基于 PCA 算法实现数据降维

学号:20192131089　姓名: 吴宇涛

2021 年 10 月 5 日

目录

摘要

PCA（Principal Component Analysis）是一种常用的数据分析方法. PCA 通过线性变换将原始数据变换为一组各维度线性无关的表示, 可用于提取数据的主要特征分量, 常用于高维数据的降维. 本文主要介绍 PCA 算法使用案例; PCA 重要参数, 属性, 方法; PCA 算法的作用.

关键词: PCA; 数据降维; 数据挖掘; 数据处理

1 PCA 算法

1.1 PCA 介绍

主成分分析（Principal Component Analysis），是一种用于探索高维数据的技术。PCA 通常用于高维数据集的探索与可视化。还可以用于数据压缩，数据预处理等。PCA 可以把可能具有线性相关性的高维变量合成为线性无关的低维变量，称为主成分（principal components），新的低维数据集会尽可能的保留原始数据的变量，可以将高维数据集映射到低维空间的同时，尽可能的保留更多变量。

注意：降维就意味着信息的丢失，这一点一定要明确，如果用原始数据在模型上没有效果，期望通过降维来进行改善这是不现实的，不过鉴于实际数据本身常常存在的相关性，我们可以想办法在降维的同时将信息的损失尽量降低。当你在原数据上跑了一个比较好的结果，又嫌它太慢模型太复杂时候才可以采取 PCA 降维。

1.2 算法实现步骤

假设有 m 条 n 维数据.

1) 将原始数据按列组成 n 行 m 列矩阵 X(这里的行是特征值的维度)

2) 将 X 的每一行 (代表一个属性字段) 进行零均值化, 即减去这一行的均值

3) 求出协方差矩阵 $C = (1/m)XX^T$

4) 求出协方差矩阵的特征值及所对应的特征向量

5) 将特征向量按对应特征值大小从上到下按行排列成矩阵, 取前 K 行组成矩阵 P

6) $Y = PX$ 即为降维到 K 维后的数据

1.3 算法代码实现

在这里引用一个基于 PCA 图像降维与重构, 用手写 Python 代码 (不是调用 sklearn 的 API) 实现 PCA 降维过程, 并且可以证明降维不是完全可逆的.

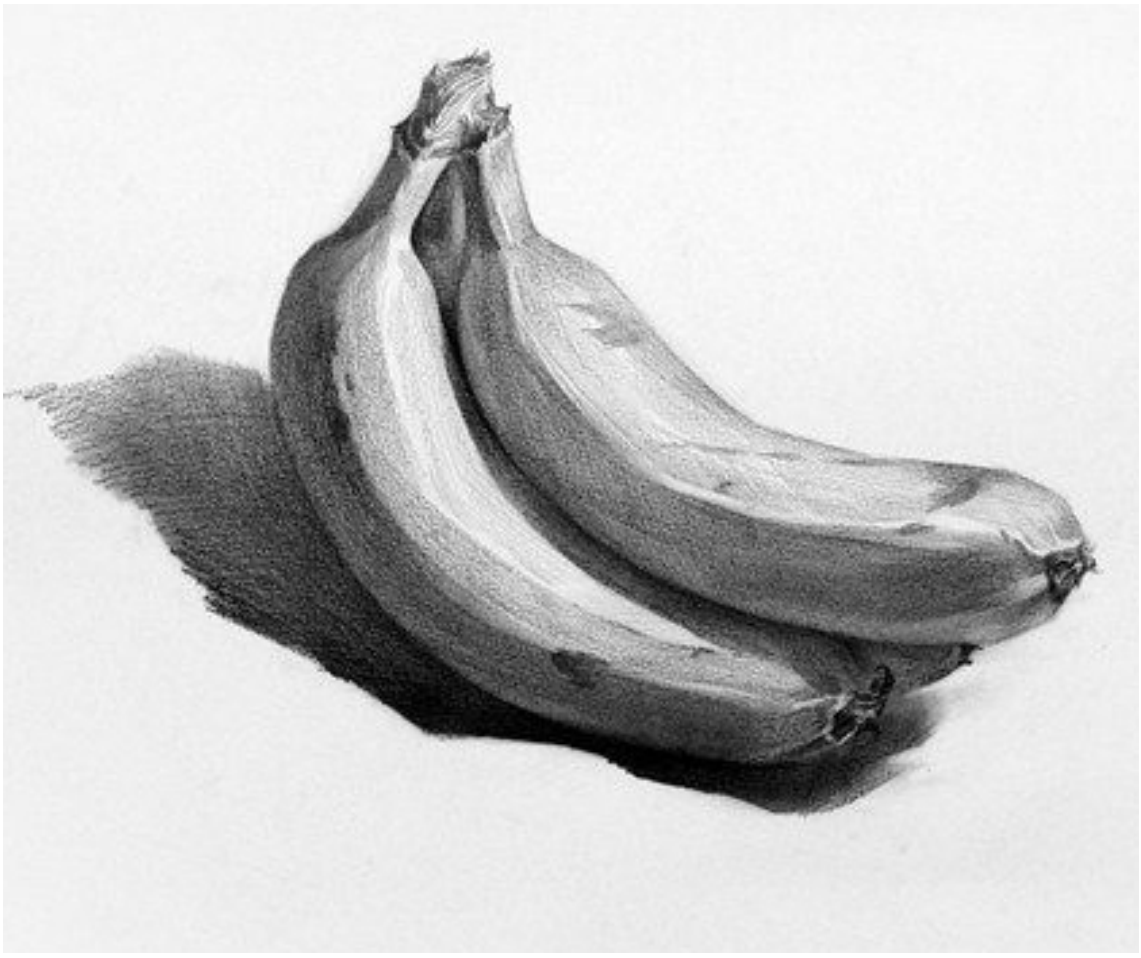

图 1: banana 原始图

1. 数据中心化

```
def Z_centered(dataMat):
        rows,cols=dataMat.shape
        meanVal = np.mean(dataMat, axis=0)  # 按列求均值, 即求各个特征的均值
        meanVal = np.tile(meanVal,(rows,1))
```

```
        newdata = dataMat-meanVal
        return newdata, meanVal
```

2. 求协方差矩阵

```
def Cov(dataMat):
    rows,cols=dataMat.shape
    meanVal = np.mean(dataMat,0) #压缩行，返回1*cols矩阵，对各列求均值
    meanVal = np.tile(meanVal, (rows,1)) #返回rows行的均值矩阵
    Z = dataMat - meanVal
    Zcov = (1/(rows-1))*Z.T * Z
    return Zcov
```

3. 最小化降维造成的损失，确定 k

```
def Percentage2n(eigVals, percentage):
        sortArray = np.sort(eigVals)  # 升序
        sortArray = sortArray[-1::-1]  # 逆转，即降序
        arraySum = sum(sortArray)
        tmpSum = 0
        num = 0
        for i in sortArray:
                tmpSum += i
                num += 1
                if tmpSum >= arraySum * percentage:
                        return num
```

4. 得到最大的 k 个特征值和特征向量

```
def EigDV(covMat, p):
        D, V = np.linalg.eig(covMat) # 得到特征值和特征向量
        k = Percentage2n(D, p) # 确定k值
        print("保留99%信息，降维后的特征个数："+str(k)+"\n")
        eigenvalue = np.argsort(D)
        K_eigenValue = eigenvalue[-1:-(k+1):-1]
        K_eigenVector = V[:,K_eigenValue]
        return K_eigenValue, K_eigenVector
```

5. 得到降维后的数据

```
def getlowDataMat(DataMat, K_eigenVector):
        return DataMat * K_eigenVector
```

6. 重构数据

```
def Reconstruction(lowDataMat, K_eigenVector, meanVal):
        reconDataMat = lowDataMat * K_eigenVector.T + meanVal
        return reconDataMat
```

7. 实现 PCA 算法

```
def PCA(data, p):
        dataMat = np.float32(np.mat(data))
        #数据中心化
```

```
    dataMat, meanVal = Z_centered(dataMat)
    #计算协方差矩阵
            #covMat = Cov(dataMat)
    covMat = np.cov(dataMat, rowvar=0)
    #得到最大的k个特征值和特征向量
    D, V = EigDV(covMat, p)
    #得到降维后的数据
    lowDataMat = getlowDataMat(dataMat, V)
    #重构数据
    reconDataMat = Reconstruction(lowDataMat, V, meanVal)
    return reconDataMat
```

8. 运行程序得到结果

```
PS C:\Users\PC_SKY_WYT\Desktop\pca> & C:/Users/PC_SKY_WYT/AppData/Local/Programs/Python/Python37/python.exe c:/Users/PC_SKY_WYT/Desktop/pca/pca.py
降维前的特征个数: 420

[[244 245 246 ... 247 248 247]
 [245 246 245 ... 247 249 248]
 [246 246 245 ... 247 248 248]
 ...
 [244 243 243 ... 236 237 237]
 [243 243 244 ... 237 238 237]
 [243 244 244 ... 237 237 237]]
----------------------------------------
保留99%信息，降维后的特征个数: 91

c:/Users/PC_SKY_WYT/Desktop/pca/pca.py:82: ComplexWarning: Casting complex values to real discards the imaginary part
  reconImage = reconImage.astype(np.uint8)
[[244 244 244 ... 241 241 241]
 [244 244 244 ... 242 241 241]
 [244 244 244 ... 242 241 241]
 ...
 [243 242 243 ... 239 238 239]
 [243 242 243 ... 239 238 239]
 [242 242 242 ... 239 238 239]]
```

图 2: 运行结果

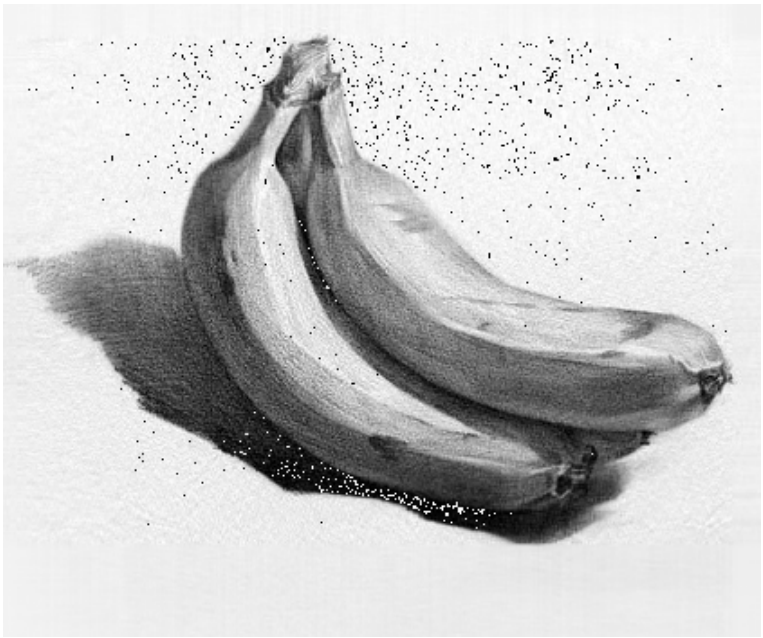

图 3: 降维重构图

观察运行结果, 我们可以得到降维前的特征个数有 420 个, 在保留 99% 的信息, 降维后的特征个数只有 91 个, 大大减少了特征个数, 可以提高我们对于图片处理的效率.

我们可以得到通过重构复原的图片还是能反映原始图片的样子, pca 算法是通过丢掉部分信息得到的, 通过重构之后得到的数据不能包含一开始的全部信息, 因此降维不是完全可逆的.

2 鸢尾花数据集降维及可视化

鸢尾花数据集共收集了三类鸢尾花，即 Setosa 鸢尾花、Versicolour 鸢尾花和 Virginica 鸢尾花，每一类鸢尾花收集了 50 条样本记录，共计 150 条。iris(鸢尾花) 特征为'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'.

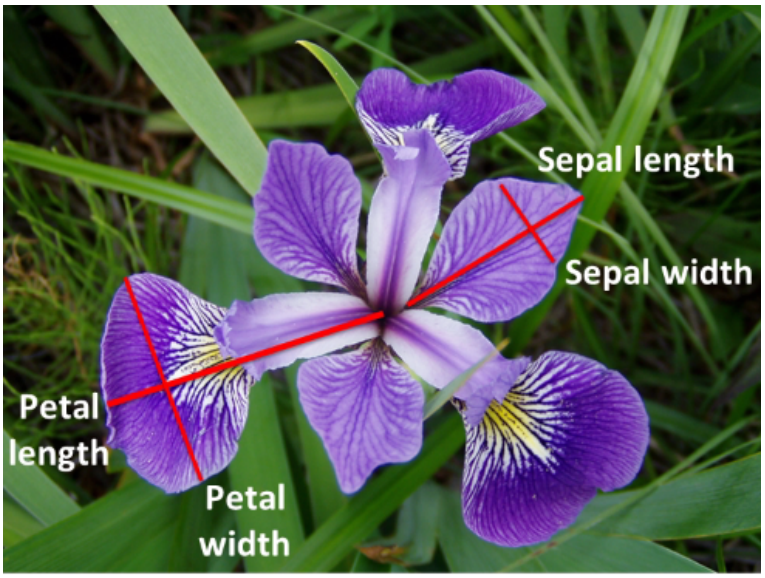

图 4: iris 的特征

2.1 获取数据

```
#导入需要的包
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
x = load_iris().data
y = load_iris().target
x.shape
```

结果为 (150, 4)

2.2 调用 sklearn 的 PCA

使用 sklearn 的方法就很简单了, 实例化一个 PCA 对象, 再调用 fit_transform 函数。

```
pca = PCA(n_components=2) #将4个特征降成2个特征， 以便在平面上展示
x_pca = pca.fit_transform(x)
```

2.3 降维结果可视化

使用 python 的 matplotlib 库可以到达可视化

```
plt.figure()
sns.scatterplot(x_pca[:,0],x_pca[:,1],hue=y,palette=sns.color_palette("Accent", 3))
plt.title('pca result')
plt.xlabel('pc_1')
plt.ylabel('pc_2')
plt.show()
```

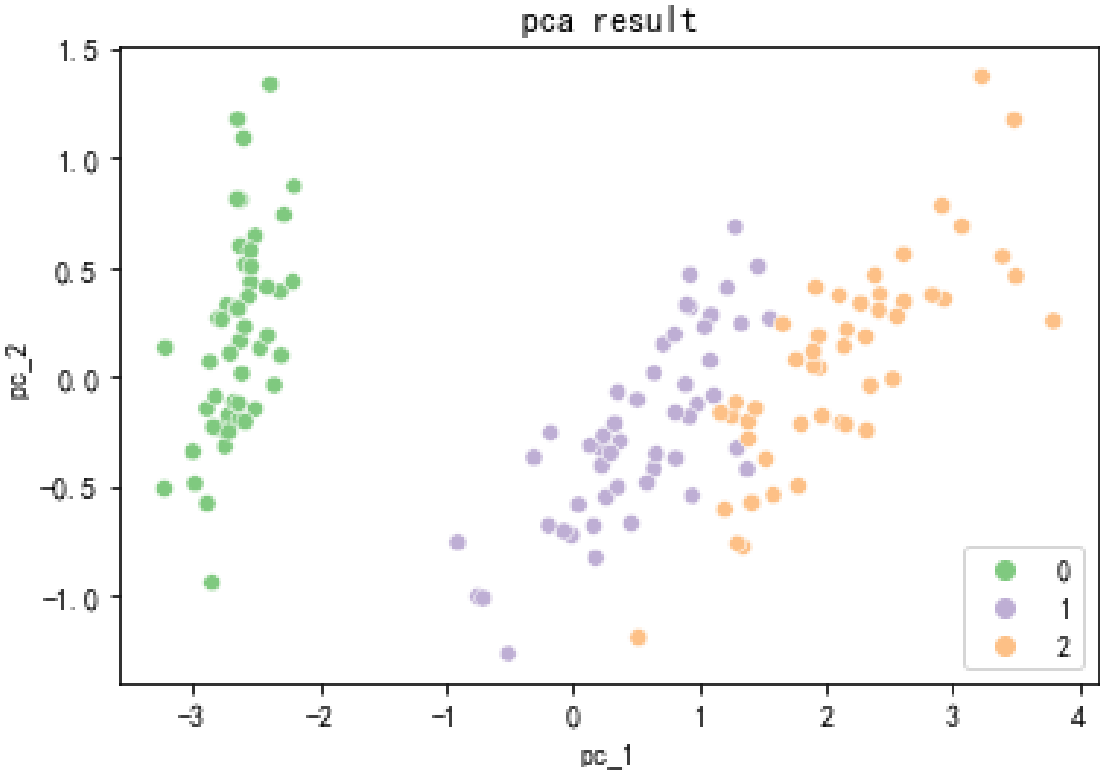

图 5: pca 四维变成二维

```
#属性explained_variance_，查看降维后每个新特征向量上所带的信息量大小（可解释性方差的大小）
pca.explained_variance_
```

[9] Python

```
array([4.22824171, 0.24267075])
```

```
#属性explained_variance_ratio，查看降维后每个新特征向量所占的信息量占原始数据总信息量的百分比
#又叫做可解释方差贡献率
print(pca.explained_variance_ratio_)
print('降维后的信息量是原始信息量的{:.2%}'.format(pca.explained_variance_ratio_.sum()))
```

[10] Python

```
[0.92461872 0.05306648]
降维后的信息量是原始信息量的97.77%
```

图 6: pca 信息结果 (详情见代码)

2.4 PCA 重要参数、属性、方法

2.4.1 重要参数

- n_components: 降到多少维。可以填入整数、0-1 之间的小数或者‘mle’ 0 1 之间的小数表示保留多少信息量，根据信息量决定降到多少维，必须搭配 svd_solver=’full’ 来使用；‘mle’ 使用极大似然估计判断要降到多少维.
- svd_solver: 可选值 ‘auto’、‘full’、‘arpack’、‘randomized’ ; 具体区别如下:

  auto：基于 X.shape 和 n_components 的默认策略来选择分解器：如果输入数据的尺寸大于 500x500 且要提取的特征数小于数据最小维度 min(X.shape) 的 80%，就启用效率更高的” randomized “方法；否则，就使用 ‘full’ 模式，截断将会在矩阵被分解完成后有选择地发生

  full：从 scipy.linalg.svd 中调用标准的 LAPACK 分解器来生成精确完整的 SVD，适合数据量比较适中，计算时间充足的情况，生成的精确完整的 SVD 的结构为: U(m,m) , (m,n) ,V(n,n)

  arpack：从 scipy.sparse.linalg.svds 调用 ARPACK 分解器来运行截断奇异值分解 (SVD truncated)，分解时就将特征数量降到 n_components 中输入的数值 k，可以加快运算速度，适合特征矩阵很大的时候，但一般用于特征矩阵为稀疏矩阵的情况，此过程包含一定的随机性. 截断后的 SVD 分解出的结构为：U(m,k) , (m,k) ,V(n,n)

randomized：在"full" 方法中，分解器会根据原始数据和输入的 n_components 值去计算和寻找符合需求的新特征向量，但是在"randomized" 方法中，分解器会先生成多个随机向量，然后一一去检测这些随机向量中是否有任何一个符合我们的分解需求，如果符合，就保留这个随机向量，并基于这个随机向量来构建后续的向量空间. 这个方法已经被 Halko 等人证明，比"full" 模式下计算快很多，并且还能够保证模型运行效果. 适合特征矩阵巨大，计算量庞大的情况.

- random_state：当 svd_solver 取值为 arpack 和 randomized 时的随机数种子

2.4.2 重要属性

- components_：设降维后的特征矩阵为 x_pca, 即 x * V 的转置 = x_pca,components_ 就是 V x 的 shape 为 (m,n),V 的 shape 是 (k,n),x_pca 的 shape 是 (m,k)
- explained_variance_: 各主成分的可解释性方差
- explained_variance_ratio_: 各主成分可解释性方差占比（分母是所有特征的方差和）

2.4.3 重要方法

- inverse_transform: 将降维后的特征矩阵升维会原始的维度

3 总结

根据上面对 PCA 算法原理的解释，我们可以了解到一些 PCA 的能力和限制。PCA 本质上是将方差最大的方向作为主要特征，并且在各个正交方向上将数据“离相关”，也就是让它们在不同正交方向上没有相关性。

因此，PCA 也存在一些限制，例如它可以很好的解除线性相关，但是对于高阶相关性就没有办法了，对于存在高阶相关性的数据，可以考虑 Kernel PCA，通过 Kernel 函数将非线性相关转为线性相关，关于这点就不展开讨论了。另外，PCA 假设数据各主特征是分布在正交方向上，如果在非正交方向上存在几个方差较大的方向，PCA 的效果就大打折扣了。

最后需要说明的是，PCA 是一种无参数技术，也就是说面对同样的数据，如果不考虑清洗，谁来做结果都一样，没有主观参数的介入，所以 PCA 便于通用实现，但是本身无法个性化的优化。

参考文献

[1] http://blog.codinglabs.org/articles/pca-tutorial.html

[2] https://blog.csdn.net/weixin_41391619/article/details/115715646

[3] https://blog.csdn.net/weixin_41824716/article/details/101166953