The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Zhijie Liu

Supervisor:
Qingyao Wu

Student ID：
201530612392

Grade:
Undergraduate

December 12, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract—

I. INTRODUCTION

Today,I try to solve two questions which named logistic regression and linear classification.This two question are all to solve classification questions.
There are the terminals of this two questions:
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.
The lab use a9a data-set witch contains 32561 training data and 16281 testing data.Every data has 123 feature.
This lab is based on python3.

II. METHODS AND THEORY

Logistic Regression
Loss function:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} log\, h_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right) log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

Gradient

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y\right)x^{(i)}$$

Linear Classification:
Loss function

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

Gradient:

$$\nabla f(\beta) = \begin{cases} \mathbf{w}^\top - C\mathbf{y}^\top \mathbf{X} & 1-y_i(\mathbf{w}^\top \mathbf{x}_i + b) >= 0 \\ \mathbf{w}^\top & 1-y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 0 \end{cases}$$

III. EXPERIMENT

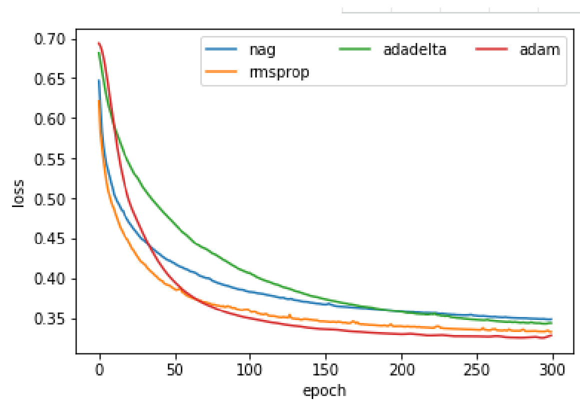
Steps
Logistic Regression:
1.Read experimental training set and verification set.
2.Logistic regression model parameter initialization, consider all-zero initialization, random initialization or normal distribution initialization.
3.Select Loss function and its derivative, the process see courseware ppt.
4.Find the gradient of some samples to Loss function.
5.Use different optimization methods to update model parameters (NAG, RMSProp, AdaDelta, and Adam).
6.Select the appropriate threshold, will verify the centralized calculation results greater than the threshold marked as positive, otherwise negative. Test on the validation set and get the Loss function values for different optimization methods
Repeat steps 4-6 several times and plot the number of iterations.

Linear Classification:
1.Read experimental training set and verification set.
2.Support vector machine model parameter initialization, you can consider all zero initialization, random initialization or normal distribution initialization.
3.Select Loss function and its derivative, the process see courseware ppt.
4.Find the gradient of some samples to Loss function.
5.Use different optimization methods to update model parameters (NAG, RMSProp, AdaDelta, and Adam).
6.Select the appropriate threshold, will verify the centralized calculation results greater than the threshold marked as positive, otherwise negative. Test on the validation set and get the Loss function values for different optimization methods
Repeat steps 4-6 several times and plot the number of iterations.

Result:
Logistic Regression:

```
# training nag
for epoch in range(300):
    random.seed()
    i=randint(0,n-1-batch_NAG)
```

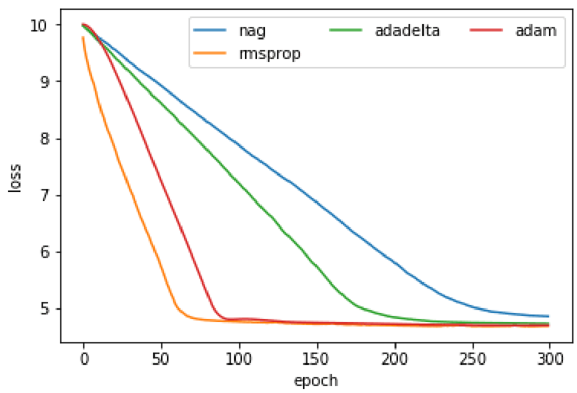
```
g=gradient(x_train[i:i+batch_NAG].reshape((batch_NAG,m)),
y_train[i:i+batch_NAG].reshape((batch_NAG,1)),W_NAG-ga
mma_NAG*v)
        v=gamma_NAG*v+eta_NAG*g
        W_NAG=W_NAG-v
        l_test=loss(x_test,y_test,W_NAG)
        L_NAG.append(l_test)
    print("L_NAG")
    print(L_NAG)

    # training rmsprop
    for epoch in range(300):
        random.seed()
        i=randint(0,n-1-batch_RMS)

g=gradient(x_train[i:i+batch_RMS].reshape((batch_RMS,m)),
y_train[i:i+batch_RMS].reshape((batch_RMS,1)),W_RMS)
        G=gamma_RMS*G+(1-gamma_RMS)*(g*g)

W_RMS=W_RMS-eta_RMS/np.sqrt(G+epsilon_RMS)*g
        l_test=loss(x_test,y_test,W_RMS)
        L_RMSProp.append(l_test)
    print("L_RMSProp")
    print(L_RMSProp)

    # training adadelta
    for epoch in range(300):
        random.seed()
        i=randint(0,n-1-batch_ADA)

g=gradient(x_train[i:i+batch_ADA].reshape((batch_ADA,m)),
y_train[i:i+batch_ADA].reshape((batch_ADA,1)),W_ADA)
        GG=gamma_ADA*GG+(1-gamma_ADA)*g*g

dw=-np.sqrt(dt+epsilon_ADA)/np.sqrt(GG+epsilon_ADA)*g
        W_ADA=W_ADA+dw
        dt=gamma_ADA*dt+(1-gamma_ADA)*dw*dw
        l_test=loss(x_test,y_test,W_ADA)
        L_AdaDelta.append(l_test)
    print("L_AdaDelta")
    print(L_AdaDelta)

    # training adam
    for epoch in range(300):
        i=randint(0,n-1-batch_ADAM)

g=gradient(x_train[i:i+batch_ADAM].reshape((batch_ADAM
,m)),y_train[i:i+batch_ADAM].reshape((batch_ADAM,1)),W
_ADAM)
        M=beta_ADAM*M+(1-beta_ADAM)*g
        G=gamma_ADAM*G+(1-gamma_ADAM)*g*g

alpha=eta_ADAM*np.sqrt(1-math.pow(gamma_ADAM,epoc
h))/(1-beta_ADAM)

W_ADAM=W_ADAM-alpha*M/np.sqrt(G+epsilon_ADAM)
        l_test=loss(x_test,y_test,W_ADAM)
        L_Adam.append(l_test)
    print("L_Adam")
    print(L_Adam)
```



Linear Classification:

```
    #NAG Training
    for epoch in range(300):
        random.seed()
        i=random.randint(0,n-1-batch_NAG)

g=gradient(x_train[i:i+batch_NAG].reshape((batch_NAG,m)),
y_train[i:i+batch_NAG].reshape((batch_NAG,1)),W_NAG-ga
mma_NAG*v,C)
        v=gamma_NAG*v+eta_NAG*g
        W_NAG=W_NAG-v
        l_test=loss(x_test,y_test,W_NAG,C)
        L_NAG.append(l_test)
    print("L_NAG")
    print(L_NAG)

    #RMS Training
    for epoch in range(300):
        random.seed()
        i=random.randint(0,n-1-batch_RMS)

g=gradient(x_train[i:i+batch_RMS].reshape((batch_RMS,m)),
y_train[i:i+batch_RMS].reshape((batch_RMS,1)),W_RMS,C)
        G=gamma_RMS*G+(1-gamma_RMS)*(g*g)

W_RMS=W_RMS-eta_RMS/np.sqrt(G+epsilon_RMS)*g
        l_test=loss(x_test,y_test,W_RMS,C)
        L_RMSProp.append(l_test)
    print("L_RMSProp")
    print(L_RMSProp)

    #Adadelta training
    for epoch in range(300):
        random.seed()
        i=random.randint(0,n-1-batch_ADA)

g=gradient(x_train[i:i+batch_ADA].reshape((batch_ADA,m)),
y_train[i:i+batch_ADA].reshape((batch_ADA,1)),W_ADA,C)
```

```
        G=gamma_ADA*G+(1-gamma_ADA)*g*g

dw=-np.sqrt(dt+epsilon_ADA)/np.sqrt(G+epsilon_ADA)*g
        W_ADA=W_ADA+dw
        dt=gamma_ADA*dt+(1-gamma_ADA)*dw*dw
        l_test=loss(x_test,y_test,W_ADA,C)
        L_AdaDelta.append(l_test)
    print("L_AdaDelta")
    print(L_AdaDelta)

    #adam training
    for epoch in range(300):
        i=random.randint(0,n-1-batch_ADAM)

g=gradient(x_train[i:i+batch_ADAM].reshape((batch_ADAM
,m)),y_train[i:i+batch_ADAM].reshape((batch_ADAM,1)),W
_ADAM,C)
        M=beta_ADAM*M+(1-beta_ADAM)*g
        G=gamma_ADAM*G+(1-gamma_ADAM)*g*g

alpha=eta_ADAM*np.sqrt(1-math.pow(gamma_ADAM,epoc
h))/(1-beta_ADAM)

W_ADAM=W_ADAM-alpha*M/np.sqrt(G+epsilon_ADAM)
        l_test=loss(x_test,y_test,W_ADAM,C)
        L_Adam.append(l_test)
    print("L_Adam")
    print(L_Adam)
```



IV. CONCLUSION

When using different optimization algorithms to optimize, the different optimization algorithms decline at different speeds. When optimizing logistic regression, adam and rmsprop may drop faster. When optimizing linear classification, the same reason may be that the learning rate may be Tonality, but if the learning rate is too high, the loss value may also reverse increase.

This experiment, compared to the first experiment more difficult to enhance the experimental formula is also more numerous, some parameters are difficult to understand at the beginning, leading to delay in getting started, through this experiment, I slowly learned In order to compare the advantages and disadvantages of different optimization algorithms for different problems, we can further study the logic regression and linear classification by adjusting the parameters of different algorithms and the convergence speed, reducing the time cost of algorithm operation.