CSE 5523 Homework 4 Report:

Linear Models and SGD on Brain Image Data

Yuxiao Zhao zhao.2379@osu.edu

Problem: In this assignment you will train linear models using stochastic gradient descent to predict whether a human subject is viewing a picture or reading a sentence from their fMRI brain image data.

Stochastic Gradient Descent (7 points)

Four functions are implemented as follows:

```
def LogisticLoss(X, Y, W, lmda):
   #TODO: Compute (regularized) Logistic Loss
   loss = 0
   regularized = lmda * sum(W*W)
   loss = sum(np.logaddexp(0, -Y*np.dot(X,W))) + regularized
   return loss

def SgdLogistic(X, Y, maxIter, learningRate, lmda):
   W = np.zeros(X.shape[1])
   loss = LogisticLoss(X,Y,W,lmda)
   #TODO: implement stochastic gradient descent using the logistic loss function
   for t in range(maxIter):
      lastloss = loss
      # take a random training example
      j = np.random.randint(0,X.shape[0])
      logsum = np.logaddexp(0, Y[j]*np.dot(X[j],W))
      if logsum > 500:
         denom = float('inf')
      else:
         denom = np.exp(logsum)
      gradient = (-Y[j]*X[j])/denom
      gradient = gradient + 2*lmda*W
      W = W - learningRate * gradient
      # compute loss
      loss = LogisticLoss(X,Y,W,lmda)
      print("iteration %s: logistic loss is %s" % (t+1,loss))
      if abs(lastloss-loss) < 0.0001:
         print("Reach Convergence in iteration %s" % (t+1))
```

```
            break
    return W

def HingeLoss(X, Y, W, lmda):
    #TODO: Compute (regularized) Hinge Loss
    loss = 0
    regularized = lmda * sum(W*W)
    loss = sum(np.maximum(np.zeros(X.shape[0]),np.ones(X.shape[0])-Y*np.dot(X,W)))
+ regularized
    return loss

def SgdHinge(X, Y, maxIter, learningRate, lmda):
    W = np.zeros(X.shape[1])
    loss = HingeLoss(X,Y,W,lmda)
    #TODO: implement stochastic (sub) gradient descent with the hinge loss function
    for t in range(maxIter):
        lastloss = loss
        # take a random training example
        j = np.random.randint(0,X.shape[0])
        gradient = np.zeros(X.shape[1])
        if Y[j]*np.dot(X[j],W) < 1:
            gradient = -Y[j]*X[j]
        gradient = gradient + 2*lmda*W
        W = W - learningRate * gradient
        # compute loss
        loss = HingeLoss(X,Y,W,lmda)
        print("iteration %s: hinge loss is %s" % (t+1,loss))
        if abs(lastloss-loss) < 0.0001:
            print("Reach Convergence in iteration %s" % (t+1))
            break
    return W
```

Parameter Tuning (6 points)

Table 1: accuracy for hinge loss

Hinge loss accuracy	lamda = 1	lambda = 0.3	lamda = 0.1
eta = 0.1	0.55	0.65	0.8
eta = 0.01	0.6	0.65	***0.95***
eta = 0.001	0.85	0.85	0.85
eta = 0.0001	0.75	0.8	0.8

Table 2: accuracy for logistic loss

logistic loss accuracy	lamda = 1	lambda = 0.3	lamda = 0.1

eta = 0.1	0.55	0.7	0.7
eta = 0.01	0.65	0.65	***0.9***
eta = 0.001	0.8	0.85	0.8
eta = 0.0001	0.8	***0.9***	0.65

hinge_loss_cv.txt and logistic_loss_cv.txt are the two output files of the total loss in each iteration of cross validation. Table 1 and Table 2 report the accuracy of different combinations of eta and lamda.
Maximum iteration is set to 500, and the first 20 examples/trials after permutation are used to tune the parameters.
It turns out eta = 0.01, lamda = 0.1 works best for hinge loss; while eta = 0.01, lamda = 0.1 and eta = 0.0001, lamda = 0.3 work best for logistic loss.

Results on Test Data (2 points)

Parameters used for hinge loss are: eta = 0.01, lamda = 0.1
Parameters used for logistic loss are:eta = 0.0001, lamda = 0.3
hinge_test.txt and logistic_test.txt are the two output files of the total loss in each iteration and accuracy for two models based on the test data.
Accuracy for both of the models on test data is: 0.8235

Inspecting the Model's Parameters (2 Extra Credit Points)

I have ended up with a function to calculate the average absolute weight of the voxels in the region and through the time.

```
def ROI(W):
    # given the weight, calculate how import the region is (average absolute weight)
    W = W[:-1].reshape(data[1][0].shape)
    for roi in rois[0]:
        voxel_index = roi[2][0]-1
        W_abs = abs(W[:,voxel_index])
        average_weight = np.sum(W_abs)/(W_abs.shape[0]*W_abs.shape[1])
        print("roi:", roi[0],"  average absolute weight:", average_weight)
```

Printed-out information is stored in logistic_ROI.txt and hinge_ROI.txt for the weights in two models. ROI information is at the end of the file.
This one below is the logistic one:

roi: ['CALC'] average absolute weight: 0.0005275803590265385
roi: ['LDLPFC'] average absolute weight: 0.0004571212730998805
roi: ['LFEF'] average absolute weight: 0.000478095285365287
roi: ['LIFG'] average absolute weight: 0.0004770977102156525
roi: ['LIPL'] average absolute weight: 0.00044434267838446457

roi: ['LIPS'] average absolute weight: 0.0004489723489935803
roi: ['LIT'] average absolute weight: 0.0004652892115892433
roi: ['LOPER'] average absolute weight: 0.0004939923589501586
roi: ['LPPREC'] average absolute weight: 0.0004247786205568448
roi: ['LSGA'] average absolute weight: 0.00046901730191943006
roi: ['LSPL'] average absolute weight: 0.0004414923128896444
roi: ['LT'] average absolute weight: 0.00044441847896250147
roi: ['LTRIA'] average absolute weight: 0.00045183049219678926
roi: ['RDLPFC'] average absolute weight: 0.0004765684690957023
roi: ['RFEF'] average absolute weight: 0.0004996210773325901
roi: ['RIPL'] average absolute weight: 0.0005533699760969577
roi: ['RIPS'] average absolute weight: 0.0004926826397368311
roi: ['RIT'] average absolute weight: 0.0005061189568336304
roi: ['ROPER'] average absolute weight: 0.00047261444572919514
roi: ['RPPREC'] average absolute weight: 0.0004439986491721522
roi: ['RSGA'] average absolute weight: 0.0005469462802360609
roi: ['RSPL'] average absolute weight: 0.00044387570009018904
roi: ['RT'] average absolute weight: 0.0004612857476656962
roi: ['RTRIA'] average absolute weight: 0.0004627998796887444
roi: ['SMA'] average absolute weight: 0.0004196196004590173

The top three regions: ‘RIPL’, ‘RSGA’, ‘CALC’. This is also the same as in hinge loss.
So these three are the most important regions in the process of prediction.