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:
Jining He

Supervisor:
Qingyao Wu

Student ID:
201721045565

Grade:
Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

***Abstract*—In this experiment, we use stochastic gradient descent and other optimization algorithms to optimize logistic regression and linear classification on larger dataset. Adam optimization algorithm has the best performance.**

I. Introduction

THIS experiment require us to predict adult whether income exceeds $50K/yr. We will use both logistic regression and linear classification. We also will use four optimization algorithms.

The Motivation of this experiment is let us compare and understand the differences between gradient descent and stochastic gradient descent, compare and understand the differences and relationships between logistic regression and linear classification, further understand the principles of SVM and practice on larger data.

II. Methods and Theory

In this experiment, we use logistic regression and linear classifier to predict adult whether income exceeds $50K/yr.

A. Logistic Regression

In logistic regression, the hypothesis function is:

$$h_{w,b}(x) = g(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}} \tag{1}$$

and the loss function is:

$$J(W,b) = -\frac{1}{m}\sum_{i=1}^{m}(y_i \log h_{W,b}(x_i) + (1-y_i)\log(1 - h_{W,b}(x_i))) \tag{2}$$

and the gradient formulas are:

$$\begin{aligned} \nabla_W J(W,b) &= \frac{1}{m}\sum_{i=1}^{m}(h_{W,b}(x_i) - y_i)x_i \\ \nabla_b J(W,b) &= \frac{1}{m}\sum_{i=1}^{m}(h_{W,b}(x_i) - y_i) \end{aligned} \tag{3}$$

B. Linear Classification

In linear classification, we add hinge loss to our loss function:

$$J(W,b) = \frac{||W||^2}{2} + \frac{C}{m}\sum_{i=1}^{m} max(0, 1 - y_i(W^T x_i + b)) \tag{4}$$

and the gradient formulas are:

$$\begin{aligned} g_w(x_i) &= \begin{cases} -y_i x_i & 1 - y_i(w^T x_i + b) >= 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases} \\ g_b(x_i) &= \begin{cases} -y_i & 1 - y_i(w^T x_i + b) >= 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases} \\ \nabla_W J(W,b) &= W + \frac{C}{m}\sum_{i=1}^{m} g_w(x_i) \\ \nabla_b J(W,b) &= \frac{C}{m}\sum_{i=1}^{m} g_b(x_i) \end{aligned} \tag{5}$$

C. Gradient Descent Optimization Algorithms

In the last experiment, we used batch gradient descent to optimize model. In fact, there are many gradient descent optimization algorithms. In this experiment, we use some of them.

1) NAG: Below is the NAG update parameters formula, and momentum term γ is around 0.9.

$$\begin{aligned} g_t &= \nabla_\theta J(\theta_{t-1} - \gamma v_{t-1}) \\ v_t &= \gamma v_{t-1} + \alpha g_t \\ \theta_t &= \theta_{t-1} - v_t \end{aligned} \tag{6}$$

2) RMSProp: The term ϵ is usually 10^{-8}.

$$\begin{aligned} g_t &= \nabla J(\theta_{t-1}) \\ G_t &= \gamma G_t + (1-\gamma) g_t^2 \\ \theta_t &= \theta_{t-1} - \frac{\alpha}{\sqrt{G_t} + \epsilon} g_t \end{aligned} \tag{7}$$

3) AdaDelta:

$$\begin{aligned} g_t &= \nabla J(\theta_{t-1}) \\ G_t &= \gamma G_t + (1-\gamma) g_t^2 \\ \theta_t &= \theta_{t-1} - \frac{\alpha}{\sqrt{G_t} + \epsilon} g_t \end{aligned} \tag{8}$$

4) Adam: The term β_1 is usually 0.9 and the term β_2 is 0.999.

$$\begin{aligned} g_t &= \nabla J(\theta_{t-1}) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t \\ \theta_t &= \theta_{t-1} - \frac{\alpha}{\sqrt{G_t} + \epsilon} m_t \end{aligned} \tag{9}$$

III. Experiments

A. Dataset

The datasets are from LIBSVM. This experiment uses a9a dataset, including 32561/16281(testing) samples and each sample has 123 features.

B. Implementation

1) Data Preprocessing: Use load_svmlight_file function in sklearn library to load train set and validation set. Then convert data to column vector.

2) Initialization: There are many ways to initialize linear model parameters. I choose to set all parameters into zero.

3) Compute Loss and Gradient: In logistic regression, use formula (2) to compute loss and formula (3) to compute gradient. In linear classification, use formula (4) to compute loss and formula (5) to compute gradient.

4) Update Parameters: The parameters are updated according to the optimization algorithm used.

5) Repeat: Repeat step 3 and 4 until reach the end condition. Get trian loss and validation loss in every iteration.

6) Implementation Code: I use a function to random and split dataset to many mini-batches:

```
def random_batches(X,Y,batch_size):
    m = X.shape[1]
    batches = []
    permutation = np.random.permutation(m)
    shuffled_X = X[:,permutation]
    shuffled_Y = Y[:,permutation]
    num_batches = math.floor(m/batch_size)
    for i in range(num_batches):
        batch_X = shuffled_X[:,
            i*batch_size:(i+1)*batch_size]
        batch_Y = shuffled_Y[:,
            i*batch_size:(i+1)*batch_size]
        batch = (batch_X, batch_Y)
        batches.append(batch)
    if m % batch_size != 0:
        batch_X = shuffled_X[:,
            num_batches*batch_size:m]
        batch_Y = shuffled_Y[:,
            num_batches*batch_size:m]
        batch = (batch_X, batch_Y)
        batches.append(batch)
    return batches
```

In logistic regression, I use this function to compute cost and gradient:

```
def propagate(W,b,X,Y):
    m = X.shape[1]
    A = sigmoid(np.dot(W.T,X)+b)
    cost = (np.dot(Y,np.log(A).T) +
        np.dot((1-Y),np.log(1-A).T))/(-m)
    gW = np.dot(X,(A-Y).T)/m
    gb = np.sum(A-Y)/m
    cost = np.squeeze(cost)
    return cost,gW,gb
```

And in linear classification, I use this function:

```
def propagate(C,W,b,X,Y):
    m = X.shape[1]
    A = np.dot(W.T,X)+b
    cost = np.sum(np.square(W))/2 +
        C*np.sum(np.maximum(0,
        1-Y*(np.dot(W.T,X)+b)))/m
    filt = (1-Y*(np.dot(W.T,X)+b))>0
    gW = W - C*np.dot(Y*filt,X.T).T/m
    gb = -C*np.sum(Y*filt*b)/m
    return cost,gW,gb
```

The following is the formula for each optimization algorithm to update the parameters. NAG:

```
vgW_prev = vgW
vgW = beta * vgW - learning_rate * gW
W = W - beta * vgW_prev + (1+beta) * vgW
vgb_prev = vgb
vgb = beta * vgb - learning_rate * gb
b = b - beta * vgb_prev + (1+beta) * vgb
```

RMSProp:

```
W_cache = decay_rate * W_cache + (1 -
    decay_rate) * gW**2
W = W - learning_rate * gW /
    (np.sqrt(W_cache) + eps)
b_cache = decay_rate * b_cache + (1 -
    decay_rate) * gb**2
b = b - learning_rate * gb /
    (np.sqrt(b_cache) + eps)
```

AdaDelta:

```
W_cache = beta * W_cache + (1-beta) * gW**2
delta_W = - np.sqrt(delta_with_W +
    eps)/(np.sqrt(W_cache)+eps) * gW
W = W + delta_W
delta_with_W = beta * delta_with_W +
    (1-beta)* delta_W ** 2
b_cache = beta * b_cache + (1-beta) * gb**2
delta_b = - np.sqrt(delta_with_b +
    eps)/(np.sqrt(b_cache)+eps) * gb
b = b + delta_b
delta_with_b = beta * delta_with_b +
    (1-beta)* delta_b ** 2
```

Adam:

```
vgW = beta1*vgW + (1-beta1)*gW
sgW = beta2*sgW + (1-beta2)*(gW**2)
W = W - learning_rate * vgW /
    (np.sqrt(sgW) + eps)
vgb = beta1*vgb + (1-beta1)*gb
sgb = beta2*sgb + (1-beta2)*(gb**2)
b = b - learning_rate * vgb /
    (np.sqrt(sgb) + eps)
```

7) Result: I plot both logistic regression and linear classification performance. Fig 1 and Fig 3 show the performance of these optimization algorithms. From the convergence speed, all four optimization algorithms are better than MSGD, and Adam is the best.

Compare Fig 1 and Fig 3, we can find that the performance is different with different model. In logistic regression, RMSProp is better than NAG. However, in linear classification, NAG is better than RMSprop. Moreover, we can also find that the performance of linear classification is better than logistic regression.

Fig 2 is the classification accuracy rate of logistic regression varies with the number of iterations graph. We also can find that as the number of iterations increases, the classification accuracy of each optimization algorithm continues to increase. And, Adam is the best performing optimization algorithm.

I also plot both logistic regression and linear classification with different batch sizes. From Fig 4, Fig 5, Fig 6 and Fig 7

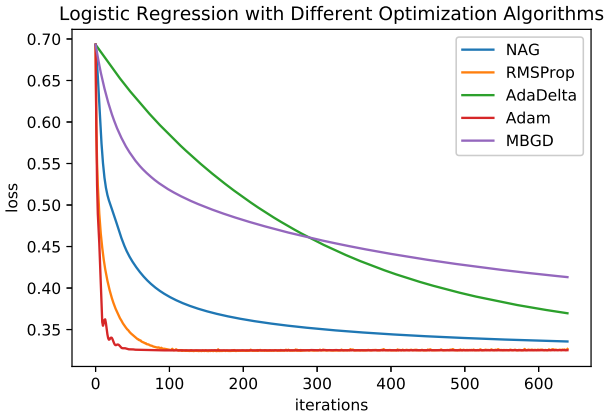

Fig. 1. The curves of the logistic regression loss function with different optimization algorithms. $Batch\_size = 4096$ and $\alpha = 0.01$.

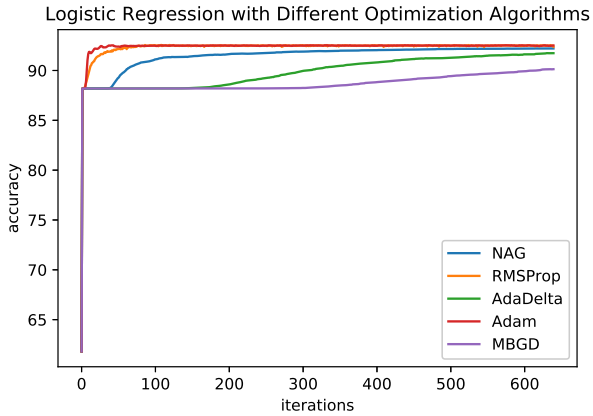

Fig. 2. The curves of the logistic regression accuracy rate with different optimization algorithms. $Batch\_size = 4096$ and $\alpha = 0.01$.

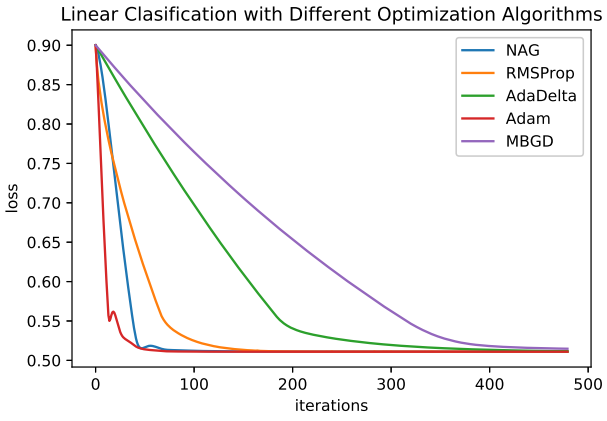

Fig. 3. The curves of the linear classification loss function with different optimization algorithms. $Batch\_size = 4096$ and $\alpha = 0.001$.

we can find that the smaller batch, the greater the loss function shock. And too small batch sizes may not fit well. So when we train a machine learning model, we need to select an appropriate batch size.

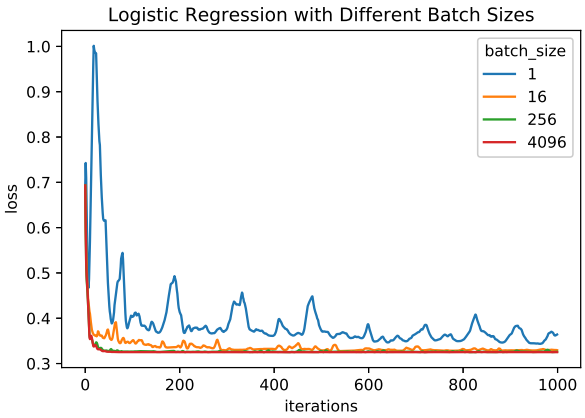

Fig. 4. The curves of the logistic regression loss function with MBGD with different batch sizes. $\alpha = 0.01$

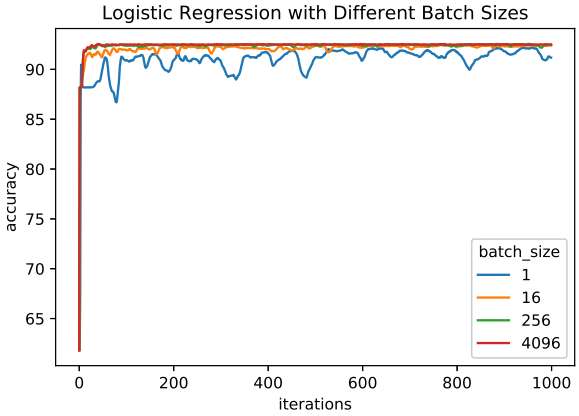

Fig. 5. The curves of the logistic regression accuracy rate with MBGD with different batch sizes. $\alpha = 0.01$

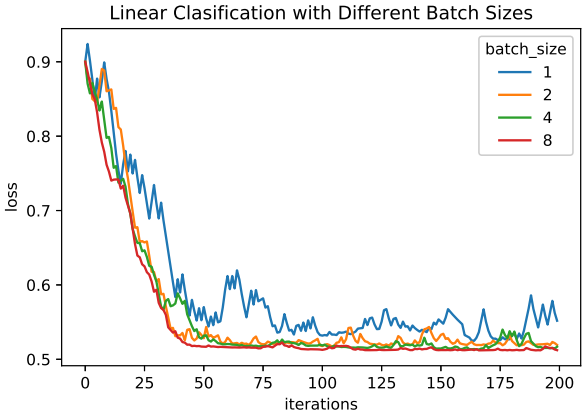

Fig. 6. The curves of the linear classification loss function with MBGD with different batch sizes. $\alpha = 0.01$

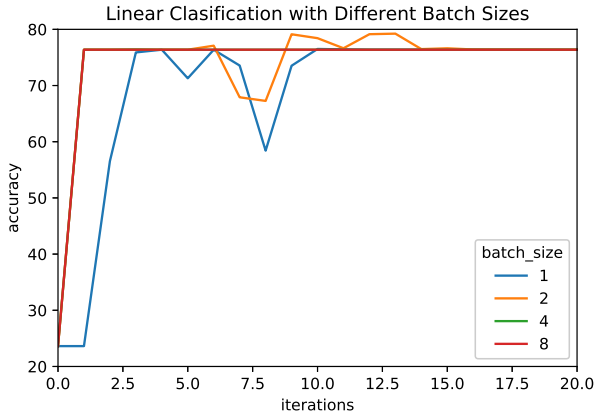

Fig. 7. The curves of the linear classification accuracy rate with MBGD with different batch sizes. $\alpha = 0.01$

IV. Conclusion

Through this experiment, I understood the difference between gradient descent and stochastic gradient descent. Stochastic gradient descent uses only a small size of data for training, which accelerates the training process. However, if the data size of each batch is too small, there will be a big shock, or even can not converge. I also found that SVM as a classifier perform better than logistic regression. At the same time, I implemented four gradient descent optimization algorithms. I found that an optimization algorithm may perform differently on different models. And Adam performed well on both models. These processes make me understand gradient descent better.