
天津大学

模式识别与深度学习课程

实验 2、PCA 降维 SVM 分类算法实验报告



学 院 智能与计算学部
专 业 计算机与科学技术
学 号 3019244140
姓 名 郭思齐

1 . 实验目标

实验二——PCA 降维与 SVM 分类算法，实验总目标是帮助我们掌握 PCA 将维算法原理及代码实现，对已有算法灵活调用获取降维重建结果。根据实验数据的特性调节 PCA 降维数以及 SVM 线性函数或核函数。其中两个小实验的目标分别是：

1. 将 PCA 人脸降维重建算法理解，更改给出的重建代码，实现对本班级采集人脸数据进行 PCA 降维重建实验，并调试参数 `n_components` 的不同取值，查看不同取值对实验结果的影响，将结果记录分析。
2. 补全 `pca_svm.py` 代码，使用本班级人脸采集数据进行 PCA 降维，然后使用降维后的数据进行 SVM 分类实验。调试不同参数，查看不同参数对实验结果的影响，将不同参数对应的训练集和测试集准确率通过表格记录，并对结果进行分析。

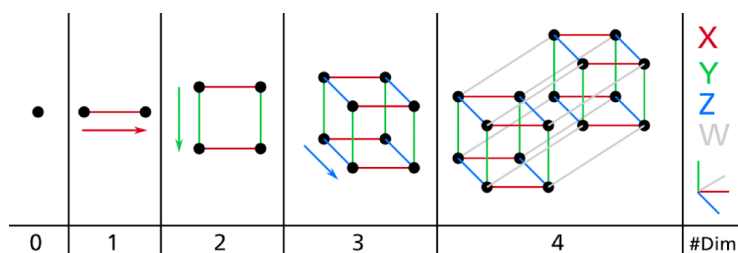
2. 实验一

2.1 算法实现及参数调节说明

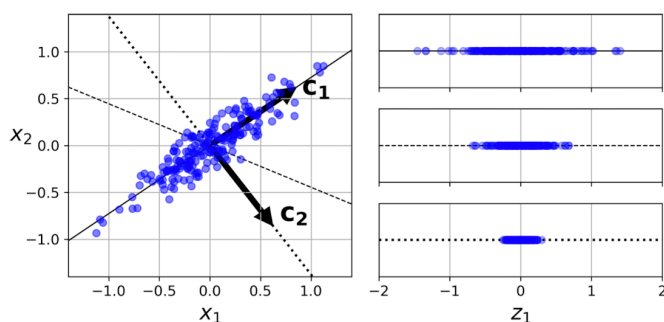
应用 PCA 降维模型对灰度化处理后的图片进行降维，并利用 PCA 重建模型和降维后的主成分向量进行人脸的重建。

PCA 的引入：

1. 随着维度的增长，点间的距离增大。维度越大，过拟合的风险越高。除此以外，空间的复杂度也会如下图一般增大。



2. 在每个训练实例涉及到成千上万的特征时，机器学习相关的训练会耗时巨大，并且不容易获得优解，导致维度灾难，采用诸如 PCA（映射）或 LLE（流形学习）这样的算法可以对高维数据维数约简、降维处理。
3. PCA 定义一个超平面并将数据映射到上面。PCA 可能会通过找到保存最多训练变量的轴或是最小化卷方差距的轴，选取为需要引射到的轴。这些轴就是 Principal Component (PC)。



通过 SVD（Singular Value Decomposition）来将训练集矩阵 X 分解为三个矩阵乘积 $U \Sigma V^T$ 。其中 V 是主成分矩阵。

$$V = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

4. PCA 用来压缩（映射到 d 维度），解压可以有如下的等式：

$$X_{d-proj} = X W_d$$

$$X_{recovered} = X_{d-proj} W_d^T = X W_d W_d^T$$

图像重建也是基于这样的思想。

算法实现的核心：

1. 数据导入、预处理以及灰度化：

```
def dataload(path):
    all_image = []
    re_path = []

    for face_path_name in os.listdir(path):
        #//print(face_path_name)
        if face_path_name == '.DS_Store':
            #guosiqi
            continue
        face_path = os.path.join(path, face_path_name)
        #./class4-facedata-rgb/guosiqi

        for image_path_name in os.listdir(face_path):
            #guosiqi.jpg

            image_path = os.path.join(face_path, image_path_name) #./class4-facedata-rgb/guosiqi/guosiqi.jpg

            class4_person_dir = os.path.join(path.split('-')[0] + '_recon', face_path_name)
            if not os.path.exists(class4_person_dir):
                os.makedirs(class4_person_dir)
            re_path.append(os.path.join(class4_person_dir, image_path_name))

            img_gray = Image.open(image_path).convert('L')
            img_gray.save(os.path.join(os.path.join(path.split('-')[0] + '_recon', face_path_name, image_path_name)))
            img_np = np.array(img_gray)
            all_image.append(img_np)

    all_image = np.array(all_image) #(51, 250, 250)
    all_image_flatten = all_image.reshape((all_image.shape[0], -1)) # (51, 62500) 62500 = 250*250
    return all_image_flatten, re_path

all_image_flatten, re_path = dataload('./class4-facedata-rgb')
```

2. PCA 降维以及图片重建：

```
model = PCA(n_components=components_value) # n_components <= min(n_samples, n_features)

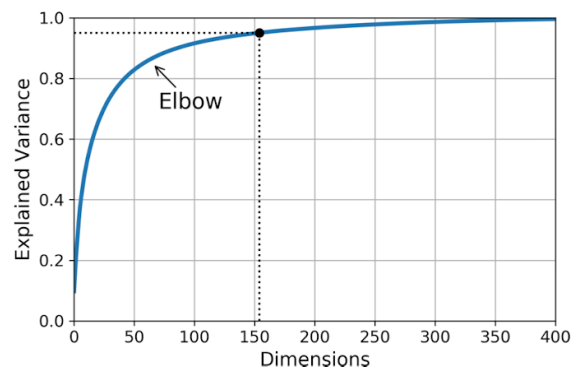
components = model.fit_transform(all_image_flatten)

face_recon = model.inverse_transform(components) # (51, 62500)
```

































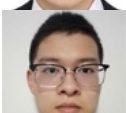







3. 数据灰度化图像保存：

```
# Reshape the reconstructions, gray them and save them.
for i in range(face_recon.shape[0]):
    each_face_recon = face_recon[i]
    each_face_recon = Image.fromarray(each_face_recon.reshape((250,250)))
    each_face_recon = each_face_recon.convert('L') #img_gray
    each_face_recon.save(re_path[i].split('.jpg')[0] + '-recon-' + str(components_value) + '.jpg')
```

4. 使用多组 $n_component$ 作为传参比较实验。其中 PCA 的 $n_components$ 与可解释变量的比例有一定的关系，如下图。(1) 当它被设置为 0-1 之间的浮点数，这就表示希望保留的信息量的比例。(2) 也可以设置其参数为 "mle"，即让 PCA 用最大似然估计自己选择超参数。(3) 表示将至的维数。



5. 实验结果整理：

Reconstruction	Original Photo	n_component = 1	n_component = 5	n_component = 10	n_component = 20	n_component = 30	n_component = 40	n_component = 50
Gao Han								
Gao Leyu								
Guo Siqi								
Guo YongXu								
Hua Yi								

2.2 结果分析

使用的 n_components 越大，在 PCA 映射的时候维度就越多，相当于在压缩数据的时候，丢失的信息就越少，重建后表示的维度就越高，恢复后可表示的信息也越多，和原图相比就越相似。这里可以看出，降维不是完全可逆的。

其次，对比不同的同学，可以看出来背景色与肤色反差度大的同学面貌经 PCA 映射的时候，信息重建时更完整。而本身图像不够清晰或者饱和度比较弱的图像自然很容易就丢失掉特有的信息，PCA 重建的时候会更困难。

3. 实验二

3.1 算法实现说明

这一部分在 sklearn 库的基础上实现较为简单。完善好 PCA 和 SVM 两个函

数，把降维的样例用 SVM 分类。

```
def pca_function(all_images, components_value):
    # 实现降维功能，返回得到的降维数组
    model = PCA(n_components=components_value)
    components = model.fit_transform(all_images)

    return components

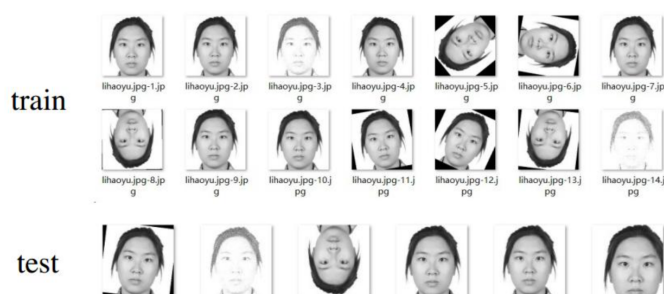
def svm_poly(X_train, X_test, Y_train, Y_test, cc):
    # 实现svm分类功能，得到分类结果
    clf = svm.SVC(C=cc, kernel='poly', decision_function_shape='ovr')
    clf.fit(X_train, Y_train)

    trainAcc = clf.score(X_train, Y_train)
    testAcc = clf.score(X_test, Y_test)
    print('training prediction:%.3f' % trainAcc)
    print('test data prediction:%.3f' % testAcc)
```

数据集说明：

1. train_num = 714 # 训练集样本数
2. 数据集增强：
 - 每位同学的人脸数据进行了20倍数据增广。包括随机旋转，上下左右翻转，加噪声，随机剪裁
 - 划分了训练集和测试集，每位同学对应的训练集包含14个样本，测试集包含7个样本

3. 数据集样例：



SVM 降维人脸分类结果展示：

1. 在实验一中发现 ovo 与 ovr 的结果差别不大，于是在实验二中没有统计该对比结果)
2. 对比实验一，使用整型的 n_components 作为 PCA 参数统计 n_components, kernel, 系数 C 的结果：

主元	C		0.01	0.1	1	10	100
	Kernel						
10	Poly	Train Acc	0.223	0.244	0.613	0.93	1
		Test Acc	0	0	0.134	0.408	0.51
	Rbf	Train Acc	0.45	0.45	0.655	0.966	1
		Test Acc	0	0	0.101	0.493	0.51
	Sigmoid	Train Acc	0.183	0.183	0.273	0.311	0.293
		Test Acc	0	0	0	0.092	0.095
20	Linear	Train Acc	1	1	1	1	1
		Test Acc	0.49	0.49	0.49	0.49	0.49
	Poly	Train Acc	0.238	0.256	0.674	0.965	1
		Test Acc	0	0	0.196	0.448	0.513
	Rbf	Train Acc	0.483	0.483	0.73	0.997	1
		Test Acc	0	0	0.267	0.529	0.529
30	Sigmoid	Train Acc	0.213	0.213	0.367	0.471	0.485
		Test Acc	0	0	0.003	0.154	0.173
	Linear	Train Acc	1	1	1	1	1
		Test Acc	0.503	0.503	0.503	0.503	0.503
	Poly	Train Acc	0.246	0.272	0.695	0.969	1
		Test Acc	0	0	0.206	0.448	0.507
40	Rbf	Train Acc	0.494	0.494	0.769	1	1
		Test Acc	0	0	0.17	0.553	0.553
	Sigmoid	Train Acc	0.235	0.234	0.41	0.534	0.555
		Test Acc	0	0	0.003	0.229	0.242
	Linear	Train Acc	1	1	1	1	1
		Test Acc	0.5	0.5	0.5	0.5	0.5
50	Poly	Train Acc	0.265	0.291	0.696	0.972	1
		Test Acc	0	0	0.206	0.431	0.484
	Rbf	Train Acc	0.504	0.504	0.709	1	1
		Test Acc	0	0	0.176	0.523	0.523
	Sigmoid	Train Acc	0.246	0.246	0.437	0.581	0.616
		Test Acc	0	0	0.007	0.268	0.281
50	Linear	Train Acc	1	1	1	1	1
		Test Acc	0.507	0.507	0.507	0.507	0.507
	Poly	Train Acc	0.275	0.301	0.697	0.976	1
		Test Acc	0	0	0.203	0.431	0.471
	Rbf	Train Acc	0.517	0.517	0.8	1	1
		Test Acc	0	0	0.186	0.526	0.526
50	Sigmoid	Train Acc	0.258	0.258	0.451	0.604	0.668
		Test Acc	0	0	0.007	0.265	0.275

3. 对比实验二，使用浮点型（比例）的 `n_components` 作为 PCA 参数统计

n_components, kernel, 系数 C 的结果：

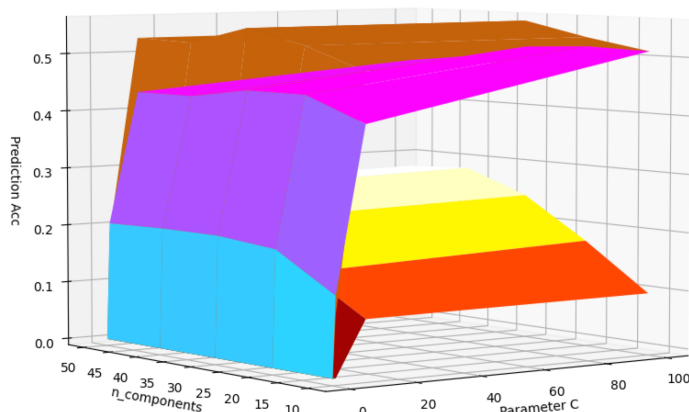
主元	C		0.01	0.1	1	10	100
	Kernel						
20%	Poly	Train Acc	0.034	0.046	0.032	0.038	0.038
		Test Acc	0	0	0	0	0
	Rbf	Train Acc	0.085	0.085	0.094	0.118	0.144
		Test Acc	0	0	0	0	0
	Sigmoid	Train Acc	0.05	0.055	0.042	0.078	0.07
		Test Acc	0	0	0	0.007	0.003
40%	Poly	Train Acc	0.098	0.105	0.164	0.227	0.255
		Test Acc	0	0	0.013	0.016	0.02
	Rbf	Train Acc	0.182	0.182	0.228	0.314	0.412
		Test Acc	0	0	0	0.033	0.078
	Sigmoid	Train Acc	0.087	0.087	0.055	0.062	0.055
		Test Acc	0	0	0	0	0.1
60%	Poly	Train Acc	0.175	0.213	0.521	0.842	0.989
		Test Acc	0	0	0.092	0.34	0.461
	Rbf	Train Acc	0.394	0.394	0.559	0.919	1
		Test Acc	0	0	0.85	0.435	0.493
	Sigmoid	Train Acc	0.151	0.154	0.19	0.211	0.221
		Test Acc	0	0	0	0.052	0.088
80%	Poly	Train Acc	0.244	0.269	0.686	0.968	1
		Test Acc	0	0	0.209	0.454	0.51
	Rbf	Train Acc	0.487	0.487	0.751	1	1
		Test Acc	0	0	0.17	0.529	0.529
	Sigmoid	Train Acc	0.225	0.225	0.385	0.496	0.517
		Test Acc	0	0	0.003	0.199	0.222
95%	Poly	Train Acc	0.311	0.335	0.718	0.989	1
		Test Acc	0	0	0.203	0.395	0.425
	Rbf	Train Acc	0.55	0.55	0.874	1	1
		Test Acc	0	0	0.18	0.493	0.493
	Sigmoid	Train Acc	0.273	0.275	0.496	0.756	0.821
		Test Acc	0	0	0.013	0.32	0.382

3.2 结果分析

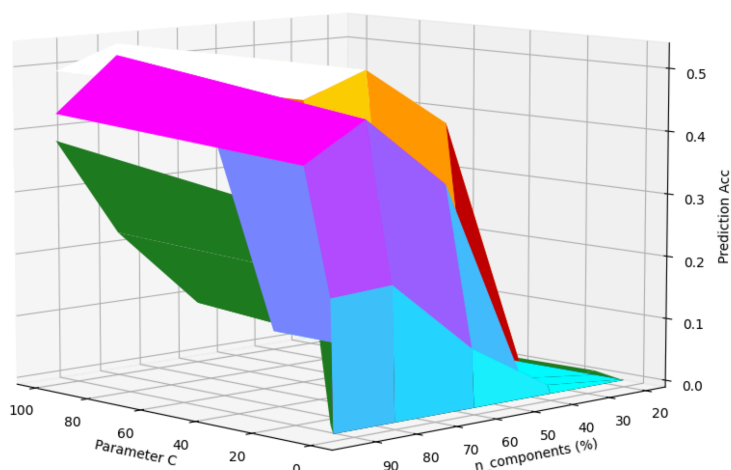
根据得到的实验结果可视化并分析：

1. 首先，在 n_components = 10, 20, 30, 40, 50 的情况下，针对 rbf(最上层棕

色)、poly (中间层冷色) 和 sigmoid (最下层暖色) 三种 kernel 作出参数 $n_components$ 和 C 与最后预测准确度的关系示意图：



2. 其次，在 $n_components = 20\%, 40\%, 60\%, 80\%, 95\%$ 的情况下，针对 rbf (最上层暖色)、poly (中间层冷色) 和 sigmoid (最下层绿色) 三种 kernel 作出参数 $n_components$ 和 C 与最后预测准确度的关系示意图：



在表达关系上，单张图片传递的信息有限，后续的分析结果已经结合其他的折线图如针对 kernel 或者 C 或者 $n_components$ 单变量分析图。

3. 总结：

- (1) 人脸图像信息在被 PCA 降维后会丢失信息，但是实际上对后续 SVM 分类的结果影响有限，甚至在出现降维后反而使得分类结果提升的个别情况，比如 $\text{Prediction Acc}(n_components=40, \text{kernel=poly}) > \text{Prediction Acc}(n_components=50, \text{kernel=poly})$ ，可能是因为又一些特征值冗余，影响了实验结果。
- (2) 在比较内核后发现，即使 PCA 对数据降维，Linear 内核的 SVM 分类结

果受到的影响较小，但是耗时巨大，可能是因为模型过于简单，或者实验数据线性不可分。今后可能会对这个现象有更深刻的认识。

但是，可以从 Acc 的表现和 n_components 两者之间的关系看出，sigmoid 受到降维影响最大。

整体来看，在这次试验中，SVM 的结果 rbf 优于 poly，更优于 sigmoid。

- (3) 惩罚系数对 Acc 的影响在 0-15 左右变化大，后续经历一个 elbow 拐点影响几乎消失。

在惩罚系数较小的时候，降维压缩会降低最终的 Acc 表现。

- (4) 对比了从主成分数量的角度和从保留变量比例的角度来看 SVM 的表现，变化趋势的区别不大，当然这也是和主成分数量和保留训练变量比例之间的关系有关的。

- (5) 虽然本实验中没有统计运行时长，但是可以肯定，经过降维后的数据进行 SVM 分类的时间消耗必然可以有所降低。而且，时间成本的降低并没有牺牲过多的转准确度。总的来说，PCA 减少了特征数，加快了计算速度；从图像对比来看，它依然保留了大部分重要特征，仍能够分辨得出人脸。因此，PCA 降维算法在特征选择中作用很大。

4. 总结

实验二相较实验一更进一步，除了应用到 SVM 分类算法以外，加入了 PCA 主成分降维算法，以及对降维后的数据进行 SVM 的分类实际应用。

在这一实验中，深入学习了使用 LFW 数据集进行 PCA 降维重建的工程案例，并迁移学习，自己动手对班级内同学们的人脸图像进行 PCA 降维重建，以及运用之前所学的 SVM 支持向量机算法对降维后的 SVM 分类。在调节变量的过程中，也感受到了 PCA 降维算法对后续 SVM 分类加速的不错的效果。