AI学习笔记--sklearn--PCA主成算法

- PCA算法介绍

主成分分析（Principal Component Analysis，PCA），是一种统计方法。通过正交变换将一组可能存在相关性的变量转换为一组线性不相关的变量，转换后的这组变量叫主成分。

在实际课题中，为了全面分析问题，往往提出很多与此有关的变量（或因素），因为每个变量都在不同程度上反映这个课题的某些信息。

主成分分析首先是由K.皮尔森（Karl Pearson）对非随机变量引入的，尔后H.霍特林将此方法推广到随机向量的情形。信息的大小通常用离差平方和或方差来衡量。

降维就是一种对高维度特征数据预处理方法。降维是将高维度的数据保留下最重要的一些特征，去除噪声和不重要的特征，从而实现提升数据处理速度的目的。在实际的生产和应用中，降维在一定的信息损失范围内，可以为我们节省大量的时间和成本。降维也成为应用非常广泛的数据预处理方法。

降维具有如下一些优点：

1) 使得数据集更易使用。
2) 降低算法的计算开销。
3) 去除噪声。
4) 使得结果容易理解。

降维的算法有很多，比如奇异值分解(SVD)、主成分分析(PCA)、因子分析(FA)、独立成分分析(ICA)。

- 算法原理

在用统计分析方法研究多变量的课题时，变量个数太多就会增加课题的复杂性。人们自然希望变量个数较少而得到的信息较多。在很多情形，变量之间是有一定的相关关系的，当两个变量之间有一定相关关系时，可以解释为这两个变量反映此课题的信息有一定的重叠。主成分分析是对于原先提出的所有变量，将重复的变量（关系紧密的变量）删去多余，建立尽可能少的新变量，使得这些新变量是两两不相关的，而且这些新变量在反映课题的信息方面尽可能保持原有的信息。

设法将原来变量重新组合成一组新的互相无关的几个综合变量，同时根据实际需要从中可以取出几个较少的综合变量尽可能多地反映原来变量的信息的统计方法叫做主成分分析或称主分量分析，也是数学上用来降维的一种方法。

协方差为正时，说明X和Y是正相关关系；协方差为负时，说明X和Y是负相关关系；协方差为0时，说明X和Y是相互独立。Cov(X,X)就是X的方差。当样本是n维数据时，它们

的协方差实际上是协方差矩阵(对称方阵)。在信号处理中认为信号具有较大的方差，噪声有较小的方差，信噪比就是信号与噪声的方差比，越大越好。样本在u1上的投影方差较大，在u2上的投影方差较小，那么可认为u2上的投影是由噪声引起的。

更多原理可以参考：
https://blog.csdn.net/program_developer/article/details/80632779

- 应用学科

主成分分析作为基础的数学分析方法，其实际应用十分广泛，比如人口统计学、数量地理学、分子动力学模拟、数学建模、数理分析等学科中均有应用，是一种常用的多变量分析方法。

- Python Demo

```
# 这是一个ｐｃａ的ｄｅｍｏ
# 这是吴恩达 教程上面的那个例子
# 写在打印出来的了讲义上面了

from sklearn import datasets
from sklearn import decomposition
import numpy as np

arr = np.array([[-1,-1,0,2,0],[-2,0,0,1,1]])
X = arr.T

# n_components=2 保留的主成分特征的数量
# 建立pca 模型
estimator = decomposition.PCA(n_components=1)
# 进行 pca 处理
# reduce_data.shape ： (1797, 2)
# X.shape : (1797, 64)
reduce_data = estimator.fit_transform(X)
print ("降维前, shape : ",X.shape)
print (X)
print ("降维后, shape : ",reduce_data.shape)
print (reduce_data)

输出结果：
➜  Python_Workplace ipython pca.py
('\xe9\x99\x8d\xe7\xbb\xb4\xe5\x89\x8d, shape : ', (5, 2))
[[-1 -2]
[-1  0]
[ 0  0]
[ 2  1]
[ 0  1]]
```

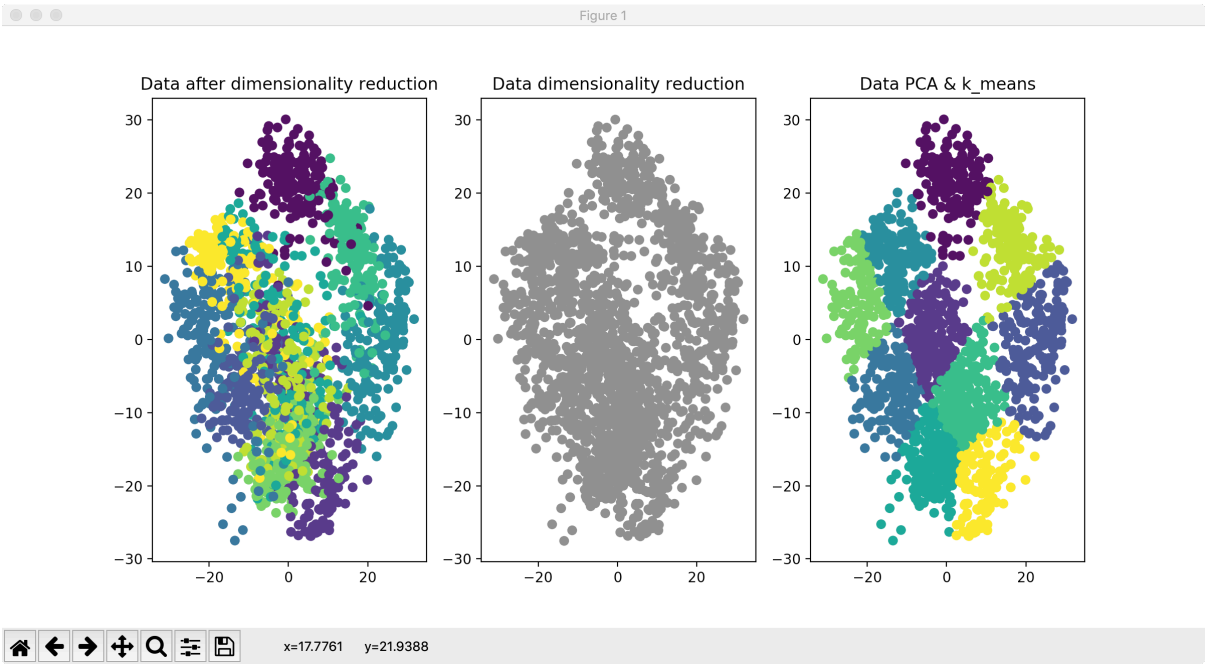
```
('\xe9\x99\x8d\xe7\xbb\xb4\xe5\x90\x8e, shape : ', (5, 1))
[[ 2.12132034]
[ 0.70710678]
[-0.        ]
[-2.12132034]
[-0.70710678]]
```

实际在用PCA和K-Means聚类的例子，我们可以理解一些作用，在聚类的前期，用降维和PCA等处理算法可以过滤干扰，保存最有效的数据：

```
"""
使用pca对数据进行降维，然后show出降维之后数据的分布
+
使用降维之后的数据最为kmeans算法的输入，进行聚类，（聚成 1  0 类）
可以观察到，聚类之后数据的分布和降维后加上标签的数据的分布有相似的部分
"""

from sklearn import datasets
from matplotlib import pyplot as plt
from sklearn import decomposition
from sklearn.cluster import k_means
import numpy as np

digits_data = datasets.load_digits()

X = digits_data.data  # shape (1797, 64)  --> 含义：  （样本 ×   每个样本的特征）
y = digits_data.target  # shape  (1797,)    --> 含义：每个样本的label,即他们对应的数字

## 使用pca降维
# n_components=2 保留的主成分特征的数量
# 建立pca 模型
estimator = decomposition.PCA(n_components=2)

# 进行 pca 处理
# reduce_data.shape：  (1797, 2)
# X.shape : (1797, 64)
reduce_data = estimator.fit_transform(X)

plt.figure(figsize=(12, 6))
ax1 = plt.subplot(1, 3, 1)
plt.title('Data after dimensionality reduction')
ax2 = plt.subplot(1, 3, 2)
plt.title('Data dimensionality reduction')
ax3 = plt.subplot(1, 3, 3)
plt.title('Data PCA & k_means')

# ax0.scatter(X,reduce_data[:,1])
# print reduce_data[:,0]
ax1.scatter(reduce_data[:,0],reduce_data[:,1],c=y)
```

```
## k-means 聚类
model = k_means(reduce_data,n_clusters=10)
cluster_centers = model[0]
cluster_labels = model[1]
cluster_inertia = model[2]

ax2.scatter(reduce_data[:,0], reduce_data[:,1], c="grey")
ax3.scatter(reduce_data[:,0], reduce_data[:,1], c=cluster_labels)
# plt.title('helloworld')
plt.show()

print ("左图表示降维之后的数据分布，不同颜色代表不同的数字；中图表示降维的数据分布；右图表示对中图的数据进行聚类后的结果")
```

最后的输出结果，如下：



接下来我们来一个实际的图形处理例子。

```
from PIL import Image
import numpy as np
from sklearn.cluster import KMeans
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def show_scatter(a):
    N = 10
    print( '原始数据: \n', a)
    density, edges = np.histogramdd(a, bins=[N,N,N], range=[(0,1), (0,1),
(0,1)])
    density /= density.max()
```

```
    x = y = z = np.arange(N)
    d = np.meshgrid(x, y, z)
    fig = plt.figure(1, facecolor='w')
    # ax = fig.add_subplot(111, projection='3d')
    ax = Axes3D(fig)
    ax.scatter(d[1], d[0], d[2], c='r', s=100*density, marker='o',
depthshade=True)
    ax.set_xlabel(u'红色分量')
    ax.set_ylabel(u'绿色分量')
    ax.set_zlabel(u'蓝色分量')
    plt.title(u'图像颜色三维频数分布', fontsize=20)
    plt.figure(2, facecolor='w')
    den = density[density > 0]
    den = np.sort(den)[::-1]
    t = np.arange(len(den))
    plt.plot(t, den, 'r-', t, den, 'go', lw=2)
    plt.title(u'图像颜色频数分布', fontsize=18)
    plt.grid(True)
    plt.show()
    return

num_vq = 50 #设置要聚类的个数
im = Image.open('background_test.png')     #
16.son.bmp(100)/16.flower2.png(200)/16.son.png(60)/16.lena.png(50)
image = np.array(im).astype(np.float) / 255
image = image[:, :, :3] # 变成3通道
image_v = image.reshape((-1, 3))
model = KMeans(num_vq)
show_scatter(image_v) #自定义的画三维立体图函数
N = image_v.shape[0]    # 图像像素总数
# 选择足够多的样本(如1000个)，计算聚类中心
#print(N)
idx = np.random.randint(0, N, size=1000)
image_sample = image_v[idx]
#print(image_sample)
model.fit(image_sample)
c = model.predict(image_v)  # 聚类结果
print( '聚类结果：\n', c)
print ('聚类中心：\n', model.cluster_centers_)

输出结果：
➜  Python_Workplace ipython pca.py
('\xe5\x8e\x9f\xe5\xa7\x8b\xe6\x95\xb0\xe6\x8d\xae\xef\xbc\x9a\n',
array([[0.16078431, 0.14901961, 0.14117647],
       [0.16078431, 0.14901961, 0.14117647],
       [0.16078431, 0.14901961, 0.14117647],
       ...,
       [0.16078431, 0.14901961, 0.14117647],
       [0.16078431, 0.14901961, 0.14117647],
       [0.16078431, 0.14901961, 0.14117647]]))
```

```
('\xe8\x81\x9a\xe7\xb1\xbb\xe7\xbb\x93\xe6\x9e\x9c\xef\xbc\x9a\n',
array([11, 11, 11, ..., 11, 11, 11], dtype=int32))
('\xe8\x81\x9a\xe7\xb1\xbb\xe4\xb8\xad\xe5\xbf\x83\xef\xbc\x9a\n',
array([[0.0146592 , 0.05042017, 0.14752568],
       [0.31577965, 0.48832866, 0.69430439],
       [0.06751457, 0.29920509, 0.57742448],
       [0.4496732 , 0.21830065, 0.10196078],
       [0.0214902 , 0.1065098 , 0.25819608],
       [0.42941176, 0.59641944, 0.76828645],
       [0.22177503, 0.41816305, 0.65376677],
       [0.2495098 , 0.18823529, 0.16960784],
       [0.72470588, 0.36235294, 0.26901961],
       [0.06544118, 0.21642157, 0.45735294],
       [0.65490196, 0.64313725, 0.69803922],
       [0.15591206, 0.14735591, 0.142959  ],
       [0.28543417, 0.39887955, 0.58235294],
       [0.13230626, 0.34033613, 0.61269841],
       [0.3372549 , 0.36960784, 0.44705882],
       [0.10147059, 0.15686275, 0.28014706],
       [0.22322775, 0.22051282, 0.28325792],
       [0.3745098 , 0.29869281, 0.27124183],
       [0.02651727, 0.02894491, 0.05546218],
       [0.39876543, 0.56245461, 0.74371823],
       [0.02909982, 0.22326203, 0.53578431],
       [0.85294118, 0.49019608, 0.35686275],
       [0.14248366, 0.2724183 , 0.51947712],
       [0.51026528, 0.63737024, 0.79354095],
       [0.26199813, 0.44780579, 0.67189542],
       [0.10174292, 0.23594771, 0.43376906],
       [0.5496732 , 0.35947712, 0.27647059],
       [0.37825312, 0.44491979, 0.59679144],
       [0.1884883 , 0.38140417, 0.63099304],
       [0.71372549, 0.56078431, 0.52287582],
       [0.02027738, 0.08110952, 0.19148733],
       [0.01319073, 0.03695781, 0.1030303 ],
       [0.04828431, 0.17916667, 0.40980392],
       [0.18431373, 0.33333333, 0.54419306],
       [0.22941176, 0.33137255, 0.44705882],
       [0.10413943, 0.32941176, 0.68409586],
       [0.20980392, 0.07745098, 0.0245098 ],
       [0.57333333, 0.27843137, 0.14745098],
       [0.03956171, 0.26793541, 0.56355248],
       [0.84836601, 0.32679739, 0.18039216],
       [0.06181139, 0.11204482, 0.20168067],
       [0.51294118, 0.53960784, 0.6627451 ],
       [0.0546003 , 0.16229261, 0.36078431],
       [0.35772549, 0.52117647, 0.71254902],
       [0.82745098, 0.71372549, 0.63137255],
       [0.4627451 , 0.46535948, 0.48235294],
       [0.31820728, 0.54789916, 0.78319328],
       [0.032493  , 0.13557423, 0.3124183 ],
```

```
        [0.09673203, 0.21339869, 0.38071895],
        [0.13594771, 0.42418301, 0.76143791]]))
```

原图：

结果：

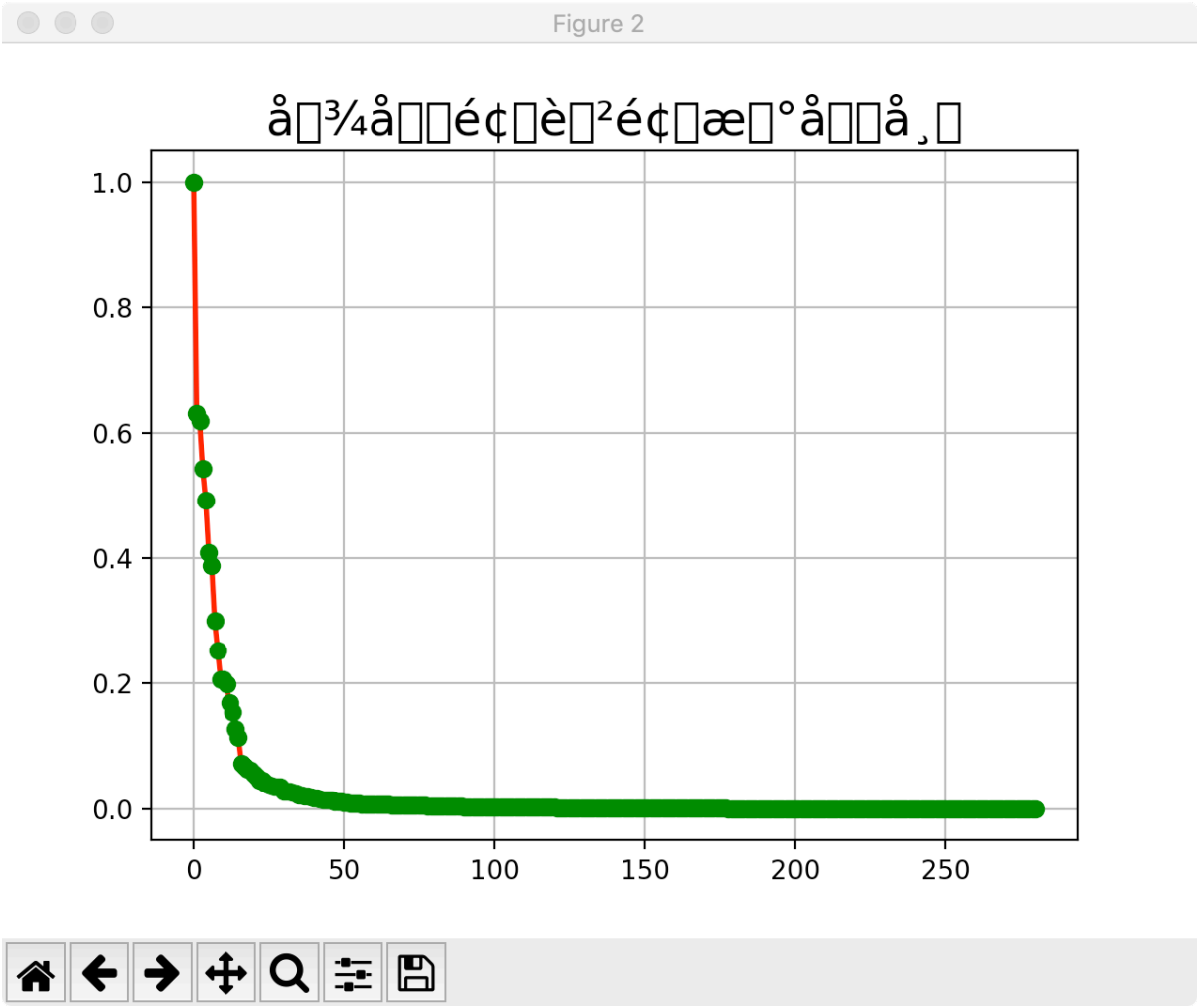
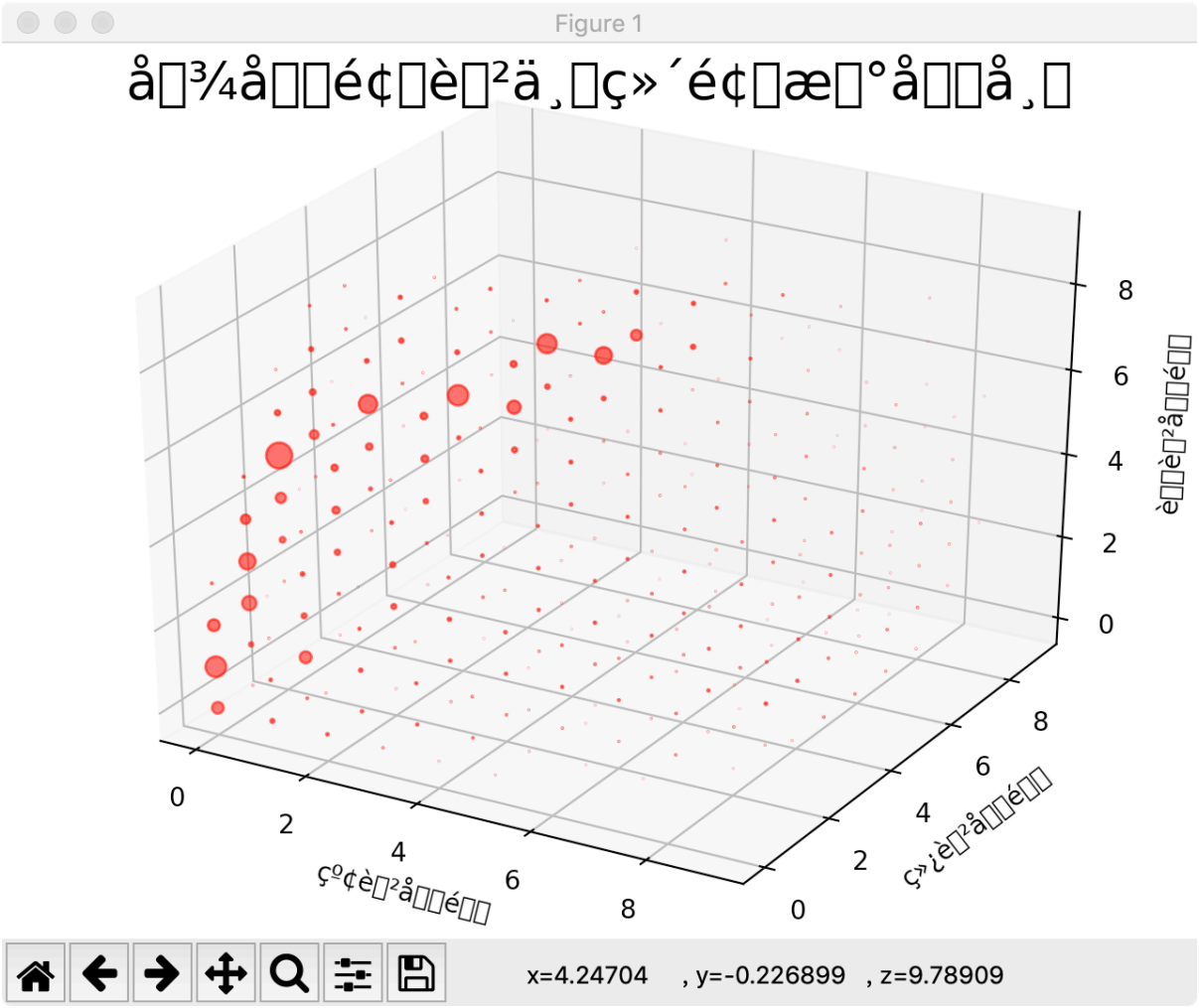


Figure 2
å□¾å□□é¢□è□²é¢□æ□°å□□å¸□
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250



By Genesis.Ling