面部图像性别识别问题

Matlab 数据文件 FaceImage.mat 中存储着四个矩阵 train_male、train_female、test_male 和 test_female，分别表示两类问题的训练样本和测试样本图像，图像大小为 30×25。

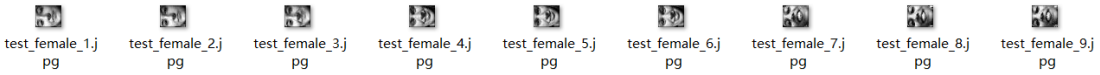
1) 自行选择一种模式分类特征，说明该特征的计算方法，编程实现特征抽取程序；
2) 编程实现感知器 Pocket 算法，解决该问题，给出分类器性能测试结果；
3) 编程实现 LMS 算法解决该问题，给出分类器性能测试结果；
4) 基于 SVM 算法（可以调用库函数）编程解决该问题，给出分类器性能测试结果（使用多项式核函数和径向基函数两种核函数）；
5) 比较三种算法的性能结果，并解释原因。

一、问题描述

设计一个分类器模型，该模型的输入为人脸图像，输出为对输入图像的分类结果（男/女，用相应的标签表示 0/1 或-1/1 等）。所谓分类器模型可以简单理解成一个算法，当然这个算法是可优化的，即我们用训练数据集作为其输入，观察其输出，根据输出结果对算法进行改进，以获得一个性能可接受的算法配置（即训练好的分类器模型）；另外我们还可以通过其他方式对模型的性能进行评测以进一步优化，例如我们可以通过测试集测试模型的泛化能力。

二、基本思路

选择模式分类特征时，首先用 python 将 matlab 数据文件中的数据恢复成 jpg 格式的图像文件，以初步获得对可用数据集中图像的感性认识，恢复的图像如下图所示：



观察到还原后的人脸图像都是经过了一定的裁剪，只保留了眼睛、鼻子、嘴巴及下巴在内的轮廓；人脸图像为灰度图像，不具备完整丰富的颜色信息，另外图像均为 30*25 像素的统一尺寸。因此难以将人脸图像的颜色，尺寸等物理和几何特征直接作为用于分类的模式特征。

于是考虑图像内部的数字特征。数据文件中的每张图像都是 30*25 的灰度图，所以将每张图像的 750 个像素值作为其原始特征，因此原始模式特征矢量是 750 维特征空间中的特征矢量，显然特征空间的维度过高，需要通过特征选择和特征提取对特征空间降维。

查阅资料后，在众多的图像数据特征选择和提取方法中选择了采用主成分分析法（PCA）降维的方式提取特征。PCA 方法从一组特征中通过求解最优的正交变换，得到一组相互间方差最大的新特征，它们是原始特征的线性组合，且相互之间是不相关的，再对新特征进行重要性排序，选取前几个主成分。用较少的主成分来表示数据，可以实现特征的降维，还可以消除数据中的噪声。PCA 的这些

特点比较符合本次任务的需求且易于实现,因此选择了 PCA 方法进行特征提取。

完成特征提取任务后即可根据几种算法设计实现相应的分类器模型。

三、算法

3.1 数据预处理

1. 如 2.1 所述，模式特征提取采用了 PCA 的方法，PCA 的实现主要有以下关键步骤：
   1) 确定数据降维后的维数 saved_components；
   2) 样本数据中心化,去平均值；
   3) 求得去平均值后样本数据的协方差矩阵；
   4) 计算协方差矩阵的特征值和特征向量；
   5) 选取最大的 saved_components 个特征值所对应的 saved_components 个特征向量构成转换矩阵；
   6) 用转换矩阵将原始数据转换到降维后的空间， 得到最终降维后的数据。

```
## 算法实现
class SimplePCA():
    def __init__(self,componets = 51, axis=0, ):
        self.reduce_axis = axis  # 降维的矩阵维度，默认对列降维
        self.saved_components = componets # 降维后保留的特征个数
        self.means = None # 样本数据降维所在维度的均值
        self.transform_mat = None # 由样本数据协方差矩阵最大的 N 个特征值对应的特征向量构成的转换矩阵

    def __call__(self, datas):
        return self.call(datas)

    def call(self, datas):
        '''
        进行 PCA 降维处理
        :param datas: 待降维的样本数据
        :return: 降维后的数据
        '''
        means = np.mean(datas, axis=self.reduce_axis)
        self.means = means
        # 样本数据中心化,去平均值,每一列减去其平均值
        mean_removed = datas - means

        self.mean_removed = mean_removed
        # 求中心化后样本数据的协方差矩阵
        covariance_mat = np.cov(mean_removed, rowvar=False)
        self.covariance_mat = covariance_mat
        # 协方差矩阵的特征值和特征向量
        # print(covariance_mat)
```

```
        # print(np.mat(covariance_mat))
        eig_values, eig_vectors = np.linalg.eigh(covariance_mat)
        self.eig_values = eig_values
        self.eig_vectors = eig_vectors
        # print("--------------------------------")
        # print(eig_values)
        # print(eig_vectors)
        # 取最大的 saved_components 个特征值
        # 特征值的索引降序排列
        _index = np.argsort(eig_values)[::-1]
        max_index = _index[ : self.saved_components]

        # 取最大的 saved_components 个特征值对应的 saved_components 个特征向
量构成转换矩阵
        transform_mat = eig_vectors[:,max_index ]
        self.transform_mat = transform_mat

        # 用转换矩阵将原始数据转换到降维后的空间， 得到最终的降维后的数据集
        transformed_datas = np.matmul(mean_removed , transform_mat)

        self.transformed_datas = transformed_datas
        # retransformed_datas = self.low_dimension_space *
self.transform_mat + self.means

        return transformed_datas
```

2. 为了方便数据集的使用，对数据集的预处理过程进行了如下封装（需要注意的是，由于给定数据集中训练样本个数为 51，因此采用 PCA 降维的方式预处理数据时，降维后的空间维数最小为 51 维）：

```
class DataLoader(object):
    def __call__(self,
path=r'D:\Daisurong\NKICS_DSR\Projects\Class_Projects\PatternRecognit
ion\ThirdPro\src\datasets\FaceImages.mat'):
        self.data_path = path
        return self.call()

    def call(self):
        datas = scio.loadmat(self.data_path)
        self.datas = datas
        test_female = datas['test_female']
        train_female = datas['train_female']
        test_male = datas['test_male']
        train_male = datas['train_male']
```

```
        self.test_nums = len(test_female) + len(test_male)  # 27
        self.train_nums = len(train_female) + len(train_male)  # 51
        x_train = np.concatenate((train_male, train_female), axis=0)
        x_test = np.concatenate((test_male, test_female), axis=0)

        # 首先将像素值处理为[0,1]范围内的值
        x_train = x_train/255.
        x_test = x_test/255.

        # 降维最少不能小于数据集的最小的一个维度
        self.n_components = min(x_train.shape)
        self.spac = SPCA(self.n_components)
        x_train = self.spac(x_train)

        # 测试集减去训练集降维时的均值， 并乘以训练集降维时的转换矩阵
        x_test = x_test - self.spac.means
        x_test = np.matmul(x_test, self.spac.transform_mat)

        self.x_train = x_train
        self.x_test = x_test
        y_train, y_test = self.set_label()
        self.y_train = np.array(y_train)
        self.y_test = np.array(y_test)
        return self.x_test, self.y_test, self.x_train, self.y_train

    def set_label(self):
        # 为数据集添加标签，设 male 对应的标签为-1， female 对应的标签为 1
        y_train = []
        y_test = []
        for i in range(len(self.datas["train_male"])):
            y_train.append(0)
        for i in range(len(self.datas["train_female"])):
            y_train.append(1)
        for i in range(len(self.datas["test_male"])):
            y_test.append(0)
        for i in range(len(self.datas["test_female"])):
            y_test.append(1)

        return y_train, y_test
# 使用样例
data_path =
r'D:\Daisurong\NKICS_DSR\Projects\Class_Projects\PatternRecognition\T
hirdPro\src\datasets\FaceImages.mat'
```

```
datas = DataLoader()
x_test, y_test, x_train, y_train = datas(data_path)
```

3. 对训练样本进行 shuffle

```
# 首先打乱训练数据集
state = np.random.get_state()
np.random.shuffle(x_train)
np.random.set_state(state)
np.random.shuffle(y_train)
```

3.2 基于感知器 Pocket 算法实现的分类器

1. 定义判别式函数

1)采用线性判别式方程的齐次坐标表示，加载训练样本时做简单的预处理：

```
data_path =
r'D:\Daisurong\NKICS_DSR\Projects\Class_Projects\PatternRecognition\ThirdPro\src\datasets\FaceImages.mat'
datas = DataLoader()
x_test, y_test, x_train, y_train = datas(data_path)
# 将 d 维的特征向量变为 d+1 维，便于采用线性判别式方程的齐次坐标表示 ，模式特征的维度+1 （+1 作为偏置项）
train_bias = np.ones(len(x_train))
test_bias = np.ones(len(x_test))
x_train = np.c_[x_train, train_bias.T]
x_test = np.c_[x_test, test_bias.T]
```

2)类别标签为 male-0；female-1，在数据预处理过程中完成：

```
def set_label(self):
    # 为数据集添加标签，设 male 对应的标签为-1， female 对应的标签为 1
    y_train = []
    y_test = []
    for i in range(len(self.datas["train_male"])):
        y_train.append(0)
    for i in range(len(self.datas["train_female"])):
        y_train.append(1)
    for i in range(len(self.datas["test_male"])):
        y_test.append(0)
    for i in range(len(self.datas["test_female"])):
        y_test.append(1)

    return y_train, y_test
```

3）齐次坐标表示的判别式函数实现

```
def fun_decide(XX, W):
    # 判别式函数
    g = np.matmul(W, XX)
    if g>0:
        return 0
    else:
        return 1
```

2. 实现感知器代价函数，并根据代价函数得到权矢量的梯度

$$\mathcal{J}(\mathrm{w}) = \sum_{X\in\Omega} \delta_X w^t X$$

$$\nabla \mathrm{w} = \sum_{X\in\Omega} \delta_X X$$

```
def fun_delta(X, W, label):
    g = fun_decide(X, W)
    if g == label: # 决策正确， 代价为 0
        return 0
    elif g == 1 : # 决策错误， x 为 w1 类，判别为 w2 类
        return -1
    else:# 决策错误， 且为 w2 类
        return 1

def fun_loss(W, X, Y):
    losses = 0
    w_gradients = np.zeros(len(W))
    WT = W.T
    for i in range(len(X)):
        delta_x = fun_delta(X[i], W, Y[i])
        losses += delta_x * np.matmul(WT,X[i])
        w_gradients += delta_x*X[i]
    return losses, w_gradients
```

3. 感知器算法的一次迭代（根据感知器算法的权矢量迭代公式更新权重）

```
def perception(W, X, Y, learning_rate=0.5):
    losses, w_gradients = fun_loss(W, X, Y)
    print("loss: ", losses)
    print("learning_rate: ", learning_rate)
    lr = learning_rate
    return W - lr*w_gradients
```

4. 基于感知器的 pocket 算法实现

1) 首先定义 accuracy（）函数以计算每次更新感知器权矢量后分类器的正确决策数目，从而确定是否更新暂存矢量 ws 和 hs

```
def accuracy(W, X, Y):
    n_correct = 0.
    x_size = X.shape[0]
    for i in range(x_size):
        predict = fun_decide(X[i], W)
        if predict == Y[i]:
            n_correct += 1
    return n_correct/x_size, n_correct
```

2) Pocket 实现

❖ 算法6.3：Pocket 算法

- 随机初始化权矢量 $\mathbf{w}^{(0)}$
- 设置迭代次数 T
- 定义：暂存矢量 $\mathbf{w}_s$ 和暂存变量 $h_s = 0$
- for $t = 1$ to T
  - ✧使用感知器算法调整权矢量
  - ✧使用调整的权矢量测试训练样本集合
  - ✧如果正确决策数目 $h > h_s$，则 $\mathbf{w}_s = \mathbf{w}^{(t+1)}, h_s = h$。

```
def pockets(x_train, y_train, iter_times=10, learning_rate=0.8):
    # 随机初始化权重向量
    n_features = x_train.shape[1]
    weights = np.random.rand(n_features)
    print(weights.shape)  # (52,)
    # print(weights)
    # 暂存权重 ws
    ws = deepcopy(weights)
    # 暂存变量 hs = 0
    hs = 0
    lr = learning_rate
    acc, n_correct = accuracy(weights, x_train, y_train)
    print("before training: " )
    print("training accuracy:", acc)
    print("correct decisions:", n_correct)
    print("-----------------------------------------")
    for i in range(iter_times):
        weights = perception(weights, x_train, y_train, lr)
        acc, n_correct = accuracy(weights, x_train, y_train)
        print("<====== % d-th training =======> "%(i+1))
        print("training accuracy:", acc)
        print("correct decisions:", n_correct)
        print("-----------------------------------------")
        if n_correct > hs:
```

```
            hs = n_correct
            ws = deepcopy(weights)
        if lr>0.00001:
            lr /= 1.1
        if n_correct==len(x_train):
            break
    return ws
```

3.3 基于 LMS 算法实现的分类器

❖ 算法6.4：最小均方(LMS)算法

⌑ 根据随机逼近方法，最小化MSE问题

$$\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}) = 2\mathbb{E}[\mathbf{x}(y - \mathbf{w}^T\mathbf{x}] = 0$$

的参数迭代调整公式为

$$\widehat{\mathbf{w}}^{(k)} = \widehat{\mathbf{w}}^{(k-1)} + \rho_k \mathbf{x}_i (y_i - \mathbf{x}_i^T \widehat{\mathbf{w}}^{(k-1)})$$

其中 $(y_i, \mathbf{x}_i)$ 是期望输出 (± 1) 和输入训练样本对。

```
def least_mean_square(x_train, y_train, iter_times=10,
learning_rate=0.05):
    # 随机初始化权重向量
    n_features = x_train.shape[1]
    weights = np.random.rand(n_features)
    print(weights.shape)  # (52,)
    lr = learning_rate
    for i in range(iter_times):
        # mloss = fun_mloss(weights, x_train, y_train)
        acc, n_correct = accuracy(weights, x_train, y_train)
        print("==========================================")
        print("learning rate:", lr)
        print("train accuracy:", acc)
        print("train n_correct: ", n_correct)
        if n_correct == len(x_train):
            print(" break at %d-th iteration"%(i))
            break
        losses = []
        for i in range(len(x_train)):
            predict = sign(np.matmul(x_train[i].T, weights))
            # print(predict)
            _delta = x_train[i] * (y_train[i]-predict)
            weights += lr*_delta
            losses.append((y_train[i] - predict) ** 2)
        lr /= 1.1
```

```
        loss = np.mean(losses)
        print("loss:", loss)
        if loss < 1e-7:
            print(" break at %d-th iteration" % (i))
            break
    return weights
```

由于预定义的样本标签为 0,1，所以自定义 sign()函数以得到模型发预测标签：

```
def sign(value):
    if value>0:
        return 1
    else:
        return 0

def accuracy(W, X, Y):
    n_correct = 0.
    x_size = X.shape[0]
    for i in range(x_size):
        predict = sign(np.matmul(X[i], W))
        if predict == Y[i]:
            n_correct += 1
    return n_correct/x_size, n_correct
```

3.4 基于 SVM 算法的分类器

调用了 scikit_learn 中库函数完成对基于 SVM 实现的分类器的测试。

1) 采用多项式核函数

```
'''
SVM 分类器， 采用多项式核函数,kernel='poly'
'''
poly_model = svm.SVC(kernel='poly', C=c, gamma=g)

poly_model.fit(x_train, y_train)
poly_score = poly_model.score(x_train, y_train)
print("poly score:", poly_score)
poly_predict = poly_model.predict(x_test)
# print("poly_predict:", poly_predict)
test_acc, test_correc = accuracy(poly_predict, y_test)
print("poly test accuracy:", test_acc)
print("correct decisions:", test_correc)
```

2) 采用径向基核函数

```
'''
    SVM 分类器， 采用多径向基核函数,kernel='rbf'
'''
rbf_model = svm.SVC(kernel='rbf', C=c, gamma=g)

rbf_model.fit(x_train, y_train)
rbf_score = rbf_model.score(x_train, y_train)
print("rbf score:", rbf_score)
rbf_predict = rbf_model.predict(x_test)
# print("rbf_predict:", rbf_predict)
test_acc, test_correc = accuracy(rbf_predict, y_test)
print("rbf test accuracy:", test_acc)
print("correct decisions:", test_correc)

print("poly_predict:", poly_predict)
print("true label:", y_test)
print("rbf_predict:", rbf_predict)
```

3) 由于 scikit-learn 库函数中实现的 SVM 测试时 predict 函数输出的是类别标签，因此需自定义计算分类精度的函数以方便进行后续的实验结果对比分析。

```
def accuracy(predict, true_labels):
    x_size = float(len(predict))
    n_correct = np.sum(predict==true_labels)
    acc = n_correct/x_size
    return n_correct, acc
```

四、结果与分析

4.1 基于感知器 pocket 算法的分类结果及分析

1. 实验结果

基于感知器 pocket 算法的分类结果如表 4-1 所示。

表 4-1 感知器分类实验结果

	第 1 组	第 2 组	第 3 组	第 4 组	第 5 组	第 6 组	第 7 组	第 8 组
最大迭代次数	10	20	30	10	20	10	10	10
初始学习率	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1
学习率更新率	lr /= 2	lr /= 2	lr /= 2	lr /= 1.5	lr /= 1.5	lr /= 1.2	lr /= 1.1	lr = lr
收敛迭代次数	未收敛	未收敛	未收敛	未收敛	未收敛	7	7	5
训练集上的分类准确率（%）	86.27	86.27	86.27	96.08	96.08	100	100	100
测试集上的分类准确率（%）	74.07	77.78	77.78	**85.18**	81.48	**70.37**	74.07	**62.97**

说明：①lr /= 2 表示学习率在上次迭代的基础上除以 2；②收敛迭代次数表示在训练未达到预设的最大迭代次数前即收敛（训练集上的分类准确率为 100%）时所经过的迭代次数；③训练集上的分类准确率（%）取每组实验在训练集上得到的最大分类准确率；

2. 实验结果分析

分析表 4-1 中的实验结果可得：

1) 学习率对收敛速度有较大的影响：从第 1、2、3 组实验可见当学习率调整幅度过大时，算法难以收敛；而以较小幅度调整学习率时，算法收敛较快；当采用常数学习率 0.1 时，算法也会很快收敛。

2) 当分类器模型在训练集上的分类精度很高时在测试集上的分类精度不一定高，第 4、6、8 三组的实验结果对比尤其明显。因此在训练集上达到 100%的分类准确率很有可能已经导致了分类器模型的过拟合问题，因此可以适当调整算法终止条件，以获得泛化能力较好的分类器模型。

4.2 基于 LMS 算法的分类结果及分析

1. 实验结果

基于 LMS 算法的分类实验结果如表 4-2 所示。

表 4-2 基于 LMS 算法分类实验结果

	第 1 组	第 2 组	第 3 组	第 4 组	第 5 组	第 6 组	第 7 组	第 8 组
最大迭代次数	2	2	10	10	10	10	10	10
初始学习率	1.0	1.0	0.5	0.5	0.1	0.1	1.0	1.0
学习率更新率	lr /= 2	lr /= 1.1	lr = lr	lr = lr	lr = lr	lr = lr	lr /= 1.1	lr /= 1.1
收敛迭代次数	未收敛	未收敛	3	3	3	4	4	3
训练集上的分类准确率（%）	88.24	90.20	100	100	100	100	100	100
测试集上的分类准确率（%）	**85.18**	77.78	70.37	66.67	**81.48**	70.37	66.67	70.37

3. 实验结果分析

分析表 4-2 中的实验结果可得：

1) 基于 LMS 的分类器会以较快的速度在训练集上到达 100%的分类准确率；

2) 与 pocket 算法相似，当分类器模型在训练集上的分类精度很高时在测试集上的分类精度不一定高，第 1、2 组特意设置了较小的最大迭代次数以阻止分类器在训练集上达到 100%的准确率，得到的测试集上的分类准确率相对其他组平均要高。因此也要考虑过拟合问题。

3) 第 5 组实验的结果在训练集上达到了 100%的分类准确率，在测试集上也有较高的准确率，但这是很少很偶然的情况，特意在多组实验结果中记录了这一组，以说明模型的通过已有的数据得到的实验结果所能呈现的问题存在一定的不确定性，需要更丰富的实验数据进行实验分析。

4.3 基于 SVM 的分类结果及分析

1. 实验结果

基于 SVM 的分类实验结果如表 4-3、4-4、4-5 所示。

表 4-3 基于 SVM 分类实验结果

实验序号		①	②	③	④	⑤	⑥	⑦	⑧	⑨
参数 C		1	2	3	4	5	6	7	8	9
参数 gamma		“auto”								
训练集上的得分	poly	0.76	0.84	0.86	0.86	0.92	0.94	0.96	0.98	1.0
	rbf	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
测试集上的分类准确率（%）	poly	62.96	62.96	62.96	66.7	66.7	66.7	66.7	66.7	**70.37**
	rbf	**74.07**	59.26	59.26	59.26	59.26	59.26	59.26	59.26	59.26

说明：①参数”C”表示误差项的惩罚系数；②参数” gamma”表示 SVM 采用径向基核函数时的参数；③”poly”表示采用多项式核函数，“rbf”表示采用径向基核函数。

表 4-4 基于 SVM 分类实验结果（仅多项式核函数）

实验序号	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫
参数 C	1	2	3	4	5	6	7	8	9	10	11	12
训练集上得分	0.76	0.84	0.86	0.86	0.92	0.94	0.96	0.98	1.0	1.0	1.0	1.0
测试集准确率（%）	62.96	62.96	62.96	66.67	66.67	66.67	66.67	66.67	**70.37**	66.67	62.96	62.96

表 4-5 基于 SVM 分类实验结果（仅径向基核函数）

实验序号	①	②	③	④	⑤	⑥	⑦	⑧	⑨
参数 C	4	3	2	1	0.85	0.75	0.5	0.25	0.1
（gamma=“auto”）									
训练集上的得分	1.0	1.0	1.0	1.0	0.98	0.94	0.90	0.78	0.64
测试集上的分类准确率（%）	59.26	59.25	59.26	74.07	74.07	**77.78**	77.78	62.96	66.67
（gamma=0.01）									
训练集上的得分	1.0	1.0	0.98	0.88	0.84	0.82	0.80	0.64	0.64
测试集上的分类准确率（%）	59.26	70.37	**77.778**	77.78	77.78	77.78	77.78	66.67	66.67
（gamma=0.1）									
训练集上的得分	1.0	1.0	1.0	1.0	1.0	1.0	0.98	0.64	0.64
测试集上的分类准确率（%）	66.67	66.67	66.67	66.67	66.67	**74.07**	62.96	66.67	66.67
（gamma=0.5）									
训练集上的得分	1.0	1.0	1.0	1.0	1.0	1.0	0.64	0.64	0.64
测试集上的分类准确率（%）	66.67	66.67	66.67	66.67	66.67	**66.67**	66.67	66.67	66.67
（gamma=10）									
训练集上的得分	1.0	1.0	1.0	1.0	1.0	1.0	0.64	0.64	0.64
测试集上的分类准确率（%）	66.67	66.67	66.67	66.67	66.67	**66.67**	66.67	66.67	66.67

4. 实验结果分析

分析表 4-3、4-4、4-5 中的实验结果可得：

1) 分类器参数的设置对分类器性能有较大影响；
2) 表 4-3 的实验结果表明基于 SVM 的分类器模型采用不同的核函数时，对惩罚系数"C"的适应性有很大不同。
3) 表 4-4 表明，基于多项式核函数的 SVM 分类器模型随着惩罚系数"C"增大，分类器在训练集上的得分越来越高，并从第⑦、⑧、⑨组实验结果可见，随着分类器在训练集上的得分越来越高，其在测试集上的精度也有所提高。但是当继续增加惩罚系数"C",即对误差项的容忍持续增大时，尽管在训练集上得分很高，但在测试集上的结果反而出现下降的趋势。
4) 表 4-5 表明，基于径向基核函数的 SVM 分类器，同时受到两个关键参数的影。对于参数"C"的影响与基于多项式核函数的类似，不在赘述。对于参数 gamma 的影响，对比参数 gamma 分别等于 0.01、0.1、0.5 三个大组的实验结果可得，参数 gamma 过大时，模型会更精确拟合训练集中的每个样本，从而容易导致过拟合问题，降低模型的泛化能力，尽管会议较快的速度在训练集上收敛，但是在测试集上的精度反而更低。

4.4 几种方法的对比分析

对每种分类器模型，取以上实验中在测试集上最好的结果记录如表 4-6 所示。由表 4-6 所示，三个分类器模型中，基于 pocket 算法和 LMS 算法的分类器模型所得到的最优的分类准确率较高，而多项式核函数相对较低。

1) 从 SVM 采用两种不同的核函数得到了不同的分类精度可以初步认为径向基核函数相对多项式核函数而言的更适合我们本次实验分类任务。
2) 基于 pocket 算法和 LMS 算法的分类器模型具有相同的最优分类结果，猜测二者具有很大的相似性。尽管二者采用了不同的代价函数，有不同的优化目标，但是二者的代价函数均与期望输出（y）和实际输出（W^TX）相关，因此无法否定二者存在一定的等价性或相似性。
3) 基于 pocket 算法和 LMS 算法的分类器模型性能优于 SVM 的性能：一方面有可能是因为没有选择最适合模式特征的核函数；另一方面粗粒度的参数调整错过了 SVM 分类结果最优时的参数配置。

表 4-6 三种分类器最优结果对比

	pocket	LMS	SVM	
			多项式核函数	径向基核函数
最好分类结果	85.18	85.18	70.37	77.78

4.5 其他分析

由于时间关系没有对可能影响模型性能的一些因素进行实验验证和分析，现在简单的对可能的情况进行讨论：

1. 在采用 PCA 提取特征，进行特征空间降维时，在选择协方差矩阵的最大的 51 个特征值时，没有对特征值取绝对值再选择最大的，因此会存在一定的影响，但是不清楚正负特征值个数所占的比例以及降维后具体如何影响新的特征空间，所以具体对模型造成了什么影响不确定，有时间可以对其验证一下。

2. 在采用 PCA 提取特征，进行特征空间降维时，固定采用了最低可降维数 51，因此可以考虑进行特征空间降维时保留更高的维数，也即保留原始特征空间中的更多特征以训练分类器模型，猜想应该能够得到性能更好的分类器模型，但是也不能排除过拟合问题的可能。
3. 在调整 SVM 分类器模型的参数时，没有进行更细粒度的参数配置（例如 C 的取值个数更多一些和间隔更小一些），因此目前记录的最好的分类结果有可能并不是最好的。