South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author: 周东东

Supervisor: Qingyao Wu

Student ID: 201530613849

Grade: Undergraduate

December 12,2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

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

2. Time: 2017/12/2

3. Reporter:周东东

4. Purposes:

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

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

4.3 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. Please download the training set and validation set.

6. Experimental steps:

6.1 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 G 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 Lnag， Lrmsprop，Ladadelta and Ladam.

7.Repeate step 4 to 6 for several times, and drawing graph of Lnag， Lrmsprop，Ladadelta and Ladamwith the number of iterations.

6.2 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 G 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 Lnag， Lrmsprop，Ladadelta and Ladam.

7.Repeate step 4 to 6 for several times, and drawing graph of Lnag， Lrmsprop，Ladadelta and Ladam. with the number of iterations.

7. Code:

7.1 Logistic Regression and Stochastic Gradient Descent

```
from sklearn.datasets import load_svmlight_file
import numpy as np
from scipy.sparse import csr_matrix, hstack
import matplotlib.pyplot as plt
import math
import random
%matplotlib inline

def data_loader(train_file):
   X,Y = load_svmlight_file(train_file)
   X = X.toarray()
   X = np.c_[np.ones((X.shape[0],1)),X]
   return X, Y

def sigmoid(inX):
   return 1.0/(1+np.exp(-inX))

def loss(X,Y,W):
   m,n = np.shape(X)
   loss=0
   for i in range(m):
      inX = Y[i]*W.T*X[i]
      loss += log(1+sigmoid(inX))
   return loss/m

def sto_batch_grad(X,Y,W):
   #Mini-batch gradient
   m,n=np.shape(X)
   dataIndex = range(m)
   randIndex = int (random.uniform(0,len(dataIndex))
   X_part = np.mat(X(randIndex,randIndex+100))
   Y_part = np.mat(Y(randIndex,randIndex+100))
   h = sigmoid(X_part*W)
   error = h - Y_part
   G = X_part.transpose() * error #G(14,1)
   return G

def sto_gradDecline(X,Y,numIter=100):
   m,n=np.shape(X)
   W = np.ones(n)
   for j in range(numIter):
```

```
        dataIndex = range(m)
        for i in range(m):
            learn_rate = 4/(1.0+j+i)+0.01
            randIndex = int (random.uniform(0,len(dataIndex)))
            h = sigmoid(sum(X[randIndex]*W))
            error = h - Y[randIndex]
            W = W - learn_rate*error*X[randIndex]
            del(randIndex)
    return W

def NAG(X,Y,learn_rate=.05, gamma=.9):
    """
    m,n=np.shape(X)
    v = []
    next_v = [gamma * v[i] + eta * gradients[i] for i in
range(para_num)]
    updates = [(v[i], next_v[i]) for i in range(para_num)]
    updates.extend([(parameters[i], parameters[i] - gamma *
next_v[i] - eta * gradients[i])
                for i in range(para_num)])
    updates.extend([(t, t + 1)])
    return updates
    """
    X_matrix = np.mat(X)
    Y_matrix = np.mat(Y)
    m,n = np.shape(X_matrix)
    maxCycle = 500
    W = np.ones((n,1))
    v = np.ones((n,1))
    for k in range(maxCycle):
        G = sto_batch_grad(X_matrix,Y_matrix,W-gamma*v)
        v = gamma * v + learn_rate * G
        W = W - v
    return W


def RMSProp(X,Y,learn_rate=0.001, gamma=0.9, epsilon=1e-
8):
    X_matrix = np.mat(X)
    Y_matrix = np.mat(Y)
    m,n = np.shape(X_matrix)
    maxCycle = 500
    G = np.ones((n,1))
    W = np.ones((n,1))
    for k in range(maxCycle):
        g = sto_batch_grad(X_matrix,Y_matrix,W)
        G = gamma * G + (1 - gamma) * np.sqr(g)
        W = W - learn_rate * g / np.sqrt(next_G[i] + epsilon)
    return W


def AdaDelta(X,Y,gamma=0.95, epsilon=1e-6):
    X_matrix = np.mat(X)
    Y_matrix = np.mat(Y)
    m,n = np.shape(X_matrix)
    maxCycle = 500
    W = np.ones((n,1))
    G = np.ones((n,1))
    dw = np.ones((n,1))
    t = np.ones((n,1))
    for k in range(maxCycle):
        g = sto_batch_grad(X_matrix,Y_matrix,W)
        G = gamma * G + (1 - gamma) *np.sqrt(g)
        dw = np.sqrt(t+epsilon)/np.sqrt(G+epsilon)
        W = W + dw
        t = gamma * t + (1-gamma)*np.sqrt(dw)
    return W


def Adam(X,Y, learn_rate=0.002, gamma=0.999, beta=0.9,
epsilon=1e-8):
    X_matrix = np.mat(X)
    Y_matrix = np.mat(Y)
    m,n = np.shape(X_matrix)
    maxCycle = 500
    W = np.ones((n,1))
    m = np.ones((n,1))
    t=1
    for k in range(maxCycle):
        g = sto_batch_grad(X_matrix,Y_matrix,W)
        m = beta * m + (1 - beta) * g
        G = gamma * G + (1 - gamma) * np.sqrt(g)
        W = W - learn_rate * np.sqrt(1 - gamma ** t)/(1 - beta **
t) * m / np.sqrt(G + epsilon)
    return W

def classify(X,W):
    prob = sigmoid(sum(X*W))
    if prob > 0.5 : return 1.0
    else : return 0.0

if __name__ =="__main__":
    X_train,Y_train = data_loader("a9a")
    X_test,Y_test = data_loader("a9a.t")
    X = array(X_train)
```

7.2 Linear Classification and Stochastic Gradient Descent

```
    from sklearn.datasets import load_svmlight_file
    import numpy as np
    from sklearn.model_selection import train_test_split
    from scipy.sparse import csr_matrix, hstack
    import matplotlib.pyplot as plt
    %matplotlib inline

    def data_loader(train_file):
        X,Y = load_svmlight_file(train_file)
        X = X.toarray()
        X = np.c_[np.ones((X.shape[0], 1)),X]
        return X, Y

    def train_test(X, Y, theta):
        #Y_prediction = X.dot(theta)
        a = np.sum(theta * X, axis = 1)
        b = Y * a
        c = 0
        for i in range(X.shape[0]):
            c = c + max(0, 1 - b[i])
        loss = np.linalg.norm(theta)**2 / 2 + c
        return loss

    def NAG(X,Y,learn_rate=.05, gamma=.9):
        X_matrix = np.mat(X)
        Y_matrix = np.mat(Y)
        m,n = np.shape(X_matrix)
        maxCycle = 500
        W = np.ones((n,1))
        v = np.ones((n,1))
```

```
    for k in range(maxCycle):
        G = sto_batch_grad(X_matrix,Y_matrix,W-gamma*v)
        v = gamma * v + learn_rate * G
        W = W - v
    return W


def RMSProp(X,Y,learn_rate=0.001, gamma=0.9,
epsilon=1e-8):
    X_matrix = np.mat(X)
    Y_matrix = np.mat(Y)
    m,n = np.shape(X_matrix)
    maxCycle = 500
    G = np.ones((n,1))
    W = np.ones((n,1))
    for k in range(maxCycle):
        g = sto_batch_grad(X_matrix,Y_matrix,W)
        G = gamma * G + (1 - gamma) * np.sqr(g)
        W = W - learn_rate * g / np.sqrt(next_G[i] + epsilon)
    return W


def AdaDelta(X,Y,gamma=0.95, epsilon=1e-6):
    X_matrix = np.mat(X)
    Y_matrix = np.mat(Y)
    m,n = np.shape(X_matrix)
    maxCycle = 500
    W = np.ones((n,1))
    G = np.ones((n,1))
    dw = np.ones((n,1))
    t = np.ones((n,1))
    for k in range(maxCycle):
        g = sto_batch_grad(X_matrix,Y_matrix,W)
        G = gamma * G + (1 - gamma) *np.sqrt(g)
        dw = np.sqrt(t+epsilon)/np.sqrt(G+epsilon)
        W = W + dw
        t = gamma * t + (1-gamma)*np.sqrt(dw)
    return W


def Adam(X,Y, learn_rate=0.002, gamma=0.999, beta=0.9,
epsilon=1e-8):
    X_matrix = np.mat(X)
    Y_matrix = np.mat(Y)
    m,n = np.shape(X_matrix)
    maxCycle = 500
    W = np.ones((n,1))
    m = np.ones((n,1))
    t=1
    for k in range(maxCycle):
        g = sto_batch_grad(X_matrix,Y_matrix,W)
        m = beta * m + (1 - beta) * g
        G = gamma * G + (1 - gamma) * np.sqrt(g)
        W = W - learn_rate * np.sqrt(1 - gamma ** t)/(1 - beta
** t) * m / np.sqrt(G + epsilon)
    return W


if __name__ =="__main__":
    X_train,Y_train = data_loader("a9a")
    X_test,Y_test = data_loader("a9a.t")
    theta = np.zeros((1,X_train.shape[1]))
    L_train = np.zeros((150))
    L_validation = np.zeros((150))
    prediction_rate = np.zeros((150))
    learning_rate = 0.000150
    threshold = -1
    for t in range(150):
        #Y_prediction = X.dot(theta)
        G = theta
        e = np.sum(theta * X_train, axis = 1)
        f = 1 - Y_train * e
        for j in range(0, 482):
            if f[j] >= 0:
                G = G - X_train[j] * Y_train[j]
            else:
                G = G

        theta -= learning_rate * G

        pre = np.sum(theta * X_train, axis = 1)

        count = 0
        for j in range(690):
            if pre[j] > threshold:
                pre[j] = 1
            elif pre[j] < threshold:
                pre[j] = -1

            if pre[j] == Y_test[j]:
                count += 1

        prediction_rate[t] = count / 16281
        L_train[t]=train_test(X_train, Y_train, theta)/(32561)
        L_validation[t]=train_test(X_train, Y_train,
theta)/(16281)

    #print('L_train:\n',L_train)
    #print('L_validation:\n',L_validation)
    plt.plot(L_train,label='train')
    plt.plot(L_validation,label='test')
    plt.ylabel('loss value')
    plt.xlabel('literation number')
    plt.legend()
```

8. The initialization method of model parameters:

9. The selected loss function and its derivatives:

NAG:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1} - \gamma \mathbf{v}_{t-1})$$
$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

RMSProp:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

AdaDelta:

$$
\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\Delta\boldsymbol{\theta}_t &\leftarrow -\frac{\sqrt{\Delta_{t-1}+\epsilon}}{\sqrt{G_t+\epsilon}} \odot \mathbf{g}_t \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t \\
\Delta_t &\leftarrow \gamma\Delta_{t-1} + (1-\gamma)\Delta\boldsymbol{\theta}_t \odot \Delta\boldsymbol{\theta}_t
\end{aligned}
$$

Adam:

$$
\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
\mathbf{m}_t &\leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\alpha &\leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t} \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t+\epsilon}}
\end{aligned}
$$

10. Results analysis: Four optimization algorithms did not succeed,Thus I did not get the correct results.

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

Generally speaking, linear regression is a real regression, and logical regression is a classifier, not a true regression. Is this a conventionally mistaken name? The main body of the NO. Logistic regression is regression: the regression is a sigmoid function , It maps the input to a decimal between 0 and 1. After getting this fraction between 0 and 1, it is interpreted as probability by human and then classified according to the threshold set in advance. Logistic regression is commonly used in dichotomous models and the objective function is the second type.

12. Summary:

Experiment did not succeed, I am ashamed. Mainly due to the four optimization algorithms I do not understand, and when I started doing it, there were such problems. Of course, mainly because I did not put too much energy on the experiment, I will certainly correct it later. I must not delay it. The tasks to be done are done as soon as possible. I can not do the thrid experiment without completing the second experiment .