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　赵云潇

学　　号　201530613757

邮　　箱

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:赵云潇

4. 实验目的:

对比理解梯度下降和随机梯度下降的区别与联系。
对比理解逻辑回归和线性分类的区别与联系。

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:

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

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

def get_data():
    data = load_svmlight_file("a9a")
    x_train = data[0].toarray()
    y_train = data[1]
    for i in range(0, len(y_train)):
        if y_train[i] == -1:
            y_train[i] = 0
    temp = np.ones(shape=[32561, 1], dtype=np.float32)
    x_train = np.concatenate([x_train, temp], axis=1)
    return x_train, y_train

def get_test():
    data = load_svmlight_file("a9a.t")
    x_test = data[0].toarray()
    y_test = data[1]
    for i in range(0, len(y_test)):
        if y_test[i] == -1:
            y_test[i] = 0
    temp = np.zeros(shape=[16281, 1], dtype=np.float32)
    temp1 = np.ones(shape=[16281, 1], dtype=np.float32)
    x_test = np.concatenate([x_test, temp, temp1], axis=1)
    return x_test, y_test

def sigmoid(input_):
    return 1 / (1 + np.exp(-input_))

def train(x_train, y_train, x_test, y_test, method, iters, test_errors):
    iterations = 100
    theta = random.rand(num_features + 1)
    num_test_samples, num_test_features = x_test.shape

    if method == 'sgd':
        rate = 0.01

        for i in range(iterations):
            output = sigmoid(np.dot(x_train[i], theta))
            error = output - y_train[i]
            theta = theta - rate * np.dot(x_train[i], error)
```

```
                predict_error = 0
                for j in range(num_test_samples):
                    predict_output = sigmoid(np.dot(x_test[j], theta))
                    predict_error -= y_test[j] * np.log(predict_output) + (1 - y_test[j]) * np.log(1 - predict_output)

                iters.append(i)
                test_errors.append(predict_error / num_test_samples)

        if method == 'nag':
            rate = 0.01
            miu = 0.9
            momentum = np.zeros(num_features + 1)

            for i in range(iterations):
                output = sigmoid(np.dot(x_train[i], theta - rate * miu * momentum))
                error = output - y_train[i]
                grad = np.dot(x_train[i], error)
                momentum = momentum * rate + grad
                theta = theta - rate * momentum

                predict_error = 0
                for j in range(num_test_samples):
                    predict_output = sigmoid(np.dot(x_test[j], theta))
                    predict_error -= y_test[j] * np.log(predict_output) + (1 - y_test[j]) * np.log(1 - predict_output)

                iters.append(i)
                test_errors.append(predict_error / num_test_samples)

        if method == 'rmsprop':
            rate = 0.1
            e = 1
            rho = 0.95
            delta = 10e-7

            for i in range(iterations):
                output = sigmoid(np.dot(x_train[i], theta))
                error = output - y_train[i]
                grad = np.dot(x_train[i], error)
                norm = grad * grad
                expectation = rho * e + (1 - rho) * norm
```

```
                theta = theta - rate * grad / (np.sqrt(expectation) + delta)

                predict_error = 0
                for j in range(num_test_samples):
                    predict_output = sigmoid(np.dot(x_test[j], theta))
                    predict_error -= y_test[j] * np.log(predict_output) + (1 - y_test[j]) * np.log(1 - predict_output)
                iters.append(i)
                test_errors.append(predict_error / num_test_samples)

        if method == 'adam':
            delta = 10e-8
            rho1 = 0.9
            rho2 = 0.999
            rate = 0.1
            s = 0
            r = 0

            for i in range(iterations):
                output = sigmoid(np.dot(x_train[i], theta))
                error = output - y_train[i]
                grad = np.dot(x_train[i], error)

                s = rho1 * s + (1 - rho1) * grad
                r = rho2 * r + (1 - rho2) * grad * grad
                s_hat = s / (1 - rho1)
                r_hat = r / (1 - rho2)
                delta_theta = (-rate * s_hat) / (np.sqrt(r_hat) + delta)
                theta = theta + delta_theta

                predict_error = 0
                for j in range(num_test_samples):
                    predict_output = sigmoid(np.dot(x_test[j], theta))
                    predict_error -= y_test[j] * np.log(predict_output) + (1 - y_test[j]) * np.log(1 - predict_output)
                iters.append(i)
                test_errors.append(predict_error / num_test_samples)

        if method == 'adadelta':
            r = 0
            e = 0
            miu = 0.9
            delta = 10e-7
            rate = 10
```

```
            for i in range(iterations):
                output = sigmoid(np.dot(x_train[i], theta))
                error = output - y_train[i]
                grad = np.dot(x_train[i], error)

                r = miu * r + (1 - miu) * grad * grad
                delta_theta = (-rate * grad * np.sqrt(e + delta)) / (np.sqrt(r + delta))
                theta = theta + delta_theta
                e = miu * e + (1 - miu) * e * e


                predict_error = 0
                for j in range(num_test_samples):
                    predict_output = sigmoid(np.dot(x_test[j], theta))
                    predict_error -= y_test[j] * np.log(predict_output) + (1 -
y_test[j]) * np.log(1 - predict_output)
                iters.append(i)
                test_errors.append(predict_error / num_test_samples)

    def main():
        x_train, y_train = get_data()
        x_test, y_test = get_test()

        methods = ['sgd', 'nag', 'rmsprop', 'adadelta', 'adam']
        for method in methods:
            iters = []
            test_errors = []
            train(x_train, y_train, x_test, y_test, method, iters, test_errors)
            plt.plot(iters, test_errors, label=method)

    main()
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    线性分类
    from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
import os
```

```
feature_size = 123
bias = np.zeros(shape=[feature_size + 1, 1])
bias[len(bias)-1][0] = 1.

def get_data():
    data = load_svmlight_file("a9a")
    x_train = data[0].toarray()
    y_train = data[1]
    temp = np.ones(shape=[32561, 1], dtype=np.float32)
    x_train = np.concatenate([x_train, temp], axis=1)
    return x_train, y_train

def get_test():
    data = load_svmlight_file("a9a.t")
    x_test = data[0].toarray()
    y_test = data[1]
    temp = np.zeros(shape=[16281, 1], dtype=np.float32)
    temp1 = np.ones(shape=[16281, 1], dtype=np.float32)
    x_test = np.concatenate([x_test, temp, temp1], axis=1)
    return x_test, y_test

def loss(x, y, w):
    pred = np.matmul(x, w)
    hinge_loss = np.maximum(1 - y * pred, 0)
    loss = np.mean(hinge_loss ** 2) + np.sum((w - bias) ** 2)
    return loss

def gradient(x, y, w):
    pred = np.matmul(x, w)
    hinge_loss = np.maximum(1 - y * pred, 0)
    hinge_loss_gradient = -np.matmul(x.transpose(), hinge_loss * y) / len(y)
    norm_gradient = 2 * (w - bias)
    gradient = hinge_loss_gradient + norm_gradient
    return gradient

def optimizer(method, x, y, w):
    global global_list

    if method == 'sgd':
        rate = 0.01
        print(w)
        w -= rate * gradient(x, y, w)

    if method == 'nag':
```

```
            rate = 0.01
            miu = 0.9
            momentum = 0
            grad = gradient(x, y, w - momentum * rate * miu)
            momentum = momentum * rate + grad
            w -= rate * momentum

        if method == 'rmsprop':
            rate = 0.1
            e = 1
            rho = 0.95
            delta = 10e-7
            grad = gradient(x, y, w)
            expectation = rho * e + (1 - rho) * (grad ** 2)
            w -= rate * grad / (np.sqrt(expectation) + delta)

        if method == 'adam':
            delta = 10e-7
            rho1 = 0.9
            rho2 = 0.999
            rate = 0.1
            s = 0
            r = 0
            grad = gradient(x, y, w)
            s = rho1 * s + (1 - rho1) * grad
            r = rho2 * r + (1 - rho2) * (grad ** 2)
            s_hat = s / (1 - rho1)
            r_hat = r / (1 - rho2)
            w -= (rate * s_hat) / (np.sqrt(r_hat) + delta)

        if method == 'adadelta':
            r = 0
            e = 0
            miu = 0.9
            delta = 10e-7
            rate = 10
            grad = gradient(x, y, w)
            r = miu * r + (1 - miu) * (grad ** 2)
            w -= (rate * grad * np.sqrt(e + delta)) / (np.sqrt(r + delta))
            e = miu * e + (1 - miu) * (w ** 2)

    def main():
        x_train, y_train = get_data()
        x_test, y_test = get_test()
```

```
    num_samples, num_features = x_train.shape
    num_test_samples, num_test_features = x_test.shape

    y_train = y_train.reshape([len(y_train), 1])
    y_test = y_test.reshape([len(y_test), 1])


    def shuffle_train():
        global x_train, y_train
        rng_state = np.random.get_state()
        np.random.shuffle(x_train)
        np.random.set_state(rng_state)
        np.random.shuffle(y_train)

    batch_size = 1024
    data_size = num_samples

    def feed_data(batch_count):
        if (1 + batch_count) * batch_size <= data_size:
            feed_dict = {'x': x_train[batch_count * batch_size:(batch_count + 1) *
batch_size],
                         'y': y_train[batch_count * batch_size:(batch_count + 1)
* batch_size]}
        else:
            feed_dict = {'x': x_train[batch_count * batch_size:data_size],
                         'y': y_train[batch_count * batch_size:data_size]}
        return feed_dict


    for method in methods:
        iters = []
        test_errors = []
        w = np.random.rand(feature_size+1, 1)
        print(w)

        count = 0
        for i in range(0, 3):
            shuffle_train()
            for batch_count in range(0, int(data_size / batch_size) + 1):
                feed_dict = feed_data(batch_count)
                iters.append(count)
                count += 1
                optimizer(method=method,    x=feed_dict['x'],    y=feed_dict['y'],
```

```
w=w)
                    test_errors.append(loss(x_test, y_test, w))
        plt.plot(iters, test_errors, label=method)

main()
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

8. 模型参数的初始化方法:

随机初始化

9.选择的 loss 函数及其导数:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p_i) + (1-y_i) log(1-p_i)$$

$$\frac{\partial Loss}{\partial w} = -\frac{1}{N}\sum_{i=1}^{N} x_i(p_i - y_i)$$

10. 实验结果和曲线图:

超参数选择：

SGD:
rate = 0.01

NAG:
rate = 0.01
miu = 0.9
momentum = 0

RMSProp:
rate = 0.1
e = 1
rho = 0.95
delta = 10e-7

Adam:
delta = 10e-7

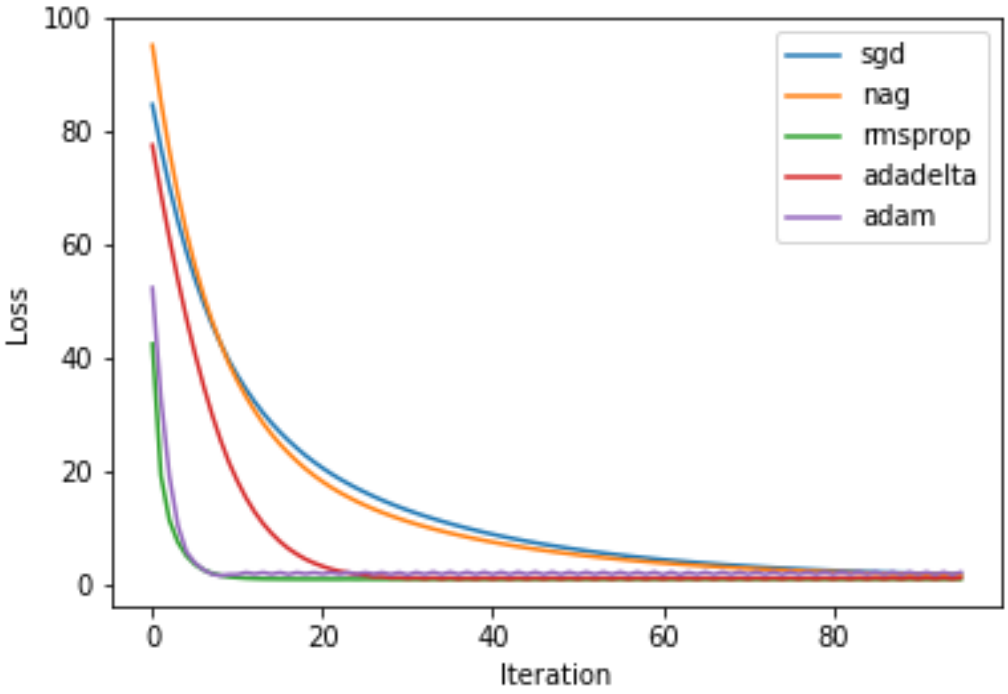
rho1 = 0.9
rho2 = 0.999
rate = 0.1
s = 0
r = 0

Adadelta
r = 0
e = 0
miu = 0.9
delta = 10e-7
rate = 10

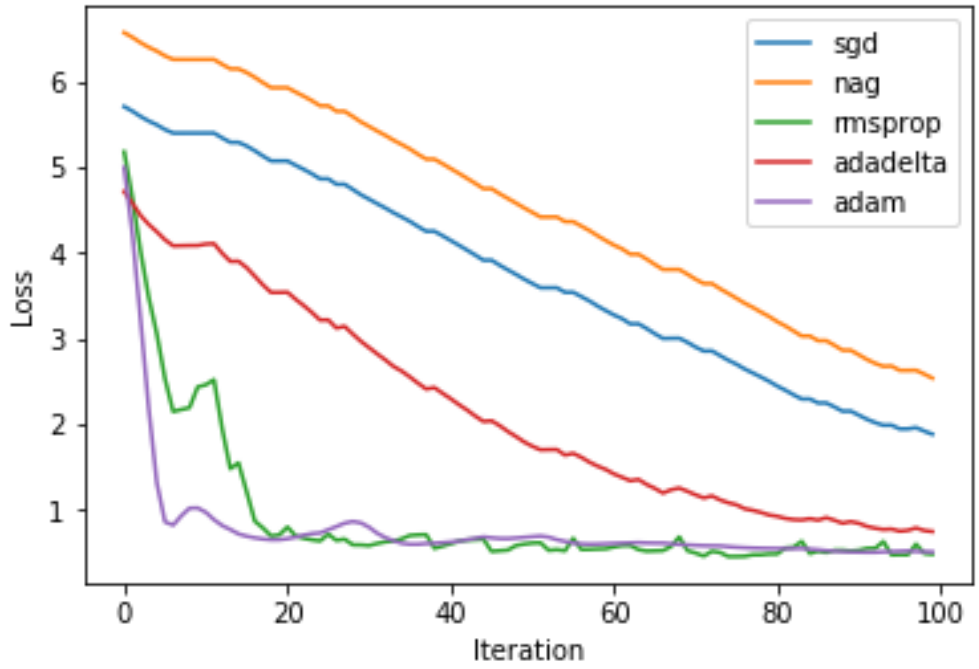
预测结果（最佳结果）：

loss 曲线图：

线性分类



逻辑回归



11.实验结果分析:

线性分类的结果优于逻辑回归

12.对比逻辑回归和线性分类的异同点：

逻辑回归通过 sigmoid 函数对梯度下降后的函数分类，先梯度下降增大两类之间的差距，再进行分类，线性分类直接分类

13.实验总结：

此次实验使用了不同的优化方法，较上次实验难度提高不少