华南理工大学

South China University of Technology

《机器学习》课程实验报告

学　　院　软件学院

专　　业　软件工程

组　　员　丁可

学　　号　201530611449

邮　　箱　952214630@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 15 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　月　日

3. 报告人:丁可

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。
Test 文件中只有 122 个属性，要添加一个

6. 实验步骤:

逻辑回归与随机梯度下降
读取实验训练集和验证集。
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。
选择 Loss 函数及对其求导，过程详见课件 ppt。
求得部分样本对 Loss 函数的梯度。
使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta 和 Adam）。
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，，和。
重复步骤 4-6 若干次，画出，，和随迭代次数的变化图。

线性分类与随机梯度下降
读取实验训练集和验证集。
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。
选择 Loss 函数及对其求导，过程详见课件 ppt。
求得部分样本对 Loss 函数的梯度。
使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta 和 Adam）。
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，，和。
重复步骤 4-6 若干次，画出，，和随迭代次数的变化图。

7. 代码内容:

逻辑回归

```
from sklearn.datasets import load_svmlight_file
```

```
from sklearn.model_selection import train_test_split
import numpy as np
import math
from math import exp,log
from random import randint
import matplotlib.pyplot as plt
import random
x_train,y_train=load_svmlight_file("C:/Users/Bean/Desktop/机器学习/机器学习实验课/a9a.txt")
x_train=x_train.toarray()
x_train=np.hstack((x_train,np.ones([np.shape(x_train)[0],1])))
y_train=y_train.reshape(np.shape(y_train)[0],1)
x_test,y_test=load_svmlight_file("C:/Users/Bean/Desktop/机器学习/机器学习实验课/a9a.t",123)
x_test=x_test.toarray()
x_test=np.hstack((x_test,np.ones([np.shape(x_test)[0],1])))
y_test=y_test.reshape(np.shape(y_test)[0],1)

n=np.shape(x_train)[0]
m=np.shape(x_train)[1]

def lossFunction(X,Y,W):
    n,m=X.shape
    loss=0
    for i in range(n):
        loss=loss+math.log((1+math.exp(-Y[i]*W.T.dot(X[i]))),math.e)
    return loss/n

def gradient(X,Y,W):
    n,m=X.shape
    g=0
    for i in range(n):
        g=g-Y[i]*X[i]/(1+exp(Y[i]*W.T.dot(X[i])))
    return (g/n).reshape((g.shape[0],1))
#NAG
W=np.zeros((m,1))
NAG=[]
eta=0.001
gamma=0.9
batch=100

v=0
for epoch in range(1500):
    random.seed()
```

```
            i=randint(0,n-1-batch)

g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W-gamma*v)
            v=gamma*v+eta*g
            W=W-v
            l_test=lossFunction(x_test,y_test,W)
            NAG.append(l_test)
      print("finish")

      #RMSProp
      W=np.zeros((m,1))
      RMSProp=[]
      eta=0.001
      gamma=0.9
      epsilon=1e-8
      batch=100
      G=0
      for epoch in range(1500):
            random.seed()
            i=randint(0,n-1-batch)

g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W)
            G=gamma*G+(1-gamma)*(g*g)
            W=W-eta/np.sqrt(G+epsilon)*g
            l_test=lossFunction(x_test,y_test,W)
            RMSProp.append(l_test)
      print('finished')

      #AdaDelta
      W=np.zeros((m,1))
      AdaDelta=[]
      gamma=0.95
      epsilon=1e-6
      batch=100
      G=0
      dt=0
      for epoch in range(1500):
            random.seed()
            i=randint(0,n-1-batch)

g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W)
```

```
        G=gamma*G+(1-gamma)*g*g
        dw=-np.sqrt(dt+epsilon)/np.sqrt(G+epsilon)*g
        W=W+dw
        dt=gamma*dt+(1-gamma)*dw*dw
        l_test=lossFunction(x_test,y_test,W)
        AdaDelta.append(l_test)
    print('finished')

    #Adam
    W=np.zeros((m,1))
    Adam=[]
    beta=0.9
    gamma=0.999
    eta=0.001
    epsilon=1e-6
    batch=100
    M=0
    for epoch in range(1500):
        i=randint(0,n-1-batch)

g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W)
        M=beta*M+(1-beta)*g
        G=gamma*G+(1-gamma)*g*g
        alpha=eta*np.sqrt(1-math.pow(gamma,epoch))/(1-beta)
        W=W-alpha*M/np.sqrt(G+epsilon)
        l_test=lossFunction(x_test,y_test,W)
        Adam.append(l_test)
    print('finished')

    plt.ylabel("loss")
    plt.plot(NAG,color='red',label='NAG')
    plt.plot(RMSProp,color='green',label='RMSProp')
    plt.plot(AdaDelta,color='blue',label='AdaDelta')
    plt.plot(Adam,color='black',label='Adam')
    plt.legend(loc='upper center',shadow=True,fontsize='x-large')
    plt.show()
```

线性分类

```
    from sklearn.datasets import load_svmlight_file
    import numpy as np
    import matplotlib.pyplot as plt
    from math import exp,log
    import math
```

```
import random
from random import randint

x_train,y_train=load_svmlight_file("C:/Users/Bean/Desktop/机器学习/机器学习实验课/a9a.txt")
x_train=x_train.toarray()
x_train=np.hstack((x_train,np.ones([np.shape(x_train)[0],1])))
y_train=y_train.reshape(np.shape(y_train)[0],1)
x_test,y_test=load_svmlight_file("C:/Users/Bean/Desktop/机器学习/机器学习实验课/a9a.t",123)
x_test=x_test.toarray()
x_test=np.hstack((x_test,np.ones([np.shape(x_test)[0],1])))
y_test=y_test.reshape(np.shape(y_test)[0],1)

n=np.shape(x_train)[0]
m=np.shape(x_train)[1]

def lossFunction(X,Y,W,C):
    loss=0
    n,m=np.shape(X)
    for i in range(n):
        loss=max(0,1-Y[i]*(W.T.dot(X[i])))
    loss=C/n*loss+(W.T.dot(W)/2)[0][0]
    return loss

def gradient(X,Y,W,C):
    g=np.zeros(np.shape(W))
    n,m=np.shape(X)
    for i in range(n):
        if 1-Y[i]*(W.T.dot(X[i]))>=0:
            g=W-C*X.T.dot(Y)
        else:
            g=W
    return g

#NAG
W=np.zeros((m,1))
NAG=[]
eta=1e-6
gamma=1e-5
batch=100
C=10
v=0
for epoch in range(1500):
```

```
            random.seed()
            i=randint(0,n-1-batch)

g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W-gamma*v,C)
            v=gamma*v+eta*g
            W=W-v
            l_test=lossFunction(x_test,y_test,W,C)
            NAG.append(l_test)
      print("finish")

      #RMSProp
      W=np.zeros((m,1))
      RMSProp=[]
      eta=1e-4
      gamma=0.9
      epsilon=1e-6
      batch=100
      C=10
      G=0
      for epoch in range(1500):
            random.seed()
            i=randint(0,n-1-batch)

g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W,C)
            G=gamma*G+(1-gamma)*(g*g)
            W=W-eta/np.sqrt(G+epsilon)*g
            l_test=lossFunction(x_test,y_test,W,C)
            RMSProp.append(l_test)
      print('finish')

      #AdaDelta
      W=np.zeros((m,1))
      AdaDelta=[]
      gamma=0.95
      epsilon=1e-9
      batch=100
      C=10
      G=0
      dt=0
      for epoch in range(1500):
            random.seed()
            i=randint(0,n-1-batch)
```

```
g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W,C)
            G=gamma*G+(1-gamma)*g*g
            dw=-np.sqrt(dt+epsilon)/np.sqrt(G+epsilon)*g
            W=W+dw
            dt=gamma*dt+(1-gamma)*dw*dw
            l_test=lossFunction(x_test,y_test,W,C)
            AdaDelta.append(l_test)
      print('finish')

      #Adam
      W=np.zeros((m,1))
      Adam=[]
      beta=0.9
      gamma=0.9
      eta=1e-5
      epsilon=1e-9
      batch=100
      M=0
      for epoch in range(1500):
            i=randint(0,n-1-batch)

g=gradient(x_train[i:i+batch].reshape((batch,m)),y_train[i:i+batch].reshape((batch,1))
,W,C)
            M=beta*M+(1-beta)*g
            G=gamma*G+(1-gamma)*g*g
            alpha=eta*np.sqrt(1-math.pow(gamma,epoch))/(1-beta)
            W=W-alpha*M/np.sqrt(G+epsilon)
            l_test=lossFunction(x_test,y_test,W,C)
            Adam.append(l_test)
      print('finish')

      plt.ylabel("loss")
      plt.plot(NAG,color='red',label='NAG')
      plt.plot(RMSProp,color='green',label='RMSProp')
      plt.plot(AdaDelta,color='blue',label='AdaDelta')
      plt.plot(Adam,color='black',label='Adam')
      plt.legend(loc='upper center',shadow=True,fontsize='x-large')
      plt.show()
```

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:logistical loss/hinge loss

10.实验结果和曲线图:（各种梯度下降方式分别填写此项）

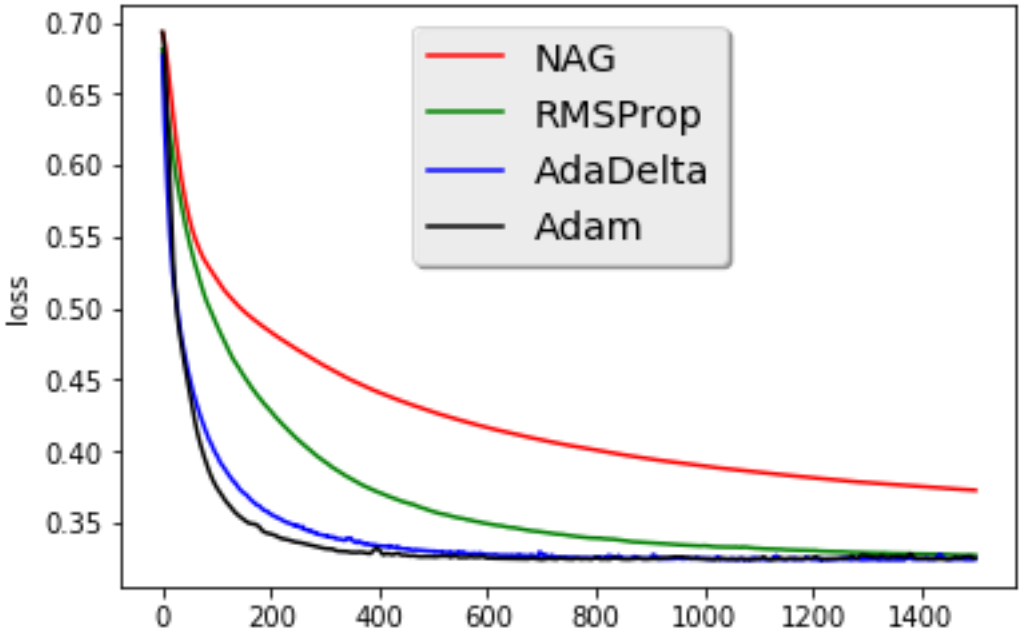
超参数选择：如代码所示

预测结果（最佳结果）：

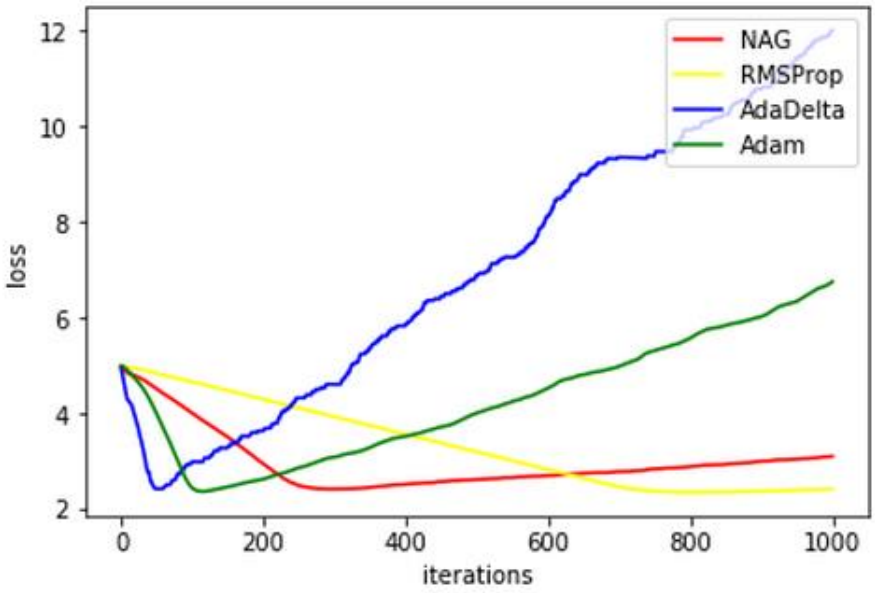
逻辑回归：min loss=0.32
线性分类：min loss=2.2

loss 曲线图：

逻辑回归：



线性分类：



11.实验结果分析:

Adam 的收敛速度最快，NAG 的收敛速度最慢

12.对比逻辑回归和线性分类的异同点：

（1）逻辑回归用的是 logistical loss，线性分类用的是 hinge loss

（2）逻辑回归模型好理解，简单容易实现，线性分类比较难理解，不易优化

（3）两种方法都是常见的分类算法

13.实验总结：

起初觉得很难，优化算法也难以理解，后来才发现原来只是打出公式就对了~不过想要理解还是要多费一些时间。