CSE 5523

HOMEWORK IV – Report

Pragya Arora

arora.170

1. Stochastic Gradient Descent

1.1. Logistic Loss

Logistic loss for a dataset given a specific set of weights is calculated using the regularized logistic loss given below:

$$\mathcal{L}_{\log}(w) = \sum_l \log(1 + \exp(-y_l w \cdot x_l)) + \lambda \sum_i w_i^2$$

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

In order to avoid overflow error logaddexp is used.

The gradient is calculated using the below derivative of logistic loss

$$\frac{\partial \mathcal{L}_{log}(W)}{\partial w_i} = (-y_d x_d) \cdot \frac{exp(-y_d w \cdot x_d)}{1 + exp(-y_d w \cdot x_d)} + 2\lambda w_i$$

```python
def LogisticGradient(x, y, W, lmda):
    grad = 2 * lmda * W
    e = y * np.dot(x, W)
    grad += (-y * x * (1/np.exp(np.logaddexp(0,e))))
    return grad
```

LogisticLoss and LogisticGradient are combined to calculate Stochastic gradient descent for logistic loss. The LogisticLoss function returns the total loss which is used for monitoring convergence during stochastic gradient descent. The gradient update is stopped when the loss change is less than 0.0001 or when iterations reached to 100.

```python
def SgdLogistic(X, Y, maxIter, learningRate, lmda):
    W = np.zeros(X.shape[1])
    iter = 0
    loss_old = 0

    while iter < maxIter:
        # Over all the training examples
        for (xi, yi) in zip(X, Y):
            grad = LogisticGradient(xi, yi, W, lmda)
            W -= learningRate * grad

        # Calculate new Loss and update old loss
        loss = LogisticLoss(X, Y, W, lmda)
        print "Iteration : ", iter, "  Loss : ", loss
        if np.abs(loss_old - loss) < 0.0001:
            break
        else:
            loss_old = loss
        iter += 1

    return W
```

1.2. Hinge Loss

Hinge loss for a dataset given a specific set of weights is calculated using the regularized hinge loss given below:

$$\mathcal{L}_{\text{hinge}}(w) = \sum_{l} \max(0, 1 - y_l w \cdot x_l) + \lambda \sum_{i} w_i^2$$

```
def HingeLoss(X, Y, W, lmda):
    loss = lmda*np.dot(W,W)
    for (xi,yi) in zip(X,Y):
        s = yi*np.dot(W,xi)
        loss += max(0,1-s)
    return loss
```

The gradient is calculated using the following derivative of hinge loss

$$\frac{\partial \mathcal{L}_{hinge}(W)}{\partial w_i} == \begin{cases} 2\lambda w_i & \text{if } y_d W \cdot x_d > 1 \\ -y_d \cdot x_d + 2\lambda w_i & \text{otherwise} \end{cases}$$

```
def HingeGradient(x, y, W, lmda):
    grad = 2 * lmda * W
    s = y * np.dot(W, x)
    grad += 0 if s > 1 else -y * x
    return grad
```

HingeLoss and HingeGradient are combined to calculate Stochastic gradient descent for hinge loss. The HingeLoss function returns the total loss which is used for monitoring convergence during stochastic gradient descent. The gradient update is stopped when the loss change is less than 0.0001 or when iterations reached to 100.

```
def SgdHinge(X, Y, maxIter, learningRate, lmda):
    W = np.zeros(X.shape[1])
    iter = 0
    loss_old = 0

    while iter<maxIter:
        #Over all the training examples
        for (xi, yi) in zip(X, Y):
            grad = HingeGradient(xi,yi,W,lmda)
            W -= learningRate*grad

        #Calculate new Loss and update old loss
        loss = HingeLoss(X, Y, W, lmda)
        print "Iteration : ",iter,"  Loss : ",loss
        if np.abs(loss_old-loss)<0.0001:
            break
        else:
            loss_old=loss
        iter+=1

    return W
```

2. Parameter Tuning

Learning rate (η) and regularization parameter (λ) are tuned for each loss on the dataset using held out development data. Several values of η and λ are tried and the accuracy for each loss is reported below.

2.1. Logistic Loss

η	λ	Accuracy
0.1	1	0.6
0.1	0.3	0.75
0.1	0.1	0.8
0.1	0.2	0.7
0.1	0.5	0.6
0.01	1	0.7
0.01	0.3	0.75
0.01	0.1	0.85
0.01	0.2	0.8
0.01	0.5	0.8
0.001	1	0.85
0.001	0.3	0.9
0.001	0.1	0.85

0.001	0.2	0.8
0.001	0.5	0.85
0.0001	**1**	**0.95**
0.0001	**0.3**	**0.95**
0.0001	**0.1**	**0.95**
0.0001	**0.2**	**0.95**
0.0001	**0.5**	**0.95**

The maximum accuracy of 0.95 over the dev set for logistic loss is attained at $\eta = 0.0001$ and $\lambda = \{1, 0.3, 0.1, 0.2, 0.5\}$.

2.2. Hinge Loss

η	λ	Accuracy
0.1	1	0.6
0.1	0.3	0.75
0.1	**0.1**	**0.9**
0.1	0.2	0.65
0.1	0.5	0.65
0.01	1	0.65
0.01	**0.3**	**0.9**
0.01	0.1	0.75
0.01	0.2	0.85
0.01	0.5	0.75
0.001	**1**	**0.9**
0.001	0.3	0.8
0.001	0.1	0.8
0.001	0.2	0.85
0.001	0.5	0.75
0.0001	**1**	**0.9**
0.0001	0.3	0.7

0.0001	0.1	0.7
0.0001	0.2	0.7
0.0001	0.5	0.7

The maximum accuracy of 0.9 over the dev set for hinge loss is attained at η = 0.1 λ = 0.1, η = 0.01 λ = 0.3, η = 0.001 λ = 1 and η = 0.0001 λ = 1

The loss at each iteration for both the losses is provided in output.txt.

3. Results on Test Data

The performance of the model is evaluated on the remaining trials at **η = 0.0001** and **λ = 1** as it gave the maximum accuracy for dev set for both the losses. The final accuracy attained is **0.852941176471** for both the loss functions.

4. Inspecting the Model's Parameters

ROI is associated with a few columns from Weight Vector W which is of dimension 258391. The Weight vector is flattened so it is collapsed to the dimension (55,4698). Then a new Weight vector is created which has sum over all-time series and is of dimension 4698. There are 25 ROIs and each ROI has column indexes of voxels in that ROI. For each ROI weights corresponding to column indexes of voxels are added and averaged out. The most positive and negative weights corresponding to all ROIs are given below for both the losses.

```
def RoisCal(W):
    W = np.delete(W, W.shape[0]-1) # remove bias
    W = np.reshape(W,data[1][0].shape)
    size = W.shape
    WNew = np.sum(W,axis=0)

    roi = rois[0]
    WRoi = np.zeros(roi.shape[0])

    for i in range(roi.shape[0]):
        col = roi[i]['columns']
        col = col[0]
        for j in col:
            WRoi[i] += WNew[j-1]
        WRoi[i] /= float(col.size)

    indices = np.argsort(WRoi)[::-1]
    c = 1
    for i in indices:
        print c, ' ', str(roi[i]['name'][0]), ' ', WRoi[i]
        c += 1
```

4.1. Hinge Loss

ROI	WEIGHT
RFEF	0.000374256
LT	0.000301538
LSGA	0.000290695
LOPER	-0.000415233
LIFG	-0.000415465
LTRIA	-0.000415813
LIT	-0.000424368
LIPS	-0.000440641
RIT	-0.000461767
RT	-0.00052224
LIPL	-0.000581742
SMA	-0.000662904
LPPREC	-0.000727877
LFEF	-0.000836142
LDLPFC	-0.000894162
RSPL	-0.001088753
LSPL	-0.001136585
RPPREC	-0.001253358
CALC	-0.001307648
RTRIA	-0.001319133
ROPER	-0.001329117
RIPS	-0.001462256
RDLPFC	-0.001895765
RIPL	-0.002544744
RSGA	-0.003114439

4.2. Logistic Loss

ROI	WEIGHT
RFEF	0.000571516
LT	0.000509996
LFEF	0.000238463
LTRIA	0.00018534
LIT	-3. 609247628e-05
RT	-9. 89635797238e-05
LIFG	-0.000105767
LIPS	-0.000143016

LSGA	-0.000196331
LOPER	-0.000300413
RIT	-0.000342502
LIPL	-0.000506917
LDLPFC	-0.000569905
LPPREC	-0.00057417
SMA	-0.000690718
RIPS	-0.000994216
RPPREC	-0.001039914
LSPL	-0.001041802
RSPL	-0.00117856
CALC	-0.00126905
ROPER	-0.001294285
RTRIA	-0.001495563
RDLPFC	-0.001744643
RSGA	-0.002508414
RIPL	-0.002799098