表情特征信息的降维和提取

杨超群 2015141212025

2017 年 12 月 5 日

目录

1 对PCA算法的学习和理解 **2**

1.1 算法介绍 2

1.2 算法核心 2

1.3 算法适用情况 2

1.4 算法的局限性 2

2 PCA算法的数学推导 **3**

3 在MATLAB上通过PCA优化表情识别 **5**

3.1 简要过程 5

3.2 结果比较 9

4 改进方向 **11**

5 自我评价 **11**

1 对PCA算法的学习和理解

1.1 算法介绍

在多元统计分析中, 主成分分析 (Principal components analysis, PCA) 是一种分析、简化数据集的技术.PCA经常用于减少数据集的维数, 同时保持数据集中的对方差贡献最大的特征(主成分).

PCA是最简单的以特征量分析多元统计分布的方法.通常情况下, 这种运算可以被看作是揭露数据的内部结构, 从而更好的解释数据的变量的方法.如果一个多元数据集能够在一个高维数据空间坐标系中被显现出来, 那么PCA就能够提供一幅比较低维度的图像, 这幅图像即为在讯息最多的点上原对象的一个'投影'. 这样就可以利用少量的主成分使得数据的维度降低了.

主成分分析在分析复杂数据时尤为有用，比如人脸识别.

1.2 算法核心

在最小均方误差意义下, 寻找数据的协方差矩阵的主轴 (特征值最大的特征向量) , 由主轴构成新的坐标系 (按特征值从大到小, 根据所需维数要求, 选取相应的特征向量) , 从而将数据由原坐标系向新坐标系投影.

1.3 算法适用情况

1. 当给定数据集具有较多特征时, 通过PCA算法降维可以降低数据的复杂性, 使我们有更直观的感受;
2. 当需要直观的观察时, PCA算法可以将高维数据降到二维和三维 (信息损失程度不同) ;
3. 当数据量很大时, PCA算法可以去除冗余信息;
4. 当需要不同视角观察时, 不同特征向量的选取, 可以得到不同的结果.

1.4 算法的局限性

1. 仅考虑了原始变量的正交变换;
2. PCA仅依赖于样本数据的均值和协方差矩阵, 有些分布(比如多元正态分布) 可以由这两个量刻画, 有些不行;
3. 当原始变量是相关的时候, 使用 PCA 可以降低维数, 如果原始变量不相关, 则不能降维;
4. PCA 受异常点影响, 没有刻度不变性.

2 PCA算法的数学推导[1]

假设在$\mathbb{R}^n$空间中有m个点$\boldsymbol{x}^{(1)},\cdots,\boldsymbol{x}^{(m)}$, 我们希望在精度损失尽可能少的情况下, 使用更少的内存去储存这些点. 如果对于每一个点$\boldsymbol{x}^{(i)}\in\mathbb{R}^n$, 有一个对应的编码向量 $\boldsymbol{c}^{(i)}\in\mathbb{R}^l$. 如果$l<n$, 那么就使用了更少的内存来储存原来的数据. 通过找到一个编码函数, 根据输入返回编码, $f(\boldsymbol{x})=\boldsymbol{c}$;再找到一个解码函数, 给定编码重构输入, $\boldsymbol{x}\approx g(f(\boldsymbol{x}))=g(\boldsymbol{c})$.

为了简化解码器, 可以使用矩阵乘法将编码映射回$\mathbb{R}^n$, 即$g(\boldsymbol{c})=\boldsymbol{D}\boldsymbol{c}$, 其中$\boldsymbol{D}\in\mathbb{R}^{n\times l}$. 为了使问题有唯一解, 需要限制$\boldsymbol{D}$ 中所有列向量都有单位范数.为了使编码问题简单一些, PCA限制$\boldsymbol{D}$的列向量彼此正交.

在PCA算法中, 使用$\boldsymbol{L}^2$范数,

$$\boldsymbol{c}^*=arg\min_{\boldsymbol{c}}\|\boldsymbol{x}-g(\boldsymbol{c})\|_2 \tag{1}$$

因为平方$\boldsymbol{L}^2$范数和$\boldsymbol{L}^2$范数在相同的$\boldsymbol{c}$上取得最小值, 故上式可变为

$$\boldsymbol{c}^*=arg\min_{\boldsymbol{c}}\|\boldsymbol{x}-g(\boldsymbol{c})\|_2^2 \tag{2}$$

即

$$\boldsymbol{c}^*=arg\min_{\boldsymbol{c}}(\boldsymbol{x}-g(\boldsymbol{c}))^{'}(\boldsymbol{x}-g(\boldsymbol{c})) \tag{3}$$

将等式右边化简得

$$\boldsymbol{x}^{'}\boldsymbol{x}-2\boldsymbol{x}^{'}g(\boldsymbol{c})+g(\boldsymbol{c})^{'}g(\boldsymbol{c}) \tag{4}$$

因为$\boldsymbol{x}^{'}\boldsymbol{x}$与$\boldsymbol{c}$无关, $g(\boldsymbol{c})=\boldsymbol{D}\boldsymbol{c}$和$\boldsymbol{D}$的列向量彼此正交, 故

$$\begin{aligned}\boldsymbol{c}^*&=arg\min_{\boldsymbol{c}}-2\boldsymbol{x}^{'}\boldsymbol{D}\boldsymbol{c}+\boldsymbol{c}^{'}\boldsymbol{D}^{'}\boldsymbol{D}\boldsymbol{c} &(5)\\ &=arg\min_{\boldsymbol{c}}-2\boldsymbol{x}^{'}\boldsymbol{D}\boldsymbol{c}+\boldsymbol{c}^{'}\boldsymbol{c} &(6)\end{aligned}$$

为解得$\boldsymbol{c}$, 对$\boldsymbol{c}$求导并使导数为0得

$$\begin{aligned}\nabla_{\boldsymbol{c}}\big(-2\boldsymbol{x}^{'}\boldsymbol{D}\boldsymbol{c}+\boldsymbol{c}^{'}\boldsymbol{c}\big)=0 &\quad(7)\\ -2\boldsymbol{D}^{'}\boldsymbol{x}+2\boldsymbol{c}=0 &\quad(8)\\ \boldsymbol{c}=\boldsymbol{D}^{'}\boldsymbol{x} &\quad(9)\end{aligned}$$

所以编码函数为

$$f(\boldsymbol{x})=\boldsymbol{D}^{'}\boldsymbol{x} \tag{10}$$

PCA重构操作为

$$r(\boldsymbol{x})=g(f(\boldsymbol{x}))=\boldsymbol{D}\boldsymbol{D}^{'}\boldsymbol{x} \tag{11}$$

[1]这一部分的内容参考了Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016 的2.12节.

再来确定编码矩阵$\boldsymbol{D}$, 使所有维数和所有点上的误差矩阵的Fribenius范数最小

$$\boldsymbol{D}^* = arg\min_{\boldsymbol{D}} \sqrt{\sum_{i,j} \big(\boldsymbol{x}_j^{(i)} - r(\boldsymbol{x}^{(i)})_j\big)^2}, \boldsymbol{D}^{'}\boldsymbol{D} = \boldsymbol{I}_l \tag{12}$$

先考虑$l = 1$的情况, 此时$\boldsymbol{D}$为向量, 记为$\boldsymbol{d}$, 则上式化简为

$$\boldsymbol{d}^* = arg\min_{\boldsymbol{d}} \sum_i \|\big(\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i)}\boldsymbol{d}\boldsymbol{d}^{'}\big)\|_2^2, \|\boldsymbol{d}\|_2 = 1 \tag{13}$$

记$\boldsymbol{X}$为各个点堆叠成的矩阵, 其中$\boldsymbol{X}_{i,:} = \boldsymbol{x}^{(i)'}$, 则上式可化为

$$\boldsymbol{d}^* = arg\min_{\boldsymbol{d}} \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\|_F^2, \boldsymbol{d}\|_2 = 1 \tag{14}$$

因为

$$arg\min_{\boldsymbol{d}} \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\|_F^2 \tag{15}$$

$$= arg\min_{\boldsymbol{d}} Tr\Big(\big(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big)^{'}\big(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big)\Big) \tag{16}$$

$$= arg\min_{\boldsymbol{d}} Tr\big(\boldsymbol{X}^{'}\boldsymbol{X} - \boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'} - \boldsymbol{d}\boldsymbol{d}^{'}\boldsymbol{X}^{'}\boldsymbol{X} + \boldsymbol{d}\boldsymbol{d}^{'}\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big) \tag{17}$$

(因为$\boldsymbol{X}^{'}\boldsymbol{X}$与$\boldsymbol{d}$无关)

$$= arg\min_{\boldsymbol{d}} -Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big) - Tr\big(\boldsymbol{d}\boldsymbol{d}^{'}\boldsymbol{X}^{'}\boldsymbol{X}\big) + Tr\big(\boldsymbol{d}\boldsymbol{d}^{'}\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big) \tag{18}$$

$$= arg\min_{\boldsymbol{d}} -2Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big) + Tr\big(\boldsymbol{d}\boldsymbol{d}^{'}\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big) \tag{19}$$

(循环改变迹运算中相乘矩阵的顺序不影响结果)

$$= arg\min_{\boldsymbol{d}} -2Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big) + Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\boldsymbol{d}\boldsymbol{d}^{'}\big) \tag{20}$$

(因为$\boldsymbol{d}^{'}\boldsymbol{d} = 1$)

$$= arg\min_{\boldsymbol{d}} -2Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big) + Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big), \boldsymbol{d}^{'}\boldsymbol{d} = 1 \tag{21}$$

$$= arg\min_{\boldsymbol{d}} -Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big), \boldsymbol{d}^{'}\boldsymbol{d} = 1 \tag{22}$$

$$= arg\max_{\boldsymbol{d}} Tr\big(\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^{'}\big), \boldsymbol{d}^{'}\boldsymbol{d} = 1 \tag{23}$$

$$= arg\max_{\boldsymbol{d}} Tr\big(\boldsymbol{d}^{'}\boldsymbol{X}^{'}\boldsymbol{X}\boldsymbol{d}\big), \boldsymbol{d}^{'}\boldsymbol{d} = 1 \tag{24}$$

所以最优的$\boldsymbol{d}$是$\boldsymbol{X}^{'}\boldsymbol{X}$最大特征值对应的特征向量.

对于$l = 1$, 以上推导得到了第一个主成分. 更一般的, 对于$1 < l < n$, 矩阵$\boldsymbol{D}$ 由$\boldsymbol{X}^{'}\boldsymbol{X}$的前$l$个最大的特征值对应的特征向量组成.

3 在MATLAB上通过PCA优化表情识别

3.1 简要过程

本项目[2] 的人脸表情识别算法采用的是前馈神经网络，通过反向传播算法优化权重矩阵，神经网络尺寸为$n \times 25 \times 5$，有一个中间层，其中n由输入数据维度决定，输出为一5维的单位向量，根据其单位1出现位置，分别表示惊讶、微笑、愤怒、失望和中性. PCA的实现由MATLAB的内置函数svd完成.

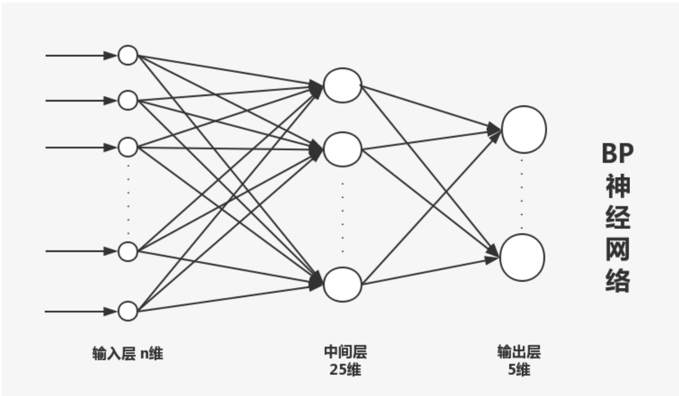


图 1: 神经网络结构

本项目的数据[3]为一1965×560的矩阵，代表1965个实例,每个实例是一560维向量，另有一1965×1的向量，其中各个元均为一1到5的整数，分别代表其对应实例的具体表情——惊讶、微笑、愤怒、失望和中性.先对数据进行特征标准化，使数据具有0均值和单位方差，以便于进行PCA处理和神经网络的权重矩阵的训练,

```
function [X_norm, mu, sigma] = featureNormalize(X)

%FEATURENORMALIZE Normalizes the features in X
%   FEATURENORMALIZE(X) returns a normalized version of X where
%   the mean value of each feature is 0 and the standard deviation
%   is 1.
mu = mean(X);
X_norm = bsxfun(@minus, X, mu);

sigma = std(X_norm);
X_norm = bsxfun(@rdivide, X_norm, sigma);

end
```

[2]由于所用代码较多，在此只列出部分，全部代码可在 https://github.com/OpenWaygate/Machine-Learning上下载.

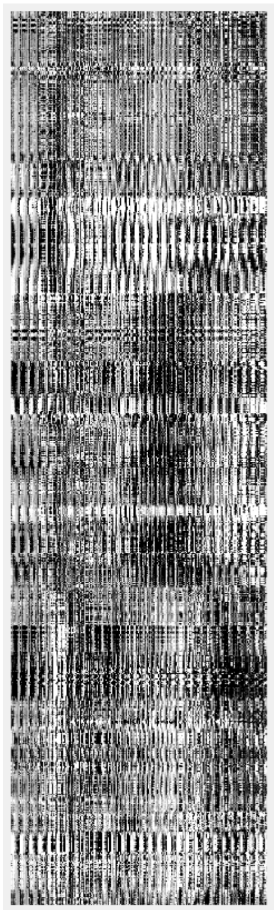
[3]数据集来自https://cs.nyu.edu/~roweis/data.html, Faces, Frey Face [data/frey_rawface.mat], From Brendan Frey. Almost 2000 images of Brendan's face, taken from sequential frames of a small video. Size: 20×28.

若不经过PCA的降维处理，神经网络的尺寸为560×25×5;在讯息损失不超过1%的情况下，经PCA处理后神经网络的尺寸降为226×25×5，输入维度变为原来的40.36%，大大减少了神经网络的复杂度.

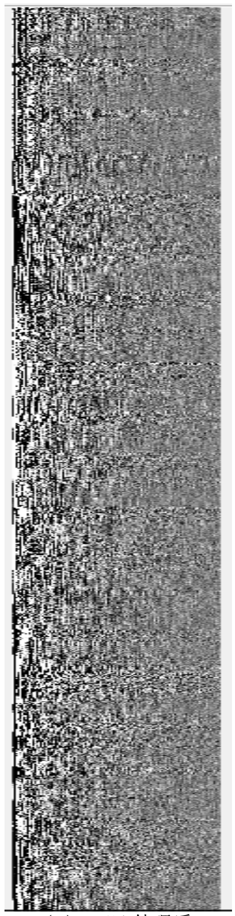
```
function [U, S] = pca(X)
%   PCA Run principal component analysis on the dataset X
%   [U, S] = pca(X) computes eigenvectors of the covariance matrix of X
%   Returns the eigenvectors U, the eigenvalues (on diagonal) in S

    [m, n] = size(X);

    U = zeros(n);
    S = zeros(n);
    sigma = X'*X/m;
    [U, S, ¬] = svd(sigma);

end
```

```
%  Run PCA
[U, S] = pca(X_pca);

% Choose input layer size K
for i = 1:size(S, 1)
    if sum(diag(S(1:i, 1:i)))/sum(diag(S)) > 0.99&&...
            sum(diag(S(1:i - 1, 1:i - 1)))/sum(diag(S)) ≤ 0.99
        K = i;
    end
end

% Show image as seen by the classifier
imshow(X_norm*U(:, 1:K), [-1, 1] );

% Setup the parameters
input_layer_size  = K;              % Input Images of Digits
hidden_layer_size = 25;             % 25 hidden units
num_labels = 5;                     % 10 labels, from 1 to 5
                                    % (note that we have mapped "0" to label 5)

% project original X to lower dimension Z
X_train = X_pca*U(:, 1:K);
y_train = y(sel(1:1179));

X_cv = X_norm(sel(1180:1572), :)*U(:, 1:K);
y_cv = y(sel(1180:1572));

X_test = X_norm(1573:end, :)*U(:, 1:K);
y_test = y(1573:end);
```

将所得数据显示以灰度图像如下:

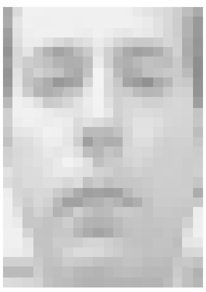


(a) PCA处理前



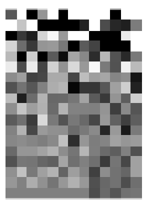
(b) PCA处理后

图 2



(a) 数据标准化前

(b) PCA处理前



(c) PCA处理后

图 3

可以看出PCA处理后被用于机器识别的特征被高度抽象.

将处理后所得数据随机分成比例为6:2:2的训练集、交叉验证集、测试集, 使用训练集训练神经网络的权重矩阵. 为防止出现过拟合, 加入惩罚参数λ, 选取范围为0.01到1.8, 步长为0.01.对每个λ, 将训练好的神经网络的权重矩阵用于预测交叉验证集的表情, 选择正确率最高的一组权重矩阵, 再用这组权重矩阵预测测试集所对应的表情, 得到最终正确率.

在使用PCA的算法中, 需先对训练集进行PCA处理, 将所得的特征向量作用于训练集、交叉验证集、测试集, 然后进行训练、预测.

```
% project original X to lower dimension Z
X_train = X_pca*U(:, 1:K);
y_train = y(sel(1:1179));

X_cv = X_norm(sel(1180:1572), :)*U(:, 1:K);
y_cv = y(sel(1180:1572));

X_test = X_norm(1573:end, :)*U(:, 1:K);
y_test = y(1573:end);

for lambda = 0.01:0.01:1.8

    % Create "short hand" for the cost function to be minimized
    costFunction = @(p) nnCostFunction(p, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, X_train, y_train, lambda);

    % CostFunction is a function that takes in only one argument (the
    % neural network parameters)
    [nn_params, cost] = fmincg(costFunction, initial_nn_params, options);

    % Obtain Theta1 and Theta2 back from nn_params
    Theta1 = reshape(nn_params(1:hidden_layer_size * (1 + input_layer_size)), ...
        hidden_layer_size, (input_layer_size + 1));

    Theta2 = reshape(nn_params((1 + (hidden_layer_size * (1 + ...
        input_layer_size))):end), num_labels, (hidden_layer_size + 1));

    % Implement Predict
    pred = predict(Theta1, Theta2, X_cv);
    fprintf('\nCross validation Set Accuracy: %f\n', mean(double(pred == y_cv))*100);
    accuracy(2) = mean(double(pred == y_cv))*100;

    % Refresh the highest predicting accuracy
    if accuracy(2) >= accuracy(1)
        theta1 = Theta1;
        theta2 = Theta2;
        Lambda = lambda;
        accuracy(1) = accuracy(2);
    end
end
```

3.2 结果比较

下图是未使用PCA的运行摘要[4]，列出了主要函数的运行情况，运行总耗时为2320.821秒，在$\lambda = 0.45$时在交叉验证集上的正确率最大，在测试集上的正确率为97.201018%，

函数名称	调用次数	**总时间**	自用时间*	总时间图 (深色条带 = 自用时间)
face	1	2320.821 s	1.347 s	
fmincg	180	2318.732 s	297.268 s	
...ze,num_labels,X_train,y_train,lambda)	271430	2021.464 s	10.291 s	
nnCostFunction	271507	2011.189 s	1837.198 s	
sigmoid	543376	174.039 s	174.039 s	
predict	181	0.422 s	0.373 s	
imshow	1	0.224 s	0.080 s	
initSize	1	0.093 s	0.013 s	
movegui	1	0.059 s	0.051 s	
checkNNGradients	1	0.027 s	0.004 s	
mean	362	0.023 s	0.023 s	
basicImageDisplay	1	0.022 s	0.007 s	
featureNormalize	1	0.021 s	0.005 s	
...den_layer_size,num_labels,X,y,lambda)	77	0.018 s	0.002 s	
optimset	1	0.016 s	0.006 s	
cell.strmatch	2	0.015 s	0.007 s	
isSingleImageDefaultPos	1	0.013 s	0.010 s	
close	1	0.013 s	0.004 s	
newplot	1	0.013 s	0.005 s	
std	1	0.012 s	0.000 s	
computeNumericalGradient	1	0.011 s	0.003 s	
var	1	0.011 s	0.011 s	
allchild	2	0.009 s	0.005 s	

图 4: 无PCA处理的运行摘要

[4]由于权重矩阵需先进行随机初始化，故每次运行的结果会有微小的差异，但使用PCA的结果都显著优于不使用PCA的结果．这里是任意选择的一组进行比较．

下图是使用PCA后的运行摘要，运行总耗时为1438.173秒，$\lambda = 0.41$时在交叉验证集上的正确率最大，在测试集上的正确率为99.745547%．

函数名称	调用次数	**总时间**	自用时间*	总时间图 (深色条带 = 自用时间)
face_pca	1	1438.173 s	1.029 s	
fmincg	180	1436.392 s	258.853 s	
...ze,num_labels,X_train,y_train,lambda)	268884	1177.540 s	9.492 s	
nnCostFunction	268961	1168.063 s	1006.537 s	
sigmoid	538284	161.574 s	161.574 s	
imshow	1	0.279 s	0.106 s	
predict	181	0.260 s	0.213 s	
pca	1	0.101 s	0.101 s	
initSize	1	0.087 s	0.014 s	
movegui	1	0.049 s	0.041 s	
basicImageDisplay	1	0.039 s	0.012 s	
checkNNGradients	1	0.027 s	0.005 s	
imageDisplayParseInputs	1	0.024 s	0.002 s	
optimset	1	0.024 s	0.017 s	
mean	362	0.023 s	0.023 s	
featureNormalize	1	0.022 s	0.005 s	
newplot	1	0.022 s	0.006 s	
close	1	0.019 s	0.005 s	
imageDisplayValidateParams	1	0.018 s	0.006 s	
...den_layer_size,num_labels,X,y,lambda)	77	0.017 s	0.002 s	
isSingleImageDefaultPos	1	0.013 s	0.010 s	
newplot>ObserveAxesNextPlot	1	0.013 s	0.002 s	
std	1	0.013 s	0.001 s	

图 5: PCA处理后的运行摘要

比较两篇运行摘要可以明显看出，使用PCA后运行时间大幅减少，后者为前者的61.97%，而且精确度在97.201018%的基础上提高到了99.745547%．这显示出使用PCA后前馈神经网络对人脸表情识别效率的巨大提升，达到了本项目的实验目的．

4 改进方向

思路1 PCA算法是在均方误差意义下, 可以尝试其他误差函数;

思路2 PCA算法是一种线性降维算法, 在对人脸图像向量化的时候破坏了图像的局部结构信息, 对识别率造成了影响;可以先使用非线性降维算法对样本集数据进行处理, 保留局部信息结构, 再用PCA算法对处理过的数据进行降维.

5 自我评价

在暑假中, 我认真学习了coursera上吴恩达的机器学习课程, 仔细分析了神经网络和主成分分析两节的作业代码, 通过课程提供的MATLAB License, 在MATLAB上实现了人脸表情识别, 通过分析比较使用PCA前后人脸表情识别的训练速度、准确率等, 得出了较好的结果.

通过这次的小火花项目的具体实践,我学习到了线性代数在现实领域的运用(PCA), 掌握了MATLAB的基本用法,接触了当下较热门的机器学习, 提高了自身的综合能力.