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　黄景浩

学　　号　201530611739

邮　　箱　394876962@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：黄景浩

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：

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

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 LNAG，LRMSProp，LAdaDelta 和 LAdam。
7. 重复步骤 4-6 若干次，**画出 LNAG，LRMSProp，LAdaDelta 和 LAdam 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。
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 LNAG，LRMSProp，LAdaDelta 和 LAdam。
7. 重复步骤 4-6 若干次，**画出 LNAG，LRMSProp，LAdaDelta 和 LAdam 和随迭代次数的变化图**。

7. 代码内容:

（针对逻辑回归和线性分类分别填写 8-11 内容）

逻辑回归与随机梯度下降

```
import sys
import numpy as np
import math
import random
import matplotlib
import matplotlib.pylab as plt
from matplotlib.patches import Rectangle


plt_color_array = ['r', 'g', 'b', 'y']
plt_dict = dict()

sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))
random.seed(1000)

num_feature = 123

opt_algo_set = [ 'NAG', 'RMSprop','Adadelta', 'Adam']


def train(x, y, opt_algo, num_epoch=30, mini_batch=100, lambda_=0.01):
    num_params = 1 * (num_feature + 1) + 2 * 2
    w = np.matrix(0.005 * np.random.random([num_params, 1]))
    data = np.column_stack([x, y])

    gamma = 0.9
    epsilon = 1e-8

    if opt_algo == 'RMSprop' or opt_algo == 'Adam':
        eta = 0.001
    else:
        eta = 0.05

    v = np.matrix(np.zeros(w.shape))
    m = np.matrix(np.zeros(w.shape))

    # Adam params
    beta1 = 0.9
    beta2 = 0.999
    beta1_exp = 1.0
```

```
beta2_exp = 1.0

# Adagrad params
grad_sum_square = np.matrix(np.zeros(w.shape))

# Adadelta & RMSprop params
grad_expect = np.matrix(np.zeros(w.shape))
delta_expect = np.matrix(np.zeros(w.shape))

first_run = True
for epoch in range(num_epoch):
    np.random.shuffle(data)
    k = 0
    cost_array = list()
    while k < len(data):
        x = data[k: k + mini_batch, 0: -1]
        y = np.matrix(data[k: k + mini_batch, -1], dtype='int32')
        if opt_algo == 'SGD':
            # Stochastic gradient descent
            cost, grad = gradient(x, y, lambda_, w)
            w = w - eta * grad

        elif opt_algo == 'Momentum':
            # Momentum
            cost, grad = gradient(x, y, lambda_, w)
            v = gamma * v + eta * grad
            w = w - v

        elif opt_algo == 'NAG':
            # Nesterov accelerated gradient
            cost, grad = gradient(x, y, lambda_, w - gamma * v)
            v = gamma * v + eta * grad
            w = w - v

        elif opt_algo == 'Adagrad':
            # Adagrad
            cost, grad = gradient(x, y, lambda_, w)
            grad_sum_square += np.square(grad)
            delta = - eta * grad / np.sqrt(grad_sum_square + epsilon)
            w = w + delta

        elif opt_algo == 'Adadelta':
            # Adadelta
            cost, grad = gradient(x, y, lambda_, w)
```

```
                grad_expect = gamma * grad_expect + (1.0 - gamma) * np.square(grad)
                # when first run, use sgd
                if first_run == True:
                    delta = - eta * grad
                else:
                    delta = - np.multiply(np.sqrt(delta_expect + epsilon) /
np.sqrt(grad_expect + epsilon), grad)
                w = w + delta
                delta_expect = gamma * delta_expect + (1.0 - gamma) * np.square(delta)

            elif opt_algo == 'RMSprop':
                # RMSprop
                cost, grad = gradient(x, y, lambda_, w)
                grad_expect = gamma * grad_expect + (1.0 - gamma) * np.square(grad)
                w = w - eta * grad / np.sqrt(grad_expect + epsilon)

            elif opt_algo == 'Adam':
                # Adam
                cost, grad = gradient(x, y, lambda_, w)
                m = beta1 * m + (1.0 - beta1) * grad
                v = beta2 * v + (1.0 - beta2) * np.square(grad)
                beta1_exp *= beta1
                beta2_exp *= beta2
                w = w - eta * (m / (1.0 - beta1_exp)) / (np.sqrt(v / (1.0 - beta2_exp)) +
epsilon)

            k += mini_batch
            cost_array.append(cost)
            if first_run == True: first_run = False

        if not opt_algo in plt_dict:
            plt_dict[opt_algo] = list()
        plt_dict[opt_algo].extend(cost_array)

    return w;


def gradient(x, y, lambda_, w):

    num_sample = len(x)

    w1 = w[0: (num_feature + 1)].reshape(1, num_feature + 1)
    w2 = w[(num_feature + 1):].reshape(2, 2)
    b = np.matrix(np.ones([num_sample, 1]))
```

```
    a1 = np.column_stack([x, b])
    s2 = sigmoid(a1 * w1.T)
    a2 = np.column_stack([s2, b])
    a3 = sigmoid(a2 * w2.T)

    y_one_hot = np.matrix(np.zeros([num_sample, 2]))
    y_one_hot[(np.matrix(range(num_sample)), y.T)] = 1

    cost = (1.0 / num_sample) * (
    - np.multiply(y_one_hot, np.log(a3)) - np.multiply(1.0 - y_one_hot, np.log(1.0 -
a3))).sum()
    cost += (lambda_ / (2.0 * num_sample)) * (np.square(w1[:, 0: -1]).sum() +
np.square(w2[:, 0: -1]).sum())

    delta3 = a3 - y_one_hot
    delta2 = np.multiply(delta3 * w2[:, 0: -1], np.multiply(s2, 1.0 - s2))
    l1_grad = delta2.T * a1
    l2_grad = delta3.T * a2

    r1_grad = np.column_stack([w1[:, 0: -1], np.matrix(np.zeros([1, 1]))])
    r2_grad = np.column_stack([w2[:, 0: -1], np.matrix(np.zeros([2, 1]))])

    w1_grad = (1.0 / num_sample) * l1_grad + (1.0 * lambda_ / num_sample) * r1_grad
    w2_grad = (1.0 / num_sample) * l2_grad + (1.0 * lambda_ / num_sample) * r2_grad
    w_grad = np.row_stack([w1_grad.reshape(-1, 1), w2_grad.reshape(-1, 1)])

    return cost, w_grad


def predict(x,w):
    num_sample = len(x)
    w1 = w[0: (num_feature + 1)].reshape(1, num_feature + 1)
    w2 = w[(num_feature + 1):].reshape(2, 2)
    b = np.matrix(np.ones([num_sample, 1]))
    h1 = sigmoid(np.column_stack([x, b]) * w1.T)
    h2 = sigmoid(np.column_stack([h1, b]) * w2.T)
    return np.argmax(h2, 1)

def test(x, y, w):
    num_sample = len(x)
    y_pred = predict(x,w)
    y_one_hot = (np.zeros(y.shape))
    y_one_hot[np.where(y_pred == y)[0]] = 1
```

```
    acc = 1.0 * y_one_hot.sum() / num_sample
    return acc


from sklearn.datasets import load_svmlight_file
data = load_svmlight_file("a9a",query_id=True)
x_train=data[0].todense()
y_train=data[1]
datatest=load_svmlight_file("a9a.t",query_id=True,n_features=123)
x_test=datatest[0].todense()
y_test=datatest[1]

for opt_algo in opt_algo_set:
    w = train(x_train, y_train, opt_algo)
    acc_train = test(x_train, y_train, w)
    acc_test = test(x_test, y_test, w)


plt.subplot(111)
plt.title('Performance of different Gradient Descent Optimization')
plt.xlabel('# of epoch')
plt.ylabel('cost')


proxy = list()
legend_array = list()
for index, (opt_algo, epoch_cost) in enumerate(plt_dict.items()):
    selected_color = plt_color_array[index % len(plt_color_array)]
    plt.plot(range(len(epoch_cost)), epoch_cost, '-%s' % selected_color[0])
    proxy.append(Rectangle((0, 0), 0, 0, facecolor=selected_color))
    legend_array.append(opt_algo)
plt.legend(proxy, legend_array)
plt.show()
```

线性分类与随机梯度下降

8. 模型参数的初始化方法: 留出法

9.选择的 loss 函数及其导数:

逻辑回归与随机梯度下降

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

$$\mathbf{w}' \to \mathbf{w} - \eta\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = (1-\eta\lambda)\mathbf{w} + \eta\frac{1}{n}\sum_{i=1}^{n}\frac{y_i\mathbf{x}_i}{1+e^{y_i\cdot\mathbf{w}^\top\mathbf{x}_i}}$$

线性分类与随机梯度下降

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：

逻辑回归与随机梯度下降
η=0.001(RMSprop，Adam),0.05(Adadelta, NAG)
epoch=30

线性分类与随机梯度下降

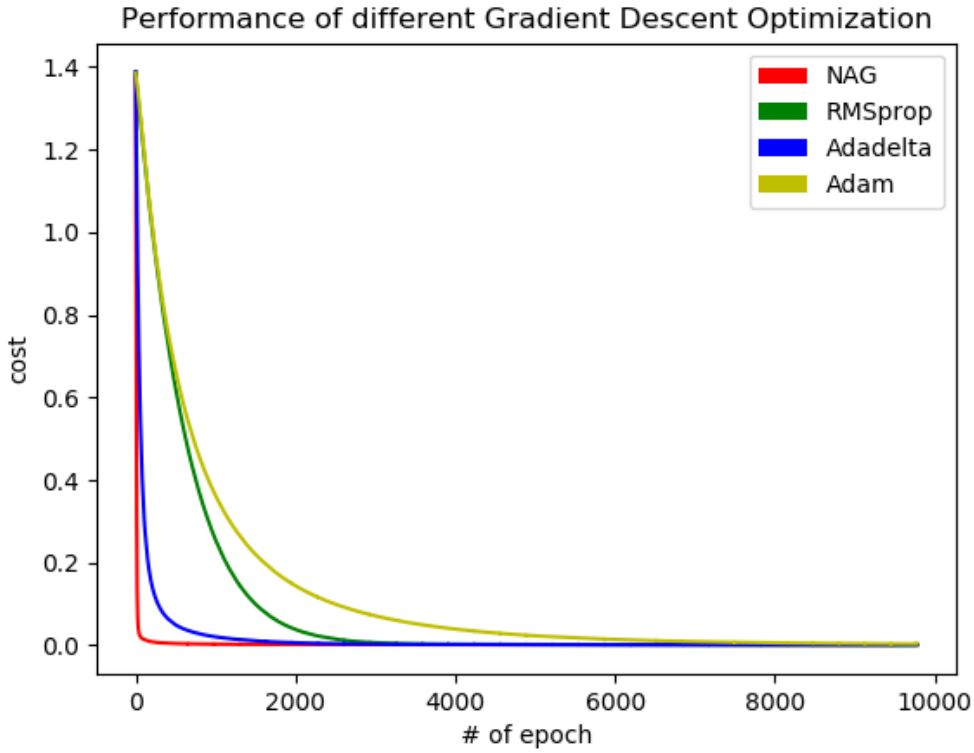
预测结果（最佳结果）：

逻辑回归与随机梯度下降
最佳结果达到 1

线性分类与随机梯度下降

loss 曲线图：

逻辑回归与随机梯度下降



线性分类与随机梯度下降

11.实验结果分析:

逻辑回归与随机梯度下降

从曲线图观察，NAG 方法能够实现快速收敛，其他优化方法也能在一定迭代次数内较快地完成收敛，并且最终结果趋同。

线性分类与随机梯度下降

12.对比逻辑回归和线性分类的异同点：

13.实验总结：