

**《机器学习》课程实验报告**

**学 院 软件学院**

**专 业 软件工程**

**组 员**  **潘文杰**

**学 号 201530612590**

**邮 箱 3213359017@qq.com**

**指导教师** 吴庆耀

**提交日期** **2017年 12 月1 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 1 日

## 3. 报告人:潘文杰

## 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。7)在验证集上测试并得到不同优化方法的Loss函数值

。

8)重复步骤4-6若干次，画出

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。

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。7)在验证集上测试并得到不同优化方法的Loss函数值

。

。

8)重复步骤4-6若干次，画出随迭代次数的变化图。

## 7. 代码内容:

**逻辑回归和随机梯度下降**

# 实验: 逻辑回归和随机梯度下降

## 1. 读取实验数据，划分训练集合验证集

import sklearn

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

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

from numpy import \*

mem = Memory("./mycache")

@mem.cache

def getData():

train\_X, train\_y = load\_svmlight\_file('data/a9a', n\_features=123)

test\_X, test\_y = load\_svmlight\_file('data/a9a.t', n\_features=123)

train\_y = train\_y.reshape(train\_y.shape[0],1)

test\_y = test\_y.reshape(test\_y.shape[0],1)

train\_y[train\_y == -1] = 0

test\_y[test\_y == -1] = 0

return train\_X, test\_X, train\_y, test\_y

train\_X, test\_X, train\_y, test\_y = getData()

## 2.初始化模型参数

import numpy as np

m, n = np.shape(train\_X)

theta = np.ones((n, 1))

alpha = 0.03

maxIteration = 500

## 3.选择Loss函数

def getLoss(x,y,theta):

return -( y \* log(sigmoid(x \* theta)) + (1 - y) \* log(1 - sigmoid(x \* theta)) ).sum() / x.shape[0]

def sigmoid(a):

return 1 / (1 + np.exp(-a))

## 4.随机梯度下降函数训练

def getGradientSGD(w):

random\_num = np.random.randint(0,m)

return (train\_X[random\_num].T \* (sigmoid(train\_X[random\_num] \* w) - train\_y[random\_num]))

train\_loss = []

evaluation\_loss = []

def SGD(theta):

for i in range(0, maxIteration):

gradient = getGradientSGD(theta)

theta = theta - alpha \* gradient

train\_loss.append(getLoss(train\_X,train\_y,theta))

evaluation\_loss.append(getLoss(test\_X,test\_y,theta))

SGD(theta)

## 5.绘制Loss train和Loss validation随迭代次数的变化图

import matplotlib.pyplot as plt

%matplotlib inline

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss, label="train")

plt.plot(evaluation\_loss,label="evaluation" )

plt.legend(loc ='upper right')

## 6.使用不同的优化方法更新模型参数

### NAG

train\_loss\_nag,evaluation\_loss\_nag,train\_accr\_nag,evaluation\_accr\_nag = [],[],[],[]

theta = np.ones((n, 1))

def NAG(theta):

gama = 0.9

vt = 0

for i in range(0, maxIteration):

gradient = getGradientSGD(theta - gama\*vt)

vt = gama\*vt + alpha \* gradient

theta = theta - vt

train\_loss\_nag.append(getLoss(train\_X,train\_y,theta))

evaluation\_loss\_nag.append(getLoss(test\_X,test\_y,theta))

NAG(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_nag, label="train")

plt.plot(evaluation\_loss\_nag,label="evaluation" )

plt.legend(loc ='upper right')

### RMSProp

train\_loss\_RMSProp,evaluation\_loss\_RMSProp,train\_accr\_RMSProp,evaluation\_accr\_RMSProp = [],[],[],[]

theta = np.ones((n, 1))

def RMSProp(theta):

gama = 0.9

vt = 0

Egt = 0

e=0.00000001

learning\_rate = 0.3

for i in range(0, maxIteration):

gradient = getGradientSGD(theta - gama\*vt)

Egt = gama \* Egt + ((1-gama)\*(gradient\*\*2)).sum()

theta = theta - learning\_rate\*gradient/math.sqrt(Egt + e)

train\_loss\_RMSProp.append(getLoss(train\_X,train\_y,theta))

evaluation\_loss\_RMSProp.append(getLoss(test\_X,test\_y,theta))

RMSProp(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_RMSProp, label="train")

plt.plot(evaluation\_loss\_RMSProp,label="evaluation" )

plt.legend(loc ='upper right')

### AdaDelta

train\_loss\_adaDelta,evaluation\_loss\_adaDelta,train\_accr\_adaDelta,evaluation\_accr\_adaDelta = [],[],[],[]

theta = np.ones((n, 1))

def adaDelta(theta):

rho = 0.9

Egt=0

Edt = 0

e=0.00000001

delta = 0

learning\_rate = 2000

for i in range(0, maxIteration):

gradient = getGradientSGD(theta)

Egt = rho \* Egt + ((1-rho)\*(gradient\*\*2) ).sum()

delta = - math.sqrt(Edt + e)\*gradient/math.sqrt(Egt + e)

Edt =rho\*Edt+( (1-rho)\*(delta\*\*2) ).sum()

theta = theta + learning\_rate\*delta

train\_loss\_adaDelta.append(getLoss(train\_X,train\_y,theta))

evaluation\_loss\_adaDelta.append(getLoss(test\_X,test\_y,theta))

adaDelta(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_adaDelta, label="train")

plt.plot(evaluation\_loss\_adaDelta,label="evaluation" )

plt.legend(loc ='upper right')

### Adam

train\_loss\_adam,evaluation\_loss\_adam,train\_accr\_adam,evaluation\_accr\_adam = [],[],[],[]

theta = np.ones((n, 1))

def adam(theta):

t = 0

m = 0

v = 0

b1 = 0.9

b2 = 0.995

learning\_rate = 0.05

for i in range(0, maxIteration):

gradient = getGradientSGD(theta)

t +=1

m = b1 \* m + ((1 - b1) \* gradient).sum()

v = b2 \* v + ((1 - b2) \* (gradient \*\* 2)).sum()

mt = m / (1 - (b1 \*\* t))

vt = v / (1 - (b2 \*\* t))

theta = theta- learning\_rate \* mt / (math.sqrt(vt) + e)

train\_loss\_adam.append(getLoss(train\_X,train\_y,theta))

evaluation\_loss\_adam.append(getLoss(test\_X,test\_y,theta))

adam(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_adam, label="train")

plt.plot(evaluation\_loss\_adam,label="evaluation" )

plt.legend(loc ='upper right')

## 五种不同随机梯度下降方法对比

plt.plot(train\_loss, label="train\_loss")

plt.plot(train\_loss\_nag, label="train\_loss\_nag")

plt.plot(train\_loss\_adaDelta, label="train\_loss\_adaDelta")

plt.plot(train\_loss\_RMSProp, label ="train\_loss\_RMSProp")

plt.plot(train\_loss\_adam, label="train\_loss\_adam")

plt.legend(loc="upper right")

**线性分类和随机梯度下降**

# 实验：线性分类和随机梯度下降

## 1. 读取实验数据，划分训练集合验证集

import sklearn

from sklearn.externals.joblib import Memory

from sklearn.datasets import load\_svmlight\_file

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

from numpy import \*

mem = Memory("./mycache")

@mem.cache

def getData():

train\_X, train\_y = load\_svmlight\_file('data/a9a', n\_features=123)

test\_X, test\_y = load\_svmlight\_file('data/a9a.t', n\_features=123)

train\_y = train\_y.reshape(train\_y.shape[0],1)

test\_y = test\_y.reshape(test\_y.shape[0],1)

train\_y[train\_y == -1] = 0

test\_y[test\_y == -1] = 0

return train\_X, test\_X, train\_y, test\_y

train\_X, test\_X, train\_y, test\_y = getData()

## 2.初始化SVM模型参数

import numpy as np

m, n = np.shape(train\_X)

theta = np.ones((n, 1))

maxIteration = 300

c = 0.5

learning\_rate = 0.01

## 3.计算随机梯度下降函数和计算Loss函数

def getStochasticGradient(theta):

index = (1 - train\_y \* (train\_X \* theta) < 0)

y = train\_y.copy()

y[index] = 0

randomNum = np.random.randint(0,train\_X.shape[0])

epsilon\_gradient = - ((train\_X)[randomNum].T \* y[randomNum]).reshape(123,1)

gradient = theta + epsilon\_gradient

return gradient

def getHingeLoss(theta,x,y):

epsilon\_loss = 1 - y \* x.dot(theta)

epsilon\_loss[epsilon\_loss<0] = 0

loss = 0.5 \* np.dot(theta.transpose(), theta).sum() + epsilon\_loss.sum()

return loss/x.shape[0]

## 4.随机梯度下降训练

train\_loss, evaluation\_loss, train\_accr, evaluation\_accr = [],[],[],[]

def gradientDescent(w):

for i in range(maxIteration):

gradient = getStochasticGradient(w)

w -= learning\_rate\*gradient

train\_loss.append(getHingeLoss(w,train\_X,train\_y))

evaluation\_loss.append( getHingeLoss(w,test\_X,test\_y))

gradientDescent(theta)

## 5.绘制Loss train和Loss validation随迭代次数的变化图

import matplotlib.pyplot as plt

%matplotlib inline

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot( train\_loss, label="train")

plt.plot( evaluation\_loss,label="evaluation" )

plt.legend(loc="upper right")

## 6.使用不同的优化方法更新模型参数

### NAG

train\_loss\_nag,evaluation\_loss\_nag,train\_accr\_nag,evaluation\_accr\_nag = [],[],[],[]

theta = np.ones((n, 1))

def NAG(w):

vt = 0

gama = 0.9

for i in range(maxIteration):

gradient = getStochasticGradient(w -gama\*vt)

vt = gama\*vt + learning\_rate \* gradient

w = w - vt

train\_loss\_nag.append(getHingeLoss(w,train\_X,train\_y))

evaluation\_loss\_nag.append( getHingeLoss(w,test\_X,test\_y))

NAG(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_nag, label="train")

plt.plot(evaluation\_loss\_nag,label="evaluation" )

plt.legend(loc ='upper right')

### RMSProp

train\_loss\_RMSProp,evaluation\_loss\_RMSProp,train\_accr\_RMSProp,evaluation\_accr\_RMSProp = [],[],[],[]

theta = np.ones((n, 1))

def RMSProp(w):

gama = 0.9

vt = 0

Egt = 0

e=0.00000001

learning\_rate = 0.3

for i in range(0, maxIteration):

gradient = getStochasticGradient(w - gama\*vt)

Egt = gama \* Egt + ((1-gama)\*(gradient\*\*2)).sum()

w -= learning\_rate\*gradient/math.sqrt(Egt + e)

train\_loss\_RMSProp.append(getHingeLoss(w,train\_X,train\_y))

evaluation\_loss\_RMSProp.append( getHingeLoss(w,test\_X,test\_y))

RMSProp(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_RMSProp, label="train")

plt.plot(evaluation\_loss\_RMSProp,label="evaluation" )

plt.legend(loc ='upper right')

### AdaDelta

train\_loss\_adaDelta,evaluation\_loss\_adaDelta,train\_accr\_adaDelta,evaluation\_accr\_adaDelta = [],[],[],[]

theta = np.ones((n, 1))

def adaDelta(w):

rho = 0.9

Egt=0

Edt = 0

e=0.00000001

delta = 0

learning\_rate = 2000

for i in range(0, maxIteration):

gradient = getStochasticGradient(w)

Egt = rho \* Egt + ((1-rho)\*(gradient\*\*2) ).sum()

delta = - math.sqrt(Edt + e)\*gradient/math.sqrt(Egt + e)

Edt =rho\*Edt+( (1-rho)\*(delta\*\*2) ).sum()

w = w + learning\_rate\*delta

train\_loss\_adaDelta.append(getHingeLoss(w,train\_X,train\_y))

evaluation\_loss\_adaDelta.append( getHingeLoss(w,test\_X,test\_y))

adaDelta(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_adaDelta, label="train")

plt.plot(evaluation\_loss\_adaDelta,label="evaluation" )

plt.legend(loc ='upper right')

### Adam

train\_loss\_adam,evaluation\_loss\_adam,train\_accr\_adam,evaluation\_accr\_adam = [],[],[],[]

theta = np.ones((n, 1))

def adam(w):

t = 0

m = 0

v = 0

b1 = 0.9

b2 = 0.995

learning\_rate = 0.05

for i in range(0, maxIteration):

gradient = getStochasticGradient(w)

t +=1

m = b1\*m + ((1-b1)\*gradient).sum()

v = b2\*v + ((1-b2)\*(gradient\*\*2)).sum()

mt = m/(1-(b1\*\*t))

vt = v/(1-(b2\*\*t))

w = w- learning\_rate \* mt/(math.sqrt(vt)+e)

train\_loss\_adam.append(getHingeLoss(w,train\_X,train\_y))

evaluation\_loss\_adam.append( getHingeLoss(w,test\_X,test\_y))

adam(theta)

plt.xlabel("Iterations")

plt.ylabel("Loss")

plt.plot(train\_loss\_adam, label="train")

plt.plot(evaluation\_loss\_adam,label="evaluation" )

plt.legend(loc ='upper right')

## 五种不同随机梯度下降方法对比

plt.plot(train\_loss, label="train\_loss")

plt.plot(train\_loss\_nag, label="train\_loss\_nag")

plt.plot(train\_loss\_adaDelta, label="train\_loss\_adaDelta")

plt.plot(train\_loss\_RMSProp, label ="train\_loss\_RMSProp")

plt.plot(train\_loss\_adam, label="train\_loss\_adam")

plt.legend(loc="upper right")

## 8. 模型参数的初始化方法:

**逻辑回归**

m, n = np.shape(train\_X)

theta = np.ones((n, 1))

alpha = 0.03

maxIteration = 500

**线性分类**

m, n = np.shape(train\_X)

theta = np.ones((n, 1))

maxIteration = 300

c = 0.5

learning\_rate = 0.01

## 9.选择的loss函数及其导数:

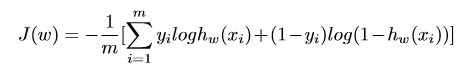
**逻辑回归**

定义逻辑函数



Label为{0,1}

Loss函数为



其导数为

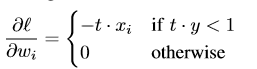


**线性分类**

定义



Hinge loss函数



则loss函数为



## 10.实验结果和曲线图:

### 逻辑回归

**1) Vanilla SGD**

## 超参数选择：

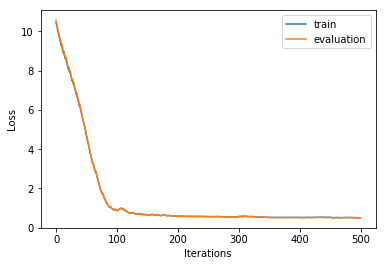
alpha = 0.03

maxIteration = 500

## 预测结果（最佳结果）：

0.495440805639

## loss曲线图：



**2)** **NAG**

## 超参数选择：

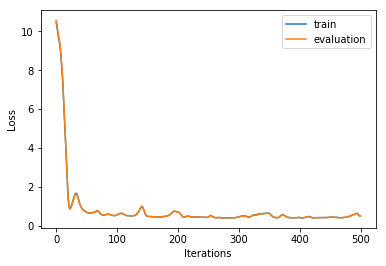
alpha = 0.03

maxIteration = 500

## 预测结果（最佳结果）：

0.424465702885

## loss曲线图：

****

**3)** **RMSProp**

## 超参数选择：

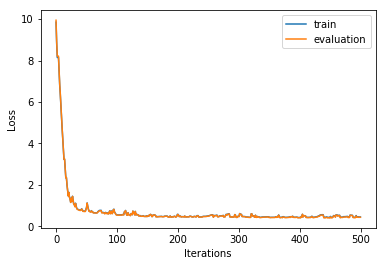
learning\_rate = 0.3

maxIteration = 500

## 预测结果（最佳结果）：

0.392042296275

## loss曲线图：

****

**4)** **AdaDelta**

## 超参数选择：

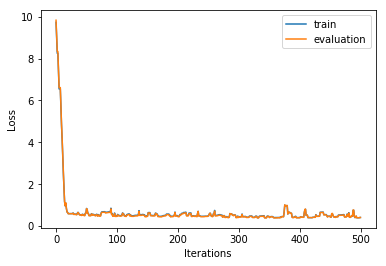
learning\_rate = 2000

maxIteration = 500

## 预测结果（最佳结果）：

0.402268825518

## loss曲线图：

****

**5)** **Adam**

## 超参数选择：

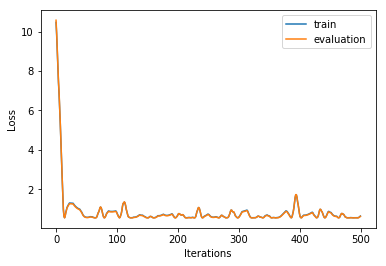
learning\_rate = 0.05

maxIteration = 500

## 预测结果（最佳结果）：

0.548395087657

## loss曲线图：

****

### 线性分类

**1) Vanilla SGD**

## 超参数选择：

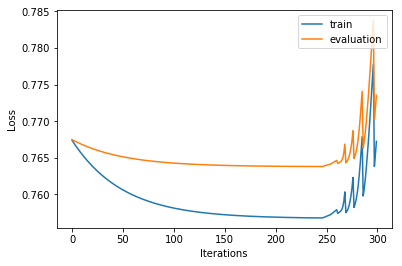
learning\_rate = 0.01

maxIteration = 300

## 预测结果（最佳结果）：

0.763805192026

## loss曲线图：

****

**2)** **NAG**

## 超参数选择：

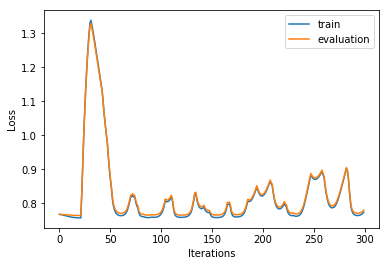
learning\_rate = 0.01

maxIteration = 300

## 预测结果（最佳结果）：

0.763822922757

## loss曲线图：

****

**3)** **RMSProp**

## 超参数选择：

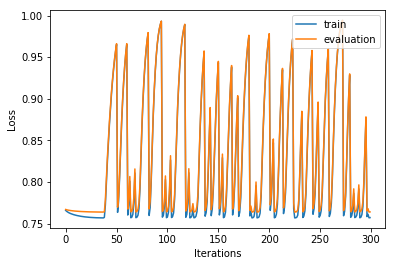
learning\_rate = 0.3

maxIteration = 300

## 预测结果（最佳结果）：

0.763797625979

## loss曲线图：

****

**4)** **AdaDelta**

## 超参数选择：

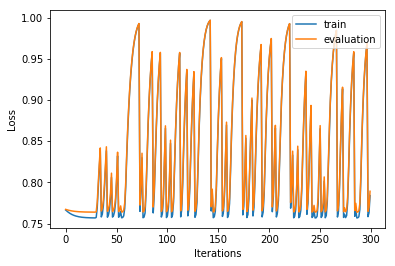
learning\_rate = 2000

maxIteration = 300

## 预测结果（最佳结果）：

0.763807229843

## loss曲线图：

****

**5)** **Adam**

## 超参数选择：

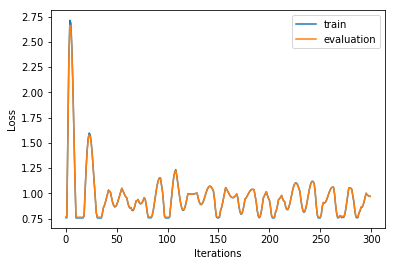
learning\_rate = 0.05

maxIteration = 300

## 预测结果（最佳结果）：

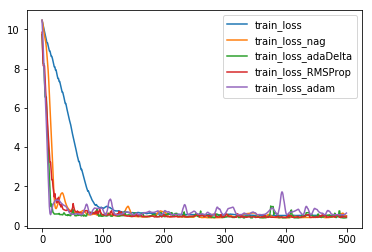
0.763805383034

## loss曲线图：

****

## 11.实验结果分析:

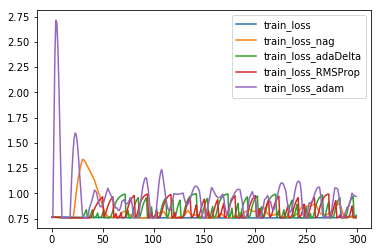
逻辑回归：



对比分析五种不同的梯度下降可以得出以下结论：

1. 就该问题，在当前数据集上，train\_loss下降较为缓慢，而其他几种方式都下降得比较快
2. RMSProp方法具有最好的效果，而Adam方法结果较差。

线性分类：



对比分析五种不同的梯度下降可以得出以下结论：

1) 就该问题，在当前数据集上，train\_loss\_adam的波动较大，较不稳定快

2) RMSProp方法具有最好的效果，而Adam方法结果较差。

## 12.对比逻辑回归和线性分类的异同点：

相同：

逻辑回归和线性分类都可以总结成为一种流程

即

1.选择一个合适的模型

2.选合适的loss函数

3.通过各种方法（这里是梯度下降的方法）求出最好的模型参数

差异：

逻辑回归致力于解决连续性问题，而线性分类致力于解决离散性问题。他们的输出具有较大的差异。

## 13.实验总结：

在本次实验中，我使用python实现逻辑回归、线性分类和随机梯度下降算法，这加深了对逻辑回归、线性分类和随机梯度下降算法的理解，有了更深的体会。

此外，还实现了多种不同的梯度下降方式，并且尝试对比他们的优势和劣势，对这几种方法有了更加深入的认识。