华南理工大学

South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
HuiHan

Supervisor:
Qingyao Wu

Student ID:
201721045572

Grade:
Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

***Abstract*—Stochastic gradient descent and gradient descent are different methods. In this experiment, Realize two kinds of different classification methods of logistic regression and linear classification, the method of Stochastic gradient descent is used to optimize the logistic regression and linear classification, optimizing the stochastic gradient descent by different algorithms.**

I. Introduction

LOGISTIC regression is a widely used classification machine learning algorithm that fits data to a logit function (or logistic function) to predict the probability of an event occurring. When training a large amount of data, it takes a lot of time to find the gradient using all the samples. The idea of stochastic gradient is to randomly extract part of the data for training in each iteration. This greatly speeds up the training efficiency. In this experiment, we use the method of stochastic gradient to optimize logistic regression and linear classification. Four different optimization algorithms have been implemented: NAG, RMSProp, AdaDelta, Adam.The purpose of the experiment includes the following points:(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.

II. Methods and Theory

A. Logistic Regression

First we should determine the model function:

$$h_W(x) = \frac{1}{1 + e^{-W^T x}} \tag{1}$$

Then we can deduce the loss function:

$$J(W) = -\frac{1}{n}\Big[\sum_{i=1}^{n} y_i \log h_W(x_i) + (1 - y_i)\log(1 - h_W(x_i))\Big] \tag{2}$$

Calculate the derivatives of parameters W:

$$G_W = \frac{1}{n}\sum_{i=1}^{n}(h_W(x_i) - y_i)x_i \tag{3}$$

In the Stochastic Gradient Descent section we will give different parameter update algorithms.

B. Linear Classification

First we should determine the model function:

$$f(x) = W^T X + b \tag{4}$$

Then we can deduce the loss function:

$$L = \frac{||W||^2}{2} + \frac{C}{n}\sum_{i=1}^{n} max(0, 1 - y_i(W^T x_i + b)) \tag{5}$$

Calculate the derivatives of parameters W and b:

$$G_W = \begin{cases} W + \dfrac{C}{n}\displaystyle\sum_{i=1}^{n}(-y_i x_i) & 1 - y_i(W_T x_i + b) > 0 \\ W & 1 - y_i(W_T x_i + b) < 0 \end{cases} \tag{6}$$

$$G_b = \begin{cases} \dfrac{C}{n}\displaystyle\sum_{i=1}^{n}(-y_i) & 1 - y_i(W_T x_i + b) > 0 \\ 0 & 1 - y_i(W_T x_i + b) < 0 \end{cases} \tag{7}$$

C. Stochastic gradient descent

Here we introduce four different improved algorithms for stochastic gradient descent

1) NAG: The core idea of the Nesterov accelerated gradient is to use Momentum to predict the next gradient rather than using the current θ. The parameter update formula are as follows:

$$g_t \leftarrow \nabla J(\theta_{t-1} - \gamma v_{t-1}) \tag{8}$$

$$v_t \leftarrow \gamma v_{t-1} + \eta g_t \tag{9}$$

$$\theta_t \leftarrow \theta_{t-1} - v_t \tag{10}$$

2) RMSProp: RMSProp is to solve the problem of AdaGrad learning rate tends to 0. The parameter update formula are as follows:

$$g_t \leftarrow \nabla J(\theta_{t-1}) \tag{11}$$

$$G_t \leftarrow \gamma G_t + (1 - r) g_t \odot g_t \tag{12}$$

$$\theta_t \leftarrow \theta_{t-1} - \frac{\eta}{\sqrt{G_t + e}} \odot g_t \tag{13}$$

The initial value of parameter γ should be set to 0.9 and the initial value of parameter η should be set to 0.001.

3) AdaDelta: AdaDelta is often seen as similar to RMSProp, feeling AdaDelta a higher order point since it does not even have to be set for initial learning rates, but AdaDelta is sometimes relatively slow. The parameter update formula are as follows:

$$g_t \leftarrow \nabla J(\theta_{t-1}) \tag{14}$$

$$G_t \leftarrow \gamma G_t + (1-r) g_t \odot g_t \tag{15}$$

$$\triangle\theta_t \leftarrow -\frac{\sqrt{\triangle_{t-1}+e}}{\sqrt{G_T+e}} \odot g_t \tag{16}$$

$$\theta_t \leftarrow \theta_{t-1} + \triangle\theta_t \tag{17}$$

$$\triangle t \leftarrow \gamma\triangle_{t-1} + (1-\gamma)\triangle\theta_t \odot \triangle\theta_t \tag{18}$$

The initial value of parameter γ should be set to 0.95 and the initial value of other parameters should be set to zero.

4) Adam: Adam takes advantage of AdaGrad and RMSProp on sparse data. The correction to the bias of initialization also makes Adam perform better. It is adaptive estimates of lower-order moments, which adaptively adjusts the 1st order (mean) and 2nd order moments (variance). The parameter update formula are as follows:

$$g_t \leftarrow \nabla J(\theta_{t-1}) \tag{19}$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{20}$$

$$G_t \leftarrow \gamma G_t + (1-r) g_t \odot g_t \tag{21}$$

$$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t} \tag{22}$$

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{m_t}{\sqrt{G_t+e}} \tag{23}$$

The initial value of x is 0.9, may need to be attenuated, the initial value of y is 0.999, the initial value of z is 0.001, and sometimes it needs to be attenuated. The initial value of other parameters should be set to zero.

III. Experiments

A. Dataset

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features.The experimental environment is a direct installation of anaconda3, which avoids the installation of many dependent packages.Each iteration of the training set to a random part of the data gradient, and then use all test sets to calculate the loss function.

B. Implementation

1) Logistic Regression and Stochastic Gradient Descent: The experimental steps are as follows:

(1).Use load_svmligh_file function in sklearn library to load the experiment data,including training set and validation set.

(2).Initalize logistic regression model parameters,Set all parameter into zore.

(3).Select the loss function and calculate its derivation.

(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 lossL_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}.

(7).Repeate step 4 to 6 for several times, and drawing graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} with the number of iterations.

Experimental implementation of the key code is as follows:

```
def selectData(X,Y,size):
    data = []
    per = np.random.permutation(X.shape[1])
    shuffled_X = X[:,per]
    shuffled_Y = Y[:,per]
    num_frag = math.floor(X.shape[1]/size)
    for i in range(num_frag):
        data_X = shuffled_X[:,i*size:(i+1)*size]
        data_Y = shuffled_Y[:,i*size:(i+1)*size]
        a = (data_X, data_Y)
        data.append(a)
    if X.shape[1] % size != 0:
        data_X =
            shuffled_X[:,num_frag*size:X.shape[1]]
        data_Y =
            shuffled_Y[:,num_frag*size:X.shape[1]]
        a = (data_X, data_Y)
        data.append(a)
    return data
def cost(W,X,Y):
    Hx = 1/(1+np.exp(-np.dot(W.T,X)))
    cost =
        (np.dot(Y,np.log(Hx).T)+np.dot((1-Y),np.log(1-Hx
    cost = np.squeeze(cost)
    return cost
def loss(W,X,Y):
    Hx = 1/(1+np.exp(-np.dot(W.T,X)))
    GW = np.dot(X,(Hx-Y).T)/X.shape[1]
    return GW
def main(X_train, y_train, X_test, y_test,
     method):
    lenrate = 0.001
    n = 100
    validCost = []
    W = np.zeros((X_train.shape[0],1))
    vW = 0
    vb = 0
    r = 0.9
    decay_rate = 0.9
    cache = 0
    eps = 1e-8
    m = 0
    v = 0
```

```
    beta1 = 0.9
    beta2 = 0.999
    rA = 0.95
    diffdireW = 0
    direW = 0
    gW = 0
    for i in range(n):
        datas = selectData(X_train, y_train,
            1000)
        for data in datas:
            (X_sel, Y_sel) = data
            GW = loss(W,X_sel,Y_sel)
            if method is "NAG":
                v_prev = vW
                vW = r*vW-lenrate*GW
                W += -r*v_prev+(1+r)*vW
            elif method is "RMSProp":
                cache = decay_rate * cache + (1 -
                    decay_rate) * GW**2
                W += - lenrate * GW /
                    (np.sqrt(cache) + eps)
            elif method is "Adam":
                t = i + 1
                m = beta1*m + (1-beta1)*GW
                mt = m / (1-beta1**t)
                v = beta2*v + (1-beta2)*(GW**2)
                vt = v / (1-beta2**t)
                W += - lenrate * mt /
                    (np.sqrt(vt) + eps)
            elif method is "AdaDelta":
                gW = rA*gW+(1-rA)*GW**2
                diffdireW =
                    -np.sqrt(direW+eps)/np.sqrt(gW+eps)*GW
                W += diffdireW
                direW =
                    rA*direW+(1-rA)*diffdireW**2
        validcost = cost(W,X_test,y_test)
        validCost.append(validcost)
    return validCost
```

The Logistic Regression and Stochastic Gradient Descent experimental results,which draws graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} with the number of iterations,are as follows Fig. 1.

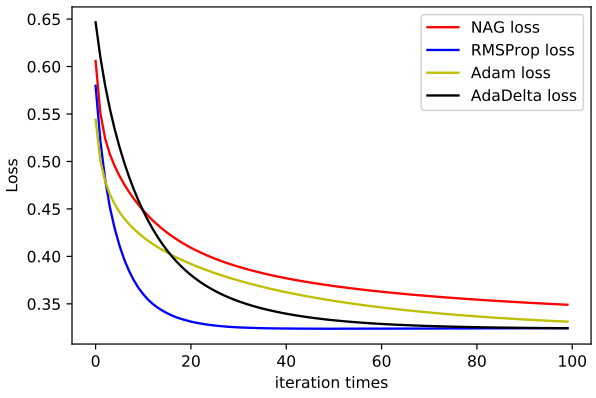

Fig. 1. Loss function curve in Logistic Regression and Stochastic Gradient Descent experiments

Logistic Regression and Stochastic Gradient Descent experimental results show that as the number of iterations increases, The four kinds of optimization algorithms Loss function with different trajectories to reduce, eventually tend to be stable. The experimental results are consistent with the predicted results, so the experiment was successful.

2) Linear Classification and Stochastic Gradient Descent: The experimental steps are as follows:

(1).Use load_svmligh_file function in sklearn library to load the experiment data,including training set and validation set.

(2).2.Initalize SVM model parameters, Set all parameter into zore.

(3).Select the loss function and calculate its derivation.

(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 lossL_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam}.

(7).Repeate step 4 to 6 for several times, and drawing graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} with the number of iterations.

Experimental implementation of the key code is as follows:

```
def cost(W,b,X,Y):
    C = 5
    cost = np.sum(np.square(W))/2 +
        C*np.sum(np.maximum(0,1 -
        Y*(np.dot(W.T,X)+b))) / X.shape[1]
    return cost
def loss(W,b,X,Y):
    C = 5
    GW = np.zeros(X.shape[1])
    Gb = 0
    filt = (1-Y*(np.dot(W.T,X)+b))>0
    GW = W - C*np.dot(Y*filt,X.T).T/X.shape[1]
    Gb = -C*np.sum(Y*filt*b)/X.shape[1]
    return GW,Gb
def main(X_train, y_train, X_test, y_test,
     method ,num):
    lenrate = 0.001
    n = 50
    validCost = []
    W = np.zeros((X_train.shape[0],1))
    b = 0
    vW = 0
    vb = 0
    r = 0.9
    decay_rate = 0.99
    cacheW = 0
    cacheb = 0
    eps = 1e-8
    mW = 0
    mb = 0
    beta1 = 0.9
    beta2 = 0.999
    rA = 0.95
    diffdireW = 0
    direW = 0
    diffdireb = 0
    direb = 0
    gW = 0
    gb = 0
```

```
for i in range(n):
    datas = selectData(X_train, y_train,
        num)
    for data in datas:
        (X_sel, Y_sel) = data
        GW,Gb = loss(W,b,X_sel,Y_sel)
        if method is "NAG":
            v_prevW = vW
            vW = r*vW-lenrate*GW
            W += -r*v_prevW+(1+r)*vW
            v_prevb = vb
            vb = r*vb-lenrate*Gb
            b += -r*v_prevb+(1+r)*vb
        elif method is "RMSProp":
            cacheW = decay_rate * cacheW + (1
                - decay_rate) * GW**2
            W += - lenrate * GW /
                (np.sqrt(cacheW) + eps)
            cacheb = decay_rate * cacheb + (1
                - decay_rate) * Gb**2
            b += - lenrate * Gb /
                (np.sqrt(cacheb) + eps)
        elif method is "Adam":
            t = i + 1
            mW = beta1*mW + (1-beta1)*GW
            mtW = mW / (1-beta1**t)
            vW = beta2*vW + (1-beta2)*(GW**2)
            vtW = vW / (1-beta2**t)
            W += - lenrate * mtW /
                (np.sqrt(vtW) + eps)
            mb = beta1*mb + (1-beta1)*Gb
            mtb = mb / (1-beta1**t)
            vb = beta2*vb + (1-beta2)*(Gb**2)
            vtb = vb / (1-beta2**t)
            b += - lenrate * mtb /
                (np.sqrt(vtb) + eps)
        elif method is "AdaDelta":
            gW = rA*gW+(1-rA)*GW**2
            diffdireW = -np.sqrt(direW+eps) /
                np.sqrt(gW+eps)*GW
            W += diffdireW
            direW =
                rA*direW+(1-rA)*diffdireW**2
            gb = rA*gb+(1-rA)*Gb**2
            diffdireb = -np.sqrt(direb+eps) /
                np.sqrt(gb+eps)*Gb
            b += diffdireb
            direb =
                rA*direb+(1-rA)*diffdireb**2
    validcost = cost(W,b,X_test,y_test)
    validCost.append(validcost)
return validCost
```

Linear Classification and Stochastic Gradient Descent experimental results,which draws graph of L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} with the number of iterations,are as follows Fig. 2.

Linear Classification and Stochastic Gradient Descent results show that as the number of iterations increases, The four kinds of optimization algorithms Loss function with different trajectories to reduce. However, the final result of RMSProp and AdaDelta is not completely stable. The experimental results can be improved by optimizing the code and adjusting the parameters.

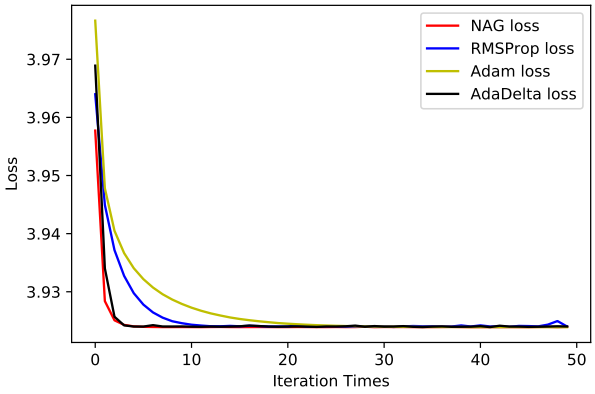

Fig. 2. Loss function curve in Linear Classification and Stochastic Gradient Descent experiments

IV. Conclusion

I learned a lot from this experiment. First, I got a deeper understanding of logistic regression, linear classification and stochastic gradient descent. Second, I learned how to use a fixed dataset to train a function model, adjust the parameters during the machine learning experiment. Finally, I learned four different optimized stochastic gradient descent algorithms, mastered the formula derivation of these algorithms, and converted the formulas into code based on references.