DBSCAN

1. 实验目的

了解 DBSCAN 算法的原理，并且可以简单应用

2. 算法原理

1)原理

DBSCAN 是一种基于密度的聚类算法，这类密度聚类算法一般假定类别可以通过样本分布的紧密程度决定。同一类别的样本，他们之间的紧密相连的，也就是说，在该类别任意样本周围不远处一定有同类别的样本存在。

通过将紧密相连的样本划为一类，这样就得到了一个聚类类别。通过将所有各组紧密相连的样本划为各个不同的类别，则我们就得到了最终的所有聚类类别结果。

2）思想

DBSCAN 的聚类定义很简单：由密度可达关系导出的最大密度相连的样本集合，即为我们最终聚类的一个类别，或者说一个簇。

这个 DBSCAN 的簇里面可以有一个或者多个核心对象。如果只有一个核心对象，则簇里其他的非核心对象样本都在这个核心对象的ϵ-邻域里；如果有多个核心对象，则簇里的任意一个核心对象的ϵ-邻域中一定有一个其他的核心对象，否则这两个核心对象无法密度可达。这些核心对象的ϵ-邻域里所有的样本的集合组成的一个 DBSCAN 聚类簇。

那么怎么才能找到这样的簇样本集合呢？DBSCAN 使用的方法很简单，它任意选择一个没有类别的核心对象作为种子，然后找到所有这个核心对象能够密度可达的样本集合，即为一个聚类簇。接着继续选择另一个没有类别的核心对象去寻找密度可达的样本集合，这样就得到另一个聚类簇。一直运行到所有核心对象都有类别为止。

基本上这就是 DBSCAN 算法的主要内容了，是不是很简单？但是我们还是有三个问题没有考虑。

第一个是一些异常样本点或者说少量游离于簇外的样本点，这些点不在任何一个核心对象在周围，在 DBSCAN 中，我们一般将这些样本点标记为噪音点。

第二个是距离的度量问题，即如何计算某样本和核心对象样本的距离。在 DBSCAN 中，一般采用最近邻思想，采用某一种距离度量来衡量样本距离，比如欧式距离。

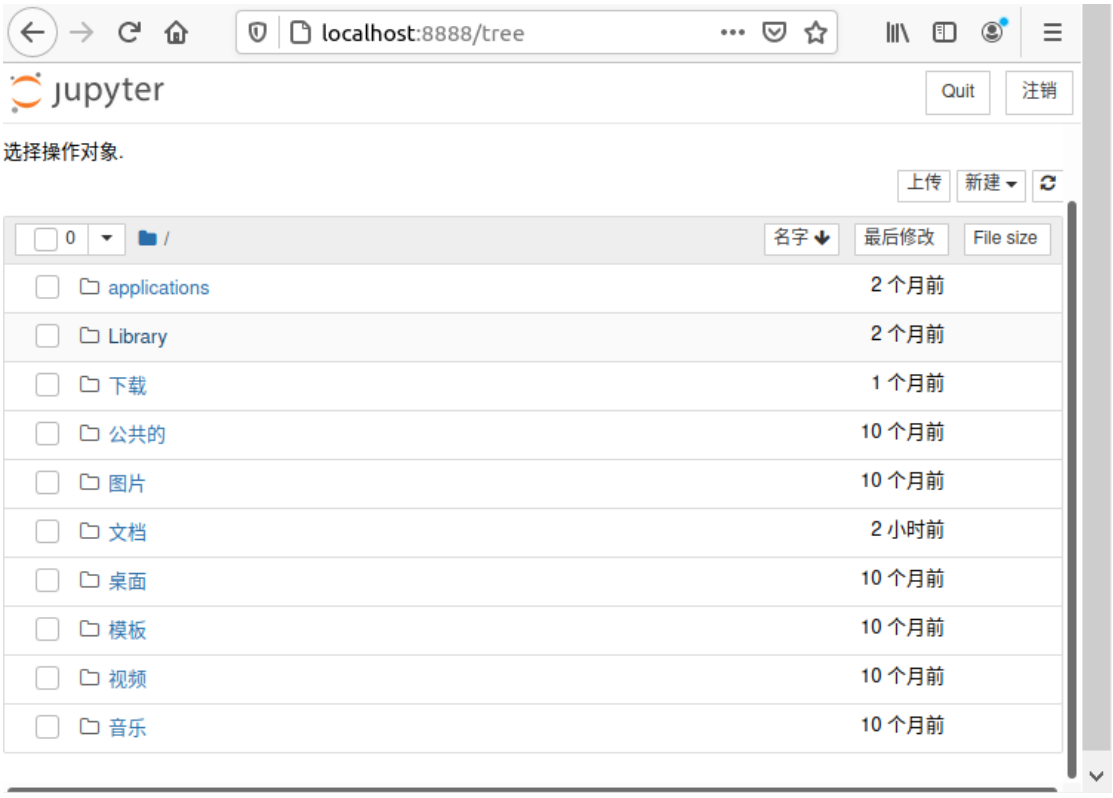
这和 KNN 分类算法的最近邻思想完全相同。对应少量的样本，寻找最近邻可以直接去计算所有样本的距离，如果样本量较大，则一般采用 KD 树或者球树来快速的搜索最近邻。如果大家对于最近邻的思想，距离度量，KD 树和球树不熟悉，建议参考之前写的另一篇文章 K 近邻法(KNN)原理小结。第三种问题比较特殊，某些样本可能到两个核心对象的距离都小于ϵ，但是这两个核心对象由于不是密度直达，又不属于同一个聚类簇，那么如何界定这个样本的类别呢？一般来说，此时 DBSCAN 采用先来后到，先进行聚类的类别簇会标记这个样本为它的类别。也就是说 BDSCAN 的算法不是完全稳定的算法。

3. 实验环境

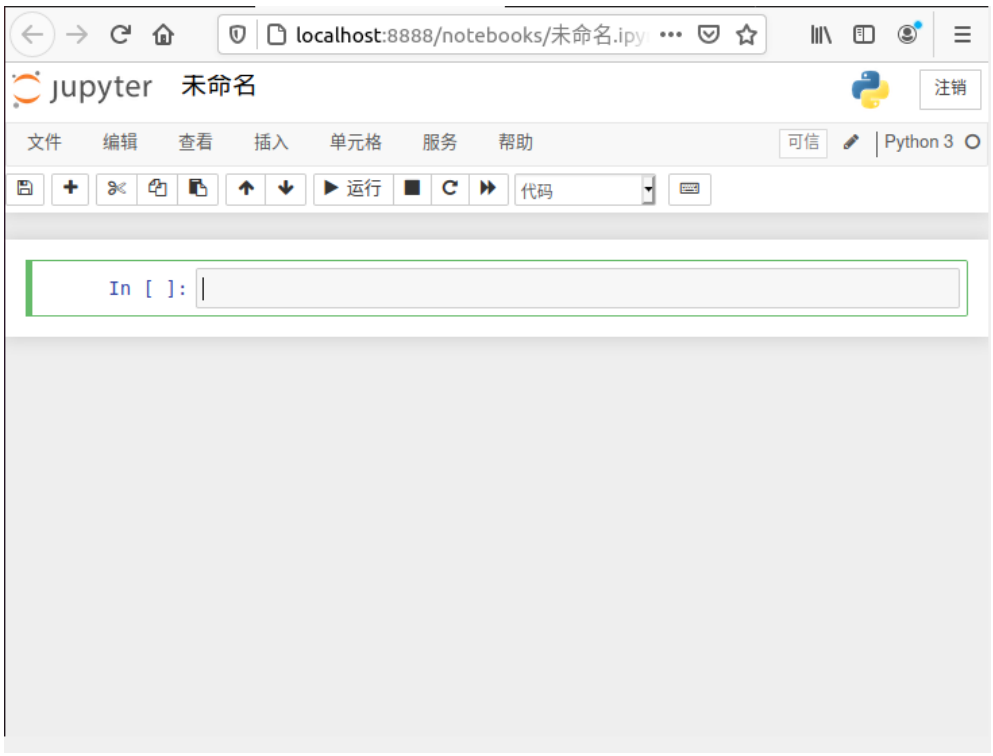
Ubuntu 20.04
Python 3.6
Jupyter notebook

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

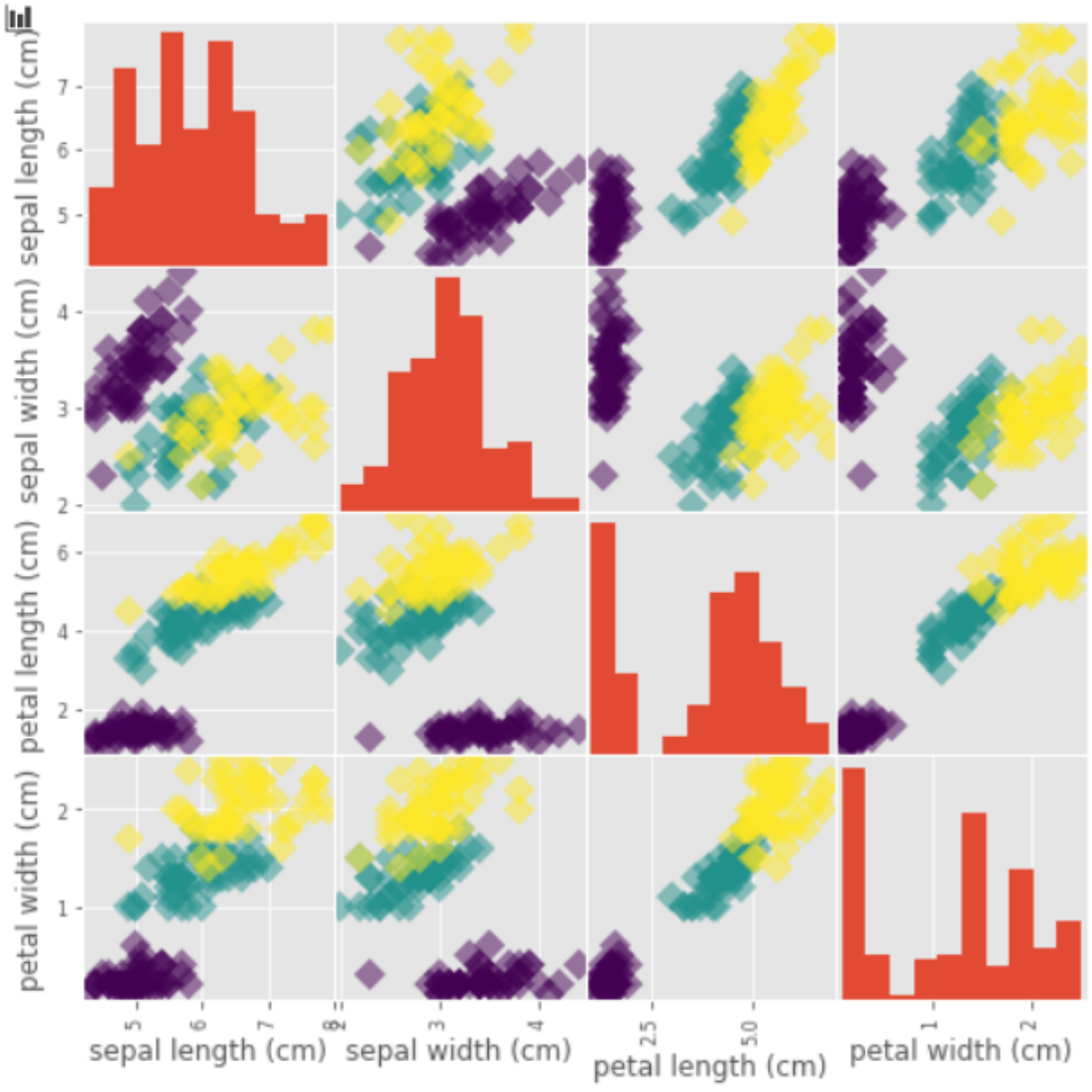
Step 1:数据预处理

1. 导入库
2. 导入数据集
3. 查看数据集
4. 绘制图表

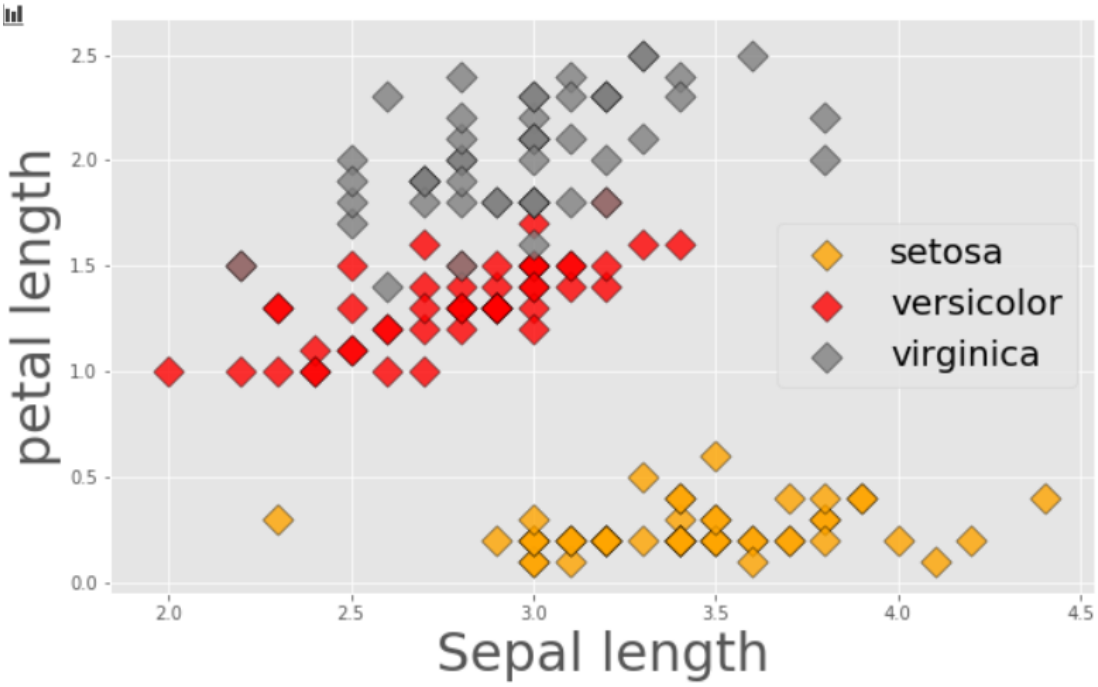
```
#导入库
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter

## import data
from sklearn.datasets import load_iris
from sklearn import datasets

## import DBSCAN model
from sklearn.cluster import DBSCAN
from sklearn import metrics


plt.style.use('ggplot')
%matplotlib inline

#导入数据集
iris = datasets.load_iris()
```

```
# 查看数据集
print(iris.keys())
X = iris.data
y = iris.target
target_names = iris.target_names
df = pd.DataFrame(X, columns=iris.feature_names)

df.head()
```

```
# 绘制图表
pd.plotting.scatter_matrix(df, c = y, figsize = [8, 8], s=100, marker='
D')
plt.show()
```



```
## Show only two features 'Sepal length' and 'petal length' for differe
nt types of flowers (target labels)
plt.figure(2, figsize=(10, 6))
colors = ['orange', 'red', 'gray']


for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X[y == i, 1], X[y == i, 3], color=color, alpha=.8,
                label=target_name, s=150, cmap=plt.cm.Set1, edgecolor='
k',marker='D')
plt.legend(loc='best', shadow=False, scatterpoints=1,fontsize=20)
plt.xlabel('Sepal length', fontsize=30)
plt.ylabel('petal length', fontsize=30)
#plt.legend(fontsize=30)
plt.show()
```



Step 2:DBSCAN 模型

```
model = DBSCAN(eps=0.8, min_samples=6).fit(X)
labels=model.labels_
labels
```

```
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_clusters_
```

Step 4：结果展示

```
plt.figure(2, figsize=(10, 6))
```

```
plt.scatter(X[:, 1], X[:, 3], c=model.labels_, s=150,  cmap=plt.cm.Set1
,alpha=.8, edgecolor='k',marker='D')
#plt.scatter(X[:, 1], X[:, 3], c=y, s=150, cmap=plt.cm.Set1, edgecolor=
'k', marker='D')   #target
plt.xlabel('Sepal length', fontsize=30)
plt.ylabel('petal length', fontsize=30)
plt.title('Estimated number of clusters: %d' % n_clusters_, fontsize=20
)
plt.show()
```