華南理工大學

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　黄班班

学　　号　201530611647

邮　　箱　863237407@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 14 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：黄班班

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：

本次实验代码及画图均在jupyter上完成。

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：

1.逻辑回归与随机梯度下降

```
import numpy
import random
import jupyter
```

```
import math
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from matplotlib import pyplot

def f_loss(x, y, w, C, random_i):
    loss = 0
    n = len(random_i)
    for m in range(n):
        loss += math.log(1+math.exp(-y[m]*(x[m,:].dot(w.T))))
    loss = loss/n + C/2 * w.dot(w.T)
    return loss[0,0]

def f_gradient(x, y, w, C):
    gradient = (-y / (1+math.exp(y*x.dot(w.T))))*x + C*w
    return gradient

def NAG_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    vt = numpy.zeros(w.shape)
    lossvalue = []
    testloss = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient  =  f_gradient(x[random_index[i],:],  y[random_index[i]],
w-gamma*vt, C)
        vt = gamma*vt - lr*gradient
        w += vt
        loss = f_loss(x, y, w, C, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, lossvalue, testloss

def RMSProp_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    lossvalue = []
    testloss = []
```

```
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = f_gradient(x[random_index[i],:], y[random_index[i]], w, C)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        w -= lr * gradient / math.sqrt(Gt+1e-8)
        loss = f_loss(x, y, w, C, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, lossvalue, testloss

def AdaDelta_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    variable_t = 0
    lossvalue = []
    testloss = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = f_gradient(x[random_index[i],:], y[random_index[i]], w, C)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        variable_w = - math.sqrt(variable_t + 1e-8) * gradient / math.sqrt(Gt
+ 1e-8)
        w += variable_w
        variable_t              =              gamma*variable_t              +
(1-gamma)*variable_w.dot(variable_w.T)
        loss = f_loss(x, y, w, C, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, lossvalue, testloss
```

```
def Adam_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    moment = numpy.zeros((1, x.shape[1]))
    B = 0.9
    lossvalue = []
    testloss = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = f_gradient(x[random_index[i],:], y[random_index[i]], w, C)
        moment = B*moment + (1-B)*gradient
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        a = lr * math.sqrt(1 - pow(gamma, iteration)) / (1-pow(B, iteration))
        w -= a * moment / math.sqrt(Gt + 1e-8)
        loss = f_loss(x, y, w, C, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, lossvalue, testloss

x, y_train = load_svmlight_file("F:\\machinelearning\\a9a.txt")
x_train = x.toarray()
x, y_test = load_svmlight_file("F:\\machinelearning\\a9at.txt")
x_test = x.toarray()

X_train = numpy.hstack([x_train, numpy.ones((x_train.shape[0], 1))])
X_test = numpy.hstack([x_test, numpy.zeros((x_test.shape[0], 1))])
X_test = numpy.hstack([X_test, numpy.ones((x_test.shape[0], 1))])

iteration = 1000

# NAG
NAG_w = numpy.zeros((1, X_train.shape[1]))
NAG_w, NAG_loss, NAGtest_loss = NAG_train(X_train, y_train, X_test, y_test,
NAG_w, 0.3, 0.001, 0.9, 0.001, iteration)

# RMSProp
```

```
    RMS_w = numpy.zeros((1, X_train.shape[1]))
    RMS_w, RMS_loss, RMStest_loss = RMSProp_train(X_train, y_train, X_test, y_test,
    RMS_w, 0.3, 0.001, 0.9, 0.001, iteration)

    # AdaDelta
    AdaDelta_w = numpy.zeros((1, X_train.shape[1]))
    AdaDelta_w, AdaDelta_loss, AdaDeltatest_loss = AdaDelta_train(X_train,
    y_train, X_test, y_test, AdaDelta_w, 0.3, 0.001, 0.9, 0.001, iteration)

    # Adam
    Adam_w = numpy.zeros((1, X_train.shape[1]))
    Adam_w, Adam_loss, Adamtest_loss = Adam_train(X_train, y_train, X_test, y_test,
    Adam_w, 0.3, 0.001, 0.9, 0.001, iteration)

    pyplot.plot(NAGtest_loss, label = 'NAG_loss')

    pyplot.plot(RMStest_loss, label = 'RMSProp_loss')

    pyplot.plot(AdaDeltatest_loss, label = 'AdaDelta_loss')

    pyplot.plot(Adamtest_loss, label = 'Adam_loss')
    pyplot.legend(loc='upper right')
    pyplot.ylabel('loss')
    pyplot.xlabel('iteration')
    pyplot.title('model')
    pyplot.show()
```

2.线性分类与随机梯度下降

```
import numpy
import random
import jupyter
import math
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from matplotlib import pyplot
def f_gradient(x, y, w):
    gradient = x * (y - x.dot(w.T))
    return gradient

def f_loss(x, y, w, random_i):
    loss = 0
    a = len(random_i)
    for m in range(a):
        loss += 0.5 * ((y[random_i[m]] - x[random_i[m],:].dot(w.T)) ** 2)
```

```
    return loss/a

def NAG_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    vt = numpy.zeros(w.shape)
    lossvalue = []
    testloss = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = f_gradient(x[random_index[i],:], y[random_index[i]],
w-gamma*vt)
        vt = gamma*vt - lr*gradient
        w -= vt
        loss = f_loss(x, y, w, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
            break
    return w, lossvalue, testloss

def RMSProp_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    lossvalue = []
    testloss = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = f_gradient(x[random_index[i],:], y[random_index[i]], w)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        w += lr * gradient / math.sqrt(Gt+1e-8)
        loss = f_loss(x, y, w, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
```

```
            break
    return w, lossvalue, testloss

def AdaDelta_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    variable_t = 0
    lossvalue = []
    testloss = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = f_gradient(x[random_index[i],:], y[random_index[i]], w)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        variable_w = - math.sqrt(variable_t + 1e-8) * gradient / math.sqrt(Gt + 
1e-8)
        w -= variable_w
        variable_t = gamma*variable_t + (1-gamma)*variable_w.dot(variable_w.T)
        loss = f_loss(x, y, w, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
            break
    return w, lossvalue, testloss

def Adam_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    moment = numpy.zeros((1, x.shape[1]))
    B = 0.9
    lossvalue = []
    testloss = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = f_gradient(x[random_index[i],:], y[random_index[i]], w)
```

```
        moment = B*moment + (1-B)*gradient
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        a = lr * math.sqrt(1 - pow(gamma, iteration)) / (1-pow(B, iteration))
        w += a * moment / math.sqrt(Gt + 1e-8)
        loss = f_loss(x, y, w, random_index)
        lossvalue.append(loss)
        testloss.append(f_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
            break
    return w, lossvalue, testloss

x, y_train = load_svmlight_file("F:\\machinelearning\\a9a.txt")
x_train = x.toarray()
x, y_test = load_svmlight_file("F:\\machinelearning\\a9at.txt")
x_test = x.toarray()

X_train = numpy.hstack([x_train, numpy.ones((x_train.shape[0], 1))])
X_test = numpy.hstack([x_test, numpy.zeros((x_test.shape[0], 1))])
X_test = numpy.hstack([X_test, numpy.ones((x_test.shape[0], 1))])

iteration = 1000
# NAG
NAG_w = numpy.zeros((1, X_train.shape[1]))
NAG_w, NAG_loss, NAGtest_loss = NAG_train(X_train, y_train, X_test, y_test, NAG_w,
0.3, 0.001, 0.9, 0.001, iteration)

# RMSProp
RMS_w = numpy.zeros((1, X_train.shape[1]))
RMS_w, RMS_loss, RMStest_loss = RMSProp_train(X_train, y_train, X_test, y_test,
RMS_w, 0.3, 0.001, 0.9, 0.001, iteration)

# AdaDelta
AdaDelta_w = numpy.zeros((1, X_train.shape[1]))
AdaDelta_w, AdaDelta_loss, AdaDeltatest_loss = AdaDelta_train(X_train, y_train,
X_test, y_test, AdaDelta_w, 0.3, 0.001, 0.9, 0.001, iteration)

#Adam
Adam_w = numpy.zeros((1, X_train.shape[1]))
Adam_w, Adam_loss, Adamtest_loss = Adam_train(X_train, y_train, X_test, y_test,
Adam_w, 0.3, 0.001, 0.9, 0.001, iteration)



pyplot.plot(NAGtest_loss, label = 'NAG_loss')
```

```
pyplot.plot(RMStest_loss, label = 'RMSProp_loss')

pyplot.plot(AdaDeltatest_loss, label = 'AdaDelta_loss')

pyplot.plot(Adamtest_loss, label = 'Adam_loss')
pyplot.legend(loc='upper right')
pyplot.ylabel('loss')
pyplot.xlabel('iteration')
pyplot.title('model')
pyplot.show()
```

8. 模型参数的初始化方法:

都是全零初始化

9. 选择的 loss 函数及其导数:

1. 逻辑回归与随机梯度下降

loss 函数：
$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\log(1+e^{-y_i\cdot\mathbf{w}^\top\mathbf{x}_i})+\frac{\lambda}{2}||\mathbf{w}||_2^2$$

导数：
$$\frac{y_i\mathbf{x}_i}{1+e^{y_i\cdot\mathbf{w}^\top\mathbf{x}_i}} + \lambda w$$

2. 线性分类与随机梯度下降

loss 函数：
$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(y^i-h_\theta(x^i))^2$$

导数：令
$$h(\theta)=\sum_{j=0}^{n}\theta_j x_j$$

导数为：
$$\cdot\frac{1}{m}\sum_{i=1}^{m}(y^i-h_\theta(x^i))x_j^i$$

10. 实验结果和曲线图:

超参数选择：

1.逻辑回归与随机梯度下降

NAG： λ =0.3 learning rate=0.001
gamma = 0.9 threshold=0.001

RMSProp： λ =0.3 learning rate=0.001
gamma = 0.9 threshold=0.001

AdaDelta： λ =0.3 learning rate=0.001
gamma = 0.9 threshold=0.001

Adam： λ =0.3 learning rate=0.001
gamma = 0.9 threshold=0.001

2.线性分类与随机梯度下降

NAG： λ =0.3 learning rate=0.001
gamma = 0.9 threshold=0.001

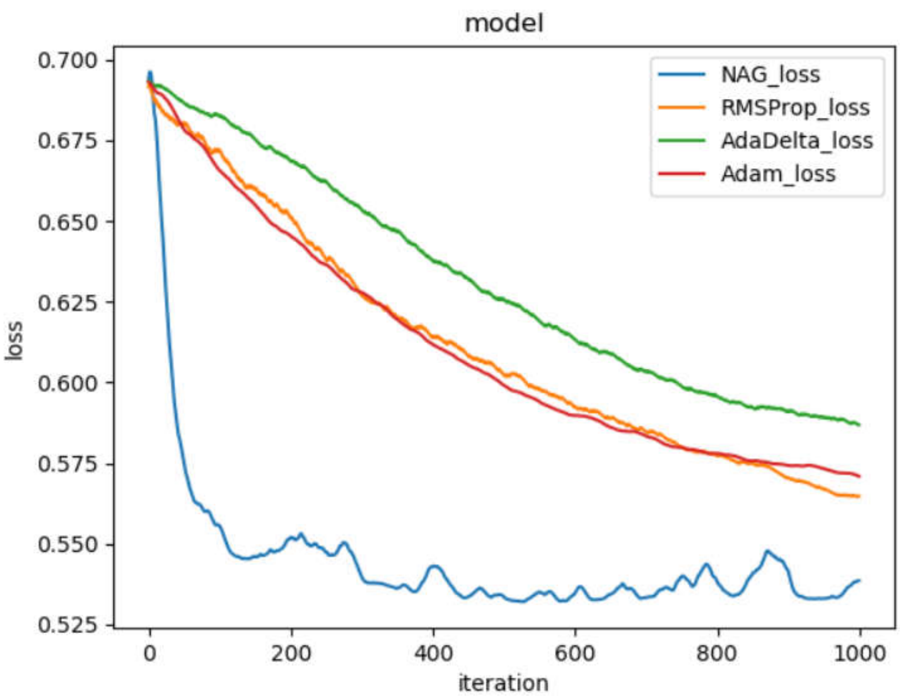
RMSProp： λ =0.3 learning rate=0.001
gamma = 0.9 threshold=0.001

AdaDelta： λ =0.3 learning rate=0.001
gamma = 0.9 threshold=0.001

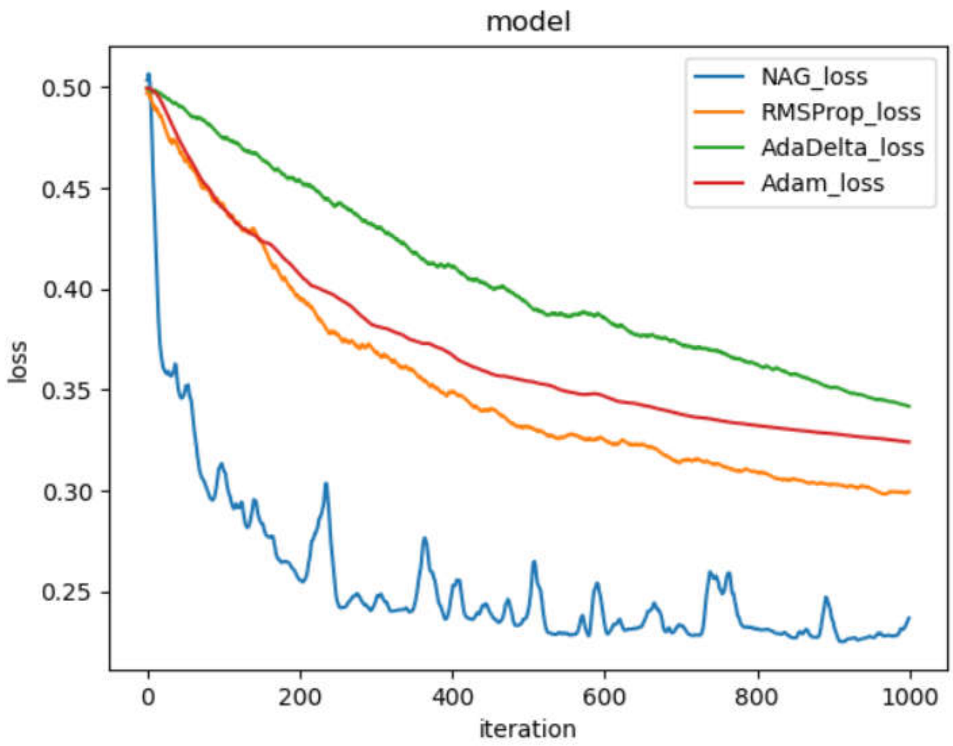
Adam： λ =0.3 learning rate=0.001

预测结果和 loss 曲线图：

1.逻辑回归与随机梯度下降



2.线性分类与随机梯度下降



11. 实验结果分析:

从图中可以看到 NAG 优化算法梯度下降最快，也最不稳定，AdaDelta 最慢但较稳定。
且由于采用的是随机梯度下降，曲线都有明显的起伏。

12. 对比逻辑回归和线性分类的异同点：

同：都能用于分类用途

异：逻辑回归寻是找使模型出现的概率最大的参数集 w 的方法。显然，参数集 w 所确定的模型，其出现概率越大，模型的准确度越高,侧重于预测其概率。

线性分类是用一个线性函数来将样本点分开，着实的将数据划分为不同的一类。

13. 实验总结：

这次试验中，我学到了随机梯度下降的方法来预测具有大规模的数据。并运用了其中的一些优化方法来改善随机下降的不稳定性。其次，实验当中免不了困难重重，调参问题、代码运行时间过长等等，都要我放平心态。本次实验也让我对分类问题有了更深入的了解，对如何解决这类问题有了一些经验。