Assignment7

Chaitra , Ganesh , Avik

July 1 2019

1 7.1

```
#!/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/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
    deltaHiddenLayer1 = np.random.uniform(low=0, high=1, size=(numberOfNeuronsIn

    # 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
    deltaHiddenLayer2 = np.random.uniform(low=0, high=1, size=(numberOfNeuronsIn

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

    # This is the bias for each layer
    bHiddenLayer1 = bHiddenLayer2 = np.random.uniform(low=0,high=1,size=(numberO
    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, deltaHidd
deltaHiddenLayer2, deltaOutputLayer


# 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)
```

```python
    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, weightedSum
activationH2,outPut[0][0], error


# In[10]:


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


# In[11]:
```

```python
def backPropogate(ouput, sample, wHiddenLayer1, wHiddenLayer2, wOutputLayer, weig
                  bHiddenLayer1, bHiddenLayer2, activationH1, activationH2, bOut
                  deltaHiddenLayer2, deltaOutputLayer):

    errorDerivative = (ouput - sample[2])/(ouput-(ouput*ouput))
    outputLayerDerivative = []
    totalCalcDerivate = 0.0

    for rowIndex in range(len(wOutputLayer)):
        #Momentum Update
        outputLayerDerivative.append(errorDerivative * sigmoidDerivative(weighted
        totalCalcDerivate = outputLayerDerivative[rowIndex] * activationH2[rowIn
        deltaOutputLayer[rowIndex][0] = (psy*deltaOutputLayer[rowIndex][0]) + to
        wOutputLayer[rowIndex][0] -= (LR * deltaOutputLayer[rowIndex][0])
    bOutputLayer[0][0] -= (LR * totalCalcDerivate)

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

    for rowIndex in range(len(wHiddenLayer2)):

        deltaForBias = 0
        hiddenLayerOutputDerivative[rowIndex][0] = wOutputLayer[rowIndex][0] * s
        for colIndex in range(len(wHiddenLayer2[rowIndex])):
            totalCalcDerivate = outputLayerDerivative[colIndex] * hiddenLayerOut
            deltaHiddenLayer2[colIndex][rowIndex] = (psy * deltaHiddenLayer2[col
            deltaForBias += totalCalcDerivate
            wHiddenLayer2[colIndex][rowIndex] -= (LR * deltaHiddenLayer2[colInde

        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]
            deltaHiddenLayer1[rowIndex][colIndex] = (psy * deltaHiddenLayer1[row
            deltaForBias += totalCalcDerivate
            wHiddenLayer1[rowIndex][colIndex] -= (LR * deltaHiddenLayer1[rowInde
```

```
        bHiddenLayer1[rowIndex][0] -= (LR * deltaForBias)

    return wHiddenLayer1, wHiddenLayer2, wOutputLayer, bHiddenLayer1, bHiddenLay
deltaHiddenLayer2, deltaOutputLayer


# 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, we
bHiddenLayer2, activationH1, activationH2, bOutputLayer, deltaHiddenLayer1 ,delta
deltaOutputLayer = initializeParamteres()

    fwHiddenLayer1 = wHiddenLayer1
    fwHiddenLayer2 = wHiddenLayer2
    fwOuputLayer = wOutputLayer
    fbHiddenLayer1 = bHiddenLayer1
    fbHiddenLayer2 = bHiddenLayer2
    fbOutputLayer = bOutputLayer
    validationIterationCount = 0
    smallestValidationError = 100
    valList = []
    LRList = []
    PsyList = []
    for randLearnParam in range(5):
        psy = random.random()
        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
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 ", validationA

                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
activationH1, activationH2, output, error = feedForward(trainingSet[randomIndex]
                                                                               wOut
                                                                               bHid
                                                                               bOut

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

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

                        wHiddenLayer1, wHiddenLayer2, wOutputLayer, bHiddenLayer1, b
deltaHiddenLayer1, deltaHiddenLayer2, deltaOutputLayer =
backPropogate(output, trainingSet[randomIndex], wHiddenLayer1, wHiddenLayer2, wO
                                            weightedSumH1, weightedSumH2, weightedSumO
                                            activationH1, activationH2, bOutputLayer,L
                                            deltaHiddenLayer2, deltaOutputLayer)

                    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,"= ",trainingAccuracy
        valList.append(smallestValidationError)
        LRList.append(LR)
        PsyList.append(psy)
        testModel(standardizedTestSet, fwHiddenLayer1, fwHiddenLayer2, fwOutputL
fbOutputLayer)
    return valList, LRList, PsyList, 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)
```

```python
    valList, LRList, PsyList, BestValScore = startNNTraining(trainingSet, valida

    #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('Psy List:',PsyList)
    print('Best Validation Score:',BestValScore)


    fig, ax1 = plt.subplots()
    ax1.set_xlabel('Learning Rate')
    ax1.set_ylabel('Validation Error')
    ax1.plot(LRList, valList, linestyle='-', marker='o', color ='blue')
    ax1.tick_params(axis='y')
    ax2 = ax1.twinx()

    ax2.set_ylabel('Psy Values')
    ax2.plot(LRList, PsyList, linestyle='-', marker='o', color = 'red')
    ax2.tick_params(axis='y')
    fig.tight_layout()
    #plt.plot(LRList, valList, linestyle='-', marker='o')
    #plt.xlabel('Learning Rate')
    #plt.ylabel('Validation Error')
    plt.title('Learning Rate vs Validation Error vs Psy Values')
    plt.show()


# In[16]:


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

INFERENCE: The momentum method helped in increasing the accuray of the model.

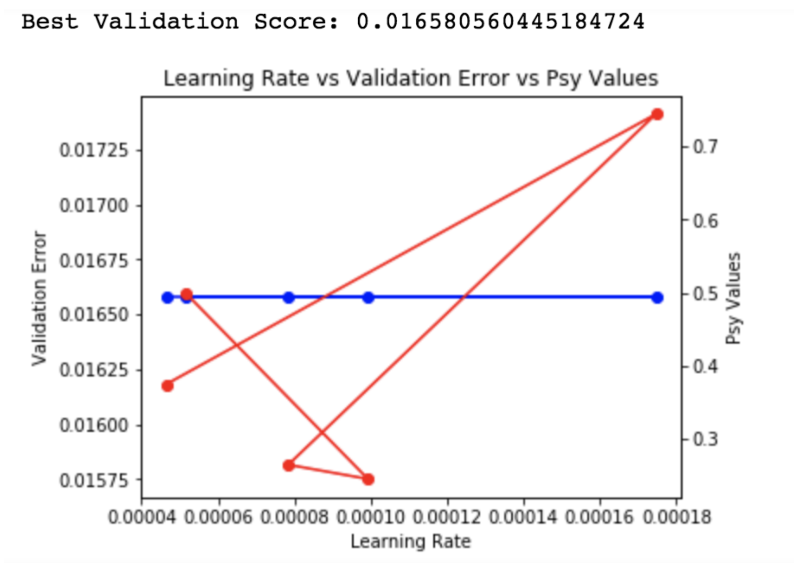

Figure 1: Learning Rate vs Validation Error vs Psy Values

```
Test accuracy =  80.0 %.
Val List: [0.016580560445184724, 0.016580560445184724, 0.016580560445184724, 0.016580560445184724, 0.01658056044518
4724]
LR List: [5.162981092376697e-05, 9.929730046679222e-05, 7.843780861649396e-05, 0.00017467847286190062, 4.6433039939
07212e-05]
Psy List: [0.49890615887308953, 0.24465007414878592, 0.2641752849712331, 0.7442272536640275, 0.3742192823910755]
Best Validation Score: 0.016580560445184724
```

Figure 2: Other Results

2 7.2

a) With Momentum,
Method 1: Random uniform weight initialization

```
Best Validation Score: 0.2676784284234564
```

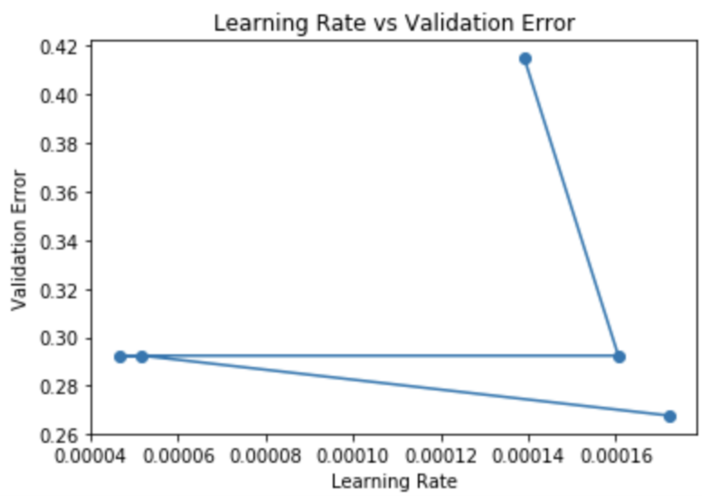

Figure 3: Learning Rate vs Validation Error (Method 1)

```
Test accuracy =  75.55555555555556 %.
Val List: [0.414895329797015, 0.2923732726401062, 0.2923732726401062, 0.2923732726401062, 0.2676784284234564]
LR List: [0.00013908576622578315, 0.00016057720164272486, 4.647224865380795e-05, 5.16916085242404e-05, 0.0001722119
9028597347]
Best Validation Score: 0.2676784284234564
```

Figure 4: Other Results (Method 1)

Method 2: Zero weight initialization

```
Best Validation Score: 0.1769086740814002
```

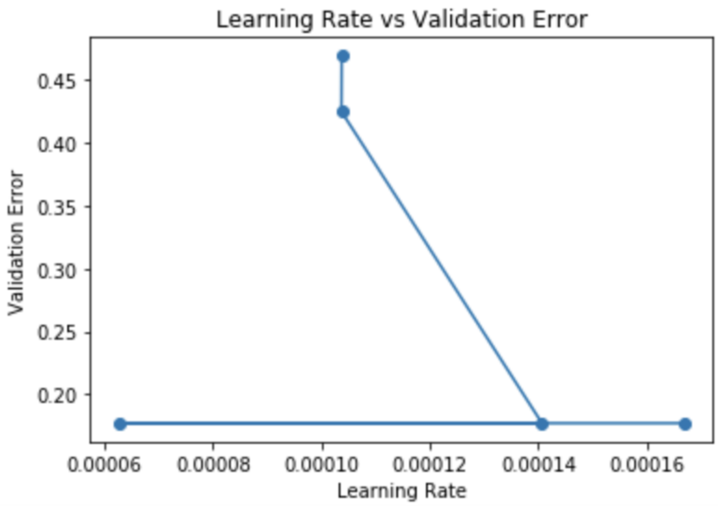

Figure 5: Learning Rate vs Validation Error (Method 2)

```
Test accuracy =  82.2222222222221 %.
Val List: [0.46939566929004994, 0.4248829151476636, 0.1769086740814002, 0.1769086740814002, 0.1769086740814002]
LR List: [0.00010373643451990835, 0.00010368624105041301, 0.000140655230238908, 6.262157900426877e-05, 0.0001669189
781983274]
Best Validation Score: 0.1769086740814002
```

Figure 6: Other Results (Method 2)

Method 3: Xavier weight initialization

Best Validation Score: 0.2794394189289825

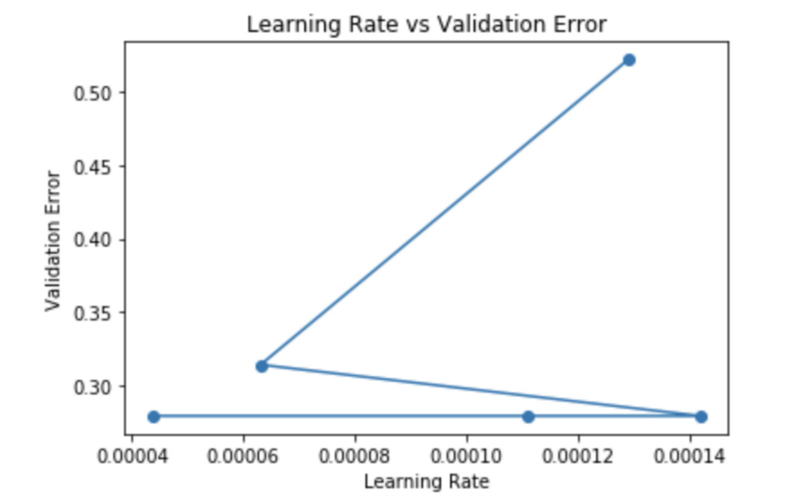

Figure 7: Learning Rate vs Validation Error (Method 3)

```
Test accuracy =  73.33333333333333 %.
Val List: [0.5219772359396917, 0.31425338979319317, 0.2794394189289825, 0.2794394189289825, 0.2794394189289825]
LR List: [0.00012911094290093633, 6.309237870325465e-05, 0.00014203999849697773, 0.00011101799368482724, 4.36042244
0198492e-05]
Best Validation Score: 0.2794394189289825
```

Figure 8: Other Results (Method 3)

b) Without Momentum,
Method 1: Random uniform weight initialization

```
Best Validation Score: 0.42947063776424066
```

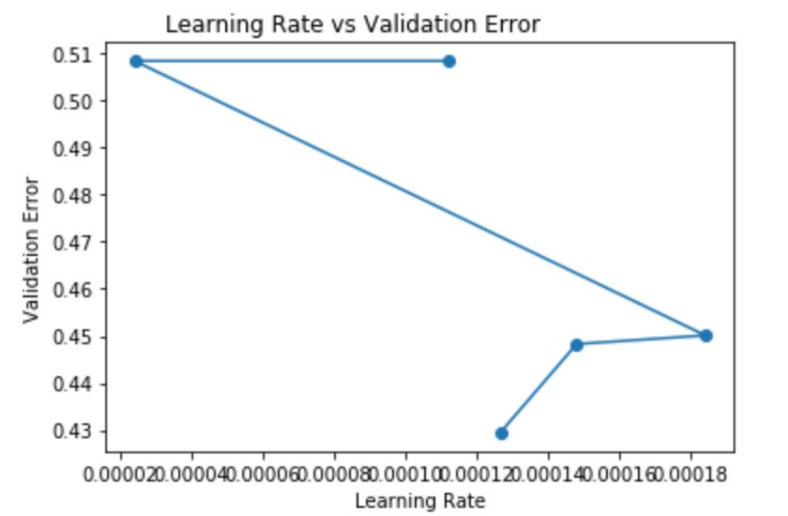

Figure 9: Learning Rate vs Validation Error (Method 1)

```
Test accuracy =  57.77777777777777 %.
Val List: [0.5084233355211298, 0.5084233355211298, 0.4501016273197388, 0.4482703367034948, 0.42947063776424066]
LR List: [0.00011220659808944273, 2.4089483225686756e-05, 0.0001844873851806272, 0.00014829773059012999, 0.00012688
305173251063]
Best Validation Score: 0.42947063776424066
```

Figure 10: Other Results (Method 1)

Method 2: Zero weight initialization

```
Best Validation Score: 0.6277491176726115
```

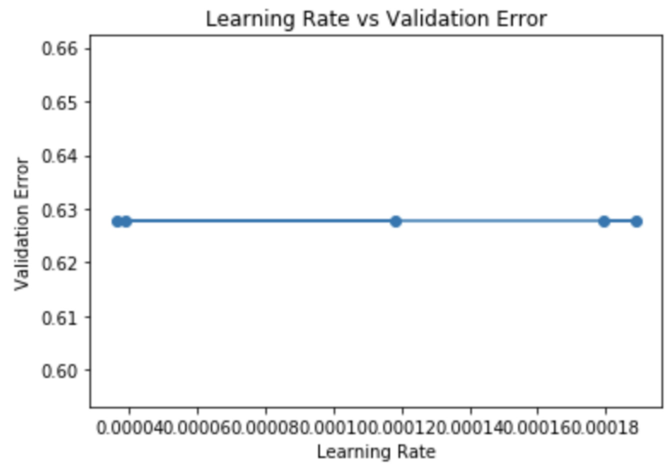

Figure 11: Learning Rate vs Validation Error (Method 2)

```
Test accuracy =  44.44444444444444 %.
Val List: [0.6277491176726115, 0.6277491176726115, 0.6277491176726115, 0.6277491176726115, 0.6277491176726115]
LR List: [0.0001793648770132244, 0.00018896752101655174, 3.6093696176431154e-05, 3.871468226154752e-05, 0.000118314
76275157661]
Best Validation Score: 0.6277491176726115
```

Figure 12: Other Results (Method 2)

Method 3: Xavier weight initialization

```
Best Validation Score: 0.12670178439978305
```

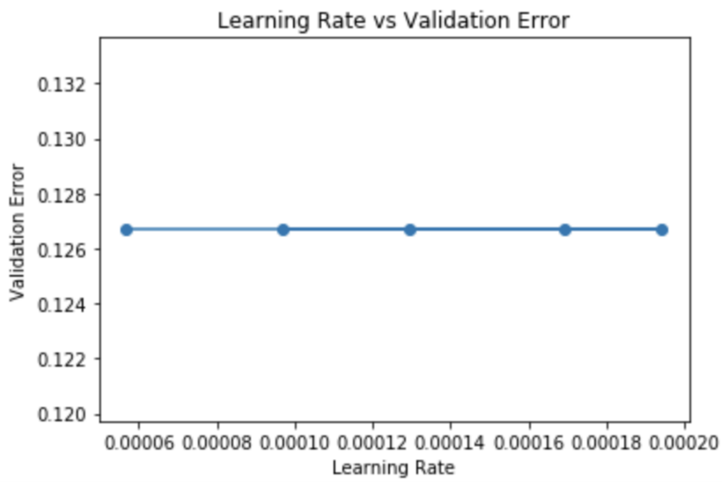

Figure 13: Learning Rate vs Validation Error (Method 3)

```
Test accuracy =  44.44444444444444 %.
Val List: [0.12670178439978305, 0.12670178439978305, 0.12670178439978305, 0.12670178439978305, 0.12670178439978305]
LR List: [5.6692424604767855e-05, 0.00019431403538870242, 9.68997344514569e-05, 0.00012957633364906279, 0.000169411
0121895974]
Best Validation Score: 0.12670178439978305
```

Figure 14: Other Results (Method 3)

c) INFERENCE: We saw that when we used momentum with the update rule, test accuracy was higher. The zero weight error has yielded us better accuracy that the Uniform weight initialization and Xavier initialization technique. Without the momentum the validation error was constant and also the test accuracy was low.

3 7.3

a) Iteration on 50 values for σ

```
import pandas as pd
import math
import numpy as np
from sklearn.svm import SVC
import statistics as stats
import matplotlib.pyplot as plt


# Module for training data set using RBF Kernel
def svmTrain(sigma,trainingData,trainingLabels):
    g = math.pow(2 * sigma, 1/2)
    clf = SVC(kernel='rbf',gamma=g)
    validationData,validationOutput,newTrainingData,newTrainingLabels = crosVail
    #print(newTrainingData)
    clf.fit(newTrainingData,newTrainingLabels)
    return svmValidate(clf,validationData,validationOutput),clf

# Module for normalizing training input values between 0 and 1
def normalizeDataSet(trainData):
    for i in range(0,len(trainData[0])-1):
        min1 = np.min(trainData[:,i])
        trainData[:,i] = trainData[:,i] - min1
        diff = np.max(trainData[:,i]) - np.min(trainData[:,i])
        trainData[:,i] = trainData[:,i] / diff
    return trainData


# Module for loading the data set
def prepeareData(dataFrame):
    X = np.zeros((len(dataFrame),2))
    Y = []
    X[:,0] = dataFrame['height'].tolist()
    X[:,1] = dataFrame['weight'].tolist()
    Y = dataFrame['gender'].tolist()
    Y[Y==-1]=0
    return X,Y
```

```python
# Module for validation using the best training weights
def svmValidate(clf, validationData, validationLabel):
    labels = []
    labels = clf.predict(validationData)
    print(labels)
    count = 0
    for i in range(0,len(labels)-1):
        if validationLabel[i] == labels[i]:
            count = count + 1
        else:
            continue
    return count/len(validationLabel)

# Module for measuring training accuracy using cross validation
def crosVailidation(trainingData, trainingLabels, numFolds):
    foldSize = len(trainingData)//numFolds
    validationData = np.zeros((foldSize,2))
    validationOutput = []
    newTrainingData = trainingData.copy()
    newTrainingLabels = trainingLabels.copy()
    for i in range(0,foldSize - 1):
        foldIndex = np.random.randint(0,len(newTrainingData))
        validationData[i,:] = trainingData[foldIndex,:]
        validationOutput.append(trainingLabels[foldIndex])
        newTrainingData = np.delete(newTrainingData, foldIndex, axis=0)
        newTrainingLabels.remove(trainingLabels[foldIndex])
    return validationData, validationOutput, newTrainingData, newTrainingLabels

def readDataFromCSV(fileName):
    return pd.read_csv(fileName)

# Main module: Entry Point of the code.
def main():
    trainFrame = readDataFromCSV('EX7data1.csv')
    trainingData, trainingLabels = prepeareData(trainFrame)
    trainingData = normalizeDataSet(trainingData)
    validationFrame = readDataFromCSV('EX7data2.csv')
    validationData, validationLabels = prepeareData(validationFrame)
    validationData = normalizeDataSet(validationData)

    trainAccuracy = []
    testAccuracy  = []
    sigmaList = []
    trainAccuracyS = []
    testAccuracyS  = []
```

```python
    sigmaListS = []

    for i in range(50):
        sigma = np.random.randint(math.pow(10,-6),math.pow(10,6))
        sigmaList.append(sigma)
        ta, clf = svmTrain(sigma, trainingData,trainingLabels)
        trainAccuracy.append(ta)
        testAccuracy.append(svmValidate(clf, validationData,validationLabels))

    for idx in np.argsort(sigmaList):
        sigmaListS.append(math.log(sigmaList[idx]))
        trainAccuracyS.append(trainAccuracy[idx])
        testAccuracyS.append(testAccuracy[idx])
    plt.plot(sigmaListS,trainAccuracyS,'red')
    plt.plot(sigmaListS,testAccuracyS,'blue')
    plt.show()
if __name__ == '__main__':
    main()
```

b)

c)

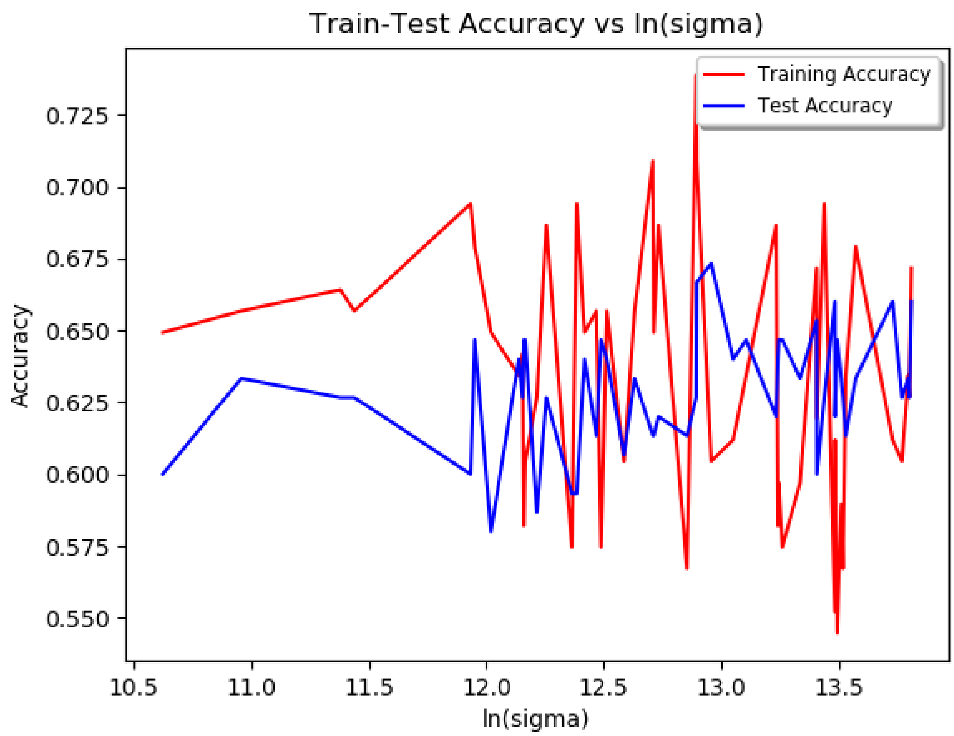

Figure 15: Plot of $\ln(\sigma)$vs Training and Testing Accuracy

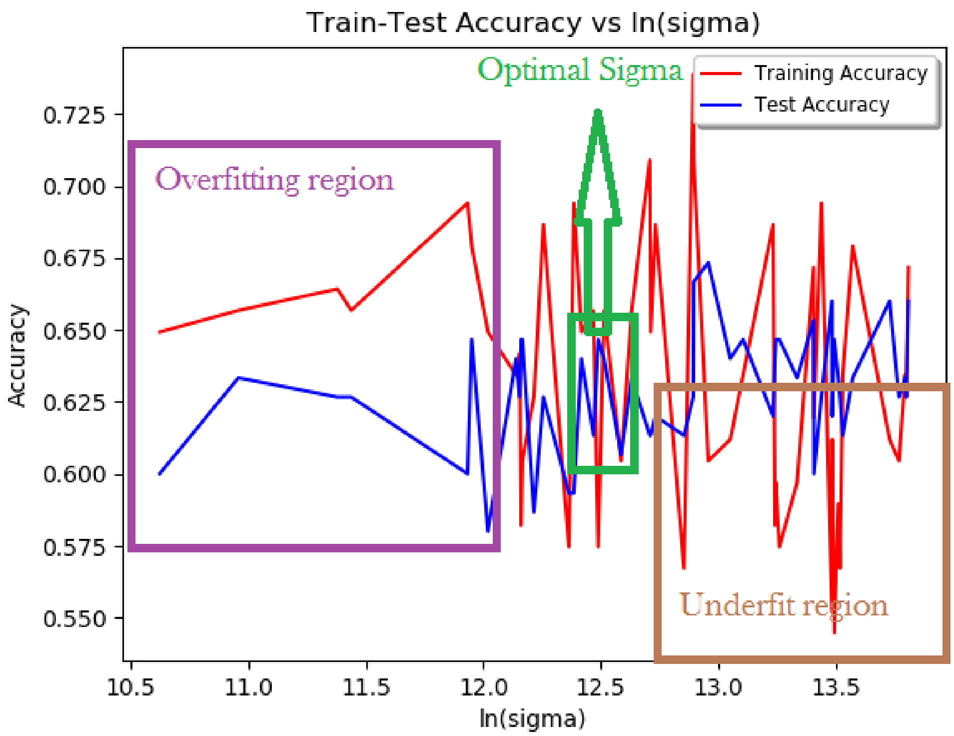

Figure 16: Overfitted, Underfitted and Optimal regions.