(Value))
    Z.append(y)
  return (Z,lval)

m =8
n0 = 0.1
n1 = 1
delta = 1
max_epoch = 10

xy,lval11 = LogisticRegression(test_full_data_norm,train_full_data_norm,train_label_datafra
```

```
value=lval11
pddat=pd.DataFrame(value, columns=['epoch','thetaval'])
plt.plot(pddat['epoch'],pddat['thetaval'])
plt.xlabel('Epoch')
plt.ylabel('Loss Function')
```

```
Text(0, 0.5, 'L(theta)')
```



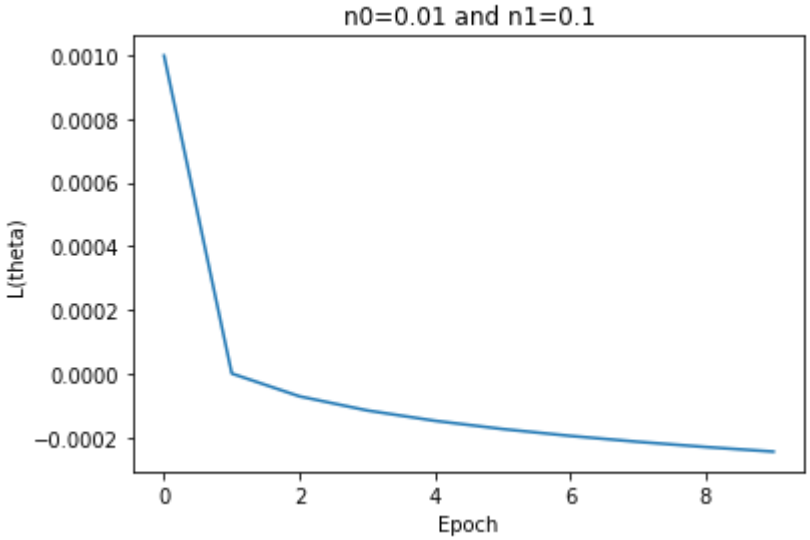
For the above values mentioned I have got the following graph. Number of epoch - **8** Loss Function - **0.0009997871**

```
m =16
n0 = 0.01
n1 = 0.1
delta = 0.00001
max_epoch = 10

x1,lval2 = LogisticRegression(test_full_data_norm,train_full_data_norm,train_label_datafram
```

```
lval2=pd.DataFrame(lval2, columns=['epoch','thetaval'])
plt.plot(lval2['epoch'],lval2['thetaval'])
plt.title('n0=0.01 and n1=0.1')
plt.xlabel('Epoch')
plt.ylabel('Loss Function')
```

```
Text(0, 0.5, 'L(theta)')
```



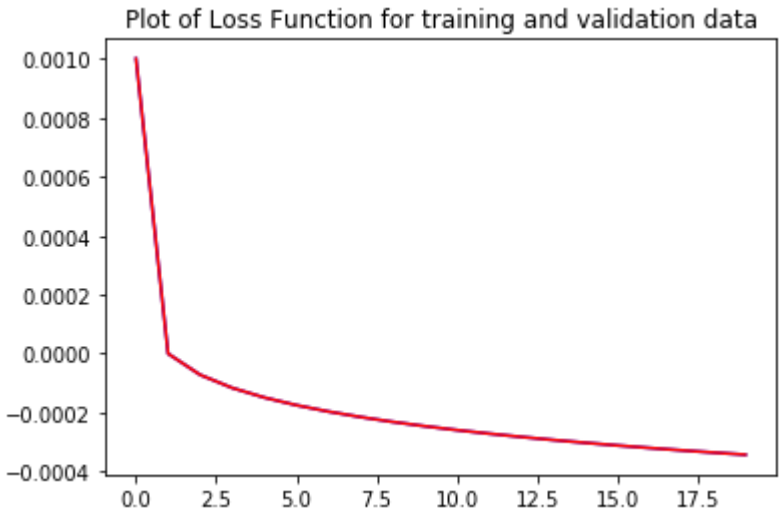
For values n0 - 0.01 and n1 = 0.1 Epoch - 8 Loss Function value - 0.0009998314425902813

**For 2.2.3 **

```
Out,plot1=LogisticRegression(test_full_data_norm,train_full_data_norm,train_label_dataframe
plot_dt=pd.DataFrame(plot1, columns=['epoch','thetaval'])
```

```
Out1,plotval=LogisticRegression(val_full_data_norm,train_full_data_norm,train_label_datafra
plot_val=pd.DataFrame(plotval, columns=['epoch','thetaval'])
```

```
plt.plot(plot_val['epoch'], plot_dt['thetaval'], 'b')
plt.plot(plot_dt['epoch'], plot_dt['thetaval'], 'r')
plt.title('Plot of Loss Function for training and validation data')
plt.show()
```



```
Loss Function as a function of Epoch. Graph for validation and training data.
```

```
Frame = pd.DataFrame(x,columns=['1','2','3','4'],dtype = float)
xy = Frame.idxmax(axis = 1)
Frame
datafr = pd.DataFrame({'Id': test_data['Id'],'Category': Frame.idxmax(axis = 1)})
datafr.to_csv('sub1.csv', header=False, index=False)
```