CSE 5523: Homework 4
SGD-Brain-fMRI-Data

Soham Mukherjee
mukherjee.126@osu.edu

November 4, 2018

Abstract

Implementation of Stochastic Gradient Descent algorithm for Logistic Loss function and Hinge Loss function on fMRI Brain Image dataset.

1 How to run the program

I wrote this program in python 3.6.3. Set your python interpreter so that it uses Python 3. Open terminal and type in:

```
> python linear_brain.py
```

2 Brief Description

2.1 SVM

```
def HingeLoss(X, Y, W, lmda):
    loss = lmda * (W.dot(W))
    dotP = X.dot(W)
    dotP = Y * dotP
    loss = loss + np.sum(np.maximum(1 - dotP, 0))  # Element wise max with 0.
    return loss
```

Computes the HingeLoss for the flattened data X and weight vector W. This is regularized HingeLoss.

```
def SgdHinge(X, Y, maxIter, learningRate, lmda):
    previousloss = 0
    W = np.zeros(X.shape[1])
    for i in range(maxIter):
        for j in range(len(Y)):
            grad = np.zeros(X.shape[1])
```

```
            val = Y[j] * np.dot(W, X[j][:])
            if val < 1:
                grad = - Y[j] * X[j][:]
            grad = grad + 2 * lmda * W
            W = W - (learningRate * grad)
        loss = HingeLoss(X, Y, W, lmda)
        print("iteration: ", i, "HingeLoss: ", loss)
        if abs(loss - previousloss) < 0.0001:
            print("Convergence reached in ", i, "iterations")
            return W
        previousloss = loss
    return W
```

Computes stochastic gradient descent to minimize regularized HingeLoss. Note that weights are computed on each sample of data. Gradient are computed in piecewise manner for HingeLoss. Gradient descent is done for 100 epochs. If the loss difference is less than 0.0001 from previous epoch weight vector is returned.

2.2 Logistic Regression

```
def LogisticLoss(X, Y, W, lmda):
    loss = lmda * np.dot(W, W)
    dotP = -Y * (X.dot(W))
    logSum = np.logaddexp(0, dotP)
    loss += np.sum(logSum)
    return loss

    return loss
```

Computes the HingeLoss for the flattened data X and weight vector W. This is regularized HingeLoss.

```
    def SgdLogistic(X, Y, maxIter, learningRate, lmda):
    W = np.zeros(X.shape[1])
    previousloss = 0
    for i in range(maxIter):
        for j in range(len(Y)):
            val = Y[j] * np.dot(W, X[j][:])
            logSum = np.logaddexp(0, val)
            logExp = np.exp(logSum)
            grad = (- Y[j] * X[j][:])/logExp
            grad = grad + 2 * lmda * W
            W = W - learningRate * grad

        loss = LogisticLoss(X, Y, W, lmda)
        if abs(loss - previousloss) < 0.0001:
```

```
            print("Convergence reached in ", i, "iterations")
            return W
        previousloss = loss
        print("iteration: ", i, "Logisticloss: ", loss)

    return W
```

Computes stochastic gradient descent to minimize regularized LogisticLoss. Note that weights are computed on each sample of data. Here **LogSumExp** trick is used to avoid floating point underflow. Logsum is calculated using **np.logaddexp**. Gradient descent is done for 100 epochs. If the loss difference is less than 0.0001 from previous epoch weight vector is returned.

2.3 ROI for Image Classification

```
def interpret(W):
    W = W[:-1]
    shape = data[1][0].shape
    W = W.reshape(shape)
    w_sum = np.sum(W, axis=0)
    w_avg = np.divide(w_sum, float(shape[0]))
    region = np.zeros(len(rois[0]))
    for i in range(len(rois[0])):
        cols = np.array(rois[0][i]['columns'])
        for idx in cols[0]:
            region[i] += w_avg[idx - 1]   # index shift
        region[i] /= float(len(cols[0]))  # Compute aggregate of weights belonging to ea

    for colId in region.argsort()[::-1]:
        print("Weight : ", region[colId], " ROI : ", str(rois[0][colId]['name'][0]))  #
```

For the dataset weight vectors are obtained and average is computed and passed to this method. The weight vectors are reshaped and again average is computed. For each component of weight vector we find which region it belongs to. As there are 25 ROIS essentially weight vectors are put into 25 bins. Weights are then sorted in descending order and corresponding ROIS are found.

3 Output

Detailed output of the program can be found in the following files:

- Output_hinge_train.pdf : SVM Result on Training data

- Output_hinge_test.pdf : SVM Result on Test data
- Output_logistic_train.pdf : Logistic Regression Result on Training data
- Output_logistic_test.pdf : Logistic Regression Result on Test data
- Output_ROI.pdf :ROIs of Brain for Image Classification

3.1 Parameter Tuning of SGDHinge

Table 1 summarizes the result of parameter estimation for **SGDHinge**.

η	λ	Accuracy
$\eta = 0.1$	$\lambda = 1.0$	0.6
	$\lambda = 0.3$	0.75
	$\lambda = 0.1$	0.9 (max)
$\eta = 0.01$	$\lambda = 1.0$	0.65
	$\lambda = 0.3$	0.9 (max)
	$\lambda = 0.1$	0.75
$\eta = 0.001$	$\lambda = 1.0$	0.9 (max)
	$\lambda = 0.3$	0.8
	$\lambda = 0.1$	0.8
$\eta = 0.0001$	$\lambda = 1.0$	0.9 (max)
	$\lambda = 0.3$	0.7
	$\lambda = 0.1$	0.7

Table 1: Parameter Tuning summary for SVM on training data

Parameters chosen for SGDHinge are $\eta = 0.0001$**,** $\lambda = 1$. Obtained accuracy on TEST data is **0.8529411764705882**.

3.2 Parameter Tuning of SGDLogistic

Table 2 summarizes the result of parameter estimation for **SGDLogistic**.

Parameters chosen for SGDLogistic are $\eta = 0.0001$, $\lambda = 0.3$. Obtained accuracy on TEST data is **0.8529411764705882**.

3.3 ROI

The Region of Interests are:

```
    Weight :  2.7242567226794666e-05  ROI :  LSGA
Weight :  1.293672584059644e-05  ROI :  RFEF
Weight :  1.2338796821595056e-05  ROI :  LT
Weight :  1.4572137910665596e-06  ROI :  RIT
Weight :  9.879808159957877e-07  ROI :  LIT
```

η	λ	Accuracy
$\eta = 0.1$	$\lambda = 1.0$	0.6
	$\lambda = 0.3$	0.75
	$\lambda = 0.1$	0.8
$\eta = 0.01$	$\lambda = 1.0$	0.7
	$\lambda = 0.3$	0.75
	$\lambda = 0.1$	0.85
$\eta = 0.001$	$\lambda = 1.0$	0.85
	$\lambda = 0.3$	0.9
	$\lambda = 0.1$	0.85
$\eta = 0.0001$	$\lambda = 1.0$	0.95 (max)
	$\lambda = 0.3$	0.95 (max)
	$\lambda = 0.1$	0.95 (max)

Table 2: Parameter Tuning summary for Logistic Regression on training data

```
Weight :  -6.283657810470323e-06  ROI :  SMA
Weight :  -6.350770948407604e-06  ROI :  RSPL
Weight :  -8.871895589818572e-06  ROI :  LIPL
Weight :  -1.1782475952804245e-05  ROI :  RIPS
Weight :  -1.2297980934492027e-05  ROI :  LIPS
Weight :  -1.2378854991623306e-05  ROI :  LOPER
Weight :  -1.2581411012022484e-05  ROI :  RT
Weight :  -1.27498285560001e-05  ROI :  LPPREC
Weight :  -1.4203738350126933e-05  ROI :  LIFG
Weight :  -1.571195162264188e-05  ROI :  LFEF
Weight :  -1.6932988682756243e-05  ROI :  LTRIA
Weight :  -1.778358480041994e-05  ROI :  RPPREC
Weight :  -2.1039778971694152e-05  ROI :  CALC
Weight :  -2.2402963166251728e-05  ROI :  LSPL
Weight :  -2.311384506142807e-05  ROI :  LDLPFC
Weight :  -2.329843133294557e-05  ROI :  RIPL
Weight :  -2.5819991841406547e-05  ROI :  ROPER
Weight :  -3.0608236096153664e-05  ROI :  RTRIA
Weight :  -3.565447287370717e-05  ROI :  RDLPFC
Weight :  -4.7082183527612176e-05  ROI :  RSGA
```