模式识别实验报告

实验三 线性分类器

学院：计算机科学与技术

姓名：张文强

学号：18S003044

一、实验内容

1、 使用 C 或 Matlab 编程实现感知器算法和最小平方误差算法；

2、 分别使用感知器算法学习区分下列两类样本的线性分类器：

$$\omega_1:(1,1)^t,(2,2)^t,(2,0)^t$$
$$\omega_2:(0,0)^t,(1,0)^t,(0,1)^t$$

3、 MNIST 数据集测试：使用 TrainSamples 中的 30000 个 17 维特征手写数字样本训练线性分类器区分 10 个类别，TrainLabels 中包含训练样本的标签；测试线性分类器对 TestSamples 中 10000 个样本的识别正确率。

二、程序代码

（感知器算法和最小平方误差算法，矩阵乘法和求逆可以调用其他函数库中的程序）

```
# label 0, 1
class Perceptron:
    def __init__(self, w=None):
        self.w = w

    def augment_and_standardize(self, data, label):
        data = np.insert(data, 0, 1, axis=1)
        for idx, l in enumerate(label):
            if l:
                data[idx] = -data[idx]
        return data

    def init_params(self, dim_data):
        if not self.w:
            self.w = np.random.randn(dim_data)

    def judge_converge(self, data):
        for x in data:
            if np.dot(self.w, x) <= 0:
                return False
        return True

    def train(self, data, label):
        data = self.augment_and_standardize(data, label)
        n_data, dim_data = data.shape
        self.init_params(dim_data)
        k = 0
        while True:
            # make mistake
            if np.dot(self.w, data[k]) <= 0:
                self.w = self.w + data[k]
            k = (k + 1) % n_data
```

```
          if self.judge_converge(data):
              break

    def test(self, test_data, label):
        test_data = np.insert(test_data, 0, 1, axis=1)
        n_data = len(label)
        predict = np.empty_like(label)
        for i, x in enumerate(test_data):
            if np.dot(self.w, x) > 0:
                predict[i] = 0
            else:
                predict[i] = 1
        # print (predict)
        correct_cnt = (predict == label).sum()
        print ('[%d/%d] acc=%.2f%%' % (correct_cnt, n_data,
correct_cnt / n_data))

# label 0, 1
class LMSE:
    def augment_and_standardize(self, data, label):
        data = np.insert(data, 0, 1, axis=1)
        for idx, l in enumerate(label):
            if l:
                data[idx] = -data[idx]
        return data

    def train(self, data, label):
        data = self.augment_and_standardize(data, label)
        n_data, dim_data = data.shape
        label = np.ones((n_data, 1))
        self.w =
np.linalg.inv(data.T.dot(data)).dot(data.T).dot(label).reshape((d
im_data,))

    def test(self, test_data, label):
        test_data = np.insert(test_data, 0, 1, axis=1)
        n_data = len(label)
        predict = np.empty_like(label)
        for i, x in enumerate(test_data):
            if np.dot(self.w, x) > 0:
                predict[i] = 0
            else:
                predict[i] = 1
        # print (predict)
```

```
        correct_cnt = (predict == label).sum()
        print ('[%d/%d] acc=%.2f%%' % (correct_cnt, n_data,
correct_cnt / n_data))


# label 0, 1, ... , c-1
class KeslerPerceptron:
    def __init__(self, c, lr = 1e-6, ws = None):
        if isinstance(ws, list):
            assert len(ws) == c, "Init ws not enough for %d
classes" % c

        self.c = c
        self.lr = lr
        self.ws = ws

    def init_params(self, dim_data):
        for i in range(self.c):
            self.ws.append(np.random.randn(dim_data))

    def augment(self, data):
        data = np.insert(data, 0, 1, axis=1)
        return data


    def judge_converge(self, data, label):
        for x,i in zip(data, label):
            g_i = np.dot(self.ws[i], x)
            for j in range(self.c):
                if j != i and np.dot(self.ws[j], x) > g_i:
                    return False
        return True

    def train(self, data, label, max_epoch = 15):
        data = self.augment(data)
        n_data, dim_data = data.shape
        if self.ws == None:
            self.ws = []
            self.init_params(dim_data)
        k = 0
        while True:
            # print ('k=%d' % k)
            x = data[k]
            i = label[k]
```

```
            g_i = np.dot(self.ws[i], x)
            for j in range(self.c):
                if j != i and np.dot(self.ws[j], x) >= g_i:
                    self.ws[i] += self.lr * x
                    self.ws[j] -= self.lr * x
            k = (k+1) % n_data
            if k == 0:
                max_epoch -= 1
                print ('epochs_remain:', max_epoch)
                print ('train:')
                self.test(mnist_train_data, mnist_train_label)
                print ('test:')
                self.test(mnist_test_data, mnist_test_label)

            if not max_epoch or self.judge_converge(data, label):
                break

    def test(self, test_data, label):
            test_data = np.insert(test_data, 0, 1, axis=1)
            n_data = len(label)
            predict = np.empty_like(label)
            for idx, x in enumerate(test_data):
                max_v = -9999999
                max_i = -1
                for i in range(self.c):
                    g_i = np.dot(self.ws[i], x)
                    if g_i > max_v:
                        max_v = g_i
                        max_i = i
                predict[idx] = max_i

            correct_cnt = (predict == label).sum()
            print ('[%d/%d] acc=%.2f%%' % (correct_cnt, n_data,
correct_cnt / n_data))


class multiclass_lmse_ova:
    def __init__(self, c):
        self.c = c
        self.lmses = [LMSE() for i in range(c)]

    def construct_data(self, data, label, cls):
        label_01 = label.copy()
        label_01[label==cls], label_01[label!=cls] = 0, 1
```

```python
        return data, label_01

    def train(self, data, label):
        for idx, lmse in enumerate(self.lmses):
            # print ('training lmse %d ... ' % idx)
            data, label_01 = self.construct_data(data, label, idx)
            lmse.train(data, label_01)

    def test(self, test_data, label):
        test_data = np.insert(test_data, 0, 1, axis=1)
        n_data = len(label)
        predict = np.empty_like(label)
        for idx, x in enumerate(test_data):
            max_v = -99999
            max_i = -1
            for i in range(self.c):
                g = np.dot(self.lmses[i].w, x)
                if g > max_v:
                    max_v = g
                    max_i = i
            predict[idx] = max_i
        correct_cnt = (predict == label).sum()
        print ('[%d/%d] acc=%.4f%%' % (correct_cnt, n_data,
correct_cnt / n_data))
```

三、实验结果

1、 仿真数据实验结果：分别给出使用感知器算法和最小平方误差算法得到的线性判别函数。

感知器算法：

G(x) = [-1.82571391,   1.66584716,   1.63244533] * y

最小平方误差算法：

G(x) = [-1.13513514,   0.91891892,   0.32432432] * y

2、 MNIST 数据集实验结果：（多类别解决方案及分类正确率）

1. **情况 3，采用扩展的感知器算法——Kesler 构造法**

   学习率设置为 1e-6，迭代 10 个 epoch，最终实验结果如下：

数据集	正确分类数	正确分类率
mnist_train (30000)	24882	82.94%
mnist_test (10000)	8241	82.41%

2. 情况 3，采用最小平方误差算法——multiclass LMSE

直接采用伪逆求解法，最终实验结果如下：

数据集	正确分类数	正确分类率
mnist_train (30000)	23341	77.80%
mnist_test (10000)	7751	77.51%