PCA

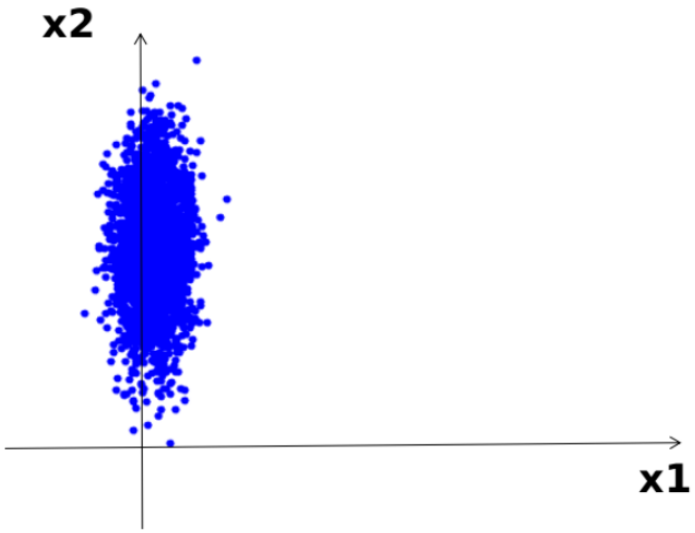
1. 实验目的

了解 PCA 算法的原理，并且可以简单应用

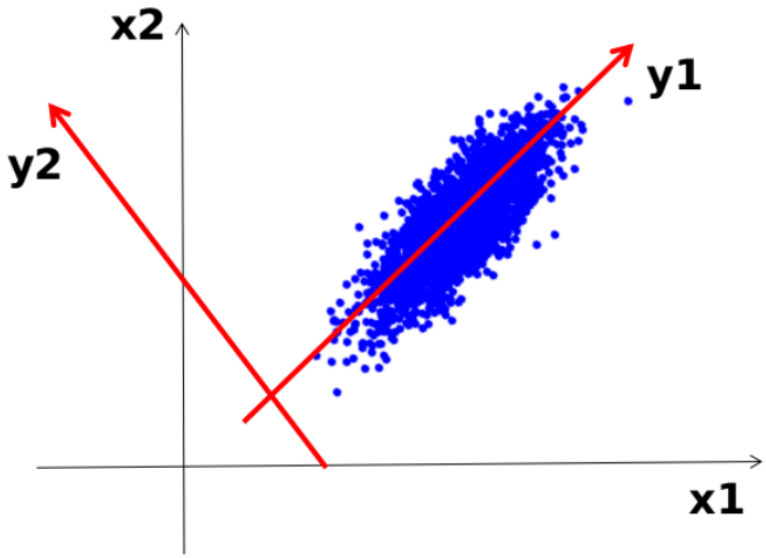
2. 算法原理

1)PCA 用来做什么

PCA 呀，用来做什么呢？这个很重要，PCA 也有很多应用，可能我们之前听过用 PCA 做人脸识别，PCA 做异常检测等等。但事实上 PCA 没有那么强大的功能，PCA 能做的事其实很有限，那就是：降维。其他拓展应用都是在这基础上做了相应额工作。为什要降维呢？很明显，数据中有些内容没有价值，这些内容的存在会影响算法的性能，和准确性。



如上图，数据点大部分都分布在 x2 方向上，在 x1 方向上的取值近似相同，那么对于有些问题就可以直接将 x1 坐标的数值去掉，只取 x2 坐标的值即可。但是有些情况不能直接这样取，例如：



上图的数据分布在 x1 和 x2 方向都比较均匀，任一去掉一个坐标的数值可能对

结果都会有很大的影响。这个时候就是 PCA 展现作用的时候了。黑色坐标系是原始坐标系，红色坐表系是我后面构建的坐标系，如果我的坐标系是红色的，那么这个问题是不是就和上面那个问题一样了，我只需要去掉 y2 坐标系的数据即可。实际上黑色坐标系和红色坐标系是等价的，差异仅仅是在空间中他们的基不同，黑色坐标系的基是我们最习惯的(1, 0), (0, 1)，红色坐标系的基是(1, 1),(-1, -1)，事实上，他们是等价的，只不过经常默认使用的就是黑色坐标系。主成分分析可以让数据的投影到那些数据分布比较分散的平面上，比如上图的 y1，从而忽视 y2 的作用，进而达到降维的目的。

2）PCA 数学原理

上面我们说 PCA 可以将数据投影到分布分散的平面内，而忽略掉分布集中的平面。我们可以这样理解，如上面的图 2,大部分数据投影到 y1 坐标系中的化，数据分布会比较分散，投影到 x1、x2 等其他坐标轴分布会相对集中，其中，投影到 y2 上面分布最集中。所以我们尽可能将数据转化到红色坐标系，然后去掉 y2 坐标。好了，问题描述完了，该想象怎样才能达到这样的目的。

任何形式的变化在数学上都可以抽象成一个映射，或者函数。好，现在我们需要构建一个函数 $f(X_{m\times n})$使得这个函数可以将矩阵 $X_{m\times n}$ 降维，矩阵 X 是原始数据，矩阵的每一行是一个样本的特征向量，即矩阵 $X_{m\times n}$ 中有 m 个样本，每个样本有 n 个特征值。所以，所谓的降维，其实是减少 n 的数量。假设降维后的结构为 $Z_{m\times k}$, 其中 $k<n$。那么 PCA 的数学表达可以这样表示：$Z_{m\times k}=f(X_{m\times n}),k<n$

3. 实验环境

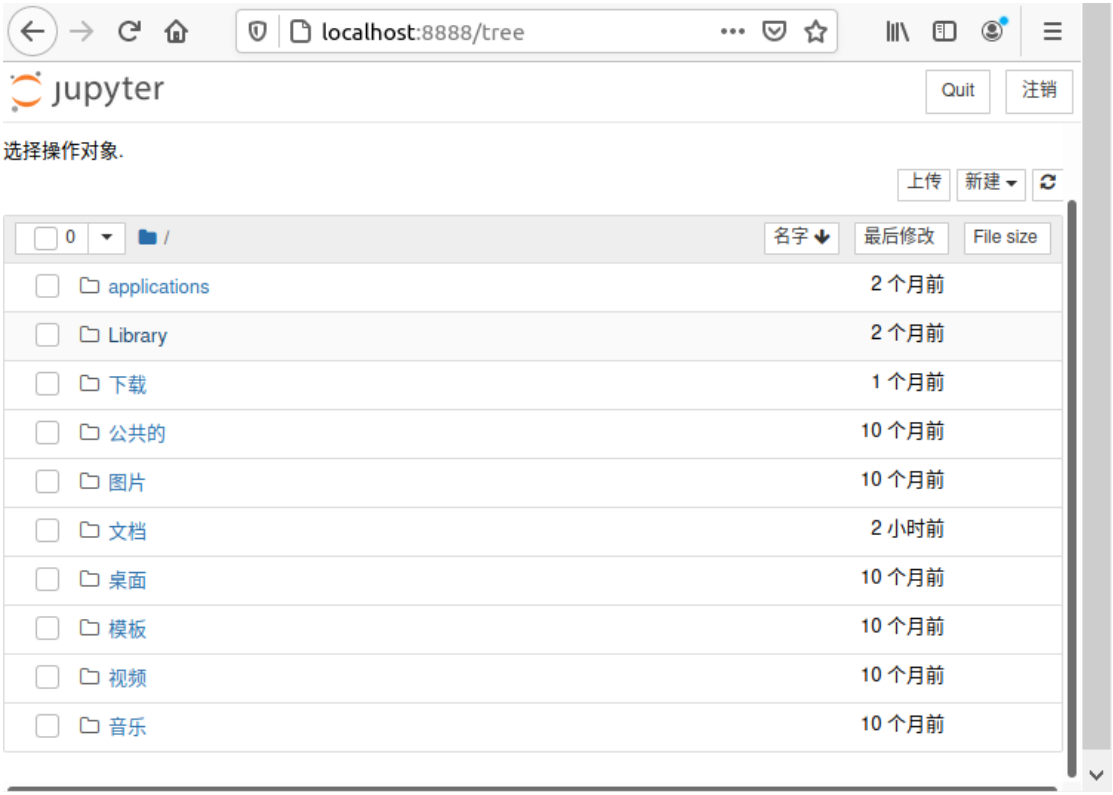
Ubuntu 20.04

Python 3.6

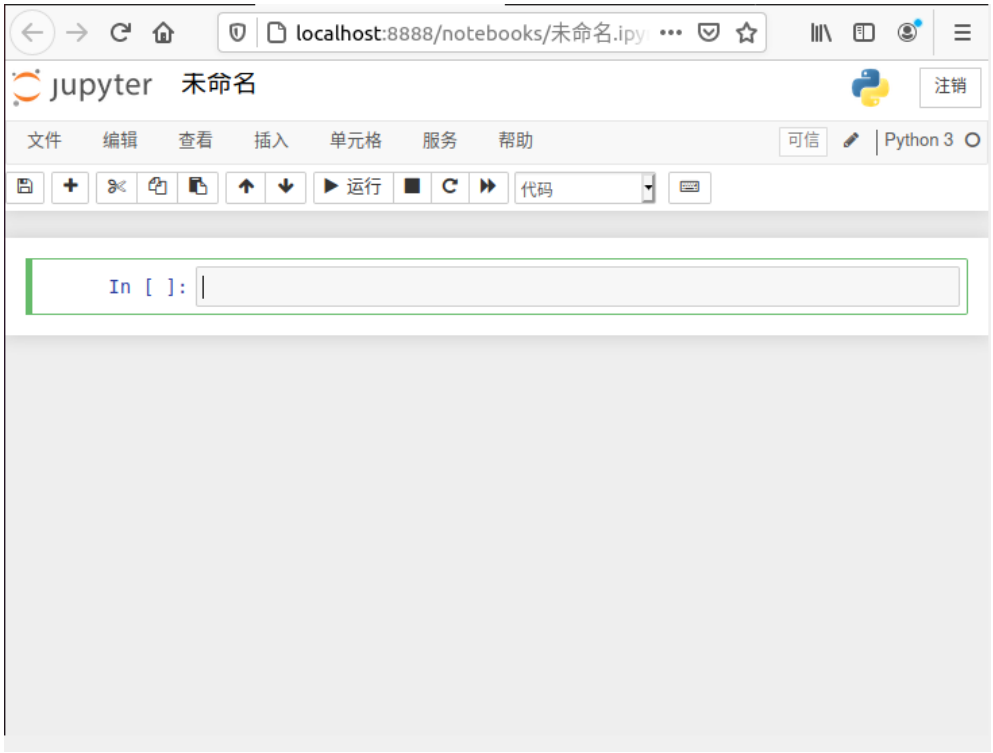
Jupyter notebook

4. 实验步骤

1)打开终端，然后输入 jupyter notebook，出现如下界面



2)选定特定文件夹，新建 ipynb 文件，在未命名出可重命名文件



5. 实操

Step 1:数据预处理

1. 导入库
2. 导入数据集
3. 设置列索引
4. 分割数据集

5. 数据集标准化

```
#导入库
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA

#导入数据集
df_wine = pd.read_csv('wine.data',header=None)


# 设置列索引
df_wine.columns =  ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                    'Alcalinity of ash', 'Magnesium', 'Total phenols',
                    'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyani
ns',
                    'Color intensity', 'Hue',
                    'OD280/OD315 of diluted wines', 'Proline']


# 数据集设置：X 为样本特征数据，y 为目标数据，即标注结果
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

# 数据集划分： 将数据集划分为训练集和测试集数据（测试集数据为 30%，训练集为
70%）
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
,
                                                    stratify=y,
                                                    random_state=0)


# 实例化
sc = StandardScaler()

# 对数据集进行标准化（一般情况下我们在训练集中进行均值和方差的计算，直接在测试
集中使用）
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

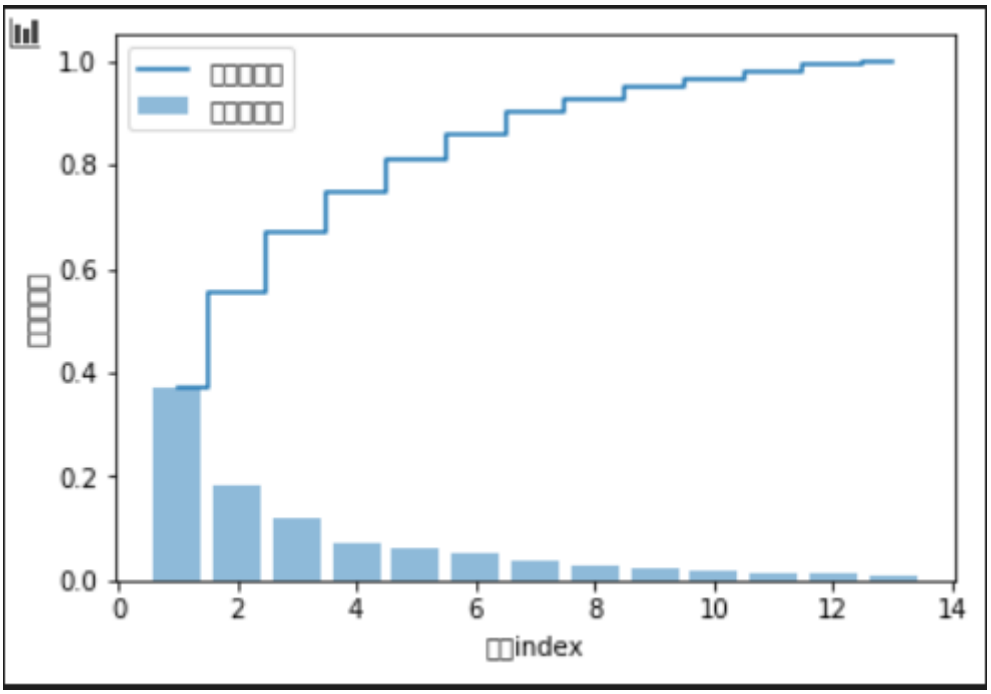
Step 2:PCA 模型

- 实例化 PCA

```
# 实例化 pca，保留所有特征
pca = PCA()
# 特征提取
X_train_pca = pca.fit_transform(X_train_std)
# 特征值结果
pca.explained_variance_ratio_
```

Step 3:特征值绘制

```
# 特征值绘制
# 绘制图像
plt.figure()
plt.bar(range(1, 14), pca.explained_variance_ratio_, alpha=0.5, align='
center',
        label='特征值分布')
plt.step(range(1, 14), np.cumsum(pca.explained_variance_ratio_), where=
'mid',
         label='累计特征值')
plt.ylabel('特征值比例')
plt.xlabel('特征 index')
plt.legend(loc='best')
```



Step 4：压缩特征

```
# 压缩到二维特征
pca = PCA(n_components=2)
# 对训练数据进行处理
X_train_pca = pca.fit_transform(X_train_std)
# 特征值结果(只保留两个特征)
```

```
print(pca.explained_variance_ratio_)
# 对测试集数据进行处理
X_test_pca = pca.transform(X_test_std)
# 特征降维后结果展示
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']

for l, c, m in zip(np.unique(y_train), colors, markers):
    # 按照样本的真实值进行展示
    plt.scatter(X_train_pca[y_train == l, 0],
                X_train_pca[y_train == l, 1],
                c=c, label=l, marker=m)

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()
```