

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   **王跃**

**Student ID 201530612941**

**E-mail 602056964@qq.com**

**Tutor**  谭明奎

**Date submitted** **2017. .**

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

**2. Time:** 2017.12.02

**3. Reporter:**王跃

**4. Purposes:**

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.
4. **Data sets and data analysis:**

Experiment uses [a9a](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html" \l "a9a" \t "https://www.zybuluo.com/chenyaofo/note/_blank) of [LIBSVM Data](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/" \t "https://www.zybuluo.com/chenyaofo/note/_blank), including 32561/16281(testing) samples and each sample has 123/123 (testing) features.

1. **Experimental steps:**

Logistic Regression and Stochastic Gradient Descent：

1. Load the training set and validation set.
2. Initialize 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 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.

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  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.
8. **Code:**

The code of logistic Regression and Stochastic Gradient Descent：

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_svmlight\_file

import random

import math

# random.randint(0,99)

# math.exp( x )

dir = "./jiqixuexi/"

# get\_testdata

def get\_testdata():

data = load\_svmlight\_file(dir + "a9a.t")

return data[0], data[1]

# get\_traindata

def get\_traindata():

data = load\_svmlight\_file(dir + "a9a\_train.txt")

return data[0], data[1]

# get\_gradiant just a sample

def gradient(w,x,y,rand\_list):

grad=np.zeros((124,))

for i in rand\_list:

grad +=(1/(1+math.exp(-np.dot((w.T),(x[i].T))))-y[i])\*(x[i].T)

return grad\*(1/100)

# get\_loss

def get\_loss(w,x,y):

los = 0

for i in range(x.shape[0]):

h = 1/(1+math.exp((-1)\*np.dot((w.T),(x[i].T))))

l = (-1)\*(y[i]\*math.log(h)+(1-y[i])\*math.log(1-h))

los += l

return los/(x.shape[0])

# read\_files

X\_train,y\_train = get\_traindata()

X\_test,y\_test = get\_testdata()

for i in range(y\_test.shape[0]):

if y\_test[i] ==(-1):

y\_test[i]=0

for i in range(y\_train.shape[0]):

if y\_train[i] ==(-1):

y\_train[i]=0

a = np.ones((X\_train.shape[0],1))

b = np.ones((X\_test.shape[0],1))

c = np.zeros((X\_test.shape[0],1))

X\_train = np.column\_stack((X\_train.toarray(),a))

X\_test = np.column\_stack((X\_test.toarray(),c))

X\_test = np.column\_stack((X\_test,b))

w1 = np.zeros((124,))

e=np.zeros((124,))

for i in range(124):

e[i]=(1\*math.e\*\*(-8))

loss = []

loss\_test = []

# NAG

w2 = np.zeros((124,))

loss\_NAG = []

loss\_test\_NAG = []

v=np.zeros((124,))

# RMSProp

w3 = np.zeros((124,))

loss\_RMSProp = []

loss\_test\_RMSProp = []

g=np.zeros((124,))

G=np.zeros((124,))

G\_e=np.zeros((124,))

# AdaDelta

w4 = np.zeros((124,))

loss\_AdaDelta = []

loss\_test\_AdaDelta = []

g=np.zeros((124,))

t=np.zeros((124,))

t\_e=np.zeros((124,)) # t+e

wt=np.zeros((124,))

G=np.zeros((124,))

G\_e=np.zeros((124,)) # G+e

# Adam

k=1

w5 = np.zeros((124,))

g=np.zeros((124,))

m=np.zeros((124,))

G=np.zeros((124,))

G\_e=np.zeros((124,)) # G+e

loss\_Adam = []

loss\_test\_Adam = []

time = 1000

while time>0:

rand\_list= random.sample(range(X\_train.shape[0]),100)

# gerneral

grad = gradient(w1,X\_train,y\_train,rand\_list)

w1 = w1 - 0.001\*grad

loss.append(get\_loss(w1,X\_train,y\_train))

loss\_test.append(get\_loss(w1,X\_test,y\_test))

# NAG

v = 0.9\*v+0.001\*(gradient(w2-0.9\*v,X\_train,y\_train,rand\_list))

w2 = w2 - v

loss\_NAG.append(get\_loss(w2,X\_train,y\_train))

loss\_test\_NAG.append(get\_loss(w2,X\_test,y\_test))

# RMSProp

g=gradient(w3,X\_train,y\_train,rand\_list)

G = 0.9\*G+(1-0.9)\*(g\*g)

G\_e= G + e

for i in range(G.shape[0]):

G\_e[i]=0.001/math.sqrt(G\_e[i])

w3 = w3-G\_e\*(g)

loss\_RMSProp.append(get\_loss(w3,X\_train,y\_train))

loss\_test\_RMSProp.append(get\_loss(w3,X\_test,y\_test))

# AdaDelta

g=gradient(w4,X\_train,y\_train,rand\_list)

G=0.95\*G+(1-0.95)\*(g\*g)

t\_e=t+e

G\_e=G+e

for i in range(G.shape[0]):

G\_e[i]=-((math.sqrt(t\_e[i]))/(math.sqrt(G\_e[i])))

wt=G\_e\*g

w4=w4+wt

t=0.99\*t+(1-0.99)\*(wt\*wt)

loss\_AdaDelta.append(get\_loss(w4,X\_train,y\_train))

loss\_test\_AdaDelta.append(get\_loss(w4,X\_test,y\_test))

# Adam

g=gradient(w5,X\_train,y\_train,rand\_list)

m=0.9\*m+(1-0.9)\*g

G=0.999\*G+(1-0.999)\*(g\*g)

G\_e=G+e

for i in range(G.shape[0]):

G\_e[i]=m[i]/(math.sqrt(G\_e[i]))

Alpha= 0.001\*((math.sqrt(1-math.pow(0.999,k)))/(math.sqrt(1-math.pow(0.9,k))))

w5=w5-Alpha\*G\_e

loss\_Adam.append(get\_loss(w1,X\_train,y\_train))

loss\_test\_Adam.append(get\_loss(w1,X\_test,y\_test))

k=k+1

time = time-1

# display diagram of the result

x = []

for i in range(1000):

x.append(i+1)

plt.figure(figsize=(10,9), dpi=80)

l1,= plt.plot(x, loss,color='black')

l2,=plt.plot(x, loss\_test, color='black', linewidth=1.0, linestyle='--')

l3,=plt.plot(x, loss\_NAG, color='sienna')

l4,=plt.plot(x, loss\_test\_NAG, color='olive', linewidth=1.0, linestyle='--')

l5,=plt.plot(x, loss\_RMSProp, color='navy')

l6,=plt.plot(x, loss\_test\_RMSProp, color='c', linewidth=1.0, linestyle='--')

l7,=plt.plot(x, loss\_AdaDelta, color='blue')

l8,=plt.plot(x, loss\_test\_AdaDelta, color='m', linewidth=1.0, linestyle='--')

l9,= plt.plot(x, loss\_Adam,color='orange')

l0,=plt.plot(x, loss\_test\_Adam, color='gray', linewidth=1.0, linestyle='--')

plt.xlabel('times')

plt.ylabel('loss')

plt.legend(handles=[l1, l2,l3,l4,l5,l6,l7,l8,l9,l0,], labels=['l1', 'l2','l3', 'l4','l5', 'l6','l7', 'l8','l9', 'l0'], loc='best')

#plt.legend(loc='upper right')

plt.show()

The code of linear Classification and Stochastic Gradient Descent：

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_svmlight\_file

from sklearn.model\_selection import train\_test\_split

import random

import math

dir = "./jiqixuexi/"

# get\_testdata

def get\_testdata():

data = load\_svmlight\_file(dir + "a9a.t")

return data[0], data[1]

# get\_traindata

def get\_traindata():

data = load\_svmlight\_file(dir + "a9a\_train.txt")

return data[0], data[1]

# gradient

def gradient(w,x,y,rand\_list):

grad = np.zeros((124,))

for i in rand\_list:

if (1-y[i]\*np.dot(w.T,x[i].T))>0 :

grad += w - y[i]\*(x[i].T)

if (1-y[i]\*np.dot(w.T,x[i].T))<=0 :

grad += w

grad = (1/100)\*grad

return grad

# get\_loss

def get\_loss(w,x,y):

los = 0

for i in range(x.shape[0]):

if (1-y[i]\*np.dot(w.T,x[i].T))>0 :

los += 1-y[i]\*np.dot((w.T),(x[i].T))

if (1-y[i]\*np.dot(w.T,x[i].T))<=0 :

los += 0

return (1/x.shape[0])\*(0.5\*np.dot((w.T),w)+los)

X\_train,y\_train = get\_traindata()

X\_test,y\_test = get\_testdata()

a = np.ones((X\_train.shape[0],1))

b = np.ones((X\_test.shape[0],1))

c = np.zeros((X\_test.shape[0],1))

X\_train = np.column\_stack((X\_train.toarray(),a))

X\_test = np.column\_stack((X\_test.toarray(),c))

X\_test = np.column\_stack((X\_test,b))

e=np.zeros((124,))

for i in range(124):

e[i]=(1\*math.e\*\*(-8))

# general

loss =[]

loss\_test= []

grad= np.zeros((124,))

w = np.zeros((124,))

# NAG

w2 = np.zeros((124,))

loss\_NAG = []

loss\_test\_NAG = []

v=np.zeros((124,))

# RMSProp

w3 = np.zeros((124,))

loss\_RMSProp = []

loss\_test\_RMSProp = []

g=np.zeros((124,))

G=np.zeros((124,))

G\_e=np.zeros((124,))

# AdaDelta

w4 = np.zeros((124,))

loss\_AdaDelta = []

loss\_test\_AdaDelta = []

g=np.zeros((124,))

t=np.zeros((124,))

t\_e=np.zeros((124,)) # t+e

wt=np.zeros((124,))

G=np.zeros((124,))

G\_e=np.zeros((124,)) # G+e

#Adam

k=1

w5 = np.zeros((124,))

g=np.zeros((124,))

m=np.zeros((124,))

G=np.zeros((124,))

G\_e=np.zeros((124,)) # G+e

loss\_Adam = []

loss\_test\_Adam = []

time = 50

while time>0:

rand\_list= random.sample(range(X\_train.shape[0]),100)

# general

grad = gradient(w,X\_train,y\_train,rand\_list)

w = w - 0.05\*grad

loss.append(get\_loss(w,X\_train,y\_train))

loss\_test.append(get\_loss(w,X\_test,y\_test))

# NAG

v=0.9\*v+0.01\*gradient(w2,X\_train,y\_train,rand\_list)

w2=w2-v

loss\_NAG.append(get\_loss(w2,X\_train,y\_train))

loss\_test\_NAG.append(get\_loss(w2,X\_test,y\_test))

#RMSProp

g=gradient(w3,X\_train,y\_train,rand\_list)

G = 0.9\*G+(1-0.9)\*(g\*g)

G\_e= G + e

for i in range(G.shape[0]):

G\_e[i]=0.001/math.sqrt(G\_e[i])

w3 = w3-G\_e\*(g)

loss\_RMSProp.append(get\_loss(w3,X\_train,y\_train))

loss\_test\_RMSProp.append(get\_loss(w3,X\_test,y\_test))

#AdaDelta

g=gradient(w4,X\_train,y\_train,rand\_list)

G=0.95\*G+(1-0.95)\*(g\*g)

t\_e=t+e

G\_e=G+e

for i in range(G.shape[0]):

G\_e[i]=(-1)\*((math.sqrt(t\_e[i]))/(math.sqrt(G\_e[i])))

wt=G\_e\*g

w4=w4+wt

t=0.99\*t+(1-0.99)\*(wt\*wt)

loss\_AdaDelta.append(get\_loss(w4,X\_train,y\_train))

loss\_test\_AdaDelta.append(get\_loss(w4,X\_test,y\_test))

#Adam

g=gradient(w5,X\_train,y\_train,rand\_list)

m=0.8\*m+(1-0.9)\*g

G=0.99\*G+(1-0.999)\*(g\*g)

G\_e=G+e

for i in range(G.shape[0]):

G\_e[i]=m[i]/(math.sqrt(G\_e[i]))

Alpha= 0.001\*((math.sqrt(1-math.pow(0.999,k)))/(math.sqrt(1-math.pow(0.9,k))))

w5=w5-Alpha\*G\_e

loss\_Adam.append(get\_loss(w5,X\_train,y\_train))

loss\_test\_Adam.append(get\_loss(w5,X\_test,y\_test))

k=k+1

time = time-1;

x = []

for i in range(50):

x.append(i+1)

plt.figure(figsize=(10,9), dpi=80)

l1,= plt.plot(x, loss,color='black')

l2,=plt.plot(x, loss\_test, color='black', linewidth=1.0, linestyle='--')

l3,=plt.plot(x, loss\_NAG, color='sienna')

l4,=plt.plot(x, loss\_test\_NAG, color='olive', linewidth=1.0, linestyle='--')

l5,=plt.plot(x, loss\_RMSProp, color='navy')

l6,=plt.plot(x, loss\_test\_RMSProp, color='c', linewidth=1.0, linestyle='--')

l7,=plt.plot(x, loss\_AdaDelta, color='blue')

l8,=plt.plot(x, loss\_test\_AdaDelta, color='m', linewidth=1.0, linestyle='--')

l9,= plt.plot(x, loss\_Adam,color='orange')

l0,=plt.plot(x, loss\_test\_Adam, color='gray', linewidth=1.0, linestyle='--')

plt.xlabel('times')

plt.ylabel('loss')

plt.legend(handles=[l1, l2,l3,l4,l5,l6,l7,l8,l9,l0,], labels=['l1', 'l2','l3', 'l4','l5', 'l6','l7', 'l8','l9', 'l0'], loc='best')

#plt.legend(loc='upper right')

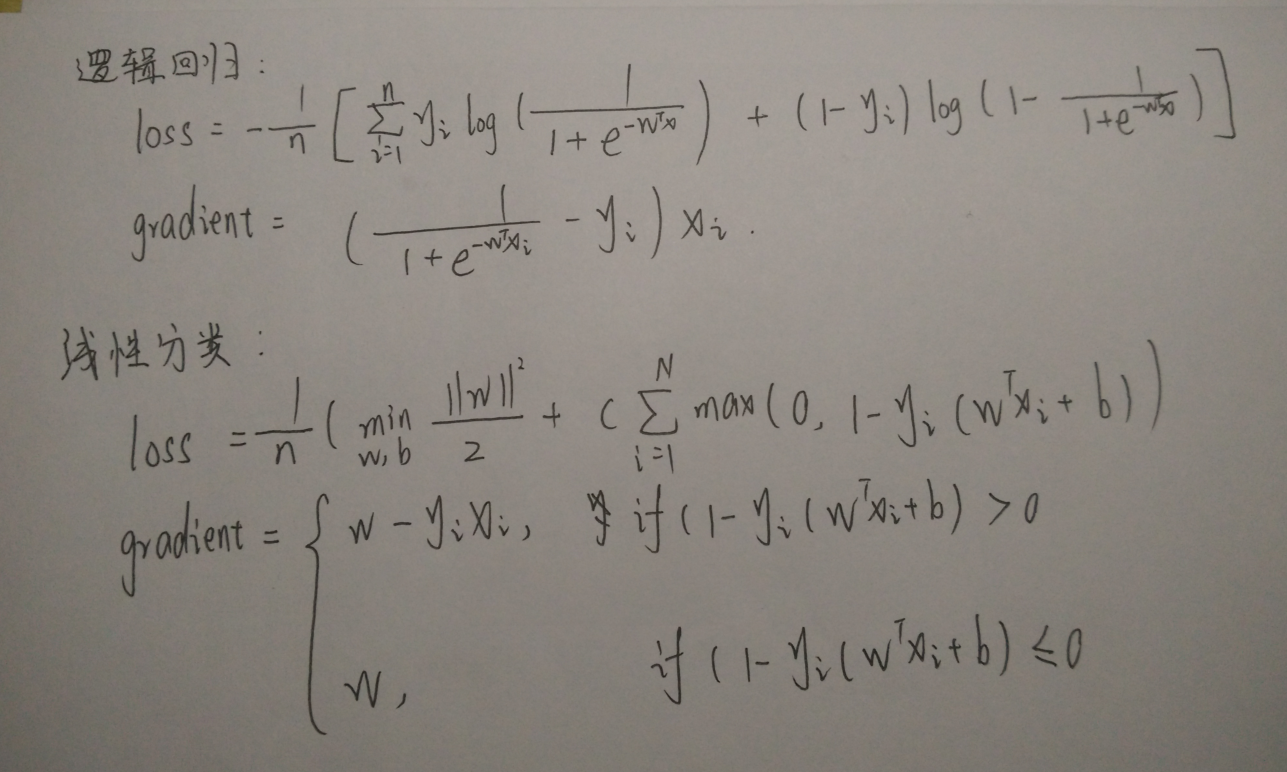
plt.show()

1. **The initialization method of model parameters:**

Logical regression: all zero initialization

Linear Category: All zero initialization

1. **The selected loss function and its derivatives:**



**10. Experimental results and curve:**

## Hyper-parameter selection:

Logistic regression: η = 0.01 β = 0. 9 γ = 0.95

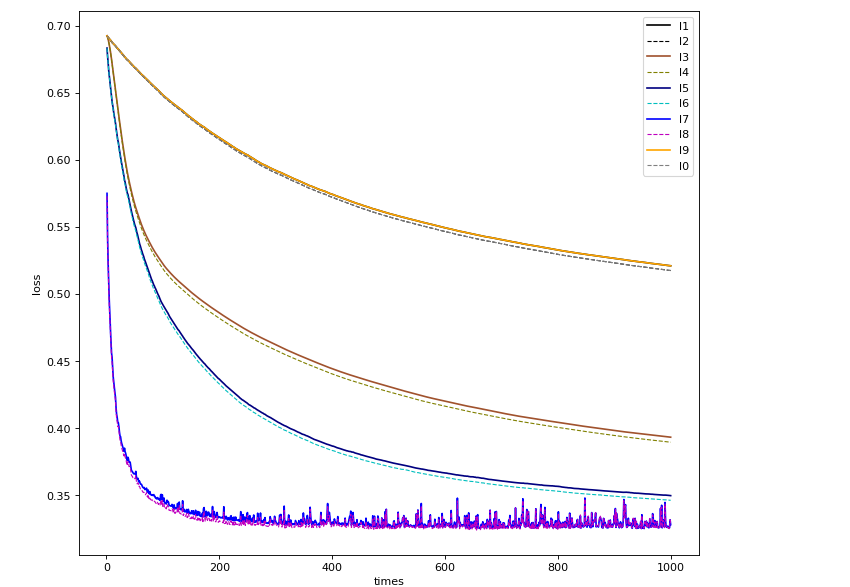
Linear classification: η = 0.01 β = 0.9 γ = 0.95

## Predicted Results (Best Results):

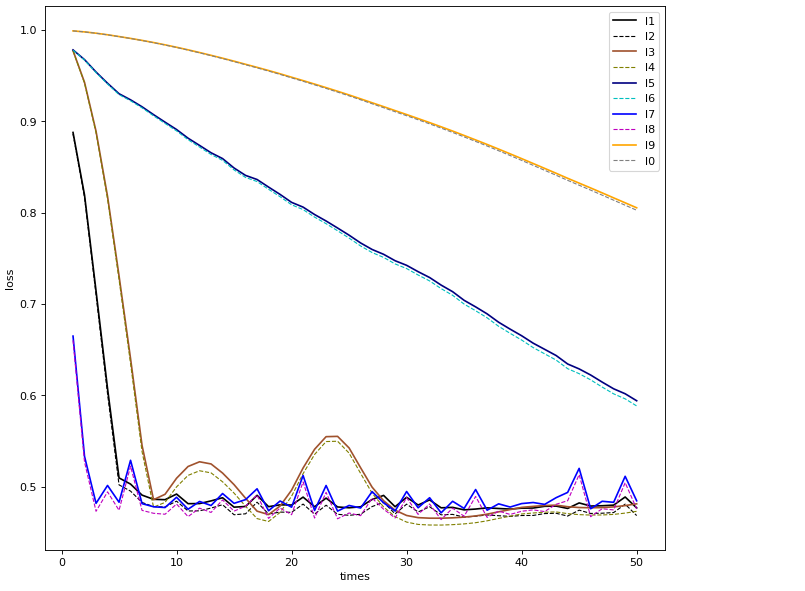
The forecast results are shown graphically.

## Loss curve:

Logistic regression:



Linear classification:



1. **Results analysis:**

Logistic regression:

Through the adjustment of parameters can get a better regression results.

Linear classification:

Through the adjustment of multiple hyper-parameters, a better SVM model can be trained on the training set.

1. **Similarities and differences between logistic regression and linear classification：**

Same point:

1, LR and SVM are classification algorithm.

2, If you do not consider the kernel function, LR and SVM are linear classification algorithm, that is, the classification decision-making surface is linear.

3, LR and SVM are supervised learning algorithm.

difference:

1, in essence, its loss function is different.

2, support vector machine only consider the local boundary line near the point, while the logical regression to consider the overall

1. **Summary:**

Through the study of this experiment, further understand the nature and realization of logistic regression and linear classification. Through the learning of gradient descent method, we further understand the important content of gradient learning. And in the course of this experiment, the gradient descent is better understood through the practice of different optimization methods.