The Experiment Report of
Machine Learning

College Software College

Subject Software Engineering

Members 林英杰

Student ID 201530612286

E-mail 1142983747@qq.com

Tutor 谭明奎

Date submitted 2017.12.15

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

2. Time: 2017.12.9

3. Reporter: 林英杰

4. Purposes:

（1）Compare and understdan 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.

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:

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 LAdam with the number of iterations.

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:

Logistic Regression and Stochastic Gradient Descent

```
from sklearn.externals.joblib import Memory
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
mem = Memory("./mycache")

@mem.cache
def get_data(path):
	data = load_svmlight_file(path)
	return data[0], data[1]

import numpy as np
import matplotlib.pyplot as plt

X_train, y_train= get_data("G:\\学习资料\\大三上\\机器学习\\experiment\\experiment2\\a9a.txt")
X_test,y_test= get_data("G:\\学习资料\\大三上\\机器学习\\experiment\\experiment2\\a9a.t")
X_train = X_train.toarray()
X_test = X_test.toarray()
y_train[y_train==-1]=0
y_test[y_test==-1]=0
X_train=np.hstack((X_train,np.ones((X_train.shape[0],1))))#add a coloum of 1 after X_train
y_train = y_train.reshape((y_train.shape[0],1))
X_test=np.hstack((X_test,np.zeros((X_test.shape[0],1))))#add a coloum of 0 after X_test
X_test=np.hstack((X_test,np.ones((X_test.shape[0],1))))#add a coloum of 1 after X_test
y_test = y_test.reshape((y_test.shape[0],1))
one_test=np.ones((X_test.shape[0],1))
```

```
w=np.ones((X_train.shape[1],1)) #initialize the w
w1=np.ones((X_train.shape[1],1)) #initialize the w1
w2=np.ones((X_train.shape[1],1)) #initialize the w2
w3=np.ones((X_train.shape[1],1)) #initialize the w3
w4=np.ones((X_train.shape[1],1)) #initialize the w4
P=0
P1=0
P2=0
P3=0
P4=0
num=500 #times
loss_test_SGD=np.zeros((num)) #the loss of test set
loss_test_NAG=np.zeros((num))
loss_test_RMSProp=np.zeros((num))
loss_test_AdaDelta=np.zeros((num))
loss_test_Adam=np.zeros((num))

def h(x,W):
	return np.exp(x.dot(W))

def sample(x,y,n):
	index_sample=np.random.randint(0,x.shape[0]-1,n)
	x_sample=np.zeros((n,x.shape[1]))
	y_sample=np.zeros((n,y.shape[1]))
	index=0
	for i in index_sample:
		x_sample[index]=x[i]
		y_sample[index]=y[i]
		index=index+1
	return x_sample,y_sample

def right_rate(W):
	p=X_test.dot(W)
	right=0
	for i in range(X_test.shape[0]):
		if p[i]>=0:
			p[i]=1
		else:
			p[i]=0
		if p[i]==y_test[i]:
			right=right+1
	P=right/X_test.shape[0]
	return P
```

```
    #SGD
    rate=0.5 #learning rate

    #NAG element
    v=0.1
    gama_NAG=0.9
    rate_NAG=0.05

    #RMSProp
    gama_RMSProp=0.9
    rate_RMSProp=0.01
    epsilon_RMSProp=0.00000001
    G_RMSProp=0

    #AdaDelta
    G_AdaDelta=0
    gama_AdaDelta=0.95
    delta=0.01
    epsilon_AdaDelta=0.00000001

    #Adam
    G_Adam=0
    beta=0.9
    gama_Adam=0.999
    rate_Adam=0.02
    epsilon_Adam=0.00000001
    m=0

    for i in range(num):
        X_sample,y_sample=sample(X_train,y_train,64)
        one_sample=np.ones((X_sample.shape[0],1))
        #SGD

J=X_sample.T.dot((h(X_sample,w)/(one_sample+h(X_sample,w))-y_sample))/X_sample.shape[0]
        w-=rate*J

loss_test_SGD[i]=(one_test.T.dot((np.log(one_test+h(X_test,w))-X_test.dot(w)*y_test)))/X_test.sha
pe[0]
        if right_rate(w)>P:
            P=right_rate(w)

        #NAG
```

```
        J1=X_sample.T.dot((h(X_sample,w1)/(one_sample+h(X_sample,w1))-y_sample))/X_sample.shape[0]
        g_NAG=J1-gama_NAG*v
        v=gama_NAG*v+rate_NAG*g_NAG
        w1=w1-v
        loss_test_NAG[i]=(one_test.T.dot((np.log(one_test+h(X_test,w1))-X_test.dot(w1)*y_test)))/X_test.shape[0]
        if right_rate(w1)>P1:
            P1=right_rate(w1)

        #RMSProp
        J2=X_sample.T.dot((h(X_sample,w2)/(one_sample+h(X_sample,w2))-y_sample))/X_sample.shape[0]
        g_RMSProp=J2
        G_RMSProp=gama_RMSProp*G_RMSProp+(1-gama_RMSProp)*g_RMSProp*g_RMSProp
        w2-=rate_RMSProp/(np.sqrt(G_RMSProp+epsilon_RMSProp))*g_RMSProp
        loss_test_RMSProp[i]=(one_test.T.dot((np.log(one_test+h(X_test,w2))-X_test.dot(w2)*y_test)))/X_test.shape[0]
        if right_rate(w2)>P2:
            P2=right_rate(w2)

        #AdaDelta
        J3=X_sample.T.dot((h(X_sample,w3)/(one_sample+h(X_sample,w3))-y_sample))/X_sample.shape[0]
        g_AdaDelta=J3
        G_AdaDelta=gama_AdaDelta*G_AdaDelta+(1-gama_AdaDelta)*g_AdaDelta*g_AdaDelta
        delta_Q=-(np.sqrt(delta+epsilon_AdaDelta)/np.sqrt(G_AdaDelta+epsilon_AdaDelta))*g_AdaDelta
        w3+=delta_Q
        delta=gama_AdaDelta*delta+(1-gama_AdaDelta)*delta_Q*delta_Q
        loss_test_AdaDelta[i]=(one_test.T.dot((np.log(one_test+h(X_test,w3))-X_test.dot(w3)*y_test)))/X_test.shape[0]
        if right_rate(w3)>P3:
            P3=right_rate(w3)

        #Adam
        J4=X_sample.T.dot((h(X_sample,w4)/(one_sample+h(X_sample,w4))-y_sample))/X_sample.shape[
```

```
0]
        g_Adam=J4
        m=beta*m+(1-beta)*g_Adam
        G_Adam=gama_Adam*G_Adam+(1-gama_Adam)*g_Adam*g_Adam
        alpha=rate_Adam*(np.sqrt(1-gama_Adam)/(1-beta))
        w4-=alpha*(m/np.sqrt(G_Adam+epsilon_Adam))

loss_test_Adam[i]=(one_test.T.dot((np.log(one_test+h(X_test,w4))-X_test.dot(w4)*y_test)))/X_test
.shape[0]
        if right_rate(w4)>P4:
            P4=right_rate(w4)

    i =range(num)
    #Draw graphs of Ltrain and Ltest with the number of iterations
    plt.rcParams['figure.figsize']= (20,10)
    plt.plot(i,loss_test_SGD,label = 'loss_SGD')
    plt.plot(i,loss_test_NAG,label = 'loss_NAG')
    plt.plot(i,loss_test_RMSProp,label = 'loss_RMSProp')
    plt.plot(i,loss_test_AdaDelta,label = 'loss_AdaDelta')
    plt.plot(i,loss_test_Adam,label = 'loss_Adam')
    plt.legend(loc='upper right')
    plt.xlabel('times')
    plt.ylabel('loss')
    plt.show()

    print(P)
    print(P1)
    print(P2)
    print(P3)
    print(P4)
```

Linear Classification and Stochastic Gradient Descent

```
from sklearn.externals.joblib import Memory
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
mem = Memory("./mycache")

@mem.cache
def get_data(path):
    data = load_svmlight_file(path)
    return data[0], data[1]

import numpy as np
```

```
import matplotlib.pyplot as plt

X_train, y_train= get_data("G:\\学习资料\\大三上\\机器学习\\experiment\\experiment2\\a9a.txt")
X_test,y_test= get_data("G:\\学习资料\\大三上\\机器学习\\experiment\\experiment2\\a9a.t")
X_train = X_train.toarray()
X_test = X_test.toarray()
X_train=np.hstack((X_train,np.ones((X_train.shape[0],1))))#add a coloum of 1 after X_train
y_train = y_train.reshape((y_train.shape[0],1))
X_test=np.hstack((X_test,np.zeros((X_test.shape[0],1))))#add a coloum of 0 after X_test
X_test=np.hstack((X_test,np.ones((X_test.shape[0],1))))#add a coloum of 1 after X_test
y_test = y_test.reshape((y_test.shape[0],1))

w=np.ones((X_train.shape[1],1)) #initialize the w
w1=np.ones((X_train.shape[1],1)) #initialize the w1
w2=np.ones((X_train.shape[1],1)) #initialize the w2
w3=np.ones((X_train.shape[1],1)) #initialize the w3
w4=np.ones((X_train.shape[1],1)) #initialize the w4
P=0
P1=0
P2=0
P3=0
P4=0
num=500 #times
loss_test_SGD=np.zeros((num)) #the loss of test set
loss_test_NAG=np.zeros((num))
loss_test_RMSProp=np.zeros((num))
loss_test_AdaDelta=np.zeros((num))
loss_test_Adam=np.zeros((num))

def h(x,W):
    return x.dot(W)

def sample(x,y,n):
    index_sample=np.random.randint(0,x.shape[0]-1,n)
    x_sample=np.zeros((n,x.shape[1]))
    y_sample=np.zeros((n,y.shape[1]))
    index=0
    for i in index_sample:
        x_sample[index]=x[i]
        y_sample[index]=y[i]
        index=index+1
    return x_sample,y_sample
```

```
def right_rate(W):
        p=X_test.dot(W)
        right=0
        for i in range(X_test.shape[0]):
                if p[i]>=0:
                        p[i]=1
                else:
                        p[i]=-1
                if p[i]==y_test[i]:
                        right=right+1
        P=right/X_test.shape[0]
        return P

#SGD
rate=0.01 #learning rate
C=100

#NAG element
v=0.1
gama_NAG=0.9
rate_NAG=0.0005
C1=100

#RMSProp
gama_RMSProp=0.9
rate_RMSProp=0.01
epsilon_RMSProp=0.00000001
G_RMSProp=0
C2=100

#AdaDelta
G_AdaDelta=0
gama_AdaDelta=0.95
delta=0.01
epsilon_AdaDelta=0.000001
C3=100

#Adam
G_Adam=0
beta=0.9
gama_Adam=0.999
rate_Adam=0.01
epsilon_Adam=0.00000001
```

```
m=0
C4=100

for i in range (num):
    X_sample,y_sample=sample(X_train,y_train,64)
    #SGD
    temp_loss=0.0
    temp_w=0    #initialize the temp_w
    temp=1-y_sample*h(X_sample,w)
    for j in range(temp.shape[0]):
        if(temp[j]>0):
            temp_w+=(y_sample[j]*X_sample[j].T).reshape((X_sample.shape[1],1))
    J=w-C*(temp_w/(X_sample.shape[0]))
    w-=rate*J    #update the w
    temp=1-y_test*h(X_test,w)
    for k in range(temp.shape[0]):
        if(temp[k]>0):
            temp_loss+=temp[k]
    loss_test_SGD[i]=C*temp_loss+0.5*np.sum(w*w)
    if right_rate(w)>P:
        P=right_rate(w)

    #NAG
    temp_loss=0.0
    temp_w=0    #initialize the temp_w
    temp=1-y_sample*h(X_sample,w1)
    for j in range(temp.shape[0]):
        if(temp[j]>0):
            temp_w+=(y_sample[j]*X_sample[j].T).reshape((X_sample.shape[1],1))
    J1=w1-C1*(temp_w/(X_sample.shape[0]))
    g_NAG=J1-gama_NAG*v
    v=gama_NAG*v+rate_NAG*g_NAG
    w1=w1-v    #update the w
    temp=1-y_test*h(X_test,w1)
    for k in range(temp.shape[0]):
        if(temp[k]>0):
            temp_loss+=temp[k]
    loss_test_NAG[i]=C1*temp_loss+0.5*np.sum(w1*w1)
    if right_rate(w1)>P1:
        P1=right_rate(w1)

    #RMSProp
    temp_loss=0.0
    temp_w=0    #initialize the temp_w
```

```
        temp=1-y_sample*h(X_sample,w2)
        for j in range(temp.shape[0]):
            if(temp[j]>0):
                temp_w+=(y_sample[j]*X_sample[j].T).reshape((X_sample.shape[1],1))
        J2=w2-C2*(temp_w/(X_sample.shape[0]))
        g_RMSProp=J2

G_RMSProp=gama_RMSProp*G_RMSProp+(1-gama_RMSProp)*g_RMSProp*g_RMSProp
        w2-=rate_RMSProp/(np.sqrt(G_RMSProp+epsilon_RMSProp))*g_RMSProp
        temp=1-y_test*h(X_test,w2)
        for k in range(temp.shape[0]):
            if(temp[k]>0):
                temp_loss+=temp[k]
        loss_test_RMSProp[i]=C2*temp_loss+0.5*np.sum(w2*w2)
        if right_rate(w2)>P2:
            P2=right_rate(w2)

        #AdaDelta
        temp_loss=0.0
        temp_w=0    #initialize the temp_w
        temp=1-y_sample*h(X_sample,w3)
        for j in range(temp.shape[0]):
            if(temp[j]>0):
                temp_w+=(y_sample[j]*X_sample[j].T).reshape((X_sample.shape[1],1))
        J3=w3-C3*(temp_w/(X_sample.shape[0]))
        g_AdaDelta=J3
        G_AdaDelta=gama_AdaDelta*G_AdaDelta+(1-gama_AdaDelta)*g_AdaDelta*g_AdaDelta

delta_Q=-(np.sqrt(delta+epsilon_AdaDelta)/np.sqrt(G_AdaDelta+epsilon_AdaDelta))*g_AdaDelta
        w3+=delta_Q
        delta=gama_AdaDelta*delta+(1-gama_AdaDelta)*delta_Q*delta_Q
        temp=1-y_test*h(X_test,w3)
        for k in range(temp.shape[0]):
            if(temp[k]>0):
                temp_loss+=temp[k]
        loss_test_AdaDelta[i]=C3*temp_loss+0.5*np.sum(w3*w3)
        if right_rate(w3)>P3:
            P3=right_rate(w3)

        #Adam
        temp_loss=0.0
        temp_w=0    #initialize the temp_w
        temp=1-y_sample*h(X_sample,w4)
        for j in range(temp.shape[0]):
```

```
        if(temp[j]>0):
            temp_w+=(y_sample[j]*X_sample[j].T).reshape((X_sample.shape[1],1))
    J4=w4-C4*(temp_w/(X_sample.shape[0]))
    g_Adam=J4
    m=beta*m+(1-beta)*g_Adam
    G_Adam=gama_Adam*G_Adam+(1-gama_Adam)*g_Adam*g_Adam
    alpha=rate_Adam*(np.sqrt(1-gama_Adam)/(1-beta))
    w4-=alpha*(m/np.sqrt(G_Adam+epsilon_Adam))
    temp=1-y_test*h(X_test,w4)
    for k in range(temp.shape[0]):
        if(temp[k]>0):
            temp_loss+=temp[k]
    loss_test_Adam[i]=C4*temp_loss+0.5*np.sum(w4*w4)
    if right_rate(w4)>P4:
        P4=right_rate(w4)


i =range(num)
#Draw graphs of Ltrain and Ltest with the number of iterations
plt.rcParams['figure.figsize']= (20,10)
plt.plot(i,loss_test_SGD,label = 'loss_SGD')
plt.plot(i,loss_test_NAG,label = 'loss_NAG')
plt.plot(i,loss_test_RMSProp,label = 'loss_RMSProp')
plt.plot(i,loss_test_AdaDelta,label = 'loss_AdaDelta')
plt.plot(i,loss_test_Adam,label = 'loss_Adam')
plt.legend(loc='upper right')
plt.xlabel('times')
plt.ylabel('loss')
plt.show()

print(P)
print(P1)
print(P2)
print(P3)
print(P4)
```

8. The initialization method of model parameters:

Logistic Regression and Stochastic Gradient Descent

zero initialization

Linear Classification and Stochastic Gradient Descent

zero initialization

9. The selected loss function and its derivatives:

Logistic Regression and Stochastic Gradient Descent

Loss function: $L=\sum_{n}-[y_n \ln f(x_n)+(1-y_n)(1-\ln f(x_n))]$

$$f(x_n)=\sigma(z)=\frac{1}{1+e^{-z}}, z = \sum w_i x_i + b$$

Gradient: the gradient of wi $G=\sum_{n}^{m}-(y_i-f(x_n))x_i^n$

For w14(b) x=1

The gradient of w $\mathbf{G}=\mathbf{X}^T(\mathbf{Xw}-\mathbf{y})$

Linear Classification and Stochastic Gradient Descent

Loss function: $L=\sum_{m}(\max(0,1-y^m(wx^m+b))+\frac{1}{2}\|\mathbf{w}\|^2$

Gradient: the gradient of wi $G=\sum_{n}-\delta(y^n(w_i x^n+b)y^n x^i$

10. Experimental results and curve:(Fill in this content for various methods of gradient descent respectively)

Logistic Regression and Stochastic Gradient Descent

Hyper-parameter selection:

SGD: η =0.5

NAG: η =0.05, γ =0.9

RMSProp: η =0.01, γ =0.9 , ε =1e-8

AdaDelta: γ =0.95, ε =1e-8

Adam: η =0.02, γ =0.999, ε =1e-8, β =0.9

Iteration: 500

Number of samples: 64

Predicted Results (Best Results):

The right rate of SGD: 0.8490264725753947

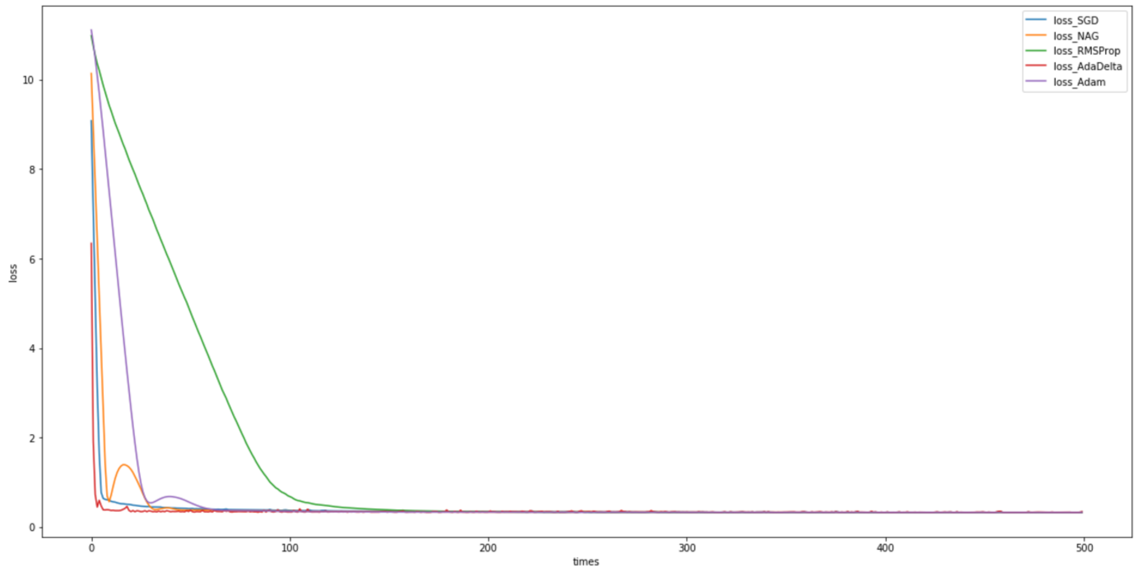
The right rate of NAG: 0.8493335790184878

The right rate of RMSProp: 0.85093.5325225723

The right rate of AdaDelta: 0.8485965235550642

The right rate of Adam: 0.8503777409250046

Loss curve:



Linear Classification and Stochastic Gradient Descent

Hyper-parameter selection:

SGD: η =0.01,C=100

NAG: η =0.0005, γ =0.9,C=100

RMSProp: η =0.01, γ =0.9 , ε =1e-8,C=100

AdaDelta: γ =0.95, ε =1e-8,C=100

Adam: η =0.01, γ =0.999, ε =1e-8, β =0.9,C=100

Iteration: 500

Number of samples: 64

Predicted Results (Best Results):

The right rate of SGD: 0.8474909403599288

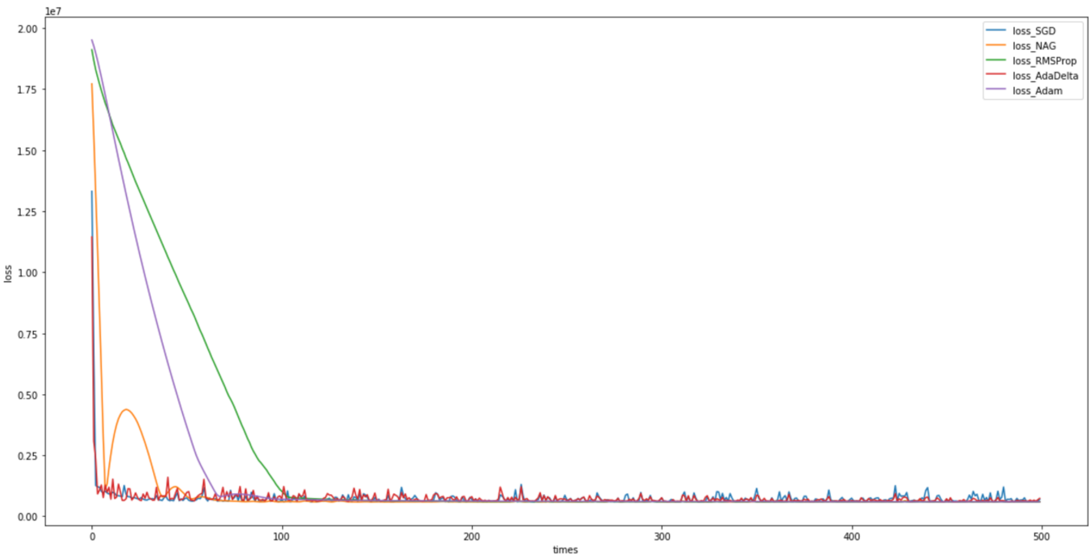
The right rate of NAG: 0.8492721577298692

The right rate of RMSProp: 0.8478594680916406

The right rate of AdaDelta: 0.8455868804127511

The right rate of Adam: 0.8449726675265647

Loss curve:



11. Results analysis:

Logistic Regression and Stochastic Gradient Descent

The five optimization models can significantly reduce the Loss value on the test set in the first 200 iterations, and then slowly decrease and stabilize in 200-500 iterations, and the corresponding correctness in the test set The rate can reach more than 84%.

The downward trend of the five optimization models is consistent, and there is no big data fluctuation, and the curve obtained is relatively smooth.

Linear Classification and Stochastic Gradient Descent

The five optimization models can significantly reduce the Loss value on the test set in the first 200 iterations, and then slowly decrease and stabilize in 200-500 iterations, and the corresponding correctness in the test set The rate can reach more than 84%.

The downward trend of the five optimization models is consistent, but the data of SGD and AdaDElta curves are more volatile than the other three curves, while the other three curves are relatively smooth.

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

The basic function of logistic regression and linear classification is to construct the model by the linear formula of $f(x) = (\sum_{i=1}^{m} w_i x_i) + b$ .Both ways are based on the classification of the problem into linear thinking.

The difference is that logistic regression is achieved by mapping the data to existing functions, whereas linear classification assumes a function to fit the data. Both methods use a gradient descent, and linear regression also uses the SVM method.

13. Summary:

This experiment let me learn the method of random gradient descent and five optimization methods. Let me know more about them. At the same time, by constantly adjusting the parameters to achieve better accuracy, I became more aware of the importance of the parameters and realized that it was not easy to learn a good model. This experiment gave me a deeper understanding of the machine learning course.