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　胡永浩

学　　号　201530611630

邮　　箱　1458732467@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 15　日

3. 报告人:胡永浩

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：

6. 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

读取数据后要将稀疏矩阵转化为特征矩阵，并对 X 增加一列全为 1，验证集需要先补零，对 y 所有的-1 用 0 代替

1）*逻辑回归与随机梯度下降*

6. 实验步骤:

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。
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，，和。
7. 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:

```
import numpy as np
import math
from matplotlib import pyplot as plt
from sklearn.datasets import load_svmlight_file


def loadDataSet():
    # 读取数据
    X_train,y_train=load_svmlight_file("D:\machine learning\experiment\lab_2\\a9a.txt")
    X_validation, y_validation = load_svmlight_file("D:\machine learning\experiment\lab_2\\a9a(testing).txt")
    #将稀疏矩阵转化为完整特征矩阵
    X_train = X_train.todense()
    X_validation = X_validation.todense()
    #对X增加一列全为1,验证集需要先补0
    X_train = np.column_stack((X_train, np.ones(y_train.shape[0])))
    X_validation = np.column_stack((X_validation, np.zeros(y_validation.shape[0])))
    X_validation = np.column_stack((X_validation, np.ones(y_validation.shape[0])))
    #对y所有的-1用0代替
    y_train = np.array(list(map((lambda x: 0 if x <= 0 else 1), y_train)))
    y_validation = np.array(list(map((lambda x: 0 if x <= 0 else 1), y_validation)))
    print(X_train.shape,y_train.shape)
    print(X_validation.shape,y_validation.shape)
    return X_train, X_validation, y_train, y_validation

def sigmoid(inX):
    return 1.0/(1+math.exp(-inX))

def loss_function(X_data,y_data,w):
    loss = 0.0
    num = y_data.shape[0]
    for i in range(num):
        y = sigmoid(np.dot(X_data[i][0].getA()[0], w))
        loss += - (y_data[i] * math.log(y) + (1 - y_data[i]) * math.log(1 - y)) / num
    return loss

def stocGradDescent(epoch,X_train,X_validation,y_train,y_validation,opt):
    num = y_train.shape[0]    #样本数量
    batch = int(5000/epoch)
    # 线性模型参数全零初始化
    w = np.zeros(X_train.shape[1])
    v = np.zeros(X_train.shape[1],dtype=np.float)
    G = np.zeros(X_train.shape[1], dtype=np.float)
    dx = np.zeros(X_train.shape[1], dtype=np.float)
    m = np.zeros(X_train.shape[1],dtype=np.float)
    t = 0
    losss = []

    # 迭代次maxCycles次
    for n in range(epoch):
        grad_w = np.zeros(X_train.shape[1])
        loss = loss_function(X_validation, y_validation, w)
        for i in range(batch):
            index = np.random.randint(0,num-1)
            y = sigmoid(np.dot( X_train[index][0].getA()[0], w))
            grad_w += ( y - y_train[index] ) * X_train[index][0].getA()[0] / batch

        #更新模型参数
        if ( opt == 'SGD'):
            updates = SGD(w, grad_w)
        elif ( opt == 'NAG'):
            updates, v = NAG(w, grad_w, v)
        elif ( opt == 'RMSProp'):
            updates, G = RMSProp(w, grad_w, G)
        elif ( opt == 'AdaDelta'):
            updates, G, dx= AdaDelta(w, grad_w, G, dx)
        elif ( opt == 'Adam'):
            updates, G, m, t = Adam(w, grad_w, G, m, t)

        for i in range(len(w)):
            w[i] = updates[i][1]
        losss.append(loss)
```

```
        print("%s_loss = %f" % (opt, loss))
    return losss

def SGD(parameters, gradients, eta=0.1):
    updates = [(parameters[i], parameters[i] - eta * gradients[i])
               for i in range(len(parameters))]
    return updates

def NAG(parameters, gradients, v, eta=0.05, gamma=.9):
    para_num = len(parameters)
    v_prev = v
    v = np.array([gamma * v[i] + eta * gradients[i] for i in range(para_num)])
    updates = [(parameters[i], parameters[i] - ( - gamma * v_prev[i] + ( 1 + gamma ) * v[i]))
                   for i in range(para_num)]

    return updates, v

def RMSProp(parameters, gradients, G, eta=.01, gamma=0.9, epsilon=1e-8):
    para_num = len(parameters)
    G = np.array([gamma * G[i] + (1 - gamma) * (gradients[i])**2 for i in range(para_num)])
    updates = [(parameters[i], parameters[i] - eta * gradients[i] / math.sqrt(G[i] + epsilon))
               for i in range(para_num)]
    return updates, G

def AdaDelta(parameters, gradients, G, dx, gamma=0.95, epsilon=1e-5):
    para_num = len(parameters)
    G = np.array([gamma * G[i] + (1 - gamma) * (gradients[i]**2) for i in range(para_num)])
    dw = [math.sqrt(dx[i] + epsilon) / math.sqrt(G[i] + epsilon) * gradients[i] for i in range(para_num)]
    updates = [(parameters[i], parameters[i] - dw[i])
               for i in range(para_num)]
    dx = np.array([gamma * dx[i] + (1 - gamma) * (dw[i]**2) for i in range(para_num)])
    return updates, G, dx

def Adam(parameters, gradients, G, m, t, eta=0.01, gamma=0.999, beta=0.9, epsilon=1e-8):
    t += 1
    para_num = len(parameters)
```

```
def Adam(parameters, gradients, G, m, t, eta=0.01, gamma=0.999, beta=0.9, epsilon=1e-8):
    t += 1
    para_num = len(parameters)
    m = np.array([beta * m[i] + (1 - beta) * gradients[i] for i in range(para_num)])
    G = np.array([gamma * G[i] + (1 - gamma) * (gradients[i]**2) for i in range(para_num)])
    updates = [(parameters[i], parameters[i] - eta * math.sqrt(1 - gamma**t) / (1 - beta**t) * m[i] / math.sqrt(G[i] + epsilon))
               for i in range(para_num)]
    return updates, G, m, t

def plotLossPerTime(epoch, sgd_losss, nag_losss, rms_losss, adad_losss, adam_losss):
    plt.xlabel('iteration times')
    plt.ylabel('loss')
    plt.title('Logistic Regression & SGD')
    n_cycles = range(1,epoch+1)
    plt.plot(n_cycles, sgd_losss, label = "Loss of SGD", linewidth=2)
    plt.plot(n_cycles, nag_losss, label="Loss of NAG", linewidth=2)
    plt.plot(n_cycles, rms_losss, label="Loss of RMSProp", linewidth=2)
    plt.plot(n_cycles, adad_losss, label="Loss of AdaDelta", linewidth=2)
    plt.plot(n_cycles, adam_losss, label="Loss of Adam", linewidth=2)
    plt.legend(loc=0)
    plt.grid()
    plt.show()

# main
X_train, X_validation, y_train, y_validation = loadDataSet()
epoch = 200
sgd_losss = stocGradDescent(epoch, X_train, X_validation, y_train, y_validation, 'SGD')
nag_losss = stocGradDescent(epoch, X_train, X_validation, y_train, y_validation, 'NAG')
rms_losss = stocGradDescent(epoch, X_train, X_validation, y_train, y_validation, 'RMSProp')
adad_losss = stocGradDescent(epoch, X_train, X_validation, y_train, y_validation, 'AdaDelta')
adam_losss = stocGradDescent(epoch, X_train, X_validation, y_train, y_validation, 'Adam')
plotLossPerTime(epoch, sgd_losss, nag_losss, rms_losss, adad_losss, adam_losss)
```

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:

10.实验结果和曲线图:

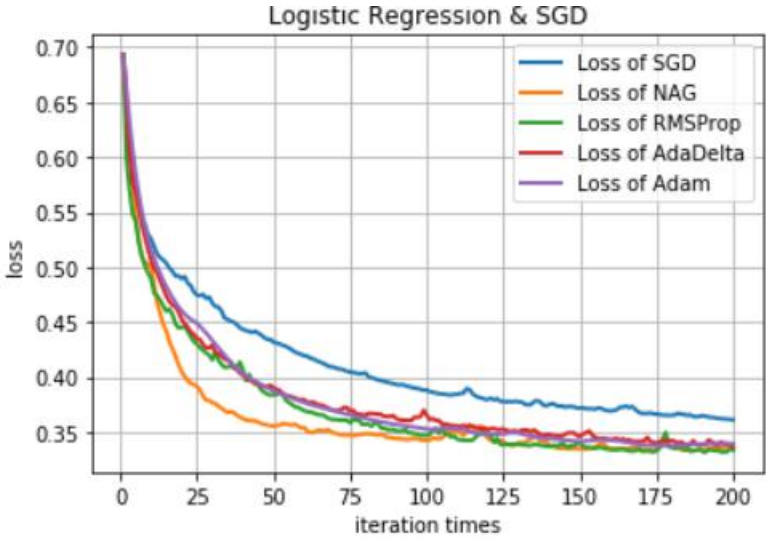
超参数选择：

NAG(parameters, gradients, v, eta=0.05, gamma=.9

RMSProp(parameters, gradients, G, eta=.01, gamma=0.9, epsilon=1e-8

AdaDelta(parameters, gradients, G, dx, gamma=0.95, epsilon=1e-5

Adam(parameters, gradients, G, m, t, eta=0.01, gamma=0.999, beta=0.9, epsilon=1e-8



2）线性分类与随机梯度下降

6.实验步骤:

- 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:

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import load_svmlight_file
import random


def loadDataSet():
    # 读取数据
    train_data = load_svmlight_file('D:\machine learning\experiment\lab_2\\a9a.txt')
    X_train = np.reshape(train_data[0].todense().data, (train_data[0].shape[0], train_data[0].shape[1]))
    y_train = np.reshape(train_data[1].data, (train_data[1].shape[0], 1))

    validation_data = load_svmlight_file('D:\machine learning\experiment\lab_2\\a9a(testing).txt')
    zeros = np.zeros(validation_data[0].shape[0])
    X_validation = np.reshape(validation_data[0].todense().data, (validation_data[0].shape[0], validation_data[0].shape[1]))
    X_validation = np.column_stack((X_validation, zeros))
    y_validation = np.reshape(validation_data[1].data, (validation_data[1].shape[0], 1))

    print(X_train.shape,y_train.shape)
    print(X_validation.shape,y_validation.shape)
    return X_train, y_train, X_validation, y_validation


def loss_function(X_data, y_data, w, C):
    hinge_loss = 0
    losses = (1 - y_data * np.dot(X_data, w))
    for one_loss in losses:
        hinge_loss += C * max(0, one_loss)
    return hinge_loss / len(X_data)


def compute_gradient(X_data, y_data, w, C):
    gradient = np.zeros((1, X_data.shape[1]))
    losses = (1 - y_data * np.dot(X_data, w))
    for i, loss in enumerate(losses):
        if loss <= 0:
```

```
    losses = (1 - y_data * np.dot(X_data, w))
    for i, loss in enumerate(losses):
        if loss <= 0:
            gradient += w.T
        else:
            gradient += w.T - C * y_data[i] * X_data[i]
    return gradient / len(X_data)


#NAG
def NAG(w, gradient, v, mu=0.9, eta=0.0003):
    v_prev = v
    v = mu * v + eta * gradient
    w += ( mu * v_prev - (1 + mu) * v).reshape((123, 1))
    return w, v


#RMSProp
def RMSProp(w, gradient, cache, decay_rate=0.9, eps=1e-8, eta=0.0005):
    cache = decay_rate * cache + (1 - decay_rate) * (gradient ** 2)
    w += (- eta * gradient / (np.sqrt(cache + eps))).reshape((123, 1))
    return w, cache


#AdaDelta
def AdaDelta(w, gradient, cache, delta_t, r=0.95, eps=1e-8):
    cache = r * cache + (1 - r) * (gradient ** 2)
    delta_theta = - np.sqrt(delta_t + eps) / np.sqrt(cache + eps) * gradient
    w = w + delta_theta.reshape((123, 1))
    delta_t = r * delta_t + (1 - r) * (delta_theta ** 2)
    return w, cache, delta_t


#Adam
def Adam(w, gradient, m, i, t, beta1=0.9, beta2=0.999, eta=0.0005, eps=1e-8):
    m = beta1 * m + (1 - beta1) * gradient
    mt = m / (1 - beta1 ** i)
    t = beta2 * t + (1 - beta2) * (gradient ** 2)
```

```
    vt = t / (1 - beta2 ** i)
    w += (-eta * mt / (np.sqrt(vt + eps))).reshape((123, 1))
    return w, m, t

def plotLossPerTime(epoch, nag_losses, rms_losses, adad_losses, adam_losses):
    plt.xlabel('iteration times')
    plt.ylabel('loss')
    plt.title('Linear Classification & SGD')
    n_cycles = range(1,epoch+1)
    plt.plot(n_cycles, nag_losses, label="Loss of NAG", linewidth=3)
    plt.plot(n_cycles, rms_losses, label="Loss of RMSProp", linewidth=3)
    plt.plot(n_cycles, adad_losses, label="Loss of AdaDelta", linewidth=3)
    plt.plot(n_cycles, adam_losses, label="Loss of Adam", linewidth=3)
    plt.legend(loc=0)
    plt.grid()
    plt.show()

X_train, y_train, X_validation, y_validation = loadDataSet()
nag_w = np.zeros((X_train.shape[1], 1))
rms_w = np.zeros((X_train.shape[1], 1))
adad_w = np.zeros((X_train.shape[1], 1))
adam_w = np.zeros((X_train.shape[1], 1))

v = np.zeros(X_train.shape[1])
cache = np.zeros(X_train.shape[1])
adad_cache = np.zeros(X_train.shape[1])
delta_t = np.zeros(X_train.shape[1])
m = np.zeros(X_train.shape[1])
t = np.zeros(X_train.shape[1])

batch_size = 5000
epoch = 200
C = 1
nag_losses = []
rms_losses = []
adad_losses = []
```

```
adad_losses = []
adam_losses = []

for i in range(epoch):
    index = list(range(len(X_train)))
    random.shuffle(index)

    # NAG
    nag_gradient = compute_gradient(X_train[index][:batch_size], y_train[index][:batch_size], nag_w, C)
    nag_w, v = NAG(nag_w, nag_gradient, v)
    nag_loss = loss_function(X_validation, y_validation, nag_w, C)
    nag_losses.append(nag_loss)
    print("NAG_loss = %f" % nag_loss)

    # RMSProp
    rms_gradient = compute_gradient(X_train[index][:batch_size], y_train[index][:batch_size], rms_w, C)
    rms_w, cache = RMSProp(rms_w, rms_gradient, cache)
    rms_loss = loss_function(X_validation, y_validation, rms_w, C)
    rms_losses.append(rms_loss)
    print("RMSProp_loss = %f" % rms_loss)

    # AdaDelta
    adad_gradient = compute_gradient(X_train[index][:batch_size], y_train[index][:batch_size], adad_w, C)
    adad_w, adad_cache, delta_t = AdaDelta(adad_w, adad_gradient, adad_cache, delta_t)
    adad_loss = loss_function(X_validation, y_validation, adad_w, C)
    adad_losses.append(adad_loss)
    print("AdaDelta_loss = %f" % adad_loss)

    # Adam
    adam_gradient = compute_gradient(X_train[index][:batch_size], y_train[index][:batch_size], adam_w, C)
    adam_w, m, t = Adam(adam_w, adam_gradient, m, i+1, t)
    adam_loss = loss_function(X_validation, y_validation, adam_w, C)
    adam_losses.append(adam_loss)
    print("Adam_loss = %f" % adam_loss)

    print()
```

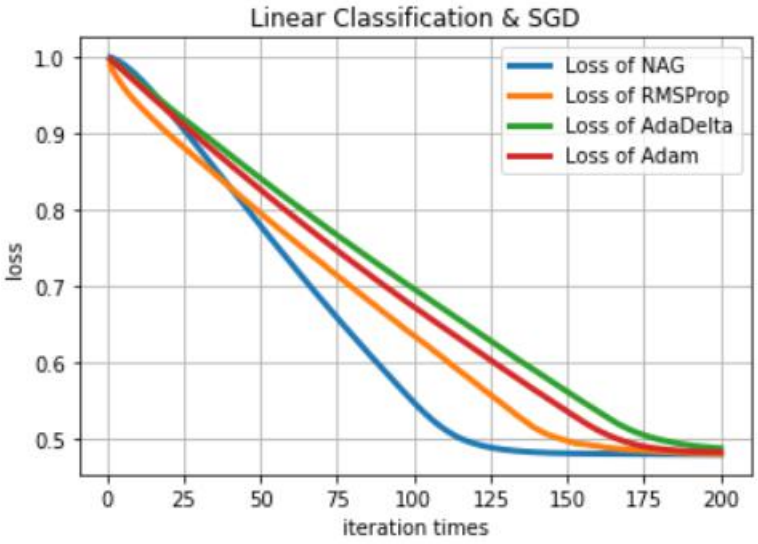
8. 模型参数的初始化方法:正态分布初始化

9.选择的 loss 函数及其导数:

10.实验结果和曲线图:

超参数选择：NAG(w, gradient, v, mu=0.9, eta=0.0003

RMSProp(parameters, gradients, G, eta=.01, gamma=0.9, epsilon=1e-8
AdaDelta(w, gradient, cache, delta_t, r=0.95, eps=1e-8
Adam(w, gradient, m, i, t, beta1=0.9, beta2=0.999, eta=0.0005, eps=1e-8



11.实验结果分析:

以 NAG 以外的其他方法更新参数收敛没有 NAG 快，但都成功收敛了

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,如 l1,l2 等等.所以在很多实验中,两种算法的结果是很接近的.

但是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,虽然一般使用的人不太会去关注.还有很重要的一点,SVM 转化为对偶问题后,分类只需要计算与少数几个支持向量的距离,这个在进行复杂核函数计算时优势很明显,能够大大简化模型和计算

svm 更多的属于非参数模型,而 logistic regression 是参数模型,本质不同.其区别就可以参考参数模型和非参模型的区别就好了.

logic 能做的 svm 能做,但可能在准确率上有问题,svm 能做的 logic 有的做不了

13.实验总结：

进一步熟悉调参过程并且了解到了逻辑回归与线性分类各自的优缺点