South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
ZhiXiang Xue

Supervisor:
Qingyao Wu

Student ID:
201530613313

Grade:
Undergraduate

December 14, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract—This experiment focuses on Linear Regression, Linear Classification and Gradient Descent.

I. Introduction

We made this experiment in order to Compare and understand the difference between gradient descent and stochastic gradient descent, Compare and understand the differences and relationships between Logistic regression and linear classification and Further understand the principles of SVM and practice on larger data.

II. Methods and Theory

Experiment steps:

Logistic Regression and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initialize logistic regression model parameters, you can consider initializing 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 L_{NAG}, $L_{RMSProp}$, $L_{AdaDelta}$ and L_{Adam} .
7. Repeat 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.

Linear Classification and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initialize SVM model parameters, you can consider initializing 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 L_{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.

III. Experiments

A. Dataset

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features.

B. Implementation

Logistic Regression and Stochastic Gradient Descent

Loss：

$$L(\theta) = -\frac{1}{m}\sum_{i=0}^{n}\left(y_i\log\left(h_\theta(X_i)\right) + (1-y_i)\log\left(1-\log\left(h_\theta(X_i)\right)\right)\right)$$

Gradient：

$$\frac{\partial L}{\partial \theta} = \frac{1}{n}\sum_{i=0}^{n} X_i\left(h_\theta(X) - y_i\right)$$

The parameters of four methods we select are as follows.

method	**parameters**

NAG	Iter_num = 2000
	Learning_rate = 0.002
	gamma = 0.9

RMSProp	gamma = 0.9
	iter_num = 2000
	learning_rate = 0.1

AdaDelta	gamma = 0.95
	delta = 0.003
	iter_num = 2000

Adam	gamma = 0.9
	learning_rate = 0.1
	iter_num = 2000

Main code:

```python
def Loss(w,x,y):
    loss=0
    tmp=w.T*x
    loss+=-(tmp*y.T)[0,0]
    for i in range(tmp.shape[1]):
        loss+=np.log(1+np.exp(tmp[0,i]))
    return loss/float(y.shape[1])
```

```python
def SGD(w,x,y,learning_rate):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient+=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    w=w-learning_rate*gradient
    return w
```

```python
def SGD_NAG(w,m,x,y,learning_rate):
    momentum=m
    gradient=np.mat(np.zeros(w.shape))

    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient+=-x[:,i]*(y[0,i]-(np.exp((w-momentum).T*x[:,i])/(1+np.exp((w-momentum).T*x[:,i]))))
    momentum=0.9*momentum+learning_rate*gradient
    w=w-momentum
    return w,momentum
```

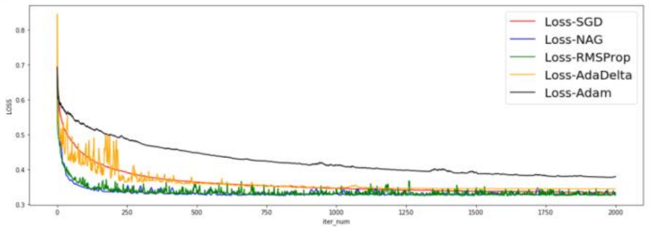
```python
def SGD_RMSProp(w,g,x,y,learning_rate):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient+=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    g=0.9*g+0.1*(gradient.T*gradient)[0,0]
    w=w-(learning_rate/(np.sqrt(g+1e-8)))*gradient
    return w,g
```

```python
def SGD_AdaDelta(w,g,delta,x,y):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    g=0.95*g+0.05*(gradient.T*gradient)[0,0]

    step_length=(np.sqrt(delta+1e-8)/np.sqrt(g+1e-8))*gradient
    w=w-step_length
    delta=0.95*delta+0.05*(step_length.T*step_length)[0,0]
    return w,g,delta
```

```python
def SGD_Adam(w,m,g,x,y,learning_rate,iter_num):
    gradient=np.mat(np.zeros(w.shape))
    for i in [np.random.randint(0,x.shape[1]) for i in range(20)]:
        gradient=-x[:,i]*(y[0,i]-(np.exp(w.T*x[:,i])/(1+np.exp(w.T*x[:,i]))))
    m=0.9*m+0.1*gradient
    g=0.999*g+0.001*gradient.T*gradient
    alpha=learning_rate*np.sqrt(1-0.999**iter_num)/(1-0.9**iter_num)
    w=w-alpha*m/np.sqrt(g+1e-8)
    return w,m,g
```

Result:



Linear Classification and Stochastic Gradient Descent

Loss:

$$L(\theta) = \frac{1}{2n}\sum_{i=0}^{n}(y_i - h_\theta(X_i))^2$$

Gradient:

$$\frac{\partial L}{\partial \theta} = \frac{1}{n}\sum_{i=0}^{n}(y_i - h_\theta(X_i)) \cdot X_i$$

The parameters of four methods we select are as follows.

method	parameters

NAG	Iter_num = 2000
	Learning_rate = 0.002
	gamma = 0.9

RMSProp	gamma = 0.9
	iter_num = 2000
	learning_rate = 0.1

AdaDelta	gamma = 0.95
	delta = 0.003
	iter_num = 2000

Adam	gamma = 0.9
	learning_rate = 0.1
	iter_num = 2000

Main code:

```python
def Loss(w,x,y):
    loss=0
    for i in range(x.shape[1]):
        if (1-y[0,i]*(w.T*x[:,i]))>0:
            loss+=1-y[0,i]*(w.T*x[:,i])
    losl=loss/float(y.shape[1])
    loss+=(w.T*w)/2.0
    return loss[0,0]
```

```python
def SGD(w_current,x,y,learning_rate):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    new_w=w_current-learning_rate*gradient
    return new_w
```

```python
def SGD_NAG(w_current,momentum,x,y,learning_rate):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current-momentum
    momentum=0.9*momentum+learning_rate*gradient

    new_w=w_current-momentum
    return new_w,momentum
```

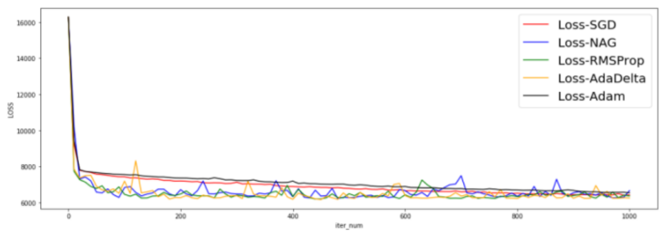
```python
def SGD_RMSProp(w_current,g,x,y,learning_rate):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    g=0.9*g+0.1*(gradient.T*gradient)[0,0]
    new_w=w_current-(learning_rate/(np.sqrt(g+1e-8)))*gradient
    return new_w,g
```

```python
def SGD_AdaDelta(w_current,g,delta,x,y):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    g=0.95*g+0.05*(gradient.T*gradient)[0,0]

    step_length=(np.sqrt(delta+1e-8)/np.sqrt(g+1e-8))*gradient
    new_w=w_current-step_length
    delta=0.95*delta+0.05*(step_length.T*step_length)[0,0]
    return new_w,g,delta
```

```python
def SGD_Adam(w_current,m,g,x,y,learning_rate,iter_num):
    gradient=np.mat(np.zeros(w_current.shape))
    for i in [np.random.randint(0,x.shape[1]) for j in range(20)]:
        if (1-y[0,i]*(w_current.T*x[:,i]))>0:
            gradient+=-y[0,i]*x[:,i]
    gradient+=w_current
    m=0.9*m+0.1*gradient
    g=0.999*g+0.001*gradient.T*gradient
    alpha=learning_rate*np.sqrt(1-0.999**iter_num)/(1-0.9**iter_num)
    new_w=w_current-alpha*m/np.sqrt(g+1e-8)
    return new_w,m,g
```

Result:



IV. Conclusion

In this experiment, I implemented logistic regression and linear classification on larger data, using SGD and four different optimization methods. During this process, I found that differences in result of using different optimization methods or just using SGD may be not large. It can't be said that four different optimization methods must be better than SGD, we should Analyze specific issues to choose the best method. But the speeds of convergence are certainly different.

Through the experiment, I compared and understand the differences and relationships between Logistic regression and linear classification and further understood the principles of SVM and practice on larger data. I feel that I have a deeper understanding of machine learning.