《机器学习》课程实验报告

学　　院	软件学院
专　　业	软件工程
组　　员	李鸿境
学　　号	201530081266
邮　　箱	420962182@qq.com
指导教师	吴庆耀
提交日期	2017 年 12 月 14 日

1. 实验题目: 逻辑回归、线性分类与随机梯度下降

2. 实验时间：2017 年 12 月 2 日

3. 报告人: 李鸿境

4. 实验目的:

对比理解梯度下降和随机梯度下降的区别与联系。

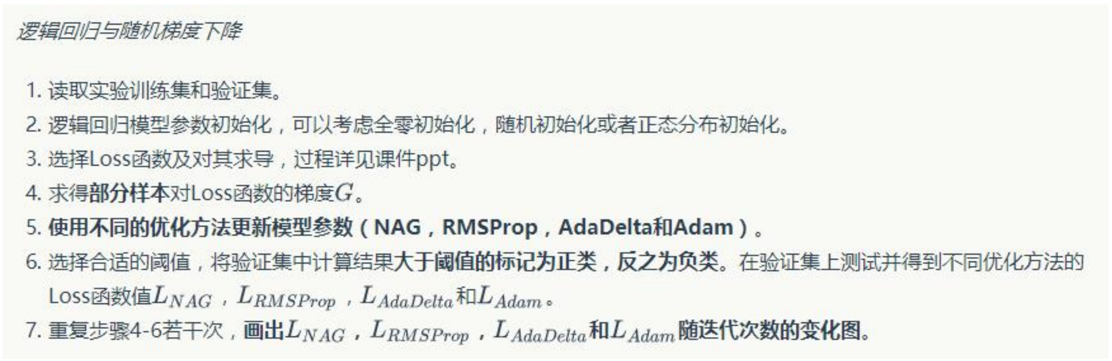
对比理解逻辑回归和线性分类的区别与联系。

进一步理解 SVM 的原理并在较大数据上实践。

5. 数据集以及数据分析：

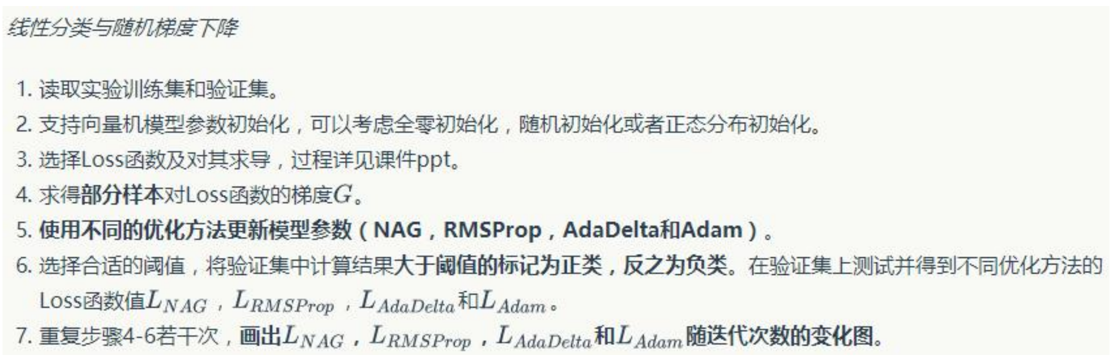
实验使用的是 **LIBSVM Data** 的中的 **a9a** 数据，包含 **32561 / 16281(testing)**个样本，每个样本有 **123/123 (testing)**个属性。

6. 实验步骤:



逻辑回归与随机梯度下降

1. 读取实验训练集和验证集。
2. 逻辑回归模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。
3. 选择Loss函数及对其求导，过程详见课件ppt。
4. 求得**部分样本**对Loss函数的梯度G。
5. **使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta和Adam）**。
6. 选择合适的阈值，将验证集中计算结果**大于阈值的标记为正类，反之为负类**。在验证集上测试并得到不同优化方法的Loss函数值L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}。
7. 重复步骤4-6若干次，**画出L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}随迭代次数的变化图**。



线性分类与随机梯度下降

1. 读取实验训练集和验证集。
2. 支持向量机模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。
3. 选择Loss函数及对其求导，过程详见课件ppt。
4. 求得**部分样本**对Loss函数的梯度G。
5. **使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta和Adam）**。
6. 选择合适的阈值，将验证集中计算结果**大于阈值的标记为正类，反之为负类**。在验证集上测试并得到不同优化方法的Loss函数值L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}。
7. 重复步骤4-6若干次，**画出L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}随迭代次数的变化图**。

7. 代码内容:

逻辑回归：

NAG：

```
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.datasets import load_svmlight_file
from sklearn.externals.joblib import Memory
import matplotlib.pyplot as plt
import math
import random

data = load_svmlight_file("a9a.txt")
X_train, y_train = data[0], data[1]
#print(X[0,0])
X_train = X_train.dot(np.eye(123))
#print(len(X))      32561
b_tr = np.ones((32561,1))
X_train = np.column_stack((X_train, b_tr))
#print(X)
#print(X.shape[1])
y_train = (y_train+1)/2
n_train = X_train.shape[0]

test = load_svmlight_file("a9a_test.txt")
X_test, y_test = test[0], test[1]
X_test = X_test.dot(np.eye(122))
a = np.zeros((16281,1))
X_test = np.column_stack((X_test, a))
b_te = np.ones((16281, 1))
X_test = np.column_stack((X_test, b_te))
y_test = (y_test+1)/2
n_test = X_test.shape[0]

w = np.zeros((124, 1))
G = np.zeros((124, 1))
v = np.zeros((124, 1))
L_test = np.zeros((1000, 1))
lossHelp = np.zeros((n_test, 1))
Y_test = np.zeros(n_test)
h = np.zeros((n_test, 1))

def sigmiod(x):
    if (x>=7):
        return 1
        # e^x/(1+e^x)
        if(x<=-7):
                return 0.0000001
        return 1.0/(1.0 + math.exp(-x))

def lossFunc(h, y):
        if(h <= 0):
                return -100
        if(h >= 1):
                return -100
        tmp = y*math.log(h)+(1-y)*math.log(1-h)
        return tmp

learningRate = 0.001

for i in range(1000):
        j = random.randint(0,32560)
        w = w - 0.9*v
        tmp = (X_train[j,:].dot(w))[0]
        output = sigmiod(tmp)
        G = ((X_train[j,:])*(output - y_train[j])).reshape(124,1)
                #s = 0
                #for k in range(124):
                        #if(X_train[j,k]!=0):
                                #s += X_train[j,k]*w[k,0]

                #output = sigmiod(s)
        v = 0.9*v + learningRate *G
        w = w - v
        h = X_test.dot(w)
        loss = 0
        for k in range(n_test):
                h[k] = sigmiod(h[k])
                loss+=lossFunc(h[k][0], y_test[k])
        L_test[i]= (-1)*loss/n_test


count = 0

for k in range(n_test):
        tmp = (X_test[k,:].dot(w))[0]
        Y_test[k] = sigmiod(tmp)
        if(Y_test[k]>=0.5):
                Y_test[k] = 1
        else:
                Y_test[k] = 0
        if(y_test[k]==Y_test[k]):
                count = count+1

true = count/n_test
print(true)
x = np.arange(0,1000,1)
%matplotlib inline
plt.plot(x,L_test,'r',label='L_test')
plt.legend(loc='upper right')
plt.xlabel('iterator times (NAG)')
plt.ylabel('Loss')
```

RMSProp：

```
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.datasets import load_svmlight_file
from sklearn.externals.joblib import Memory
import matplotlib.pyplot as plt
import math
import random

data = load_svmlight_file("a9a.txt")
X_train, y_train = data[0], data[1]
#print(X[0,0])
X_train = X_train.dot(np.eye(123))
#print(len(X))     32561
b_tr = np.ones((32561,1))
X_train = np.column_stack((X_train, b_tr))
#print(X)
#print(X.shape[1])
y_train = (y_train+1)/2
n_train = X_train.shape[0]

test = load_svmlight_file("a9a_test.txt")
X_test, y_test = test[0], test[1]
X_test = X_test.dot(np.eye(122))
a = np.zeros((16281,1))
X_test = np.column_stack((X_test, a))
b_te = np.ones((16281, 1))
X_test = np.column_stack((X_test, b_te))
y_test = (y_test+1)/2
```

```
n_test = X_test.shape[0]

w = np.zeros((124, 1))
g = np.zeros((124, 1))
G = np.zeros((124, 1))
v = np.zeros((124, 1))
L_test = np.zeros((1000, 1))
lossHelp = np.zeros((n_test, 1))
Y_test = np.zeros(n_test)
h = np.zeros((n_test, 1))

def sigmiod(x):
	if (x>=7):
		return 1
	# e^x/(1+e^x)
	if(x<=-7):
		return 0.0000001
	return 1.0/(1.0 + math.exp(-x))

def lossFunc(h, y):
	if(h <= 0):
		return -100
	if(h >= 1):
		return -100
	tmp = y*math.log(h)+(1-y)*math.log(1-h)
	return tmp

learningRate = 0.001
a = 0.00000001

for i in range(1000):
	j = random.randint(0,32560)
	#w = w - 0.9*v
	tmp = (X_train[j,:].dot(w))[0]
	output = sigmiod(tmp)
	g = ((X_train[j,:])*(output - y_train[j])).reshape(124,1)

	G = 0.9*G + 0.1*(g*g)
	for l in range(124):
		w[l][0] = w[l][0] - learningRate/(math.sqrt(G[l][0]+a))*g[l][0]
	h = X_test.dot(w)
	loss = 0
	for k in range(n_test):
		h[k] = sigmiod(h[k])
```

```
        loss+=lossFunc(h[k][0], y_test[k])
    L_test[i]= (-1)*loss/n_test


count = 0

for k in range(n_test):
    tmp = (X_test[k,:].dot(w))[0]
    Y_test[k] = sigmiod(tmp)
    if(Y_test[k]>=0.5):
        Y_test[k] = 1
    else:
        Y_test[k] = 0
    if(y_test[k]==Y_test[k]):
        count = count+1

true = count/n_test
print(true)
x = np.arange(0,1000,1)
%matplotlib inline
plt.plot(x,L_test,'r',label='L_test')
plt.legend(loc='upper right')
plt.xlabel('iterator times (RMSProp)')
plt.ylabel('Loss')
```

AdaDelta：

```
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.datasets import load_svmlight_file
from sklearn.externals.joblib import Memory
import matplotlib.pyplot as plt
import math
import random

data = load_svmlight_file("a9a.txt")
X_train, y_train = data[0], data[1]
#print(X[0,0])
X_train = X_train.dot(np.eye(123))
#print(len(X))    32561
b_tr = np.ones((32561,1))
X_train = np.column_stack((X_train, b_tr))
#print(X)
#print(X.shape[1])
```

```
y_train = (y_train+1)/2
n_train = X_train.shape[0]

test = load_svmlight_file("a9a_test.txt")
X_test, y_test = test[0], test[1]
X_test = X_test.dot(np.eye(122))
a = np.zeros((16281,1))
X_test = np.column_stack((X_test, a))
b_te = np.ones((16281, 1))
X_test = np.column_stack((X_test, b_te))
y_test = (y_test+1)/2
n_test = X_test.shape[0]

w = np.zeros((124, 1))
g = np.zeros((124, 1))
G = np.zeros((124, 1))
v = np.zeros((124, 1))
transin = np.zeros((124, 1))
tran = np.zeros((124, 1))
L_test = np.zeros((1000, 1))
lossHelp = np.zeros((n_test, 1))
Y_test = np.zeros(n_test)
h = np.zeros((n_test, 1))

def sigmiod(x):
        if (x>=7):
                return 1
        # e^x/(1+e^x)
        if(x<=-7):
                return 0.0000001
        return 1.0/(1.0 + math.exp(-x))

def lossFunc(h, y):
        if(h <= 0):
                return -100
        if(h >= 1):
                return -100
        tmp = y*math.log(h)+(1-y)*math.log(1-h)
        return tmp

learningRate = 0.001
a = 0.00000001

for i in range(1000):
        j = random.randint(0,32560)
        #w = w - 0.9*v
        tmp = (X_train[j,:].dot(w))[0]
        output = sigmiod(tmp)
        g = ((X_train[j,:])*(output - y_train[j])).reshape(124,1)

        G = 0.95*G + 0.05*(g*g)
        for l in range(124):
            transin[l][0] = - (math.sqrt(tran[l][0]+a))/(math.sqrt(G[l][0]+a))*g[l][0]

        w = w+transin
        tran = 0.95*tran + 0.05*(transin*transin)
        h = X_test.dot(w)
        loss = 0
        for k in range(n_test):
            h[k] = sigmiod(h[k])
            loss+=lossFunc(h[k][0], y_test[k])
        L_test[i]= (-1)*loss/n_test


count = 0

for k in range(n_test):
    tmp = (X_test[k,:].dot(w))[0]
    Y_test[k] = sigmiod(tmp)
    if(Y_test[k]>=0.5):
        Y_test[k] = 1
    else:
        Y_test[k] = 0
    if(y_test[k]==Y_test[k]):
        count = count+1

true = count/n_test
print(true)
x = np.arange(0,1000,1)
%matplotlib inline
plt.plot(x,L_test,'r',label='L_test')
plt.legend(loc='upper right')
plt.xlabel('iterator times (AdaDelta)')
plt.ylabel('Loss')
```

Adam：

```
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.datasets import load_svmlight_file
from sklearn.externals.joblib import Memory
import matplotlib.pyplot as plt
import math
import random

data = load_svmlight_file("a9a.txt")
X_train, y_train = data[0], data[1]
#print(X[0,0])
X_train = X_train.dot(np.eye(123))
#print(len(X))      32561
b_tr = np.ones((32561,1))
X_train = np.column_stack((X_train, b_tr))
#print(X)
#print(X.shape[1])
y_train = (y_train+1)/2
n_train = X_train.shape[0]

test = load_svmlight_file("a9a_test.txt")
X_test, y_test = test[0], test[1]
X_test = X_test.dot(np.eye(122))
a = np.zeros((16281,1))
X_test = np.column_stack((X_test, a))
b_te = np.ones((16281, 1))
X_test = np.column_stack((X_test, b_te))
y_test = (y_test+1)/2
n_test = X_test.shape[0]

w = np.zeros((124, 1))
g = np.zeros((124, 1))
G = np.zeros((124, 1))
v = np.zeros((124, 1))
m = np.zeros((124, 1))
transin = np.zeros((124, 1))
tran = np.zeros((124, 1))
L_test = np.zeros((1000, 1))
lossHelp = np.zeros((n_test, 1))
Y_test = np.zeros(n_test)
h = np.zeros((n_test, 1))

def sigmiod(x):
        if (x>=7):
                return 1
        # e^x/(1+e^x)
        if(x<=-7):
                return 0.0000001
        return 1.0/(1.0 + math.exp(-x))

def lossFunc(h, y):
        if(h <= 0):
                return -100
        if(h >= 1):
                return -100
        tmp = y*math.log(h)+(1-y)*math.log(1-h)
        return tmp

learningRate = 0.001
e = 0.00000001
a = 0

for i in range(1000):
        j = random.randint(0,32560)
        #w = w - 0.9*v
        tmp = (X_train[j,:].dot(w))[0]
        output = sigmiod(tmp)
        g = ((X_train[j,:])*(output - y_train[j])).reshape(124,1)
        m = 0.9*m + 0.1*g
        G = 0.999*G + 0.001*(g*g)
        a  =  learningRate*(math.sqrt(1  -  math.pow(0.999,  i+1)))/(1-math.pow(0.9,
i+1))
        for l in range(124):
                w[l][0] = w[l][0] - a*m[l][0]/(math.sqrt(G[l][0]+e))

        h = X_test.dot(w)
        loss = 0
        for k in range(n_test):
                h[k] = sigmiod(h[k])
                loss+=lossFunc(h[k][0], y_test[k])
        L_test[i]= (-1)*loss/n_test


count = 0

for k in range(n_test):
        tmp = (X_test[k,:].dot(w))[0]
        Y_test[k] = sigmiod(tmp)
        if(Y_test[k]>=0.5):
                Y_test[k] = 1
```

```
    else:
        Y_test[k] = 0
    if(y_test[k]==Y_test[k]):
        count = count+1

true = count/n_test
print(true)
x = np.arange(0,1000,1)
%matplotlib inline
plt.plot(x,L_test,'r',label='L_test')
plt.legend(loc='upper right')
plt.xlabel('iterator times (Adam)')
plt.ylabel('Loss')
```

线性分类：

```
from sklearn.datasets import load_svmlight_file
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

train = load_svmlight_file('a9a.txt')
test = load_svmlight_file('a9a.t')
X_train = train[0]
y_train = train[1]
X_test = test[0]
y_test = test[1]

X_train1,X_train,y_train1,y_train                                    =
train_test_split(X_train,y_train,test_size=0.1,random_state=100)
X_test1,X_test,y_test1,y_test                                        =
train_test_split(X_test,y_test,test_size=0.1,random_state=100)

y_train = map(lambda x: x if x==1 else 0, y_train)
y_test = map(lambda x: x if x==1 else 0, y_test)

X_train = X_train.todense()
y_train = np.mat(y_train).T
X_test = X_test.todense()
y_test = np.mat(y_test).T

X_train = np.column_stack((X_train, [1 for i in range(X_train.shape[0])]))
X_test = np.column_stack((X_test, [0 for i in range(X_test.shape[0])]))
X_test = np.column_stack((X_test, [1 for i in range(X_test.shape[0])]))
```

```
def lossFunc(w,x,y):
    m = x.shape[0]
    C = 1
    tot = 0.0
    for i in range(m):
        tot = tot + max(0,1-y[i]*(w*x[i].T))
    return np.double(w*w.T + C * tot)/y.shape[0]

#无优化
def SGD(X_train, X_test, y_train, y_test):
    Loss_train = []
    Loss_test = []
    epochs = 200
    learning_rate = 0.01
    w = np.mat(np.zeros(X_train.shape[1]))
    for i in range(epochs):
        loss_train = lossFunc(w,X_train,y_train)
        loss_test = lossFunc(w,X_test,y_test)
        print "%d\tloss_train%f\tloss_test%f"%(i,loss_train,loss_test)
        Loss_train.append(loss_train)
        Loss_test.append(loss_test)
        r = np.random.randint(0,X_train.shape[0]-20)
        tot = np.mat(np.zeros(X_train.shape[1]))
        for i in range(X_train.shape[0]):
            if 1 - y_train[i]*(w*X_train[i].T) > 0:
                tot+=X_train[i]*np.double(y_train[i])
        tot = w - tot
        w -= learning_rate*tot
    return Loss_train,Loss_test

#NAG
def SGD_NAG(X_train, X_test, y_train, y_test):
    Loss_train = []
    Loss_test = []
    epochs = 200
    learning_rate = 0.01
    gama = 0.001
    w = np.mat(np.zeros(X_train.shape[1]))
    v = np.mat(np.zeros(X_train.shape[1]))
    for i in range(epochs):
        loss_train = lossFunc(w,X_train,y_train)
        loss_test = lossFunc(w,X_test,y_test)
        print "%d\tloss_train%f\tloss_test%f"%(i,loss_train,loss_test)
        Loss_train.append(loss_train)
        Loss_test.append(loss_test)
        r = np.random.randint(0,X_train.shape[0]-20)
        temp = w-gama*v
        g = np.mat(np.zeros(X_train.shape[1]))
        for i in range(20):
            if 1 - y_train[r+i]*(temp*X_train[r+i].T) > 0:
                g+=X_train[r+i]*np.double(y_train[r+i])
        g = w - g
        v = gama*v + learning_rate*g
        w -= v
    return Loss_train,Loss_test

#RMSProp
def SGD_RMSProp(X_train, X_test, y_train, y_test):
    Loss_train = []
    Loss_test = []
    epochs = 200
    learning_rate = 0.01
    gama = 0.9
    epxl = 1e-8
    w = np.mat(np.zeros(X_train.shape[1]))
    G = np.mat(np.zeros(X_train.shape[1]))
    for i in range(epochs):
        loss_train = lossFunc(w,X_train,y_train)
        loss_test = lossFunc(w,X_test,y_test)
        print "%d\tloss_train%f\tloss_test%f"%(i,loss_train,loss_test)
        Loss_train.append(loss_train)
        Loss_test.append(loss_test)
        r = np.random.randint(0,X_train.shape[0]-20)
        g = np.mat(np.zeros(X_train.shape[1]))
        for i in range(20):
            if 1 - y_train[r+i]*(w*X_train[r+i].T) > 0:
                g+=X_train[r+i]*np.double(y_train[r+i])
        g = w - g
        G = gama * G + (1-gama)*np.multiply(g,g)
        w -= np.multiply(learning_rate/np.sqrt(G+epxl),g)
    return Loss_train,Loss_test

#AdaDelta
def SGD_AdaDelta(X_train, X_test, y_train, y_test):
    Loss_train = []
    Loss_test = []
    epochs = 200
    learning_rate = 0.01
    gama = 0.95
    epxl = 1e-8
    w = np.mat(np.zeros(X_train.shape[1]))
    G = np.mat(np.zeros(X_train.shape[1]))
    deltaT = np.mat([0.001 for i in range(X_train.shape[1])])
    for i in range(epochs):
        loss_train = lossFunc(w,X_train,y_train)
        loss_test = lossFunc(w,X_test,y_test)
        print "%d\tloss_train%f\tloss_test%f"%(i,loss_train,loss_test)
        Loss_train.append(loss_train)
        Loss_test.append(loss_test)
        r = np.random.randint(0,X_train.shape[0]-20)
        g = np.mat(np.zeros(X_train.shape[1]))
        for i in range(20):
            if 1 - y_train[r+i]*(w*X_train[r+i].T) > 0:
                g+=X_train[r+i]*np.double(y_train[r+i])
        g = w - g
        G = gama * G + (1-gama)*np.multiply(g,g)
        deltaW = np.multiply(np.sqrt(deltaT+epxl)/np.sqrt(G+epxl),g)
        w -= deltaW
        deltaT = gama*deltaT+(1-gama)*np.multiply(deltaW,deltaW)
    return Loss_train,Loss_test

#Adam
def SGD_Adam(X_train, X_test, y_train, y_test):
    Loss_train = []
    Loss_test = []
    epochs = 200
    learning_rate = 0.001
    gama = 0.999
    beta = 0.9
    epxl = 1e-8
    w = np.mat(np.zeros(X_train.shape[1]))
    G = np.mat(np.zeros(X_train.shape[1]))
    m = np.mat(np.zeros(X_train.shape[1]))
    for i in range(epochs):
        loss_train = lossFunc(w,X_train,y_train)
        loss_test = lossFunc(w,X_test,y_test)
        print "%d\tloss_train%f\tloss_test%f"%(i,loss_train,loss_test)
        Loss_train.append(loss_train)
        Loss_test.append(loss_test)
        r = np.random.randint(0,X_train.shape[0]-20)
        g = np.mat(np.zeros(X_train.shape[1]))
        for i in range(20):
            if 1 - y_train[r+i]*(w*X_train[r+i].T) > 0:
                g+=X_train[r+i]*np.double(y_train[r+i])
        g = w - g
        m = beta*m+(1-beta)*g
        G = gama * G + (1-gama)*np.multiply(g,g)
        alpha                                                        =
learning_rate*np.sqrt(1-np.power(gama,i+1))/(1-np.power(beta,i+1))
        w -= alpha*m/np.sqrt(G+epxl)
    return Loss_train,Loss_test

Loss_train,Loss_test = SGD(X_train, X_test, y_train, y_test)
Loss_train_NAG,Loss_test_NAG = SGD_NAG(X_train, X_test, y_train, y_test)
Loss_train_RMS,Loss_test_RMS  =  SGD_RMSProp(X_train,  X_test,  y_train,
y_test)
Loss_train_AD,Loss_test_AD = SGD_AdaDelta(X_train, X_test, y_train, y_test)
Loss_train_ADAM,Loss_test_ADAM  =  SGD_Adam(X_train,  X_test,  y_train,
y_test)
```

（针对逻辑回归和线性分类分别填写 8-11 内容）

逻辑回归：

8. 模型参数的初始化方法：

全零初始化

9. 选择的 loss 函数及其导数：

Loss 函数：

$$cost(h_\theta(x),y)=\sum_{i=1}^{m}-y_i log(h_\theta(x))-(1-y_i)log(1-h_\theta(x))$$

导数：

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)) \end{aligned}$$

10.实验结果和曲线图:（各种梯度下降方式分别填写此项）

NAG:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1} - \gamma \mathbf{v}_{t-1})$$
$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

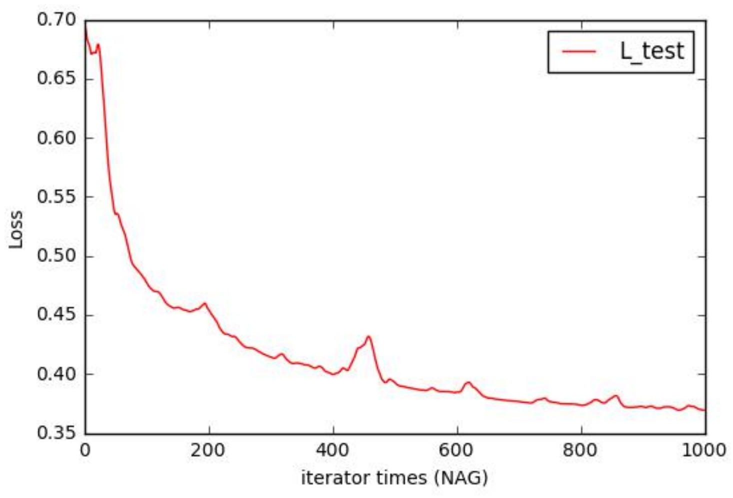
超参数选择：

γ 取 0.9

预测结果（最佳结果）：

学习率 learningRate = 0.001

loss 曲线图：



RMSProp:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

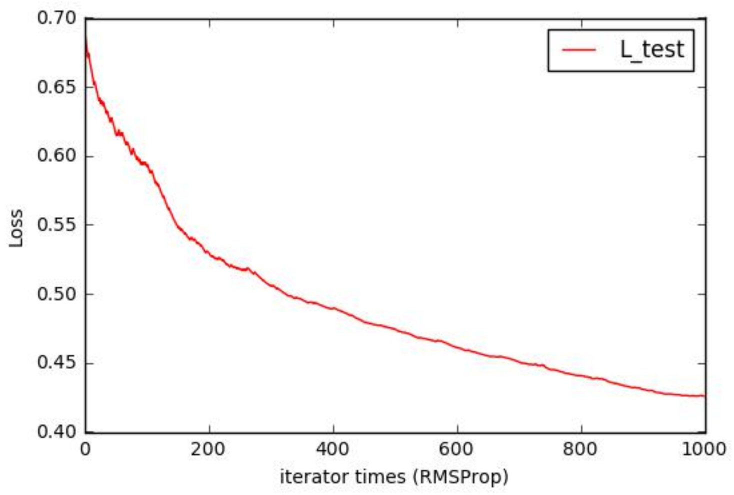
超参数选择：

γ 取 0.9

预测结果（最佳结果）：

学习率 learningRate = 0.001

loss 曲线图：



<u>AdaDelta:</u>

$$
\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\Delta\boldsymbol{\theta}_t &\leftarrow -\frac{\sqrt{\Delta_{t-1}+\epsilon}}{\sqrt{G_t+\epsilon}} \odot \mathbf{g}_t \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t \\
\Delta_t &\leftarrow \gamma\Delta_{t-1} + (1-\gamma)\Delta\boldsymbol{\theta}_t \odot \Delta\boldsymbol{\theta}_t
\end{aligned}
$$

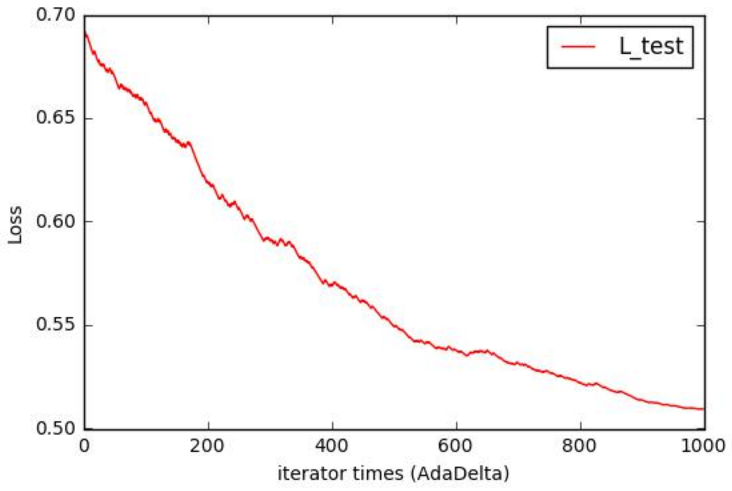
超参数选择：

γ 取 0.95

预测结果（最佳结果）：

学习率 learningRate = 0.001

loss 曲线图：



Adam:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t+\epsilon}}$$

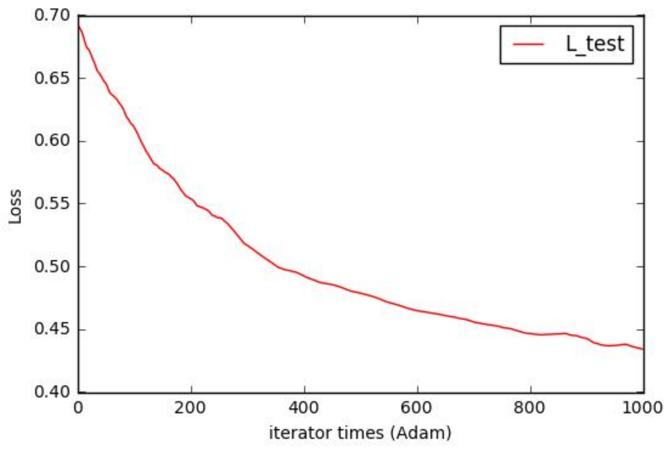
超参数选择：

γ 取 0.999 ， β_1 取 0.9

预测结果（最佳结果）：

学习率 learningRate=0.001

loss 曲线图：



11.实验结果分析:

用 SGD 算法时，收敛速度跟学习速率关系很大，大的学习率容易震荡，小的学习率收敛很慢。人为地在训练中调节是比较困难的，也难以适应数据的特征。
用改进型的算法可以减小震荡，容易跳出局部最小，也能自动调节学习率。

线性分类:

8. 模型参数的初始化方法:

全零初始化

9 .选择的 loss 函数及其导数:

Hinge 损失函数：

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

导数（梯度函数）：

$$\nabla_{w_{y_i}} L_i = \sum_{j\neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)(-x_i)$$

和

$$\nabla_{w_j} L_i = \sum_{j\neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)(x_i)$$

10.实验结果和曲线图:（各种梯度下降方式分别填写此项）

NAG:

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1} - \gamma \mathbf{v}_{t-1})$$
$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

超参数选择：

γ 取 0.9

预测结果（最佳结果）：

学习率 learningRate = 0.001

RMSProp:

$$\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t
\end{aligned}$$

超参数选择：

γ 取 0.9

预测结果（最佳结果）：

学习率 learningRate = 0.001

AdaDelta:

$$\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\Delta\boldsymbol{\theta}_t &\leftarrow -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t \\
\Delta_t &\leftarrow \gamma \Delta_{t-1} + (1-\gamma)\Delta\boldsymbol{\theta}_t \odot \Delta\boldsymbol{\theta}_t
\end{aligned}$$

超参数选择：

γ 取 0.95

预测结果（最佳结果）：

学习率 learningRate = 0.001

Adam:

$$\begin{aligned}
\mathbf{g}_t &\leftarrow \nabla J(\boldsymbol{\theta}_{t-1}) \\
\mathbf{m}_t &\leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \\
G_t &\leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t \\
\alpha &\leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t} \\
\boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t+\epsilon}}
\end{aligned}$$

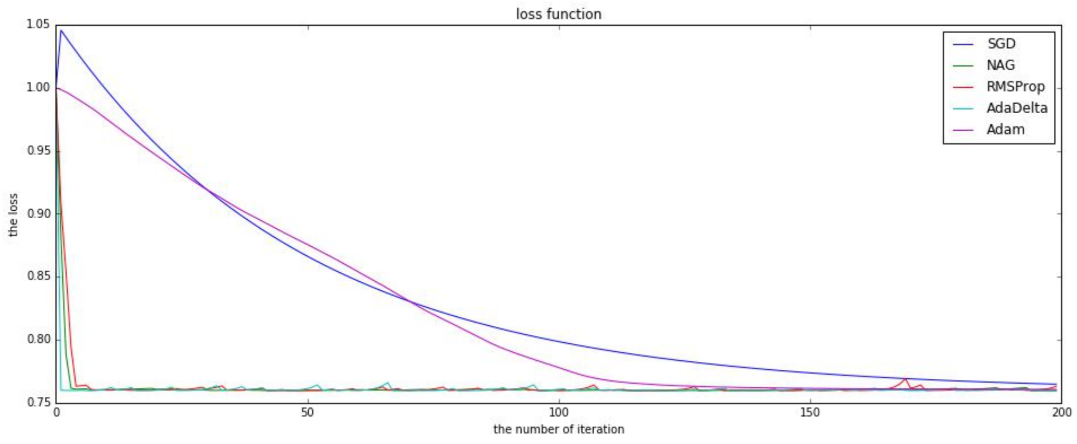
超参数选择：

γ 取 0.999 ， β_1 取 0.9

预测结果（最佳结果）：

学习率 learningRate=0.001

loss 曲线图：



11.实验结果分析:

用 SGD 算法时，收敛速度跟学习速率关系很大，大的学习率容易震荡，小的学习率收敛很慢。人为地在训练中调节是比较困难的，也难以适应数据的特征。
用改进型的算法可以减小震荡，容易跳出局部最小，也能自动调节学习率。

12.对比逻辑回归和线性分类的异同点：

.SVM 的处理方法是只考虑 support vectors,也就是和分类最相关的少数点,去学习分类器.而逻辑回归通过非线性映射,大大减小了离分类平面较远的点的权重,

相对提升了与分类最相关的数据点的权重.两者的根本目的都是一样的.此外,根据需要,两个方法都可以增加不同的正则化项。

但是逻辑回归相对来说模型更简单,好理解,实现起来,特别是大规模线性分类时比较方便.而 SVM 的理解和优化相对来说复杂一些.但是 SVM 的理论基础更加牢固,有一套结构化风险最小化的理论基础,虽然一般使用的人不太会去关注.还有很重要的一点,SVM 转化为对偶问题后,分类只需要计算与少数几个支持向量的距离,这个在进行复杂核函数计算时优势很明显,能够大大简化模型和计算。

13.实验总结:

此次实验可以说是实验一的一次扩展，在处理大数据量的时候基本上都要用到随即梯度算法，而以实验一的算法用随机梯度来处理数据时得到的损失函数会有较大的震荡，而且不能自动调节学习率，不能适应数据的特征。所以要用到改进型的 SGD 算法，本实验用 NAG，RMSProp，AdaDelta 和 Adam 四个算法，得到的损失函数也较平滑，收敛也较快。基本掌握了四个改进算法的基本用法，但其背后为什么要这么改的原因（即公式的由来）还没弄懂，接下来会花时间来看相关论文。