数字图像实验

PCA 人脸识别 PCA

算法描述&结果对比

目 录

1.	算法描述	
1	数学原理(公式)	1
2.	结果	2
3.	思考与优化2	
1	思考	2
2	优化	3

1. 算法描述

1. 数学原理

1. 基于课件上的资料与网上的资料,我对 PCA 的理解大致如下:在误差允许的范围下,将一个高维的数据集,寻找一个数据集分布较为集中的低维空间,将数据投影到此空间上,即,其他坐标设为 0。(如三维空间 XYZ,将数据投影到 XY 平面上即将 z 坐标设为 0)。在矩阵中,这个空间,可用矩阵的特征向量来构成。具体操作为,求此矩阵的协方差矩阵 Σ (n*n),利用此其最大的 h 个特征值对应的特征向量,构成 (n*h) 的矩阵 Ω ,用 Ω 的逆矩阵乘以原数据集,即可获得降维数据集。

2. 数学步骤如下:

1. 假设有 n 幅尺寸为 112*92 的图像,将每一副图像排成一个列向量,构成(112*92)* n 的 ${}_{\mathrm{H}}{}_{\mathrm{E}}X=[x_1,\ldots,x_n].$

2. 计算均值
$$\mu = \frac{1}{n} \sum_{j=1}^{n} + x_j$$
, $\Rightarrow H = \frac{1}{\sqrt{n-1}} [x_1 - \mu, \dots, x_n - \mu]$,

- 3. 根据定义, 计算协方差矩阵 $\Sigma = HH^T$;
- 4. 计算 Σ 的特征值与特征向量,取前 h 个最大特征值所对应的特征向量,构成矩阵 Φ ;
- 5. 矩阵 Φ 可对数据降维: $\Phi^T X = Y$, Y 是 h 行 n 列的矩阵, 也就是将数据从 112*92 维降为 h 维。
- 6. 利用降维数据集的每一个种类,(此次实验中有 40 种), 对其降维后的数据取均值向量, 将此向量作为此种类的匹配向量。
- 7. 对于每一个进行测试的向量,利用矩阵 Φ 对其降维。之后与每个种类的匹配向量作最小二范数匹配。匹配程度最高的,记录下匹配组号。最后计算正确率。将结果输出到文件中。

2. 结果

1. 储存在主目录的 Output 目录下的 output. txt 中。

□ 名称	修改日期	类型	大小
output(QR).txt	2018/2/13 18:11	文本文档	1 KB
output.txt	2018/2/12 2:21	文本文档	1 KE
有3个错误图像为s5的7.pgm 育4个错误图像为s10的7.pgm 育4个错误图像为s15的8.pgm 育5个错误图像为s18的9.pgm 育6个错误图像为s23的7.pgm 育7个错误图像为s27的8.pgm 育8个错误图像为s28的9.pgm 育9个错误图像为s30的8.pgm 育10个错误图像为s30的9.pgm 育11个错误图像为s35的7.pgm 育13个错误图像为s40的7.pgm 育14个错误图像为s40的7.pgm			

3. 思考与优化

1. 思考

1)运行的匹配正确率仅为88.33,说实话有点低了。经过思考,不难发现,这是因为我偷懒了,没有按照随机取7张作训练集3张作测试集,而是固定选取前七张作训练集,后三张当测试集,证明前七张和后三张有毒。此问题,不予优化。2)程序运行时,效率出奇的低,而且占用电脑的CPU和内存也很大。

名称	状态	80% CPU	84%	27% 磁盘	0%
应用 (5)					
▶ MATLAB R2016a		54.9%	4,429.0	0 MB/秒	0 Mbps

运行时间在 15 分钟~20 分钟不等。且我在程序中及时清除无关的变量,释放内存,但是依旧没有什么改观。经过断点运行排查发现,是利用 Matlab 内部算法计算特征值时,由于矩阵过大,占用的资源过多。所以考虑使用 QR 分解进行优化。

```
函数调用堆栈:
                   步进 步出
23 2 3
              暂停
                    区 运行到光标处
10
项目 ▶ 基于PCA的人脸识别-元东霖-15331383 ▶ Code
☑ 编辑器 - Q:\大三上\数图\期末\期末项目\基于PCA的人脸识别-元;
   train.m × QRtrain.m × +
 22
 23
       %用于计算协方差矩阵的H矩阵
 24 -
       H=zeros(height*width, group*num);
 25 - ☐ for i=1:group*num
        H(:, i) = imgdata(:, i)-imgu;
 26 -
 27 -
       - end
 28
        %协方差矩阵
       Cov=H*H'/(group*num-1);
 29 -
 30
        %主成分分析
 31 🔵
       [PC, ~, ~] = pcacov(Cov');
 32 —
        clear Cov H imgu;%清除变量释放内存
 33 —
       Dimension = 80;%设置维数为80
        lowDimension=(PC(:,1:Dimension))';%降维矩阵
 34 -
 35 —
       ldImgData = lowDimension * imgdata;%降维后的数据矩阵。
 36 -
       clear imgdata PC;
 37
```

2. 优化

```
1.已知\Sigma = HH^T,其中\Sigma为d^*d,H为d^*n,d代表原始数据的维数,n代表样本数,d远大于n;
```

2.对H作QR分解, h = QR, 其中Q为d*t, R为t*n, $1 \le t \le n$;

 $3.则\Sigma = QRR^TQ^T$, 对 R^T 作奇异值分解 $R^T = UDV^T$, 其中U为n*t, V为t*t, $D = diag(\sigma_1, \dots, \sigma_t)$;

4.于是 $\Sigma = QVDU^TUDV^TQ^T = QVD^2V^TQ^T = QV\Lambda V^TQ^T$,其中 $\Lambda = D^2$;

5.由于 $(QV)^T(QV) = V^TQ^TQV = V^TV = I$,所以QV可将 Σ 对角化,QV为 Σ 的特征向量矩阵, Λ 为 Σ 的特征值矩阵;

6.选取D前h个最大对角元所对应于V中的h个列,构成t*h的矩阵 V_h ,则降维矩阵 $\Phi = QV_h$;

优化后部分代码:

```
%QR分解
[Q,R] = qr(H);
clear H imgu;%清除变量释放内存
%奇异值分解
[U,S,V] = svd(R');
```

```
Dimension = 80;%设置维数为80
lowDimension=(Q*V(:,1:Dimension))';%降维矩阵
ldImgData = lowDimension * imgdata;%降维后的数据矩阵。
clear imgdata;
```

优化后结果保存在 Output 目录下的 output(QR),txt,可喜可贺的是,经过优化,不仅运行时间和占用内存比以前大大降低,匹配准确率也有一丁点增长。可能是优化之后,内部计算的

时候由于维度变小更精准了吧。

正确率: 百分之89.17 第1个错误图像为s4的9.pgm 第2个错误图像为s5的7.pgm 第3个错误图像为s10的7.pgm 第4个错误图像为s15的8.pgm 第5个错误图像为s18的9.pgm 第6个错误图像为s23的7.pgm 第7个错误图像为s27的8.pgm 第8个错误图像为s28的9.pgm 第10个错误图像为s30的8.pgm 第11个错误图像为s35的7.pgm 第11个错误图像为s36的7.pgm 第13个错误图像为s36的7.pgm