《机器学习》课程实验报告

学　　院　　软件学院

专　　业　　软件工程

组　　员　　黄景超

学　　号　　201530611722

邮　　箱　　1620265690@qq.com

指导教师 吴庆耀

提交日期 2017 年 12 月 9 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:黄景超

4. 实验目的:

A.对比理解梯度下降和随机梯度下降的区别与联系。

B.对比理解逻辑回归和线性分类的区别与联系。

C.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:

实验步骤

本次实验代码及画图均在jupyter上完成。

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. （逻辑回归）代码内容:

```
import numpy as np
```

```
import    random
import matplotlib.pyplot as plt
from sklearn.externals.joblib import Memory
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split

def get_data(path,feature):
    data = load_svmlight_file(path,feature)
    return data

def predict(W , X, y ,threshold):
    z= np.dot(W.T ,X)
    temp= 1./(1+np.exp( -z))
    y_pred=np.zeros(temp.shape)
    y_pred[temp> threshold]=1;
    y_pred[temp<=threshold]=0;
    cmp=y_pred==y
    accuracy=len(cmp[cmp==True])/y.shape[1]
      return y_pred,accuracy

#  读取数据
data = get_data(path="a9a",feature=123)

#  数据预处理
x_train=data[0].toarray()
x_train=np.column_stack((x_train,np.ones([x_train.shape[0],1])))
x_train=x_train.T
```

```
y_train=data[1]

y_train=y_train.reshape(1,len(y_train))

y_train=y_train.astype(np.int)

y_train[y_train== -1]=0

D_in, N =x_train.shape

D_out = y_train.shape[0]


# 读取数据

data = get_data(path="a9a.t",feature=123)


# 数据预处理

x_test=data[0].toarray()

x_test=np.column_stack((x_test,np.ones([x_test.shape[0],1])))

x_test=x_test.T

y_test=data[1]

y_test=y_test.reshape(1,len(y_test))

y_test=y_test.astype(np.int)

y_test[y_test== -1]=0


# 参数初始化

maxIterations=10000 # 迭代

eta = 0.05 # 学习率

threshold=0.5 # 大于阈值的标记为正类，反之为负类


def logisticRegression(W ,xtrain, ytrain, xtest ,ytest):

    N = xtrain.shape[1]

    gradNum=10
```

```
        ind=random.sample(range(0,N),gradNum)

        xtrain_batch=xtrain[:,ind]

        ytrain_batch=ytrain[:,ind]


        train_loss = 0

        test_loss =0


        dW = np.zeros(W.shape)

        z_train= np.dot(W.T ,xtrain)

        a_train= 1./(1+np.exp( -z_train))

        a_train_batch=a_train[:,ind]

        z_test= np.dot(W.T ,xtest)

        a_test= 1./(1+np.exp( -z_test))


        #逻辑回归 Loss

        train_loss= -1/N *(np.dot(np.log(a_train),ytrain.T)+np.dot(np.log(1.0-a_train),(1-ytrain).T))

        test_loss=-1/N *(np.dot(np.log(a_test),ytest.T)+np.dot(np.log(1.0-a_test),(1-ytest).T))


        #loss

        dz = a_train_batch-ytrain_batch

        dW = 1/N * np.dot(xtrain_batch, dz.T)


          return train_loss, test_loss, dW


# NAG

#  参数初始化

W = np.zeros((D_in, D_out)) # weights
```

```
pre_d = np.zeros_like(W)

pre_grad = np.zeros_like(W)

gamma =0.9 #动量因子

L_NAG =[]; #  验证 loss

for t in range(maxIterations):


    #  计算 loss

    train_loss,test_loss ,grad =logisticRegression(W, x_train, y_train, x_test, y_test)


    #  保存

    L_NAG.append ( test_loss)


    #  更新 weight

    d    = gamma * pre_d    + grad + gamma * (grad - pre_grad)

    dW = -eta * d

    W += dW

    pre_d = d

    pre_grad = grad


L_NAG=np.array(L_NAG)

L_NAG=L_NAG[:,:,0]

y_pred_NAG_train,training_accuracy_NAG =predict(W , x_train, y_train ,threshold)

y_pred_NAG_test,test_accuracy_NAG=predict(W , x_test, y_test ,threshold)


# NAG

#  参数初始化

W = np.zeros((D_in, D_out)) # weights
```

```
pre_d = np.zeros_like(W)

pre_grad = np.zeros_like(W)

gamma =0.9 #动量因子

L_NAG =[]; #  验证 loss

for t in range(maxIterations):


    #  计算 loss

    train_loss,test_loss ,grad =logisticRegression(W, x_train, y_train, x_test, y_test)


    #  保存

    L_NAG.append ( test_loss)


    #  更新 weight

    d    = gamma * pre_d    + grad + gamma * (grad - pre_grad)

    dW = -eta * d

    W += dW

    pre_d = d

    pre_grad = grad


L_NAG=np.array(L_NAG)

L_NAG=L_NAG[:,:,0]

y_pred_NAG_train,training_accuracy_NAG =predict(W , x_train, y_train ,threshold)

y_pred_NAG_test,test_accuracy_NAG=predict(W , x_test, y_test ,threshold)


# AdaDelta

#  参数初始化

W = np.zeros((D_in, D_out)) # weights
```

```
E_g2 = np.zeros_like(W)

E_dW2 = np.zeros_like(W)

gamma =0.9 #  衰退因子

epsilon = 1e-3

L_AdaDelta =[]    #  验证 loss


for t in range(maxIterations):


    train_loss,test_loss ,grad =logisticRegression(W, x_train, y_train, x_test, y_test)


    L_AdaDelta.append ( test_loss)


    E_g2 = gamma * E_g2 + (1-gamma) * np.power(grad,2)

    dW = - np.sqrt(E_dW2+epsilon) / np.sqrt(E_g2+epsilon) * grad

    W += dW

    E_dW2 = gamma * E_dW2 + (1-gamma) * np.power(dW , 2)


L_AdaDelta=np.array(L_AdaDelta)

L_AdaDelta=L_AdaDelta[:,:,0]

y_pred_AdaDelta_train,training_accuracy_AdaDelta =predict(W , x_train, y_train ,threshold)

y_pred_AdaDelta_test,test_accuracy_AdaDelta =predict(W , x_test, y_test ,threshold)


# Adam

#  参数初始化

W = np.zeros((D_in, D_out)) # weights

n = np.zeros_like(W)

m = np.zeros_like(W)
```

```
mu = 0.9 # m 的衰退因子

v = 0.9 # n 的衰退因子

epsilon = 1e-3

L_adam=[]    #  验证 loss


for t in range(maxIterations):


    train_loss,test_loss ,grad =logisticRegression(W, x_train, y_train, x_test, y_test)


    L_adam.append ( test_loss)


    #  梯度估计

    m = mu * m + (1-mu) * grad

    n = v    * n + (1-v) * np.power(grad,2)


    m_hat = m / (1-np.power(mu,t)+epsilon)

    n_hat = n / (1-np.power(v,t)+epsilon)

    W -= m_hat * eta /(np.sqrt(n_hat) + epsilon)


L_adam=np.array(L_adam)

L_adam=L_adam[:,:,0]

y_pred_Adam_train,training_accuracy_Adam =predict(W , x_train, y_train ,threshold)

y_pred_Adam_test,test_accuracy_Adam =predict(W , x_test, y_test ,threshold)


#  制图

plt.plot(L_NAG,'b',label='L_NAG')

plt.plot(L_RMSProp,'g',label='L_RMSProp')
```

```
plt.plot(L_AdaDelta,'r',label='L_AdaDelta')

plt.plot(L_adam,'y',label='L_adam')

plt.title('Loss Curve')

plt.xlabel('Iterations')

plt.ylabel('Loss')

plt.legend()

plt.show()


# 评估和预测结果

print('training accuracy_NAG=',training_accuracy_NAG,

      '\ntraining accuracy_RMSProp=',training_accuracy_RMSProp,

      '\ntraining accuracy_AdaDelta=',training_accuracy_AdaDelta,

      '\ntraining accuracy_Adam=',training_accuracy_Adam)


print('\ntest accuracy_NAG=',test_accuracy_NAG,

      '\ntest accuracy_RMSProp=',test_accuracy_RMSProp,

      '\ntest accuracy_AdaDelta=',test_accuracy_AdaDelta,

      '\ntest accuracy_Adam=',test_accuracy_Adam)
```

8. （逻辑回归）模型参数的初始化方法:

模型参数的初始化方法采用的是全零初始化。

9.（逻辑回归）选择的 loss 函数及其导数:

$$L(\theta) = -\frac{1}{m}\sum_{i=0}^{n}(y_i \log(h_\theta(X_i)) + (1-y_i)\log(1-\log(h_\theta(X_i)))$$

其中，$h_\theta(X) = \frac{1}{1+e^{-\theta^T \cdot X}}$

$$\frac{\partial L}{\partial \theta} = \frac{1}{n}\sum_{i=0}^{n} X_i(h_\theta(X) - y_i)$$

10.（逻辑回归）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：

NAG：

threshold= 0.5

eta=0.05

maxIterations=10000

gamma=0.9

RMSProp：

threshold=0.5

eta=0.05

maxIterations=10000

gamma=0.9

epsilon=0.001

AdaDelta：

threshold=0.5

maxIterations=10000

gamma=0.9

epsilon=0.001

Adam：

threshold=0.5

maxIterations=10000

mu=0.9

v=0.9

epsilon=0.001

预测结果（最佳结果）：

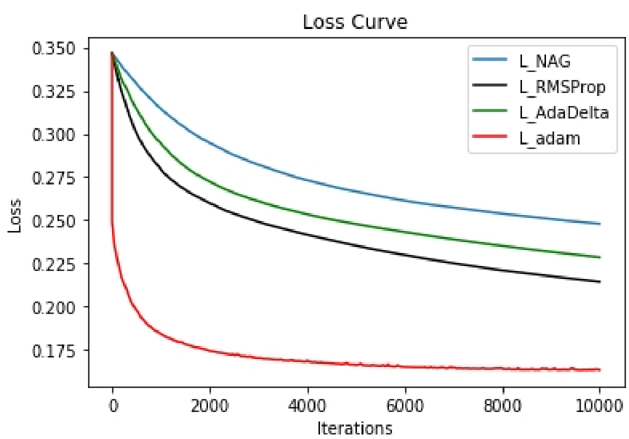
NAG：0.764

RMSProp：0.784

AdaDelta：0.765

Adam：0.850

loss 曲线图：



11.（逻辑回归）实验结果分析:

从上述预测结果可以看出，所有的预测精度都很高，这说明模型的预测效果是比较好的。

从损失曲线来看，随着迭代次数增加，损失收敛到一个很小的数且接近于零。也就是说，我们所训练的模型是比较好的。

12.（线性分类）代码内容:

```
import numpy as np

import  random

import matplotlib.pyplot as plt

from sklearn.externals.joblib import Memory

from sklearn.datasets import load_svmlight_file

from sklearn.model_selection import train_test_split



def get_data(path,feature):

    data = load_svmlight_file(path,feature)

    return data



# 读取数据

data = get_data(path="a9a",feature=123)

data = get_data(path="a9a.t",feature=123)



# 数据预处理

x_train=data[0].toarray()

x_train=np.column_stack((x_train,np.ones([x_train.shape[0],1])))

y_train=data[1]
```

```
C=len(list(set(y_train)))

y_train=y_train.astype(np.int)

y_train[y_train== -1]=0

N,D    =x_train.shape


x_test=data[0].toarray()

x_test=np.column_stack((x_test,np.ones([x_test.shape[0],1])))

y_test=data[1]

y_test=y_test.astype(np.int)

y_test[y_test== -1]=0


# 参数初始化

maxIterations=200

eta = 1e-3    # 学习率


def predict(W , X, y ):

    # 所有样例的 score

    score = np.dot(X,W)

    # 找到最大 score

    y_pred= np.argmax(score, axis = 1)

    # 计算准确度

    cmp=(y_pred == y)

    accuracy=len(cmp[cmp==True])/len(cmp)


      return y_pred,accuracy
```

```
def svm(W, xtrain, ytrain, xtest, ytest, reg):

    dW = np.zeros(W.shape)

    num_classes = W.shape[1]



    #train

    train_loss = 0

    scores_train = xtrain.dot(W)    # num_train by C

    num_train = xtrain.shape[0]

    scores_train_correct = scores_train[np.arange(num_train), ytrain]               # 1 by num_train

    scores_train_correct = np.reshape(scores_train_correct, (num_train, 1))    # num_train by 1

    margins_train = scores_train - scores_train_correct + 1.0                             # num_train by C

    margins_train[np.arange(num_train), ytrain] = 0.0

    margins_train[margins_train <= 0] = 0.0

    train_loss += np.sum(margins_train) / num_train

    train_loss += 0.5 * reg * np.sum(W * W)

    margins_train[margins_train > 0] = 1.0

    row_sum = np.sum(margins_train, axis=1)                                 # 1 by num_train

    margins_train[np.arange(num_train), ytrain] = -row_sum



    gradNum =20 #  样本数量

    ind=random.sample(range(0,num_train),gradNum)

    xtrain_batch=xtrain[ind,:]

    margins_train_batch=margins_train[ind,:]

    dW += np.dot(xtrain_batch.T, margins_train_batch)/gradNum + reg * W          # D by C
```

```
        #test
        test_loss = 0
        scores_test = xtest.dot(W)    # num_test by C
        num_test = xtest.shape[0]
        scores_test_correct = scores_test[np.arange(num_test), ytest]                  # 1 by N
        scores_test_correct = np.reshape(scores_test_correct, (num_test, 1))    # N by 1
        margins_test = scores_test - scores_test_correct + 1.0                            # N by C
        margins_test[np.arange(num_test), ytest] = 0.0
        margins_test[margins_test <= 0] = 0.0
        test_loss += np.sum(margins_test) / num_test
        test_loss += 0.5 * reg * np.sum(W * W)

         return train_loss, test_loss, dW

# NAG
#  参数初始化
W = np.zeros((D, C)) # weights
pre_d = np.zeros_like(W)
pre_grad = np.zeros_like(W)
gamma =0.9 #动量因子
L_NAG =[]; #  验证 loss
for t in range(maxIterations):
        #  计算 loss
        train_loss,test_loss ,grad =svm(W, x_train, y_train, x_test, y_test, reg=0.1)
```

```
    # 保存
    L_NAG.append ( test_loss)
    # 更新 weights
    d   = gamma * pre_d   + grad + gamma * (grad - pre_grad)
    dW = -eta * d
    W += dW
    pre_d = d
    pre_grad = grad

L_NAG=np.array(L_NAG)
# 预测结果
y_pred_NAG_train,training_accuracy_NAG = predict(W , x_train, y_train )
y_pred_NAG_test,test_accuracy_NAG = predict(W , x_test, y_test )

# RMSProp
# 参数初始化
W = np.zeros((D, C))
n = np.zeros_like(W)
gamma =0.9 # 衰退因子
epsilon = 0.001
L_RMSProp =[]   # 验证 loss

for t in range(maxIterations):

    train_loss,test_loss ,grad =svm(W, x_train, y_train, x_test, y_test, reg=0.1)
```

```
    L_RMSProp.append ( test_loss)

    n = gamma * n + (1-gamma) * np.power(grad,2)
    dW = -eta /np.sqrt(n + epsilon ) * grad
    W += dW

L_RMSProp=np.array(L_RMSProp)

y_pred_RMSProp_train,training_accuracy_RMSProp = predict(W , x_train, y_train )
y_pred_RMSProp_test,test_accuracy_RMSProp = predict(W , x_test, y_test )

# AdaDelta
# 参数初始化
W = np.zeros((D, C))
E_g2 = np.zeros_like(W)
E_dW2 = np.zeros_like(W)
gamma =0.8
epsilon = 1e-6
L_AdaDelta =[]

for t in range(maxIterations):

    train_loss,test_loss ,grad =svm(W, x_train, y_train, x_test, y_test, reg=0.1)
```

```
    L_AdaDelta.append ( test_loss)

    E_g2 = gamma * E_g2 + (1-gamma) * np.power(grad,2)

    dW = - np.sqrt(E_dW2+epsilon) / np.sqrt(E_g2+epsilon) * grad

    W += dW

    E_dW2 = gamma * E_dW2 + (1-gamma) * np.power(dW , 2)

L_AdaDelta=np.array(L_AdaDelta)

y_pred_AdaDelta_train,training_accuracy_AdaDelta = predict(W , x_train, y_train )

y_pred_AdaDelta_test,test_accuracy_AdaDelta= predict(W , x_test, y_test )

# Adam

# 参数初始化

W = np.zeros((D, C))

n = np.zeros_like(W)

m = np.zeros_like(W)

mu = 0.9

v = 0.9

epsilon = 1e-3

L_adam=[]

for t in range(maxIterations):

    train_loss,test_loss ,grad =svm(W, x_train, y_train, x_test, y_test, reg=0.1)
```

```
        L_adam.append ( test_loss)

        m = mu * m + (1-mu) * grad
        n = v    * n + (1-v) * np.power(grad,2)
        m_hat = m / (1-np.power(mu,t)+epsilon)
        n_hat = n / (1-np.power(v,t)+epsilon)
        W -= m_hat * eta /(np.sqrt(n_hat) + epsilon)

L_adam=np.array(L_adam)

y_pred_Adam_train,training_accuracy_Adam = predict(W , x_train, y_train )
y_pred_Adam_test,test_accuracy_Adam = predict(W , x_test, y_test )

# 制图
plt.plot(L_NAG,'blue',label='L_NAG')
plt.plot(L_RMSProp,'black',label='L_RMSProp')
plt.plot(L_AdaDelta,'green',label='L_AdaDelta')
plt.plot(L_adam,'red',label='L_adam')
plt.title('Error Curve')
plt.xlabel('iterations')
plt.ylabel('error')
plt.legend()
plt.show()
```

```
# 评估和预测结果
print('training accuracy_NAG=',training_accuracy_NAG,
      '\ntraining accuracy_RMSProp=',training_accuracy_RMSProp,
      '\ntraining accuracy_AdaDelta=',training_accuracy_AdaDelta,
      '\ntraining accuracy_Adam=',training_accuracy_Adam)

print('\ntest accuracy_NAG=',test_accuracy_NAG,
      '\ntest accuracy_RMSProp=',test_accuracy_RMSProp,
      '\ntest accuracy_AdaDelta=',test_accuracy_AdaDelta,
      '\ntest accuracy_Adam=',test_accuracy_Adam)
```

13. （线性分类）模型参数的初始化方法:

模型参数的初始化方法采用的是全零初始化。

14.（线性分类）选择的 loss 函数及其导数:

$$L(\theta)=\frac{1}{2n}\sum_{i=0}^{n}(y_i-h_\theta(X_i))^2$$

其中，$h_\theta(X)=\sum_{i=0}^{n}\theta_i X_i$

$$\frac{\partial L}{\partial \theta}=\frac{1}{n}\sum_{i=0}^{n}(y_i-h_\theta(X_i))\cdot X_i$$

15.（线性分类）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：

NAG：

eta=0.001

maxIterations=200

gamma=0.9

RMSProp：

eta=0.001

maxIterations=200

gamma=0.9

epsilon=0.001

AdaDelta：

maxIterations=200

gamma=0.8

epsilon=1e-6

Adam：

maxIterations=200

mu=0.9

v=0.9

epsilon=0.001

预测结果（最佳结果）：

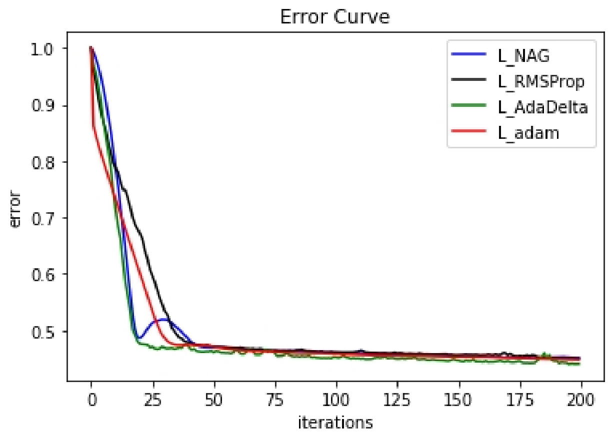
NAG：0.765

RMSProp：0.764

AdaDelta：0.798

Adam：0.768

loss 曲线图：



16.（线性分类）实验结果分析:

从上述预测结果可以看出，所有的预测精度都很高，这说明模型的预测效果是比较好的。

从损失曲线来看，随着迭代次数增加，损失收敛到一个很小的数且接近于零。也就是说，我们所训练的模型是比较好的。

17.对比逻辑回归和线性分类的异同点：

同：

都属于分类问题，都用于预测。

异：

找最优超平面的方法不同，，形象点说，logistic 模型找的那个超平面，是尽量让所有点都远离它，而 SVM 线性分类寻找的那个超平面，是只让最靠近中间分割线的那些点尽量远离，即只用到那些“支持向量”的样

本。

逻辑回归只可以处理线性可分情况；SVM 则二者皆可。

18.实验总结：

本次实验当中，我学习到了很多 SGD 在实践运用的经验，将课程中学习到的知识运用在实际问题上。但是由于自身对于知识把握懂得程度不高，在实现的过程中遇到了诸如无法正确实现 SGD 优化算法，调参不够灵活的问题，在总结反思之后解决了问题并顺利完成了实验。

在对模型的训练过程中，我体会到了灵活调参的重要性。在一开始，因为超参数 C, learning rate 等设置得不合理，导致 loss 图像与预期相差甚远，模型参数无法收敛或者收敛过慢，跑数据集时间过长等问题，导致无法拟合数据；或者是因为迭代次数过少，参数还未收敛便停止了训练。在进行数次的不同的调参后，模型往预期方向改变，我也从中学得一些经验。