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　张雄帅

学　　号　201530613641

邮　　箱　1648321269@qq.com

指导教师　吴庆耀

提交日期　2017 年　月　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:张雄帅

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。请自行下载训练集和验证集。

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：

```
#loss function
```

```
def loss(X,Y,W):
    n,m=X.shape
    l=0
    for i in range(n):

l+=math.log((1+exp(-Y[i]*W.T.dot(X[i]))),math.e
)
    return l/n

#graident function
def gradient(X,Y,W):
    n,m=X.shape
    g=0
    for i in range(n):
        g-=Y[i]*X[i]/(1+exp(Y[i]*W.T.dot(X[i])))
    return (g/n).reshape((g.shape[0],1))
```

线性分类：

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

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

```
    return g
```

四种随机梯度下降方法：

NAG：

```
def NAG():
    #initialize parameters with zero
    W=np.zeros((m,1))
    loss_NAG=[]
    #set parameter
    eta=0.1
    gamma=0.1
    batch=100

    v=0
    for epoch in range(1000):
        #calculate gradient g from partial samples
        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=loss(x_test,y_test,W)
        loss_NAG.append(l_test)
    print('NAG finished training')
    plt.plot(loss_NAG,color='blue',label='NAG')
```

RMSProp：

```
def RMSProp():
    #initialize parameters with zero
    W=np.zeros((m,1))
    loss_RMSProp=[]
    #set parameter
    eta=0.01
    gamma=0.9
    eps=0.001
    batch=100
```

```
    G=0
    for epoch in range(1000):
        #calculate gradient g from partial samples
        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+eps)*g
        l_test=loss(x_test,y_test,W)
        loss_RMSProp.append(l_test)
    print('RMSProp finished training')

plt.plot(loss_RMSProp,color='red',label='RMSPro
p')
```

AdaDelta:

```
def AdaDelta():
    #initialize parameters with zero
    W=np.zeros((m,1))
    loss_AdaDelta=[]
    #set parameter
    gamma=0.9
    eps=1e-6
    batch=100

    G=0
    dt=0
    for epoch in range(1000):
        #calculate gradient g from partial samples
        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+eps)/np.sqrt(G+eps)*g
```

```
        W=W+dw
        dt=gamma*dt+(1-gamma)*dw*dw
        l_test=loss(x_test,y_test,W)
        loss_AdaDelta.append(l_test)
    print('AdaDelta finished training')

plt.plot(loss_AdaDelta,color='green',label='Ada
Delta')
```

Adam：

```
def Adam():
    #initialize parameters with zero
    W=np.zeros((m,1))
    loss_Adam=[]
    #set parameter
    beta=0.9
    gamma=0.999
    eta=0.001
    eps=1e-6
    batch=100

    M=0
    G=0
    for epoch in range(1000):
        #calculate gradient g from partial samples
        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-b
eta)
        W=W-alpha*M/np.sqrt(G+eps)
        l_test=loss(x_test,y_test,W)
        loss_Adam.append(l_test)
    print('Adam finished training')
```

```
plt.plot(loss_Adam,color='yellow',label='Adam')
```

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:

逻辑回归:

Loss function：

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^{m}y^{(i)}log\,h_\theta\left(x^{(i)}\right)+\left(1-y^{(i)}\right)log\,\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

导数：

$$\frac{\partial J(\theta)}{\partial\theta}=\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y\right)x^{(i)}$$

线性分类:

loss function:

$$\frac{\|\mathbf{w}\|^2}{2}+C\sum_{i=1}^{N}max(0,1-y_i(\mathbf{w}^\top\mathbf{x}_i+b))$$

导数：

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

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：

逻辑回归：

NAG：

```
eta=0.1
gamma=0.1
```

```
batch=100
```

RMSProp：

```
eta=0.01
gamma=0.9
eps=0.001
batch=100
```

AdaDelta：

```
gamma=0.9
eps=1e-6
batch=100
```

Adam：

```
beta=0.9
gamma=0.999
eta=0.001
eps=1e-6
batch=100
```

线性分类：

NAG：

```
eta=1e-6
gamma=1e-5
batch=100
C=10
```

RMSProp：

```
eta=0.0003
gamma=0.9
eps=1e-6
batch=100
C=10
```

AdaDelta：

```
gamma=0.9
eps=1e-8
batch=100
C=10
```

Adam：

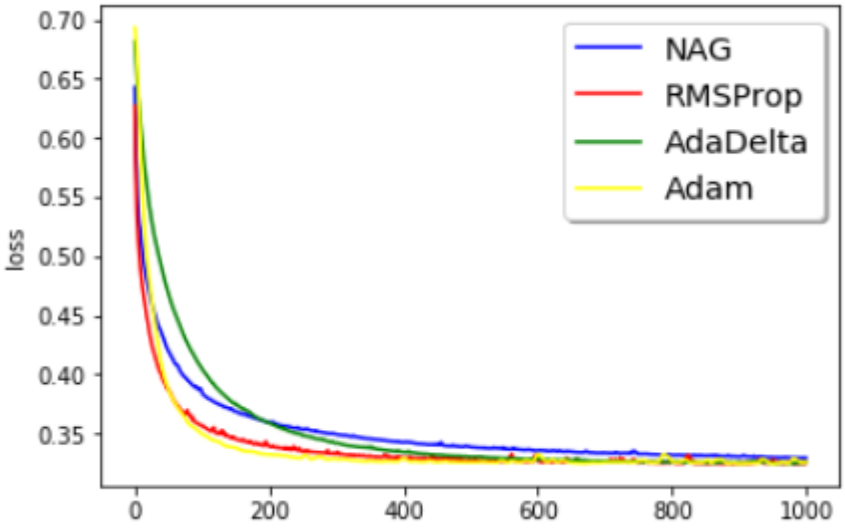
```
beta=0.9
gamma=0.99999
eta=0.000025
eps=1e-8
batch=100
```

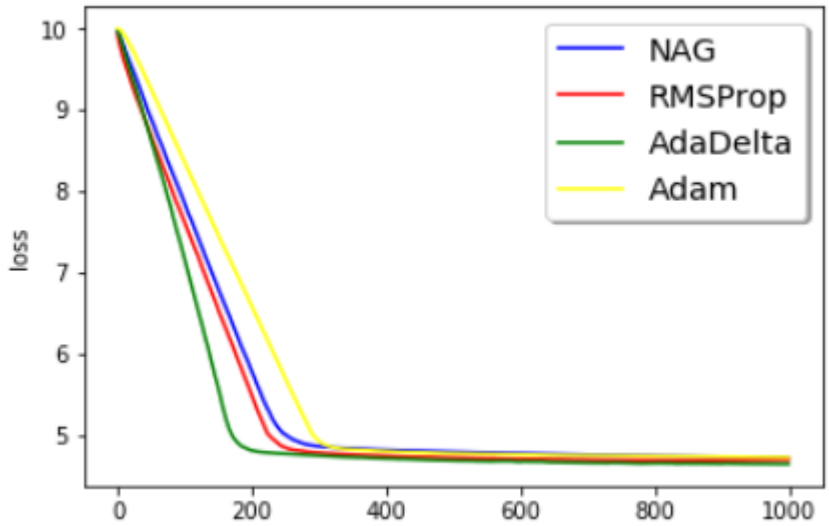
```
C=10
```

预测结果（最佳结果）：

所有随机梯度下降方法的曲线都将平滑地收敛。

loss 曲线图：





11.实验结果分析:

逻辑回归的结果与预期基本符合，在一定的迭代之后收敛。其中 NAG 和 RMSProp 有一定的抖动，而 AdaDelta 和 Adam 则比较光滑。这与随机梯度下降的性质是相对应的。

而线性回归的曲线则相对难看，像是一个转折点，突然就收敛了。这是线性分类所比不上逻辑回归的（这里不知道是不是我的代码问题，但是我检查是没有发现错误的。）。

12.对比逻辑回归和线性分类的异同点：

同：都是求出 W（包含 b）来把数据分成两类的二分类问题。

异：逻辑回归使用 SIGMOD 函数来分类，将数据约束到（0,1）的范围内。线性分类则是求出一条直线将数据分成两个区域。

13.实验总结：

实验使用了多个随机梯度下降方法，是对写算法代码的一个考验。一路跌跌撞撞，但是最后还是勉强完成了，这必然提升了我的写算法的能力。随机梯度下降有着比梯度下降更好的表现，是值得学习的。

这里数万的数据跑起来已经是耗时非常久了，想想如果是跑“大数据”，那得非常久了。这么一想觉得把数据放到 GPU 上去跑是很有必要的。好像 pytorch 就有放数据到 GPU 的 API。

逻辑回归表现得比线性分类要好，这点从图中可以看出，上面的结果分析也有提到。