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　蔡兴阳

学　　号　201530611074

邮　　箱　779762069@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 12 月 2 日

3. 报告人:

蔡兴阳

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**。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

```
def NAG(theta):

    gama = 0.9

    vt = 0

    for i in range(0, maxIteration):

        gradient = cal_gradient_sgd(theta - gama*vt)

        vt = gama*vt + alpha * gradient

        theta = theta - vt



        train_loss_n.append(cal_loss(x_train,y_train,theta))

        test_loss_n.append(cal_loss(x_test,y_test,theta))

        train_accr.append(cal_accur(x_train,y_train,theta))

        test_accr.append(cal_accur(x_test,y_test,theta))
```

```
NAG(theta)


def RMSProp(theta):
    gama = 0.9
    vt = 0
    Egt = 0
    e=0.00000001
    learning_rate = 0.3
    for i in range(0, maxIteration):
        gradient = cal_gradient_sgd(theta - gama*vt)
        Egt = gama * Egt + ((1-gama)*(gradient**2)).sum()
        theta = theta - learning_rate*gradient/math.sqrt(Egt + e)

        train_loss_r.append(cal_loss(x_train,y_train,theta))
        test_loss_r.append(cal_loss(x_test,y_test,theta))
        train_accr.append(cal_accur(x_train,y_train,theta))
```

```
        test_accr.append(cal_accur(x_test,y_test,theta))

RMSProp(theta)
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 = cal_gradient_sgd(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_aDe.append(cal_loss(x_train,y_train,theta))

        test_loss_aDe.append(cal_loss(x_test,y_test,theta))

        train_accr.append(cal_accur(x_train,y_train,theta))

        test_accr.append(cal_accur(x_test,y_test,theta))


adaDelta(theta)


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 = cal_gradient_sgd(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_ad.append(cal_loss(x_train,y_train,theta))
            test_loss_ad.append(cal_loss(x_test,y_test,theta))
            train_accr.append(cal_accur(x_train,y_train,theta))
            test_accr.append(cal_accur(x_test,y_test,theta))

    adam(theta)
```

线性分类：

```
def NAG(w):
    vt = 0
    gama = 0.9

    for i in range(maxIteration):
        gradient = cal_stochastic_gradient(w -gama*vt)
        vt = gama*vt + learning_rate * gradient
        w = w - vt

        train_loss_n.append(cal_hinge_loss(w,x_train,y_train))
        test_loss_n.append( cal_hinge_loss(w,x_test,y_test))
        train_accr.append(cal_accur(x_train,y_train,w))
        test_accr.append(cal_accur(x_test,y_test,w))

NAG(theta)
```

```
def RMSProp(w):
    gama = 0.9
    vt = 0
    Egt = 0
    e=0.00000001
    learning_rate = 0.3

    for i in range(0, maxIteration):
        gradient = cal_stochastic_gradient(w - gama*vt)
        Egt = gama * Egt + ((1-gama)*(gradient**2)).sum()
        w -= learning_rate*gradient/math.sqrt(Egt + e)

        train_loss_r.append(cal_hinge_loss(w,x_train,y_train))
        test_loss_r.append( cal_hinge_loss(w,x_test,y_test))
        train_accr.append(cal_accur(x_train,y_train,w))
        test_accr.append(cal_accur(x_test,y_test,w))
```

```
RMSProp(theta)

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 = cal_stochastic_gradient(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_aDe.append(cal_hinge_loss(w,x_train,y_train))

            test_loss_aDe.append( cal_hinge_loss(w,x_test,y_test))

            train_accr.append(cal_accur(x_train,y_train,w))

            test_accr.append(cal_accur(x_test,y_test,w))

adaDelta(theta)



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 = cal_stochastic_gradient(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_ad.append(cal_hinge_loss(w,x_train,y_train))

        test_loss_ad.append( cal_hinge_loss(w,x_test,y_test))

        train_accr.append(cal_accur(x_train,y_train,w))

        test_accr.append(cal_accur(x_test,y_test,w))


adam(theta)
```

8. 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 1 初始化。

9. 选择的 loss 函数及其导数:

逻辑回归：

The loss function is:

$$J(w) = -\frac{1}{m}[\sum_{i=1}^{m} y_i log h_w(x_i) + (1-y_i) log(1-h_w(x_i))]$$

The derivation of loss function is:

$$\frac{\partial J(w)}{\partial w} = -y\frac{1}{h_w(x)}\frac{\partial h_w(x)}{\partial w} + (1-y)\frac{1}{1-h_w(x)}\frac{\partial h_w(x)}{\partial w}$$

线性分类：

Loss Function:

$$\mathcal{L}(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} a_i(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$

$$\nabla f(\beta) = \begin{cases} \mathbf{w}^\top - C\mathbf{y}^\top\mathbf{X} & 1 - y_i(\mathbf{w}^\top x_i + b) >= 0 \\ \mathbf{w}^\top & 1 - y_i(\mathbf{w}^\top x_i + b) < 0 \end{cases}$$

10.实验结果和曲线图:

超参数选择：

逻辑回归：

NAG：

gama = 0.9

vt = 0

alpha = 0.005

maxIteration = 1000

RMSProp:

gama = 0.9

vt = 0

Egt = 0

e=0.00000001

learning_rate = 0.3

AdaDelta：

rho = 0.9

Egt=0

Edt = 0

e=0.00000001

delta = 0

learning_rate = 2000

Adam：

t = 0

m = 0

v = 0

b1 = 0.9

b2 = 0.995

learning_rate = 0.05

线性分类：

NAG：

gama = 0.9

vt = 0

alpha = 0.005

maxIteration = 500

RMSProp:

gama = 0.9

vt = 0

Egt = 0

e=0.00000001

learning_rate = 0.3

AdaDelta：

rho = 0.9

Egt=0

Edt = 0

e=0.00000001

delta = 0

learning_rate = 2000

Adam：

t = 0

m = 0

v = 0

b1 = 0.9

b2 = 0.995

learning_rate = 0.05

预测结果（最佳结果）：

逻辑回归：

NAG：0.827

RMSProp：0.808

AdaDelta：0.806

Adam：0.759

线性回归：

NAG：0.776

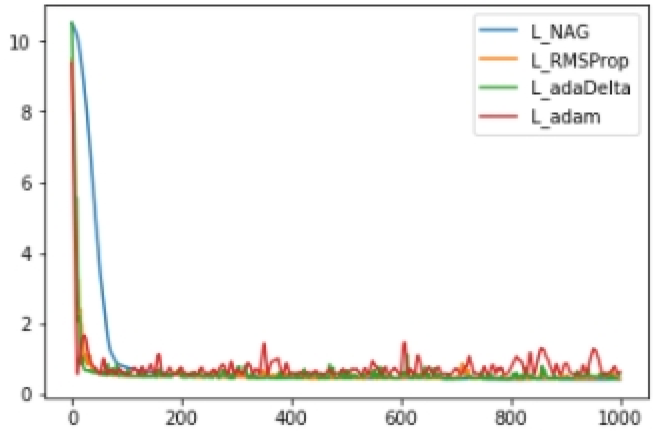
RMSProp：0.768

AdaDelta：0.789

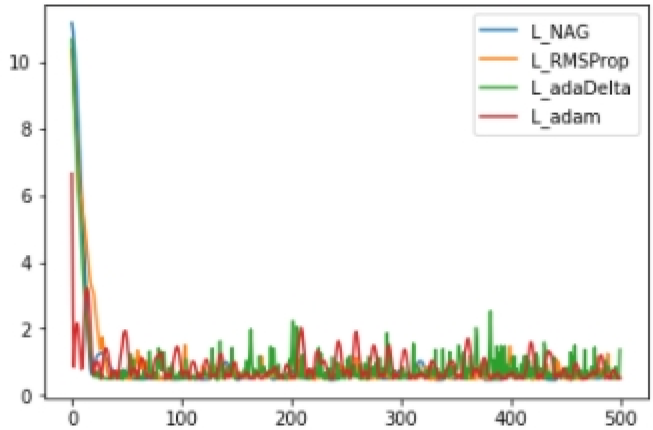
Adam：0.786

loss 曲线图：

逻辑回归：



线性分类：



11.实验结果分析:

实验预测精度都很高，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。

12.对比逻辑回归和线性分类的异同点：

两种方法都是常见的分类算法,从目标函数来看,区别在于逻辑回归采用的是 logistical loss,svm 采用的是 hinge loss.这两个损失函数的目的都是增加对分类影响较大的数据点的权重,减少与分类关系较小的数据点的权重.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.所以在很多实验中,两种算法的结果是很接近的.

13.实验总结：

通过本次实验，我了解到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。