華南理工大學

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Shoukai Xu and Yaofu Chen

Supervisor:
Mingkui Tan or Qingyao Wu

Student ID：
201530611111 and 201530600000

Grade:
Undergraduate or Graduate

December 9, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract—

I. INTRODUCTION

逻辑回归、线性分类与随机梯度下降

1. 对比理解梯度下降和随机梯度下降的区别和联系

2. 对比理解逻辑回归和线性分类的区别和联系

3. 进一步理解 svm 的原理并在较大数据上实践

II. METHODS AND THEORY

逻辑回归：全零初始化，随机梯度下降

Loss 函数：

$$L(w,b)=f_{w,b}(x^1)f_{w,b}(x^2)\left(1-f_{w,b}(x^3)\right)\cdots f_{w,b}(x^N)$$

线性分类：全零初始化，随机梯度下降

Loss 函数：

$$L(f)=\sum_{n}\delta(f(x^n)\neq\hat{y}^n)$$

III. EXPERIMENT

6. 实验步骤:

逻辑回归和随机梯度下降

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 ppt

4. 求得部份样品对 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的优化方法的 Loss

7. 重复步骤 4-6 若干次，画出 Loss 随迭代次数变化图

代码内容：

```python
def NAG(train_x,  train_y, text_x, text_y, gamma =0.9, threshold =0.5, rate =0.1, size=64, epoch=500):
    gradients = np.zeros((train_x.shape[1],1))
    weights = np.random.rand(train_x.shape[1],1)
    momentum = np.zeros((train_x.shape[1],1))

    epoch_set = []
    l_nag = []
    a_nag = []


    for k in range(epoch+1):
        batch_step = random.randint(0,train_x.shape[0]-size-1)
        for i in range(batch_step,batch_step+size):
            gradients = gradients+((sigmoid(np.dot(train_x[i],(weights-gamma*momentum)))-train_y[i])*train_x[i]).reshape((train_x.shape[1],1))
        gradients = gradients/size
        momentum = gamma*momentum+rate*gradients
        weights = weights - momentum
        if(k>0)or(k%100==0):
            epoch_set.append(k)
            loss = 0
            res = 0
            for j in range(test_x.shape[0]):
                h = sigmoid(np.dot(test_x[j],(weights-gamma*momentum)))
                loss = loss+(test_y[j]*np.log(h)+(1-test_y[j])*np.log(1-h))
                if sign(h,threshold) == test_y[j]:
                    res = res+1
            accuracy = res/test_x.shape[0]
            loss = -loss/test_x.shape[0]
```

```python
            a_nag.append(accuracy)
            l_nag.append(loss)
    return a_nag,l_nag,epoch_set
```

```python
def RMSProp(train_x,train_y,test_x,test_y,gamma =0.9,epsilon=1e-10,threshold=0.5,rate=0.1,size=64,epoch=500):
    gradients = np.zeros((train_x.shape[1],1))
    weights = np.random.rand(train_x.shape[1],1)
    g = np.zeros((train_x.shape[1],1))
    epoch_set =[]
    l_rmsp = []
    a_rmsp =[]
    for k in range(epoch+1):
        batch_step = random.randint(0,train_x.shape[0]-size-1)
        for i in range(batch_step,batch_step+size):
            gradients = gradients+((sigmoid(np.dot(train_x[i],weights))-train_y[i])*train_x[i]).reshape((train_x.shape[1],1))
        gradients = gradients/size
        g = gamma*g+np.multiply((1-gamma)*gradients,gradients)
        weights = weights - np.multiply(rate/(np.sqrt(g+epsilon)),gradients)
        if(k>0)or(k%100==0):
            epoch_set.append(k)
            loss = 0
            res = 0
            for j in range(test_x.shape[0]):
                h = sigmoid(np.dot(test_x[j],weights))
                loss = loss+(test_y[j]*np.log(h)+(1-test_y[j])*np.log(1-h))
                if sign(h,threshold) == test_y[j]:
                    res = res+1
            accuracy = res/test_x.shape[0]
            loss = -loss/test_x.shape[0]
            a_rmsp.append(accuracy)
            l_rmsp.append(loss)
    return a_rmsp,l_rmsp
```

```python
def AdaDelta(train_x,train_y,test_x,test_y,gamma=0.9,epsilon=1e-10,threshold=0.5,dx=0.001,size=64,epoch=500):
    gradients = np.zeros((train_x.shape[1],1))
    weights = np.random.rand(train_x.shape[1],1)
    g = np.zeros((train_x.shape[1],1))
    epoch_set =[]
    l_adad = []
    a_adad =[]
    for k in range(epoch+1):
        batch_step = random.randint(0,train_x.shape[0]-size-1)
        for i in range(batch_step,batch_step+size):
            gradients = gradients+((sigmoid(np.dot(train_x[i],weights))-train_y[i])*train_x[i]).reshape((train_x.shape[1],1))
        gradients = gradients/size
        g = gamma*g+np.multiply((1-gamma)*gradients,gradients)
        dw = -np.multiply((np.sqrt(dx+epsilon))/(np.sqrt(g+epsilon)),gradients)
        weights = weights +dw
        dx = gamma*dx+np.multiply((1-gamma)*dw,dw)
        if(k>0)or(k%100==0):
            epoch_set.append(k)
            loss = 0
            res = 0
            for j in range(test_x.shape[0]):
                h = sigmoid(np.dot(test_x[j],weights))
                loss = loss+(test_y[j]*np.log(h)+(1-test_y[j])*np.log(1-h))
                if sign(h,threshold) == test_y[j]:
                    res = res+1
            accuracy = res/test_x.shape[0]
            loss = -loss/test_x.shape[0]
            a_adad.append(accuracy)
```

```python
            l_adad.append(loss)
    return a_adad,l_adad
```

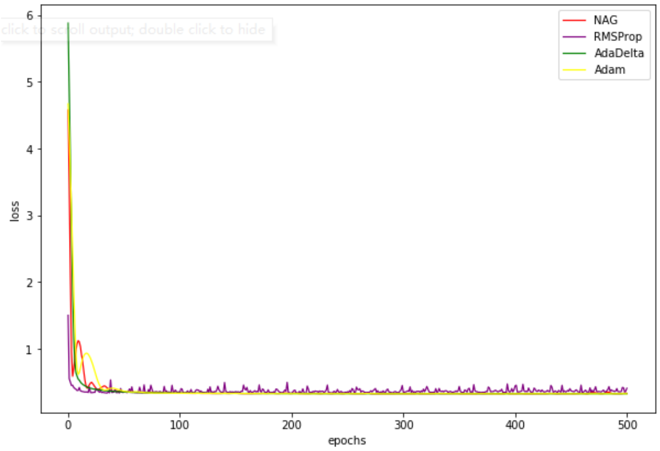
```python
def Adam(train_x,train_y,test_x,test_y,beta=0.9,gamma=0.999,epsilon=1e-8,threshold=0.5,rate=0.01,size=64,epoch=500):
    gradients = np.zeros((train_x.shape[1],1))
    weights = np.random.rand(train_x.shape[1],1)
    g = np.zeros((train_x.shape[1],1))
    moments = np.zeros((train_x.shape[1],1))
    epoch_set =[]
    l_adam = []
    a_adam =[]
    for k in range(epoch+1):
        batch_step = random.randint(0,train_x.shape[0]-size-1)
        for i in range(batch_step,batch_step+size):
            gradients = gradients+((sigmoid(np.dot(train_x[i],weights))-train_y[i])*train_x[i]).reshape((train_x.shape[1],1))
        gradients = gradients/size
        moments = beta*moments+(1-beta)*gradients
        g = gamma*g+np.multiply((1-gamma)*gradients,gradients)
        alpha = rate
        dw = -alpha*moments/(np.sqrt(g+epsilon))
        weights = weights +dw
        if(k>0)or(k%100==0):
            epoch_set.append(k)
            loss = 0
            res = 0
            for j in range(test_x.shape[0]):
                h = sigmoid(np.dot(test_x[j],weights))
                loss = loss+(test_y[j]*np.log(h)+(1-test_y[j])*np.log(1-h))
                if sign(h,threshold) == test_y[j]:
                    res = res+1
            accuracy = res/test_x.shape[0]
```

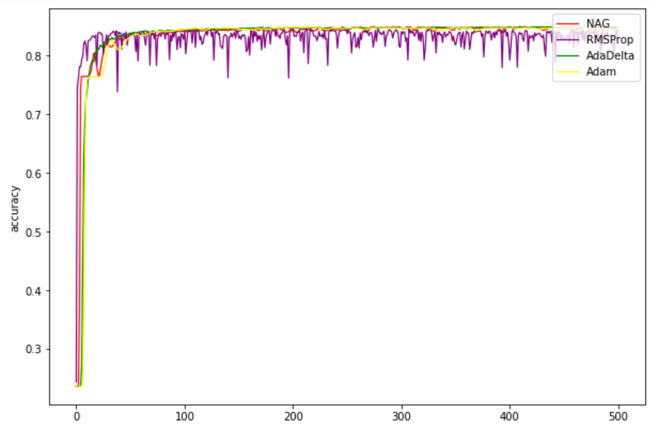
```
            loss = -loss/test_x.shape[0]
            a_adam.append(accuracy)
            l_adam.append(loss)
    return a_adam,l_adam
```

*此处为部分代码，详情见附录 ipynb 文件

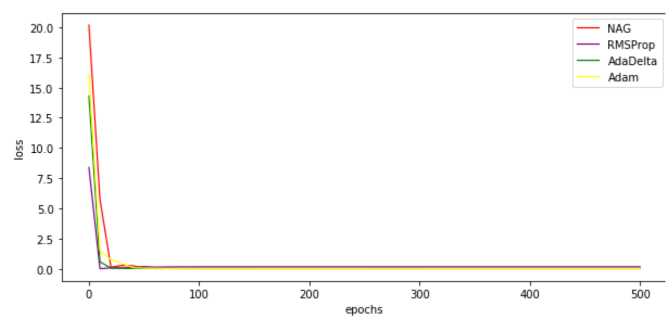
结果：

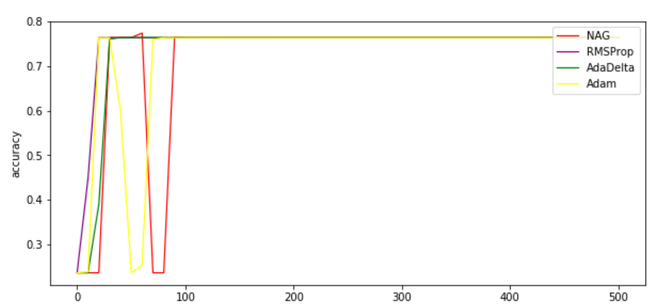
逻辑回归：





线性分类：





IV. CONCLUSION

这次的实验运用了比上一次更大的数据集做训练，使用多种梯度下降方法，从结果可以看出，其各有优劣，同时随着迭代次数的增加，最终得出的结果趋于一致。比起线性分类，逻辑回归的走向前期更为稳妥，但是后期有锯齿产生

对比逻辑回归和线性分类的异同点：

相同点：LR 和 SVM 都是分类算法，都可以处理离散的 label 的数据集

如果不考虑核函数，二者都是线性分类算法

二者都是监督学习算法

都是判别模型

不同点：LR 和 SVM 的 Loss 函数不同，后者只考虑局部边界线附近的点，而前者考虑全局