K-means

1. 算法原理

K 均值交替执行两个步骤：

1）原理

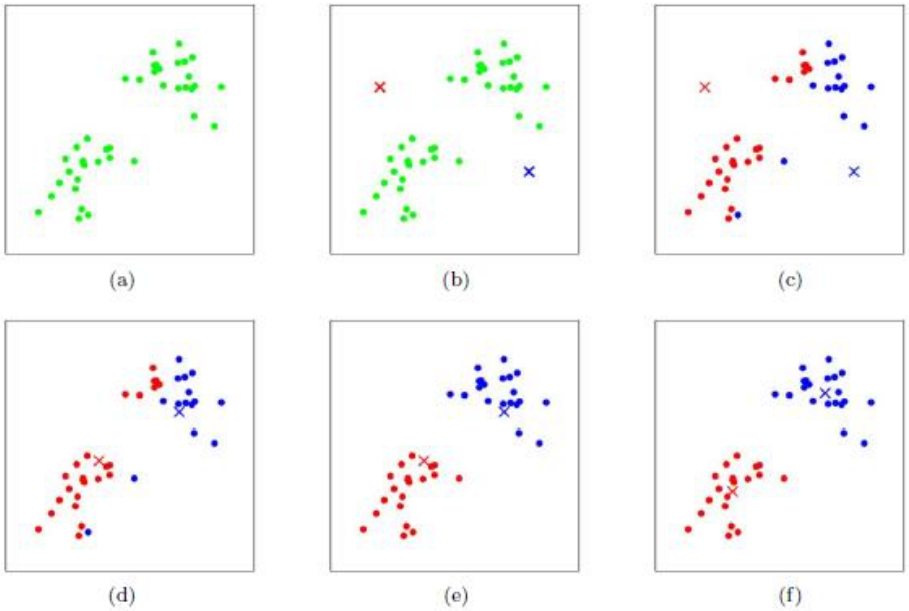
K-Means 算法的思想很简单，对于给定的样本集，按照样本之间的距离大小，将样本集划分为 K 个簇。让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。

如果用数据表达式表示，假设簇划分为(C1,C2,...Ck)，则我们的目标是最小化平方误差 $E = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||_2 2$

其中 μ i 是簇 Ci 的均值向量，有时也称为质心，表达式为：$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$

如果我们想直接求上式的最小值并不容易，这是一个 NP 难的问题，因此只能采用启发式的迭代方法。

K-Means 采用的启发式方式很简单，用下面一组图就可以形象的描述。



上图 a 表达了初始的数据集，假设 k=2。在图 b 中，我们随机选择了两个 k 类所对应的类别质心，即图中的红色质心和蓝色质心，然后分别求样本中所有点到这两个质心的距离，并标记每个样本的类别为和该样本距离最小的质心的类别，如图 c 所示，经过计算样本和红色质心和蓝色质心的距离，我们得到了所有样本点的第一轮迭代后的类别。此时我们对我们当前标记为红色和蓝色的点分别求其新的质心，如图 4 所示，新的红色质心和蓝色质心的位置已经发生了变动。图 e 和图 f 重复了我们在图 c 和图 d 的过程，即将所有点的类别标记为距离最近的质心的类别并求新的质心。最终我们得到的两个类别如图 f。

当然在实际 K-Mean 算法中，我们一般会多次运行图 c 和图 d，才能达到最终的

比较优的类别。
2）算法流程
在上一节我们对 K-Means 的原理做了初步的探讨，这里我们对 K-Means 的算法做一个总结。
首先我们看看 K-Means 算法的一些要点。
1）对于 K-Means 算法，首先要注意的是 k 值的选择，一般来说，我们会根据对数据的先验经验选择一个合适的 k 值，如果没有什么先验知识，则可以通过交叉验证选择一个合适的 k 值。
2）在确定了 k 的个数后，我们需要选择 k 个初始化的质心，就像上图 b 中的随机质心。由于我们是启发式方法，k 个初始化的质心的位置选择对最后的聚类结果和运行时间都有很大的影响，因此需要选择合适的 k 个质心，最好这些质心不能太近。

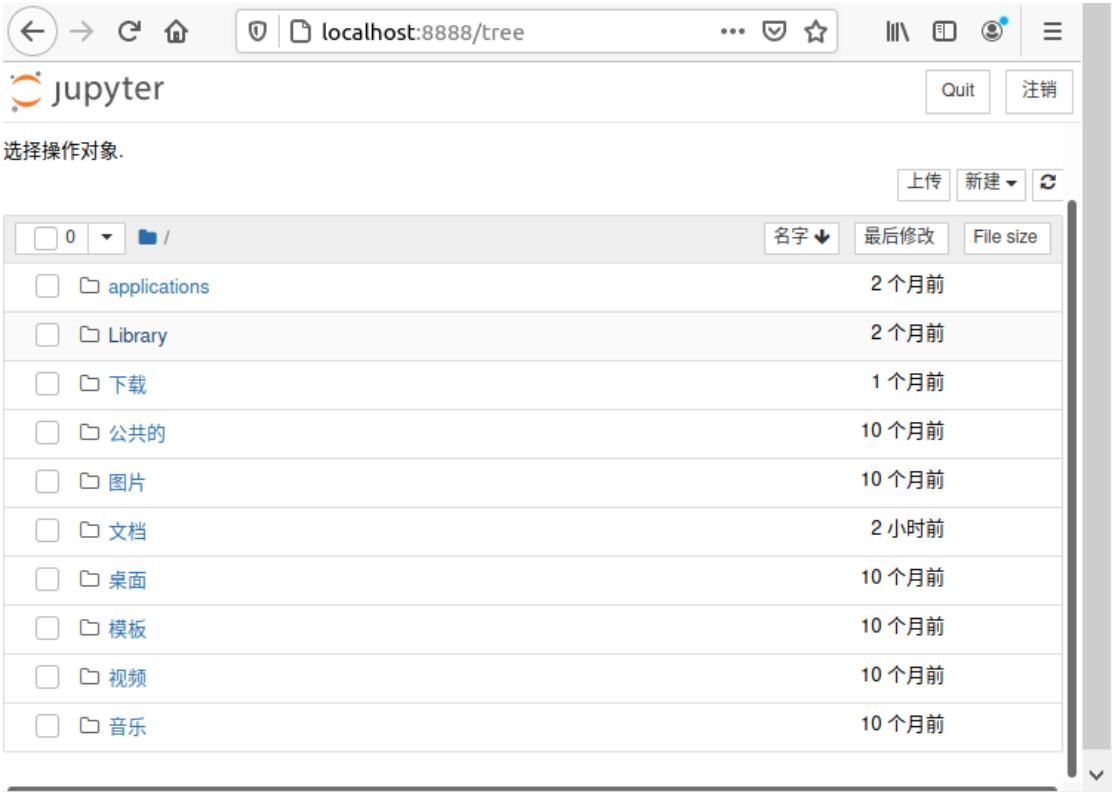
好了，现在我们来总结下传统的 K-Means 算法流程。
1、从数据集 D 中随机取 k 个元素，作为 k 个簇的各自的中心
2、分别计算剩下的元素到 k 个簇中心的相似度，将这些元素分别划归到相似度最高的簇
3、根据聚类结果，重新计算 k 个簇各自的中心，取簇中所有元素各自维度的算术平均值
4、将 D 中全部元素按照新的中心重新聚类
5、重复第 4 步，直到聚类结果不再变化
6、输出结果

2. 实验环境

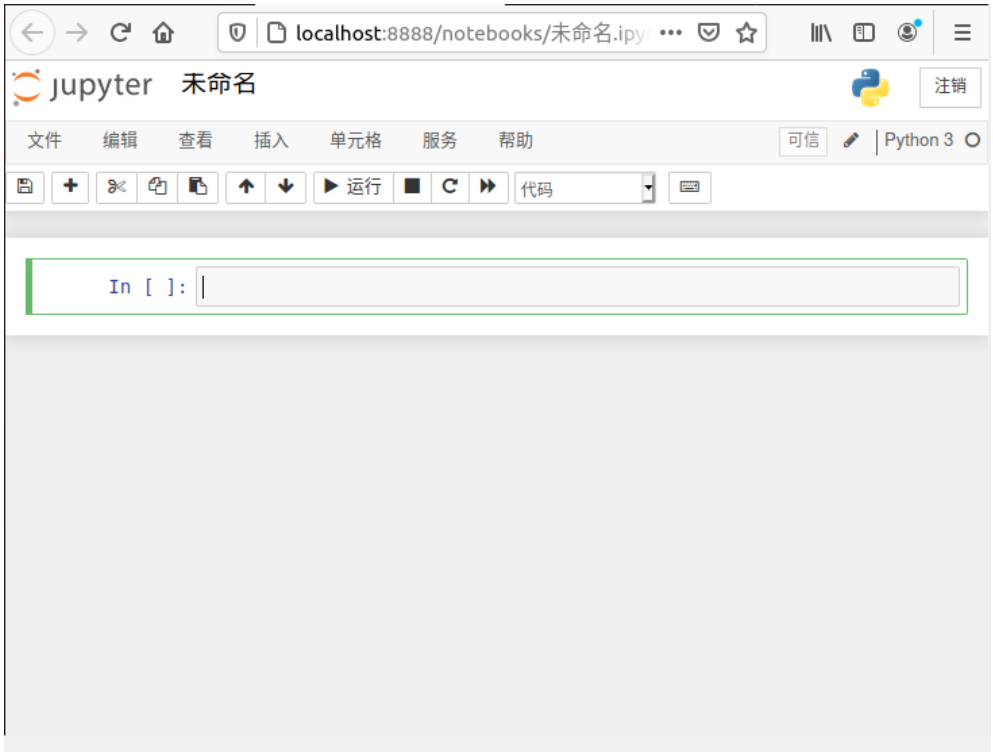
Ubuntu 20.04
Python 3.6
Jupyter notebook

3. 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



4. 实操

1)导入库

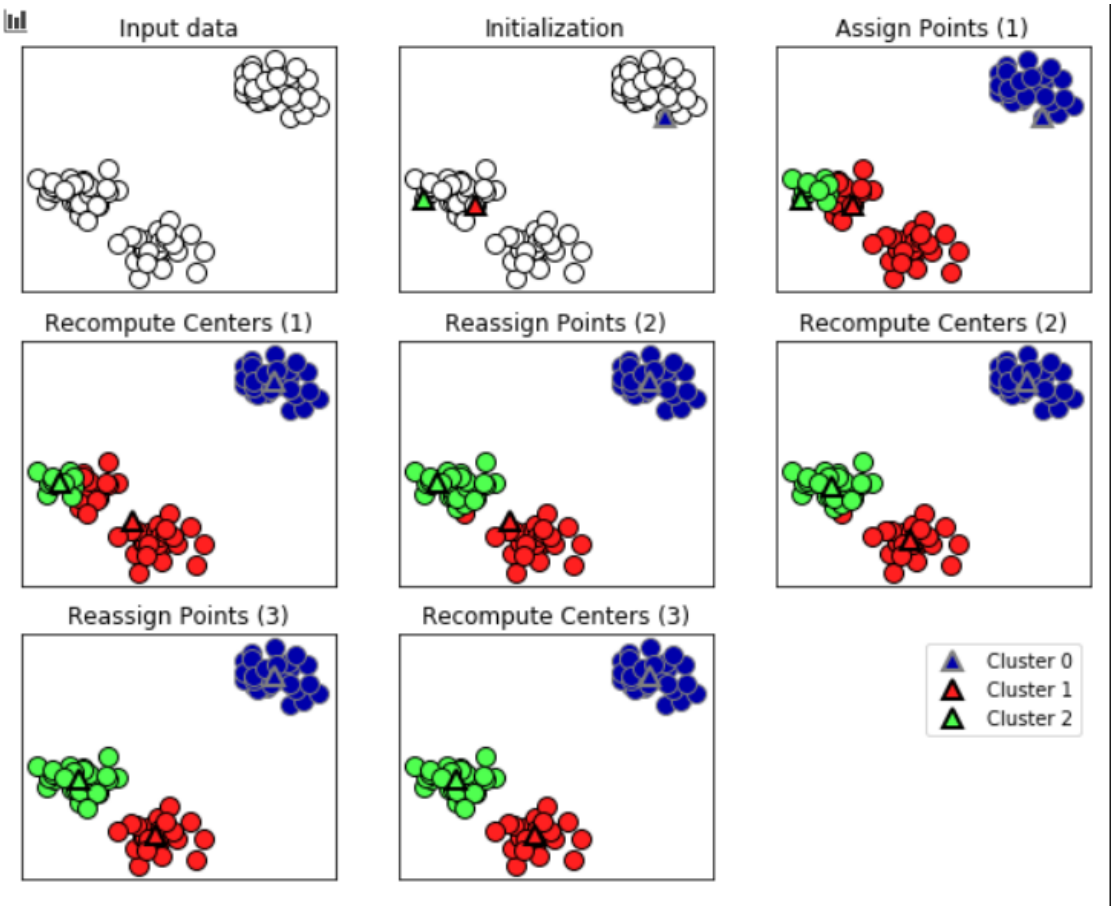
代码：

```
import numpy as np
import sklearn
import mglearn
```

```
import matplotlib.pyplot as plt
```

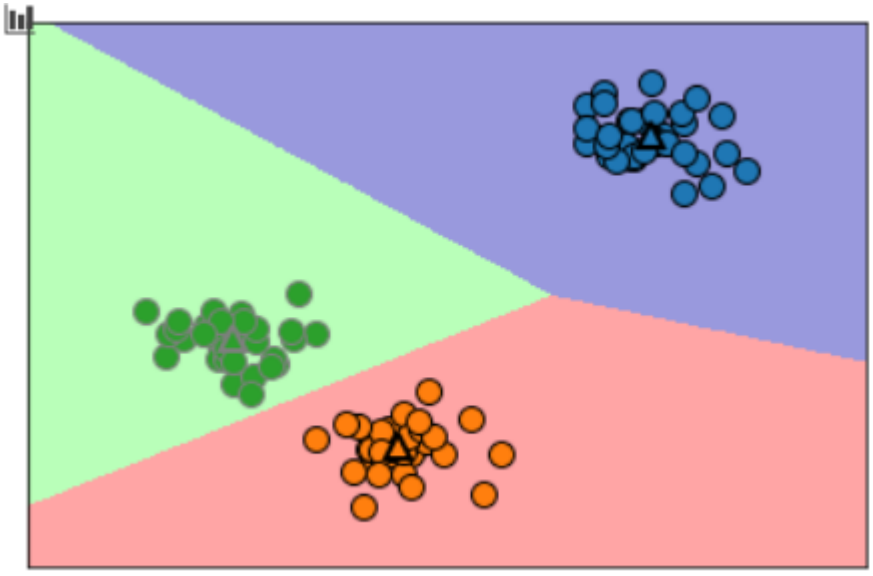
2)聚类算法流程

```
mglearn.plots.plot_kmeans_algorithm()
```



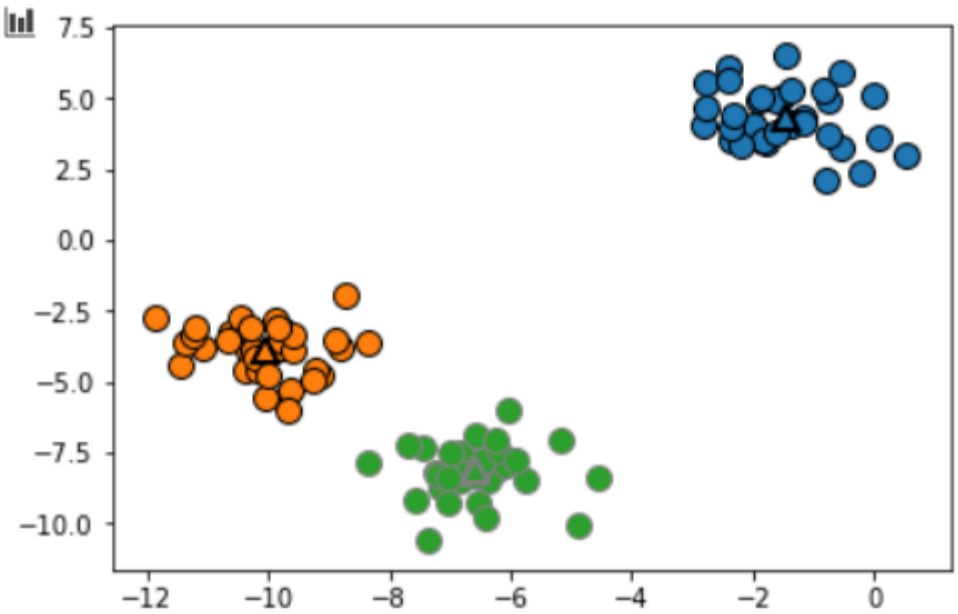
3)聚类决策边界

```
mglearn.plots.plot_kmeans_boundaries()
```



4)k 均值算法找到簇中心和边界

```
 from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# 生成模拟的二维数据
X, y = make_blobs(random_state=1)
print('shape of X:', X.shape)

# 构建聚类模型
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

# 聚类为每一个样本生成类标签
print('Cluster memberships:\n', kmeans.labels_)

# 该结果与在训练集上预测的结果一致
print('predit of X:\n', kmeans.predict(X))

# 可视化聚类结果
mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers='o')
mglearn.discrete_scatter(kmeans.cluster_centers_[:, 0],
                         kmeans.cluster_centers_[:, 1],
                         [0, 1, 2], markers='^', markeredgewidth=2)
plt.show()
```



5)使用不同簇中心数目的结果

```
 fig, axes = plt.subplots(1, 2, figsize=(10, 5))
# 2 个簇中心
kmeans = KMeans(n_clusters=2).fit(X)
assigments = kmeans.labels_
```

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], assigments, ax=axes[0])
# 5 个簇中心
kmeans = KMeans(n_clusters=5).fit(X)
assigments = kmeans.labels_

mglearn.discrete_scatter(X[:, 0], X[:, 1], assigments, ax=axes[1])
plt.show()
```