华南理工大学

South China University of Technology

The Experiment Report of
Machine Learning

College Software College

Subject Software Engineering

Members : 1

Student ID :201530612101

E-mail :2435489989@qq.com

Tutor : Ming Kui. Tan

Date submitted 2017. 12 . 15

1. Topic: Logistic Regression, Linear Classification and Stochastic Gradient Descent

2. Time: 2017/12/9

3. Reporter: 梁功达

4. Purposes:

1. Compare and understand the difference between gradient descent and stochastic gradient descent.

2. Compare and understand the differences and relationships between Logistic regression and linear classification.

3. Further understand the principles of SVM and practice on larger data.

5. Data sets and data analysis:

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. As we can see that the dataset’s labels are -1 and +1,and our purpose is to predict the labels using logistic regression and linear classification. So we need to read the testing data and validation data into our code.

6. Experimental steps:

For Logistic Regression and Stochastic Gradient Descent:

1. Load the training set and validation set.

2. Initalize logistic regression model parameters, you can consider

initalizing zeros, random numbers or normal distribution.

3. Select the loss function and calculate its derivation, find more detail in PPT.

4. Calculate gradient toward loss function from partial samples.

5.Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).

6. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method Loss_NAG, Loss_RMSProp, Loss_AdaDelta, Loss_Adam.

7. Repeate step 4 to 6 for several times, and drawing graph of Loss_NAG, Loss_RMSProp, Loss_AdaDelta and Loss_Adam with the number of iterations.

For Linear Classification and Stochastic Gradient Descent

1. Load the training set and validation set.

2. Initalize SVM model parameters, you can consider initalizing zeros, random numbers or normal distribution.

3. Select the loss function and calculate its derivation, find more detail in PPT.

4. Calculate gradient toward loss function from partial samples.

5.Update model parameters using different optimized methods(NAG， RMSProp， AdaDelta and Adam).

6. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method Loss_NAG, Loss_RMSProp, Loss_AdaDelta, Loss_Adam.

7. Repeate step 4 to 6 for several times, and drawing graph of Loss_NAG, Loss_RMSProp, Loss_AdaDelta and Loss_Adam with the number of iterations.

7. Code:

For Logistic Regression and Stochastic Gradient Descent:

①Randomly read the samples and combinating them into an array:

```
def select_sample(X, Y, n, index_list):#批处理抽出样本
    sample_index= random.sample(index_list, n)
    X_samples =np.ones((0, X.shape[1]))
                                        hape[1]))
    for item in sample_index:
        X_samples = np.r_[X_samples, X[item].reshape(1, X.shape[1])]
        Y_samples = np.r_[Y_samples, Y[item].reshape(1, Y.shape[1])]
    return X_samples, Y_samples
```

PEP 8: missing whitespace around operator

② SGD,not yet optimizztion:

```
def SGD(theta,learning_rate,epoch=200):#得到随机梯度

    for j in range(1,epoch+1):#迭代200次，每次samplenumber个样本
     m_gradient = np.zeros(theta.shape)
     X, Y = select_sample(X_train, Y_train, samplesnumber, index_list)  # 选取样本
     for i in range(0,Y.shape[0]):

       Xi=X[i].reshape(1,X.shape[1]).T
       Yi=Y[i][0]
       m_gradient += -  (Xi * Yi) / (1 + math.exp(Yi*np.dot(theta.T,Xi)[0][0]))
     gradient =m_gradient/samplesnumber+  lunda * theta
     theta =  theta -learning_rate*gradient
     loss = loss_funtion(X_test, Y_test, theta)
     print("Loss_SGD:", loss)
     Loss_SGD.append(loss)
    return  Loss_SGD
```

③NAG

```
def NAG(theta,learning_rate,lunda,epoch=200):#NAG
    Vt = np.zeros(theta.shape)
    for j in range(1,epoch+1):
      m_gradient = np.zeros(theta.shape)
      X, Y = select_sample(X_train, Y_train, samplesnumber, index_list)  # 选取样本
      for i in range(0,Y.shape[0]):
       #对theta(t-1)-lengta*V(t-1)求导数
         Xi = X[i].reshape(1, X.shape[1]).T
         Yi = Y[i][0]
         m_gradient+=-((Xi*Yi)/(1+math.exp(Yi*np.dot((theta-lengta*Vt).T,Xi)[0][0])) )
      gradient=m_gradient/samplesnumber+lunda*(theta-lengta*Vt)
      Vt=lengta*Vt+learning_rate*gradient
      theta=theta-Vt
      loss = loss_funtion(X_test, Y_test, theta)
      print("Loss_NAG:", loss)
      Loss_NAG.append(loss)
    return Loss_NAG
```

④RMSProp

```
def RMSProp(theta,epoch=200):#RMSProp
    epsilon = 1e-8
    lengta=0.9
    learning_rate=0.01
    Gt = np.zeros(theta.shape)
    for j in range(1,epoch+1):
        #这里的学习率 0.001
        X, Y = select_sample(X_train, Y_train, samplesnumber, index_list)  # 选取样本
        m_gradient = np.zeros(theta.shape)
        for i in range(0, Y.shape[0]):
            Xi = X[i].reshape(1, X.shape[1]).T
            Yi = Y[i][0]
            m_gradient += -  (Xi * Yi) / (1 + math.exp(Yi * np.dot(theta.T, Xi)[0][0]))
        gradient = m_gradient / samplesnumber + lunda * theta
        Gt=lengta*Gt+(1-lengta)*gradient*gradient
        theta=theta-learning_rate*gradient/np.sqrt(Gt+epsilon)
        loss = loss_funtion(X_test, Y_test, theta)
        print("Loss_RMSProp:", loss)
        Loss_RMSProp.append(loss)
```

⑤AdaDelta

```
def AdaDelta(theta,epoch=200):#AdaDelta,无初始学习率,用sqrt{Delta_{t-1} + epsilon}来估计学习速率,lengda=0.95
    lengda=0.999#yuan0.95
    epsilon = 1e-8
    delta_t = np.zeros(theta.shape)
    Gt = np.zeros(theta.shape)
    for j in range(1,epoch+1):
        m_gradient = np.zeros(theta.shape)
        X, Y = select_sample(X_train, Y_train, samplesnumber, index_list)  # 选取样本
        for i in range(0, Y.shape[0]):
            Xi = X[i].reshape(1, X.shape[1]).T
            Yi = Y[i][0]
            m_gradient += -  (Xi * Yi) / (1 + math.exp(Yi * np.dot(theta.T, Xi)[0][0]))

        gradient = m_gradient / samplesnumber + lunda * theta
        Gt =lengda* Gt + np.sum((1 - lengda) * (gradient**2))
        delta_theta=-np.sqrt(delta_t+epsilon)/np.sqrt(Gt+epsilon)*gradient
        theta = theta +delta_theta # 偷*learning_rate
        delta_t=lengda*delta_t+(1-lengda)*delta_theta*delta_theta
        loss = loss_funtion(X_test, Y_test, theta)
```

⑥Adam

```
def Adam(theta,epoch=200):#和Adam \beta_1取个0.9（可能需要衰减）
    beta_1=0.9
    lengda=0.999
    learning_rate=0.01
    epsilon = 1e-8
    m_t = np.zeros(theta.shape)
    Gt = np.zeros(theta.shape)
    for j in range(1,epoch+1):
        m_gradient = np.zeros(theta.shape)
        X, Y = select_sample(X_train, Y_train, samplesnumber, index_list)  # 选取样本
        for i in range(0, Y.shape[0]):
            Xi = X[i].reshape(1, X.shape[1]).T
            Yi = Y[i][0]
            m_gradient += -  (Xi * Yi) / (1 + math.exp(Yi * np.dot(theta.T, Xi)[0][0]))
        gradient = m_gradient / samplesnumber + lunda * theta
        m_t=beta_1*m_t+(1-beta_1)*gradient
        Gt=lengda*Gt+(1-lengda)*gradient*gradient
        alpha=learning_rate*math.sqrt(1-pow(lengda,i))/(1-pow(beta_1,j))
        theta=theta-alpha*m_t/np.sqrt(Gt+epsilon)
        loss = loss_funtion(X_test, Y_test, theta)
```

Loss function:

```
def loss_funtion(X,Y,theta):
    m_loss=0
    for i in range (0,Y.shape[0]):
        Xi = X[i].reshape(1, X.shape[1]).T
        Yi = Y[i][0]
        m_loss+=math.log(1+math.exp(-Yi*np.dot(theta.T,Xi)[0][0]))
    loss=1/Y.shape[0]*m_loss+0.5 * np.dot(theta.T, theta).sum()
    #print("first: ",1/samplesnumber*m_loss)
    #print("second:",0.5 * np.dot(theta.transpose(), theta).sum())
    return loss
```

Gradient:

```
for i in range(0,Y.shape[0]):

    Xi=X[i].reshape(1,X.shape[1]).T
    Yi=Y[i][0]
    m_gradient += -  (Xi * Yi) / (1 + math.exp(Yi*np.dot(theta.T,Xi)[0][0]))
gradient =m_gradient/samplesnumber+ lunda * theta
```

For Linear Classification and Stochastic Gradient Descent:

The optimization algorithm of linear classification is the same as before. However, I change the loss function and the compute of

gradient.

Loss :

```
def loss_funtion(X, Y, theta):
    epsilon_loss = 1 - Y * X.dot(theta)
    epsilon_loss[epsilon_loss < 0] = 0  # 把小于0的设为0,
    loss = 0.5 * np.dot(theta.transpose(), theta).sum()/ X.shape[0] + C * epsilon_loss.sum()  # .sum()函数将矩阵内的数全部相加
    return loss
```

Gradient:

```
for j in range(1, epoch + 1):  # 迭代200次，每次samplenumber个样本
    X, Y = select_sample(X_train, Y_train, samplesnumber, index_list)  # 选取样本


    epsilon_gredient = -np.dot(X.transpose(), Y)  #
    gradient = theta + C * epsilon_gredient/Y.shape[0]
```

(Fill in the contents of 8-11 respectively for logistic regression and linear classification)

8. The initialization method of model parameters:

logistic regression : Initial to random numbers ;

linear classification : Initial to random numbers

9.The selected loss function and its derivatives:

logistic regression :

$$\min_{\mathbf{w},b} \quad L(\mathbf{w},b) = \frac{\|\mathbf{w}\|^2}{2} + \frac{C}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

and

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b) = \mathbf{w} + \frac{C}{n} \sum_{i=1}^{n} g_{\mathbf{w}}(\mathbf{x}_i)$$

$$\nabla_b L(\mathbf{w}, b) = \frac{C}{n} \sum_{i=1}^{n} g_b(\mathbf{x}_i)$$

linear classification :

$$J(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i \cdot \mathbf{w}^\top \mathbf{x}_i}) + \frac{\lambda}{2} ||\mathbf{w}||_2^2$$

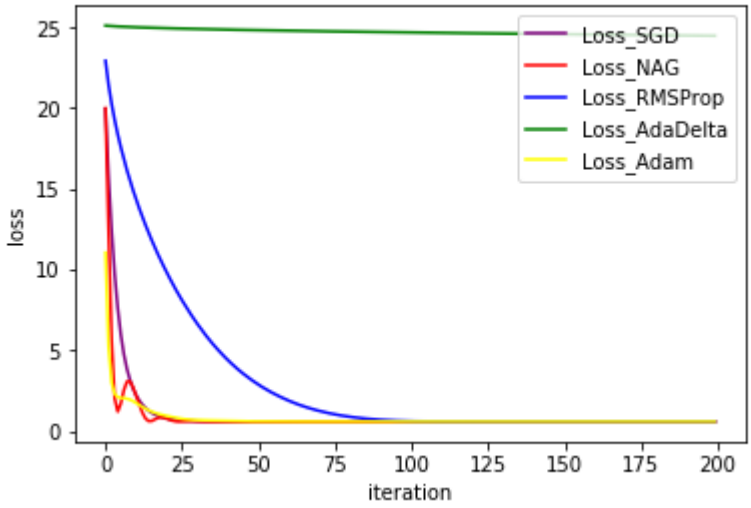
and

$$\mathbf{w}' \to \mathbf{w} - \eta \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = (1 - \eta\lambda)\mathbf{w} + \eta \frac{1}{n} \sum_{i=1}^{n} \frac{y_i \mathbf{x}_i}{1 + e^{y_i \cdot \mathbf{w}^\top \mathbf{x}_i}}$$

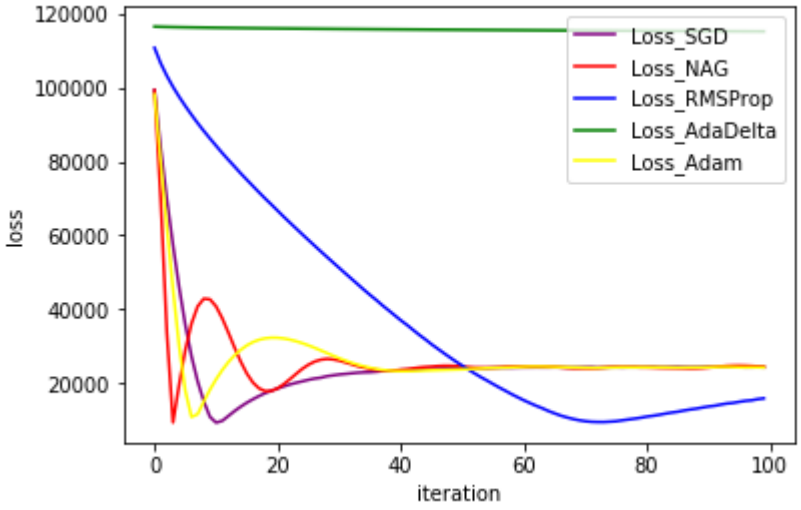
10.Experimental results and curve:(Fill in this content for various methods of gradient descent respectively)

logistic regression :

Loss curve:



linear classification :



11. Results analysis:

As you can see that , the RMSProp is wrong, and I can not find the bug. And theoretically, the other four loss is supposed to lower than the loss of SGD. However, in my curve, the loss of RMSProp and AdaDelta are not suitable, and the trouble is that I already

have no time to debug the code.

And for the result, I can not get the real answer. And I will do it well later.

12. Similarities and differences between logistic regression and linear classification：

Similarities: both of them use the hyperplane as the decision boundary

Difference: The logical regression can be judged by the threshold value after the output prediction probability, but the SVM directly outputs the segmentation hyperplane, and then uses the 0/1 function to classify the distance, and can not directly output the probability value.

13.Summary:

In all, it is really a challenge for me .I' sorry I can't get out the real answer. However, I've got a lot. I've knowed the loss and gradient of logistic regression. And I have knowed the four ways to optimize SGD. The main idea of NAG is to use Momentum to predict the next step, rather than using the current theta. The AdaGrad uses the previous gradient information to determine whether the corresponding feature J is often updated. And the AdaDelta uses sqrt{Delta_{t-1} + epsilon} to estimate learning rate.

And the last one, the Adam makes use of the advantages of

AdaGrad and RMSProp on sparse data. The correction of the initialized deviation also makes the Adam better performance.

The four optimization is difficult for me , but I've got the main idea of them. And I've realized the difference between logistic regression and linear classification. It was a hard but happy journey. I like it and I will make myself better.