Assignment$_6$

Chaitra , Ganesh , Avik

June 29 2019

1 Artificial Neural Network

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import math
import random
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


# In[2]:


def readTrainigData():

    return pd.read_csv('./data/Big_DWH_Training.csv')


# In[3]:


def initializeParamteres():

    numberOfNeuronsInHiddenLayer = 5
    numberOfHiddenLayers = 2

    np.random.seed(37)
```

```
    # This will create a 5x2 matrix which are 10 weights of input layer to first
    wHiddenLayer1 = np.random.uniform(low=0, high=1, size=(numberOfNeuronsInHidd

    # This will create a 5x5 matrix which are 25 weights of first hidden layer t
    wHiddenLayer2 = np.random.uniform(low=0, high=1, size=(numberOfNeuronsInHidd

    # This is the weight of the ouput neuron.
    wOutputLayer = np.random.uniform(low=0, high=1, size=(numberOfNeuronsInHidde

    # This is the bias for each layer
    bHiddenLayer1 = bHiddenLayer2 = np.random.uniform(low=0,high=1,size=(number(
    bOutputLayer = np.random.uniform(low=0,high=1,size=(1,1))

    # This is is the weighed sum from input layer to first hidden layer.
    weightedSumH1 = np.zeros((len(wHiddenLayer1),1))

    # This is the activation of the first hidden layer
    activationH1 = np.zeros((len(wHiddenLayer1),1))

    # This is is the weighed sum from first hidden layer to the second hidden la
    weightedSumH2 = np.zeros((len(wHiddenLayer2),1))

    # This is the activation of the second hidden layer
    activationH2 = np.zeros((len(wHiddenLayer2),1))

    # This the weighted sum from 2nd hidden layer to the output layer.
    weightedSumOp = np.zeros((1,1))

    return wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, weightedSum
bHiddenLayer1, bHiddenLayer2, activationH1, activationH2, bOutputLayer


# In[4]:


def getTrainingSamples(trainingData):

    featureSet = np.zeros((len(trainingData),3))

    for index,row in trainingData.iterrows():

        featureSet[index][0] = row['Height']
        featureSet[index][1] = row['Weight']

        # Here the labels that are originally [1,-1] are coverted
to [1,0]
```

```
        if row['Gender'] != 1:
            featureSet[index][2] = 0
        else :
            featureSet[index][2] = 1

    return featureSet
```

In[5]:

```
def getStandardizedFeatureSet(featureSet):

    totalHeight = meanCorrectedHeight = 0.0
    totalWeight = meanCorrectedWeight = 0.0

    sampleSize = len(featureSet)
    normalizedFeatureSet = np.zeros((len(featureSet),3))

    for row in featureSet:

        totalHeight += row[0]
        totalWeight += row[1]

    hMean = totalHeight/sampleSize
    wMean = totalWeight/sampleSize
    print("Height mean: ",hMean)
    print("Weight mean: ",wMean)
    for rowIndex in range(len(featureSet)):

        normalizedFeatureSet[rowIndex][0] += (featureSet[rowIndex][0] - hMean)
        normalizedFeatureSet[rowIndex][1] += (featureSet[rowIndex][1] - wMean)
        normalizedFeatureSet[rowIndex][2] = featureSet[rowIndex][2]

        meanCorrectedHeight += math.pow(normalizedFeatureSet[rowIndex][0],2)
        meanCorrectedWeight += math.pow(normalizedFeatureSet[rowIndex][1],2)


    sdHeight = math.sqrt((meanCorrectedHeight/sampleSize))
    sdWeight = math.sqrt((meanCorrectedWeight/sampleSize))
    print("Height SD: ",sdHeight)
    print("Weight SD: ",sdWeight)
    for rowIndex in range(len(featureSet)):

        normalizedFeatureSet[rowIndex][0] = (normalizedFeatureSet[rowIndex][0] /
        normalizedFeatureSet[rowIndex][1] = (normalizedFeatureSet[rowIndex][1] /
```

```
    return normalizedFeatureSet


# In[6]:


def splitTrainingData(validationDataPercentage, featureSet):

    trainingSet, validationSet = train_test_split(featureSet, test_size=0.05)
    #print(validationSet.shape)
    return trainingSet, validationSet


# In[7]:


def applySigmoidActivation(weightedSum):

    activationOutput = np.zeros((len(weightedSum),1))

    for index in range(len(weightedSum)):

        activationOutput[index][0] = 1/(1+math.exp(-(weightedSum[index][0])))

    return activationOutput


# In[8]:


def computeError(output, label):

    # Log loss computation -(y*log(output) + (1 - y)log(1-output))

    if label == 0:
        return -(math.log(1 - output))
    else:
        return -(math.log(output))


# In[9]:


def feedForward(sample, wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH
                weightedSumOp, bHiddenLayer1, bHiddenLayer2, activationH1, activa
```

```
    heightOfSample = sample[0]
    weightOfSample = sample[1]
    label = sample[2]

    for index in range(len(wHiddenLayer1)):

        weightedSumH1[index][0] = ((heightOfSample * wHiddenLayer1[index][0]) +
                                   (weightOfSample * wHiddenLayer1[index][1])
                                  ) + bHiddenLayer1[index][0]
    activationH1 = applySigmoidActivation(weightedSumH1)
    for rowIndex in range(len(wHiddenLayer2)):

        tempSum = 0
        for colIndex in range(len(wHiddenLayer2[rowIndex])):

            tempSum += (activationH1[colIndex][0] * wHiddenLayer2[rowIndex][colI

        weightedSumH2[rowIndex][0] = tempSum
    activationH2 = applySigmoidActivation(weightedSumH2)
    tempSum = 0
    for rowIndex in range(len(activationH2)):
        tempSum += (activationH2[rowIndex][0] * wOutputLayer[rowIndex][0]) + bOu
    weightedSumOp[0][0] = tempSum

    outPut = applySigmoidActivation(weightedSumOp)
    error = computeError(outPut[0][0], label)

    #return outPut[0][0], error
    return wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, weightedSun
weightedSumOp, activationH1, activationH2,outPut[0][0], error


# In[10]:


def sigmoidDerivative(outputOfPreviousLayer):
    sigmoid = 1/(1+math.exp(-(outputOfPreviousLayer)))
    return ((sigmoid)*(1-sigmoid))


# In[11]:


def backPropogate(ouput, sample, wHiddenLayer1, wHiddenLayer2, wOutputLayer, weig
weightedSumOp, bHiddenLayer1, bHiddenLayer2, activationH1, activationH2, bOutput
```

```python
    #learningRate = 0.005
    errorDerivative = (ouput - sample[2])/(ouput-(ouput*ouput))
    outputLayerDerivative = []
    totalCalcDerivate = 0.0


    for rowIndex in range(len(wOutputLayer)):

        outputLayerDerivative.append(errorDerivative * sigmoidDerivative(weighted
        totalCalcDerivate = outputLayerDerivative[rowIndex] * activationH2[rowIn
        wOutputLayer[rowIndex][0] -= (LR * totalCalcDerivate)
    bOutputLayer[0][0] -= (LR * totalCalcDerivate)

    hiddenLayerOutputDerivative = np.zeros((5, 5))

    for rowIndex in range(len(wHiddenLayer2)):

        deltaForBias = 0
        hiddenLayerOutputDerivative[rowIndex][0] =
wOutputLayer[rowIndex][0] * sigmoidDerivative(weightedSumH2[rowIndex][0])
        for colIndex in range(len(wHiddenLayer2[rowIndex])):
            totalCalcDerivate = outputLayerDerivative[colIndex] * hiddenLayerOut
            deltaForBias += totalCalcDerivate
            wHiddenLayer2[colIndex][rowIndex] -= (LR * totalCalcDerivate)

        bHiddenLayer2[rowIndex][0] -= (LR * deltaForBias)

    for rowIndex in range(len(wHiddenLayer1)):
        tempSum = 0.0
        deltaForBias = 0
        tempDerivative = sigmoidDerivative(weightedSumH1[rowIndex][0])
        for colIndex in range(len(wHiddenLayer1[rowIndex])):

            for elementIndex in range(len(hiddenLayerOutputDerivative)):

                tempSum += (hiddenLayerOutputDerivative[elementIndex][0] * wHidd
            totalCalcDerivate = outputLayerDerivative[rowIndex] *
tempSum * tempDerivative * sample[colIndex]
            deltaForBias += totalCalcDerivate
            wHiddenLayer1[rowIndex][colIndex] -= (LR * totalCalcDerivate)
        bHiddenLayer1[rowIndex][0] -= (LR * deltaForBias)

    return wHiddenLayer1, wHiddenLayer2, wOutputLayer, bHiddenLayer1, bHiddenLay
```

In[12]:

```
def startNNTraining(trainingSet, validationSet, standardizedTestSet):

    print("Neural Network Started")
    epochs = 50000
    threshold = 0.5
    prediction = None
    correctResult = None
    validationAccuracy = None
    trainingAccuracy = None
    wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, weightedSumH2,
weightedSumOp, bHiddenLayer1, bHiddenLayer2, activationH1, activationH2, bOutput

    fwHiddenLayer1 = wHiddenLayer1
    fwHiddenLayer2 = wHiddenLayer2
    fwOuputLayer = wOutputLayer
    fbHiddenLayer1 = bHiddenLayer1
    fbHiddenLayer2 = bHiddenLayer2
    fbOutputLayer = bOutputLayer
    validationIterationCount = 0
    smallestValidationError = 100
    valList = []
    LRList = []
    for randLearnParam in range(10):
        LR = random.uniform((np.divide(1,epochs)),(np.divide(10,epochs)))
        for iteration in range(epochs):
            randomIndicesForSGD = random.sample(range(0, len(trainingSet)), 50)
            validationIterationCount += 1
            if validationIterationCount == 250:

                correctResult = 0
                validationAccuracy = 0.0
                for sample in validationSet:

                    wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, w
weightedSumOp, activationH1, activationH2, output, error =
feedForward(sample, wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, w
                    weightedSumOp, bHiddenLayer1, bHiddenLayer2, activationH1, a

                    if output >= threshold:
                        prediction = 1.0
                    else:
                        prediction = 0.0
```

```
            if sample[2] == prediction:
                correctResult += 1

        validationAccuracy = (correctResult / len(validationSet))*100

        print("Validation accuracy after 250 iterations is ", validation

        if (error < smallestValidationError):

            smallestValidationError = error

            fwHiddenLayer1 = wHiddenLayer1
            fwHiddenLayer2 = wHiddenLayer2
            fwOutputLayer = wOutputLayer
            fbHiddenLayer1 = bHiddenLayer1
            fbHiddenLayer2 = bHiddenLayer2
            fbOutputLayer = bOutputLayer

        validationIterationCount = 0
        #print('Smallest Val Error',smallestValidationError)
        #print('LR', LR)

    else:
        correctResult = 0.0
        for randomIndex in randomIndicesForSGD:

            wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, w
weightedSumOp, activationH1, activationH2, output, error =
feedForward(trainingSet[randomIndex], wHiddenLayer1, wHiddenLayer2, wOutputLayer
weightedSumOp, bHiddenLayer1, bHiddenLayer2, activationH1, activationH2, bOutput

            if output >= threshold:
                prediction = 1.0
            else:
                prediction = 0.0

            if trainingSet[randomIndex][2] == prediction:
                correctResult += 1

            wHiddenLayer1, wHiddenLayer2, wOutputLayer, bHiddenLayer1, b
bOutputLayer, LR = backPropogate(output, trainingSet[randomIndex], wHiddenLayer1
                                                weightedSumH1, weightedSumH2

        trainingAccuracy = (correctResult/50)*100
```

```
            if ((iteration +1) % 100 == 0):
                print("Training Loss after",iteration+1,"of",epochs, "is:",error
                print("Training accuracy after",iteration+1,"= ",trainingAccurac
        valList.append(smallestValidationError)
        LRList.append(LR)
        testModel(standardizedTestSet, fwHiddenLayer1, fwHiddenLayer2, fwOutputL
fbOutputLayer)
    return valList, LRList, smallestValidationError
    #return fwHiddenLayer1, fwHiddenLayer2, fwOutputLayer, fbHiddenLayer1, fbHid
    #             fbOutputLayer, valList, LRList, smallestValidationError


# In[13]:


def readTestData():

    return pd.read_csv('./data/DWH_test.csv')


# In[14]:


def testModel(standardizedTestSet, wHiddenLayer1, wHiddenLayer2, wOutputLayer, b
bOutputLayer):

    threshold = 0.5
    correctResult = 0
    # This is the activation of the first hidden layer
    activationH1 = np.zeros((len(wHiddenLayer1),1))

    # This is the activation of the second hidden layer
    activationH2 = np.zeros((len(wHiddenLayer2),1))

    # This the weighted sum from 2nd hidden layer to the output layer.
    weightedSumOp = np.zeros((1,1))


    # This is is the weighed sum from first hidden layer to the second hidden la
    weightedSumH2 = np.zeros((len(wHiddenLayer2),1))


    # This is is the weighed sum from first hidden layer to the second hidden la
    weightedSumH1 = np.zeros((len(wHiddenLayer2),1))
```

```
    for sample in standardizedTestSet:
        wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, weightedSumH2
weightedSumOp, activationH1, activationH2, output, error =
feedForward(sample, wHiddenLayer1, wHiddenLayer2, wOutputLayer, weightedSumH1, w
        weightedSumOp, bHiddenLayer1, bHiddenLayer2, activationH1, activationH2,

        #output, error = feedForward(sample, wHiddenLayer1, wHiddenLayer2, wOutp
        #         weightedSumOp, bHiddenLayer1, bHiddenLayer2, activationH1, acti

        if output >= threshold:
            prediction = 1.0
        else:
            prediction = 0.0

        if sample[2] == prediction:
            correctResult += 1

    testAccuracy = (correctResult / len(standardizedTestSet))*100
    print("Test accuracy = ",testAccuracy,"%.")



# In[15]:


def main():

    print('Stats of Training Data')
    trainingData = readTrainigData()
    featureSet = getTrainingSamples(trainingData)
    standardizedFeatureSet = getStandardizedFeatureSet(featureSet)
    trainingSet, validationSet = splitTrainingData(5, standardizedFeatureSet)

    print('Stats of Testing Data')
    testData = readTestData()
    featureTestSet = getTrainingSamples(testData)
    standardizedTestSet = getStandardizedFeatureSet(featureTestSet)

    valList, LRList, BestValScore = startNNTraining(trainingSet, validationSet,

    #wHiddenLayer1, wHiddenLayer2, wOutputLayer, bHiddenLayer1, bHiddenLayer2, \
    #             bOutputLayer, valList, LRList, BestValScore = startNNTraining(t

    #testModel(standardizedTestSet, wHiddenLayer1, wHiddenLayer2, wOutputLayer,
                #bOutputLayer)
```

```
    print('Val List:', valList)
    print('LR List:', LRList)
    print('Best Validation Score:', BestValScore)

    plt.plot(LRList, valList, linestyle='-', marker='o')
    plt.xlabel('Learning Rate')
    plt.ylabel('Validation Error')
    plt.title('Learning Rate vs Validation Error (10 values)')
    plt.show()


# In[16]:


if __name__ == '__main__':
    main()
```

Accuracy Comparison (in the test set):
Accuracy using SVM Stochastic Subgradient Descent = 83.6869 %
Accuracy using LIBLINEAR SVM = 88.6486 %
Accuracy of our neural network, Highest - 66.66 % and Lowest - 57.77 %

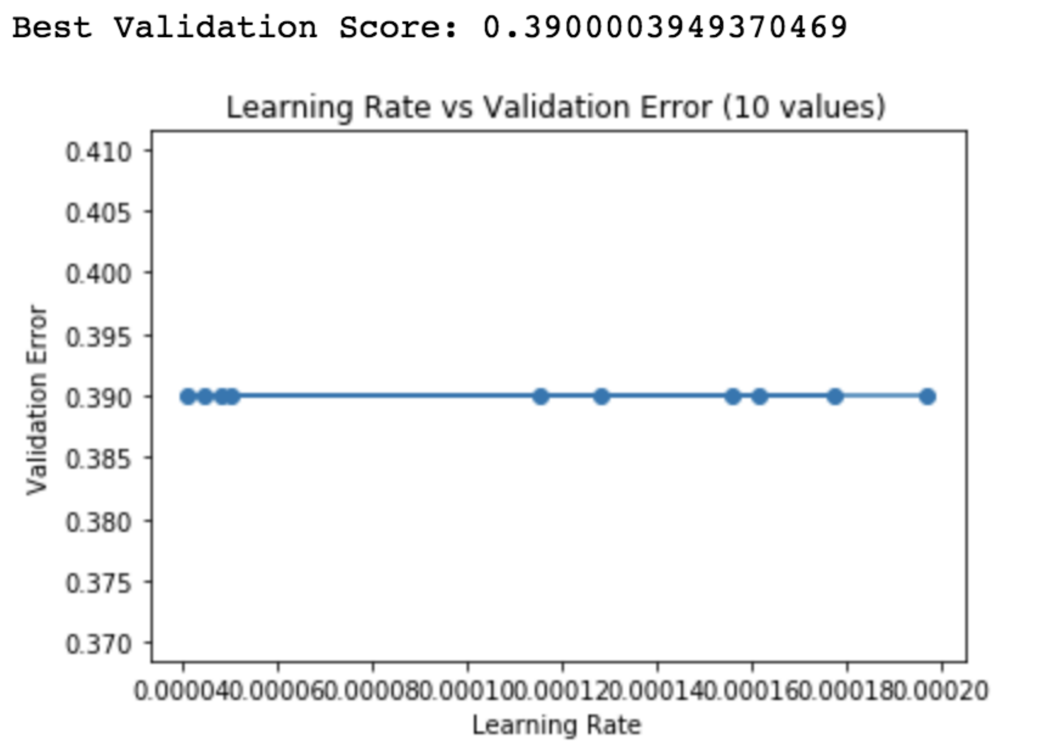

Figure 1: Learning Rate vs Validation Error

```
Test accuracy =  57.777777777777777 %.
Val List: [0.3900003949370469, 0.3900003949370469, 0.3900003949370469, 0.3900003949370469, 0.3900003949370469, 0.39
00003949370469, 0.3900003949370469, 0.3900003949370469, 0.3900003949370469, 0.3900003949370469]
LR List: [0.0001283516020178539, 4.1139211161739895e-05, 0.00016193875234906551, 0.00015594586512321194, 4.49406954
8588786e-05, 5.067386786468698e-05, 4.8335376949265507e-05, 0.0001778350437431043, 0.00011537493051296396, 0.000197
31381480919683]
Best Validation Score: 0.3900003949370469
```

Figure 2: Other Results