

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员 黄亦昕**

**学 号 201530611784**

**邮 箱** [**562145211@qq.com**](mailto:562145211@qq.com)

**指导教师** **吴庆耀**

**提交日期** **2017年 12 月 12 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 8 日

## 3. 报告人:黄亦昕

## 4. 实验目的:

1. 对比理解梯度下降和随机梯度下降的区别与联系。
2. 对比理解逻辑回归和线性分类的区别与联系。
3. 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](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/" \t "_blank)的中的[a9a](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html" \l "a9a" \t "_blank)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:



## 7. 代码内容:

（针对逻辑回归和线性分类分别填写8-11内容）

**逻辑回归：**

# write your code here

from sklearn.datasets import load\_svmlight\_file

from numpy import \*

%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

import random

#sigmoid函数

def sigmoid(inX):

    return 1.0/(1+exp(-inX))

#只对数组索引进行随机排序

def shuffle\_data\_index(data\_size):

    arr\_index = []

    for i in range(data\_size):

        arr\_index.append(i)

    random.shuffle (arr\_index)

    return arr\_index

#loss函数

def loss(y,h):

#     print("-(",y,"\*","log(",h,")+(1-",y,")\*log(1-",h,")","=",-(y\*log(h)+(1-y)\*log(1-h)))

    return -(y\*log(h)+(1-y)\*log(1-h))

#验证集上测试并得到Loss函数值

def testLogRegres(w, X\_test, y\_test):

    total\_cost=0

    shuffled\_arr = shuffle\_data\_index(y\_test.size)

    for m in range(1000):

        index = shuffled\_arr[m]

        if(y\_test[index]==-1):

            ytr=0

        else:

            ytr=1

        h = float(sigmoid(w\*X\_test[index].T))

        total\_cost += loss(ytr,h)

    return total\_cost/(m+1)

#Loss函数对w求导

def loss\_derivatived(w,X\_train,y\_train,index):

    h = float(sigmoid(w\*X\_train[index].T))

    error = (h - y\_train)

    return X\_train[index]\* error

#数据处理

def data\_process():

    t\_X,t\_y=load\_svmlight\_file("./a9a.txt")

    t\_X=t\_X.todense()

    t\_X\_row\_num,t\_X\_column\_num=shape(t\_X)

    #在X矩阵中添加一列“1”

    ones\_column = ones((t\_X\_row\_num,1))

    t\_X=hstack((ones\_column,t\_X))

    t\_X\_column\_num = t\_X[0].size

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

    X\_train, X\_test, y\_train, y\_test = train\_test\_split( t\_X, t\_y, test\_size=0.33, random\_state=43)

    return X\_train, X\_test, y\_train, y\_test

def Regression\_gradAscent(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles):

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    p\_x=[]

    p\_test\_loss=[]

    train\_size = y\_train.size

    test\_size=y\_test.size

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        if(y\_train[index]==-1):

            ytr=0

        else:

            ytr=1

        w = w - alpha \* loss\_derivatived(w,X\_train,ytr,index)

        p\_x.append(m+1)

        p\_test\_loss.append(testLogRegres(w, X\_test, y\_test))

    plt.plot(p\_x, p\_test\_loss,label="default")

    print(p\_test\_loss[499])

def Regression\_gradAscent\_NAG(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles):

    t\_X\_column\_num=X\_train[0].size

    train\_size = y\_train.size

    test\_size=y\_test.size

    w = zeros((1,t\_X\_column\_num))

    v = zeros((1,t\_X\_column\_num))

    gamma=0.7

    p\_x=[]

    p\_test\_loss=[]

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        if(y\_train[index]==-1):

            ytr=0

        else:

            ytr=1

        v=gamma\*v+alpha\*loss\_derivatived(w-gamma\*v,X\_train,ytr,index)

        w = w - v

        p\_x.append(m+1)

        p\_test\_loss.append(testLogRegres(w, X\_test, y\_test))

    plt.plot(p\_x, p\_test\_loss,label="NAG")

    plt.legend()

    print(p\_test\_loss[499])

def Regression\_gradAscent\_Adam(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles):

    learning\_rate=alpha

    D1\_decay\_rate=0.9

    D2\_decay\_rate=0.995

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    s=0

    r=0

    p\_x=[]

    p\_test\_loss=[]

    train\_size = y\_train.size

    test\_size=y\_test.size

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        if(y\_train[index]==-1):

            ytr=0

        else:

            ytr=1

        dl=loss\_derivatived(w,X\_train,ytr,index)

        s=(D1\_decay\_rate\*s+(1-D1\_decay\_rate)\*dl)

        s1=s/(1-D1\_decay\_rate)

        r=(D2\_decay\_rate\*r+(1-D2\_decay\_rate)\*multiply(dl,dl))

        r1=r/(1-D2\_decay\_rate)

        w=w-multiply(learning\_rate\*s1,1/(np.sqrt(r1)+10\*\*-8))

        p\_x.append(m+1)

        p\_test\_loss.append(testLogRegres(w, X\_test, y\_test))

    plt.plot(p\_x, p\_test\_loss,label="Adam")

    plt.legend()

    print(p\_test\_loss[499])

def Regression\_gradAscent\_RMSprop(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles):

    learning\_rate=alpha

    decay\_rate=0.995

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    r=0

    p\_x=[]

    p\_test\_loss=[]

    train\_size = y\_train.size

    test\_size=y\_test.size

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        if(y\_train[index]==-1):

            ytr=0

        else:

            ytr=1

        dl=loss\_derivatived(w,X\_train,ytr,index)

        r=decay\_rate\*r+(1-decay\_rate)\*multiply(dl,dl)

        w=w-multiply(learning\_rate/(np.sqrt(r)+1e-7),dl)

        p\_x.append(m+1)

        p\_test\_loss.append(testLogRegres(w, X\_test, y\_test))

    plt.plot(p\_x, p\_test\_loss,label="RMSprop")

    plt.legend()

    print(p\_test\_loss[499])

def Regression\_gradAscent\_Adadelta(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles):

#     learning\_rate=alpha

    decay\_rate=0.9

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    delta=0

    g=0

    e=alpha/10

#     e=0

    p\_x=[]

    p\_test\_loss=[]

    train\_size = y\_train.size

    test\_size=y\_test.size

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        if(y\_train[index]==-1):

            ytr=0

        else:

            ytr=1

        dl=loss\_derivatived(w,X\_train,ytr,index)

        g=decay\_rate\*g+(1-decay\_rate)\*multiply(dl,dl)

        e=decay\_rate\*e+(1-decay\_rate)\*multiply(delta,delta)

        delta=multiply(np.sqrt(e+1e-7)/(np.sqrt(g)+1e-7),dl)

        w=w-delta

        p\_x.append(m+1)

        p\_test\_loss.append(testLogRegres(w, X\_test, y\_test))

    plt.plot(p\_x, p\_test\_loss,label="Adadelta")

    plt.legend()

    print(p\_test\_loss[499])

def main():

    X\_train, X\_test, y\_train, y\_test=data\_process()

#     alpha = 0.04

    alpha = 0.01

    maxCycles = 500

    Regression\_gradAscent(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles)

    Regression\_gradAscent\_NAG(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles)

    Regression\_gradAscent\_RMSprop(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles)

    Regression\_gradAscent\_Adam(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles)

    Regression\_gradAscent\_Adadelta(X\_train, X\_test, y\_train, y\_test,alpha,maxCycles)

main()

**线性分类：**

# write your code here

from sklearn.datasets import load\_svmlight\_file

from numpy import \*

%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

#定义cost函数c()

def c(x,y,w):

    if(y\*(w\*x.T)<1):

        return -y

    return 0

#Loss函数对w求导

def loss\_derivatived(w,X\_train,y\_train,index):

    return (c(X\_train[index],y\_train[index],w\_temple))\*X\_train[index]

def shuffle\_data\_index(data\_size):

    arr\_index = []

    for i in range(data\_size):

        arr\_index.append(i)

    random.shuffle (arr\_index)

    return arr\_index

def testLoss(w, X\_test, y\_test):

    total\_cost=0

    shuffled\_arr = shuffle\_data\_index(y\_test.size)

    for m in range(500):

        index = shuffled\_arr[m]

        cost = float(max(0,1-y\_train[index]\*(w\*X\_train[index].T)))

        total\_cost += cost

    j = total\_cost/(m+1)

    return j

def data\_process():

    t\_X,t\_y=load\_svmlight\_file("./a9a.txt")

    t\_X=t\_X.todense()

    t\_X\_row\_num,t\_X\_column\_num=shape(t\_X)

    #在X矩阵中添加一列“1”

    ones\_column = ones((t\_X\_row\_num,1))

    t\_X=hstack((ones\_column,t\_X))

    t\_X\_column\_num = t\_X[0].size

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

    X\_train, X\_test, y\_train, y\_test = train\_test\_split( t\_X, t\_y, test\_size=0.33, random\_state=31)

    return X\_train, X\_test, y\_train, y\_test

def Classification\_gradAscent(X\_train, X\_test, y\_train, y\_test,eta,maxCycles):

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    train\_size = y\_train.size

    test\_size=y\_test.size

    p\_x = []

    p\_test\_loss=[]

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        dl=loss\_derivatived(w,X\_train,y\_train,index)

        w=w-eta\*dl

        p\_x.append(m+1)

        test\_loss=testLoss(w, X\_test, y\_test)

        p\_test\_loss.append(test\_loss)

    plt.plot(p\_x, p\_test\_loss,label="default")

#     print(p\_test\_loss[499])

def Classification\_gradAscent\_NAG(X\_train, X\_test, y\_train, y\_test,eta,maxCycles):

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    v = zeros((1,t\_X\_column\_num))

    gamma=0.9

    train\_size = y\_train.size

    test\_size=y\_test.size

    p\_x = []

    p\_test\_loss=[]

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        dl=loss\_derivatived(w-gamma\*v,X\_train,y\_train,index)

        v=gamma\*v+eta\*dl

        w=w-eta\*dl

        p\_x.append(m+1)

        test\_loss=testLoss(w, X\_test, y\_test)

        p\_test\_loss.append(test\_loss)

    plt.plot(p\_x, p\_test\_loss,label="NAG")

#     print(p\_test\_loss[499])

def Classification\_gradAscent\_RMSprop(X\_train, X\_test, y\_train, y\_test,eta,maxCycles):

    learning\_rate=eta

    decay\_rate=0.995

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    r=0

    train\_size = y\_train.size

    test\_size=y\_test.size

    p\_x=[]

    p\_test\_loss=[]

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        dl=loss\_derivatived(w,X\_train,y\_train,index)

        r=decay\_rate\*r+(1-decay\_rate)\*multiply(dl,dl)

        w=w-multiply(learning\_rate/(np.sqrt(r)+1e-7),dl)

        p\_x.append(m+1)

        p\_test\_loss.append(testLoss(w, X\_test, y\_test))

    plt.plot(p\_x, p\_test\_loss,label="RMSprop")

def Classification\_gradAscent\_Adadelta(X\_train, X\_test, y\_train, y\_test,eta,maxCycles):

    decay\_rate=0.9

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    delta=0

    g=0

    e=0

    train\_size = y\_train.size

    test\_size=y\_test.size

    p\_x = []

    p\_test\_loss=[]

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        dl=loss\_derivatived(w,X\_train,y\_train,index)

        g=decay\_rate\*g+(1-decay\_rate)\*multiply(dl,dl)

        e=decay\_rate\*e+(1-decay\_rate)\*multiply(delta,delta)

        delta=multiply(np.sqrt(e+1e-7)/(np.sqrt(g)+1e-7),dl)

        w=w-delta

        p\_x.append(m+1)

        test\_loss=testLoss(w, X\_test, y\_test)

        p\_test\_loss.append(test\_loss)

    plt.plot(p\_x, p\_test\_loss,label="Adadelta")

#     print(p\_test\_loss[499])

def Classification\_gradAscent\_Adam(X\_train, X\_test, y\_train, y\_test,eta,maxCycles):

    learning\_rate=eta

    D1\_decay\_rate=0.9

    D2\_decay\_rate=0.995

    t\_X\_column\_num=X\_train[0].size

    w = zeros((1,t\_X\_column\_num))

    s=0

    r=0

    train\_size = y\_train.size

    test\_size=y\_test.size

    p\_x = []

    p\_test\_loss=[]

    shuffled\_arr = shuffle\_data\_index(train\_size)

    for m in range(maxCycles):

        index = shuffled\_arr[m]

        dl=loss\_derivatived(w,X\_train,y\_train,index)

        s=(D1\_decay\_rate\*s+(1-D1\_decay\_rate)\*dl)

        s1=s/(1-D1\_decay\_rate)

        r=(D2\_decay\_rate\*r+(1-D2\_decay\_rate)\*multiply(dl,dl))

        r1=r/(1-D2\_decay\_rate)

        w=w-multiply(learning\_rate\*s1,1/(np.sqrt(r1)+10\*\*-8))

        p\_x.append(m+1)

        test\_loss=testLoss(w, X\_test, y\_test)

        p\_test\_loss.append(test\_loss)

    plt.plot(p\_x, p\_test\_loss,label="Adam")

    plt.legend()

#     print(p\_test\_loss[499])

def main():

    X\_train, X\_test, y\_train, y\_test=data\_process()

#     eta = 0.000603

    eta = 0.0006

    maxCycles = 600

    Classification\_gradAscent(X\_train, X\_test, y\_train, y\_test,eta,maxCycles)

    Classification\_gradAscent\_NAG(X\_train, X\_test, y\_train, y\_test,eta,maxCycles)

    Classification\_gradAscent\_RMSprop(X\_train, X\_test, y\_train, y\_test,eta,maxCycles)

    Classification\_gradAscent\_Adadelta(X\_train, X\_test, y\_train, y\_test,eta,maxCycles)

    Classification\_gradAscent\_Adam(X\_train, X\_test, y\_train, y\_test,eta,maxCycles)

main()

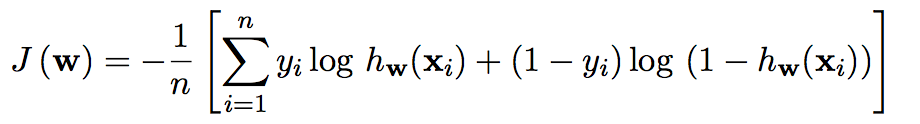
## 8. 模型参数的初始化方法:

全零初始化

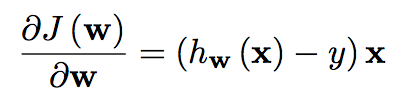
## 9.选择的loss函数及其导数:

**逻辑回归**

loss函数

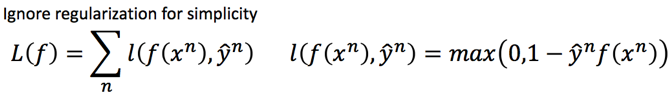


导数

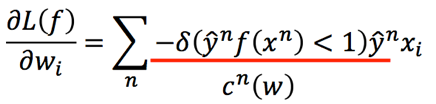


**线性分类**

loss函数



导数



## 10.实验结果和曲线图:（各种梯度下降方式分别填写此项）

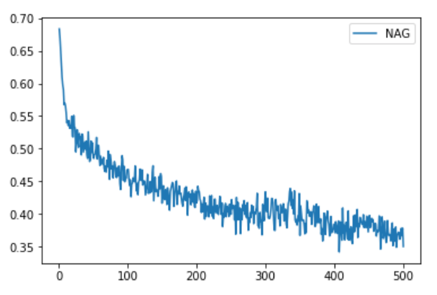
**逻辑回归**

**NAG**

超参数选择：gamma=0.7

预测结果（最佳结果）：0.35018

loss曲线图：



**RMSProp**

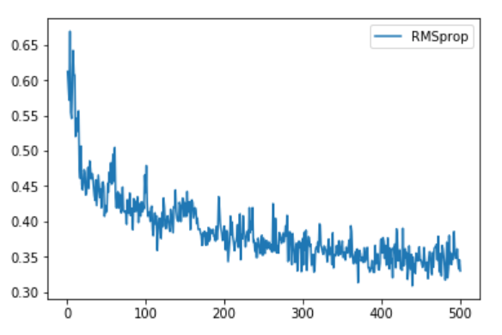
超参数选择：

learning\_rate = 0.01

decay\_rate=0.995

预测结果（最佳结果）：0.32960

loss曲线图：

****

**AdaDelta**

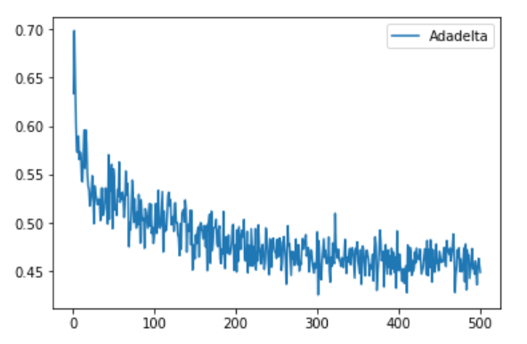
超参数选择：

decay\_rate=0.9

e=0.0001

预测结果（最佳结果）：0.43923

loss曲线图：

****

**Adam**

超参数选择：

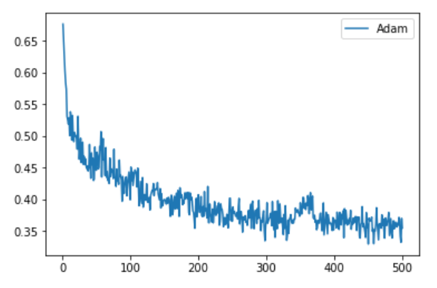
learning\_rate=0.01

D1\_decay\_rate=0.9

D2\_decay\_rate=0.995

预测结果（最佳结果）：0.34492

loss曲线图：



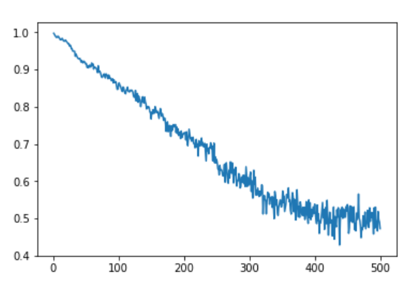
**线性分类**

**NAG**

超参数选择：gamma=0.9

预测结果（最佳结果）：0.42171

loss曲线图：



**RMSProp**

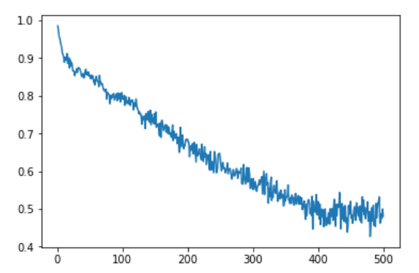
超参数选择：

learning\_rate=0.0003

decay\_rate=0.995

预测结果（最佳结果）：0.43102

loss曲线图：



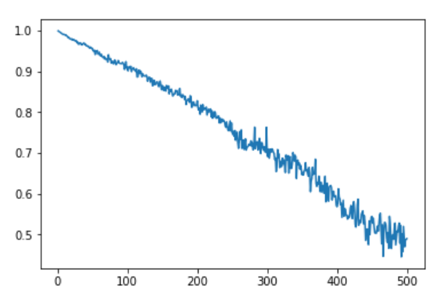
**AdaDelta**

超参数选择：

decay\_rate=0.9

预测结果（最佳结果）：0.45967

loss曲线图：

****

**Adam**

超参数选择：

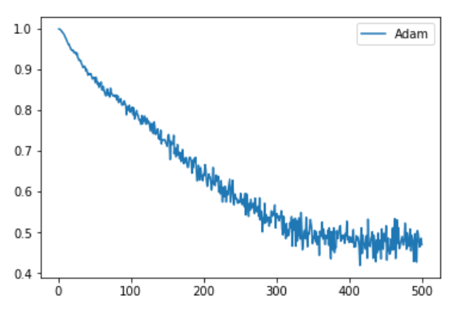
learning\_rate= 0.0005

D1\_decay\_rate=0.9

D2\_decay\_rate=0.995

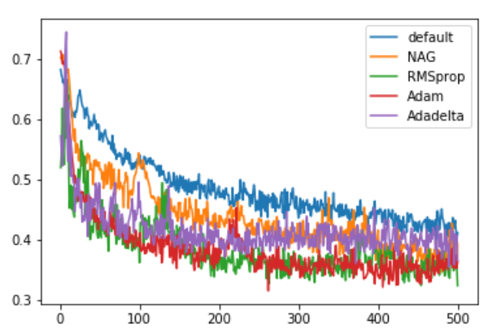
预测结果（最佳结果）：0.43531

loss曲线图：

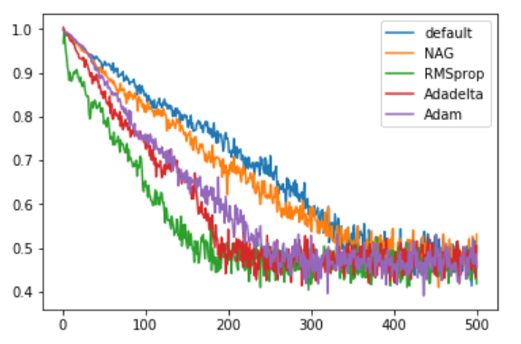


## 11.实验结果分析:

逻辑回归



线性分类（在learning\_rate = 0.0006）



**由上面的实验结果可知**

1.Adam可以不依赖于全局学习速率

2.在相同学习率（在使用到学习率的方法中）的情况下，四种优化方法更新模型参数（NAG，RMSProp，AdaDelta和Adam）使loss函数的收敛都有不同程度上的加快。在依赖学习率的算法中，RMSProp，AdaDelta普遍优于NAG。

在单独训练各使用优化方法更新模型参数的模型中，使用优化方法更新模型参数普遍可以得到比原始随机梯度下降更好的结果。

3.对于不同的训练模型，NAG，RMSProp，AdaDelta和Adam的效果有一些区别。

通过查阅资料知：

Adadelta与RMSprop在损失曲面上能够立即转移到正确的移动方向上达到快速的收敛。而NAG会导致偏离(off-track)。同时NAG能够在偏离之后快速修正其路线，因为其根据梯度修正来提高响应性。

在鞍点（saddle points）处(即某些维度上梯度为零，某些维度上梯度不为零)，SGD与NAG一直在鞍点梯度为零的方向上振荡，很难打破鞍点位置的对称性，RMSprop与Adadelta能够很快地向梯度不为零的方向上转移

链接：http://blog.csdn.net/heyongluoyao8/article/details/52478715

## 12.对比逻辑回归和线性分类的异同点：

同

1.逻辑回归和线性分类都可以解决分类问题。

2.同属于监督学习，在本次试验中，既要给定影响因子，也要给出最终的分类结果，用于更新改进测试模型。

3.都可以通过梯度下降的方法得到较优的模型

异

1.逻辑回归采用的是logistical loss，svm采用的是hinge loss。

2.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。

## 13.实验总结：

相比较于实验1，本次实验的数据规模明显增大，在这种情况下，随机梯度下降的方法更适用于本实验，这样算法会很快，但是收敛的过程会比较曲折，整体效果上，大多数时候它只能接近局部最优解，而无法真正达到局部最优解。

以上，可以使用优化方法更新模型参数来达到更好的训练效果。

在本次试验中，分别使用NAG，RMSProp，AdaDelta和Adam四种方法对线性分类和逻辑回归的梯度下降做了优化，在依赖学习率的算法中，RMSProp，AdaDelta收敛更快，Adam可以不依赖于全局学习速率。

本次实验进一步加深了我对线性分类，逻辑回归和梯度下降算法的理解。