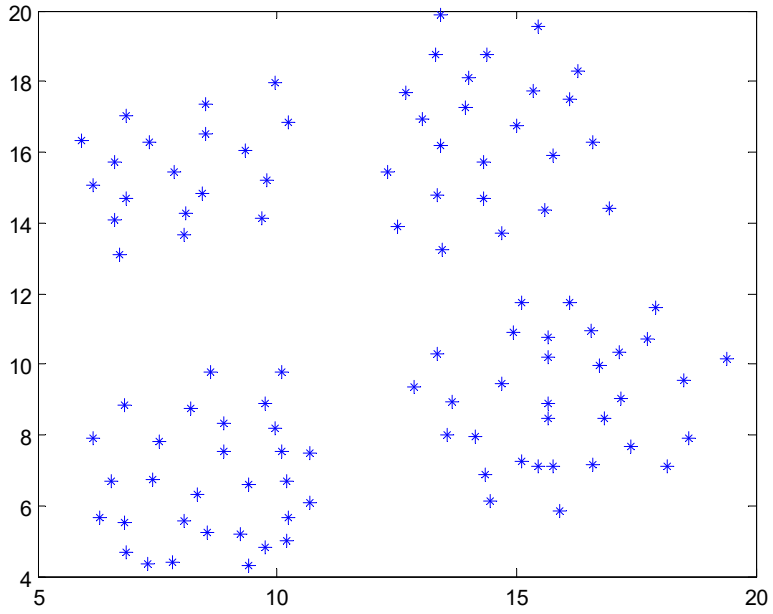
实验

实验要求：

用 k-means 算法对附件 dataforkmeans - .txt 中的数据进行聚类，k-means 算法具体可网上查询具体算法进行学习。

环境：python 3

数据分布：



验收方式：上机时验收+报告

一、k-means 算法

KMeans 聚类是最基础常用的聚类算法，聚类属于非监督学习。它的基本思想是，通过迭代寻找 K 个簇（Cluster）的一种划分方案，使得聚类结果对应的损失函数最小。其中，损失函数可以定义为各个样本距离所属簇中心点的误差平方和 J.

与分类、序列标注等任务不同，聚类是在事先并不知道任何样本标签的情况下，通过数据之间的内在关系把样本划分为若干类别，使得同类别样本之间的相似度高，不同类别之间的样本相似度低（即增大类内聚，减少类间距）。

KMeans 最核心的部分就是先固定中心点，调整每个样本所属的类别来减少 J；再固定每个样本的类别，调整中心点继续减小 J。两个过程交替循环，J 单调递减直到最（极）小值，中心点和样本划分的类别同时收敛。

二、程序实现

“程序实现”部分首先采用了已有的库函数，理解和观察了 KMeans 的实现过程，再根据原理手写代码实现。

根据附件中的数据，使用 *matplotlib.pyplot* 生成初始数据分布，如图 1 所示

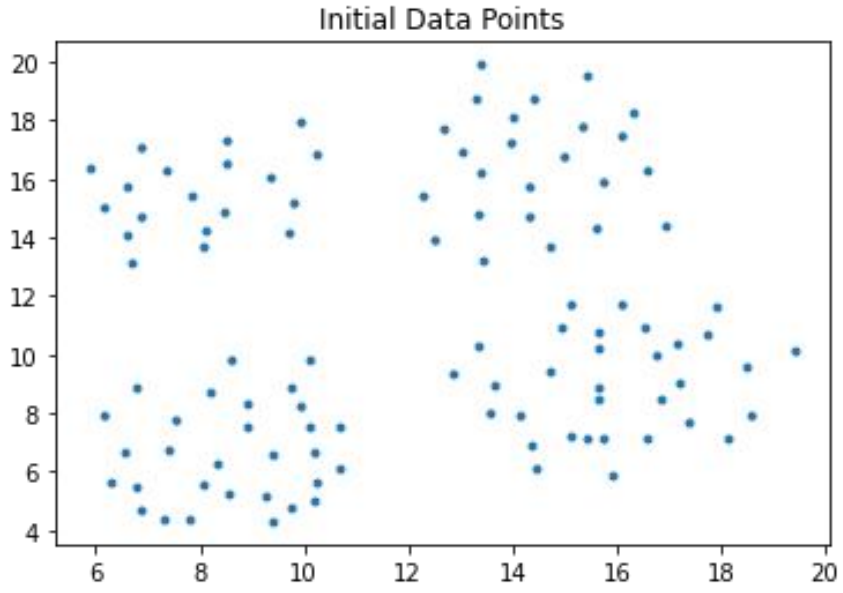


图 1 样本数据分布

1 使用 *sklearn.cluster* 库中的 *KMeans* 实现

数据预处理过程简单使用 numpy 库实现，故此处略去。通过导入所需的包，根据官方文档（https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html）简单学习其参数后，即可实现。

```
# 导入包
from sklearn.cluster import KMeans
# 数据准备
fit_data = np.vstack(list(zip(data_sample.GetX(), data_sample.GetY())))
# 指定簇的个数
kmeans = KMeans(n_clusters=4)
kmeans.fit(fit_data)
for point in kmeans.cluster_centers_:
        plt.scatter(point[0], point[1])
         print(point[0], point[1])
# 展示结果（将于“分类结果”部分展示）
plt.scatter(data_sample.GetX(), data_sample.GetY(), marker='.')
plt.show()
```

2 根据原理手写 KMeans

- 先定义总共有多少个簇（Cluster）；
- 随机选取样本点作为每个簇的簇心；

```
# 规定有多少个簇（cluster）
n_cluster = 4
# 从 105 个样本点中随机选取 4 个点作为簇心
random_choice = np.random.choice(105, 4)
cluster_init_center = [fit_data[index] for index in random_choice]
for point in cluster_init_center:
        plt.scatter(point[0], point[1])
plt.scatter(data_sample.GetX(), data_sample.GetY(), marker='.')
plt.show()
```

初始随机选取四个点的结果如图 2 所示，可以看出没有什么规律只是随机选取，接下来将进行根据这四个点的不断迭代。

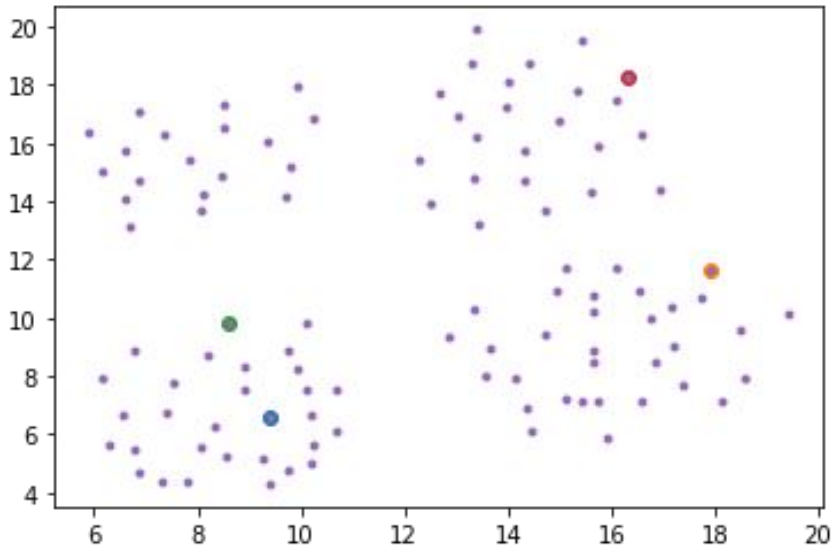


图 2 随机选取簇心

- 计算每个数据点到每个簇心的欧氏距离，将距离最近的簇心作为这个点的所属簇；
- 计算每个划分簇中样本点的均值，作为这个簇新的簇心；
- 再次计算每个样本点到新的簇心的距离，观察其所属簇是否发生改变；
- 如改变，重复上述步骤，直到每个样本点的所属簇不改变。

```
import math
cluster_center = cluster_init_center
stop = False
while(not stop):
    stop = True
    # 计算每一个样本点到初始簇心的欧氏距离
    for every_point in points:
        temp_list = []
        for every_center in cluster_center:
            temp_list.append(math.dist((every_point.x_, every_point.y_), every_center))
        index = temp_list.index(min(temp_list))
        # 终止迭代条件
        if every_point.label_ != temp_list.index(min(temp_list)):
            stop = False
        every_point.SetLabel(index)
```

```
        # 确定每一个簇的平均点
        sum_point = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
        for every_point in points:
            sum_point[every_point.label_] = [i + j for i, j in zip(sum_point[every_point.label_],
[every_point.x_, every_point.y_, 1])]
        # 新的簇心
        for center in sum_point:
            cluster_center[sum_point.index(center)] = [x / center[-1] for x in [center[0], center[1]]]
            pt = cluster_center[sum_point.index(center)]
            print(pt[0], pt[1])
            plt.scatter(pt[0], pt[1])
        plt.scatter(data_sample.GetX(), data_sample.GetY(), marker='.')
        plt.show()
```

三、分类结果

使用库函数和手写 KMeans 的方法得到了几乎相同的结果，如图 3 所示，

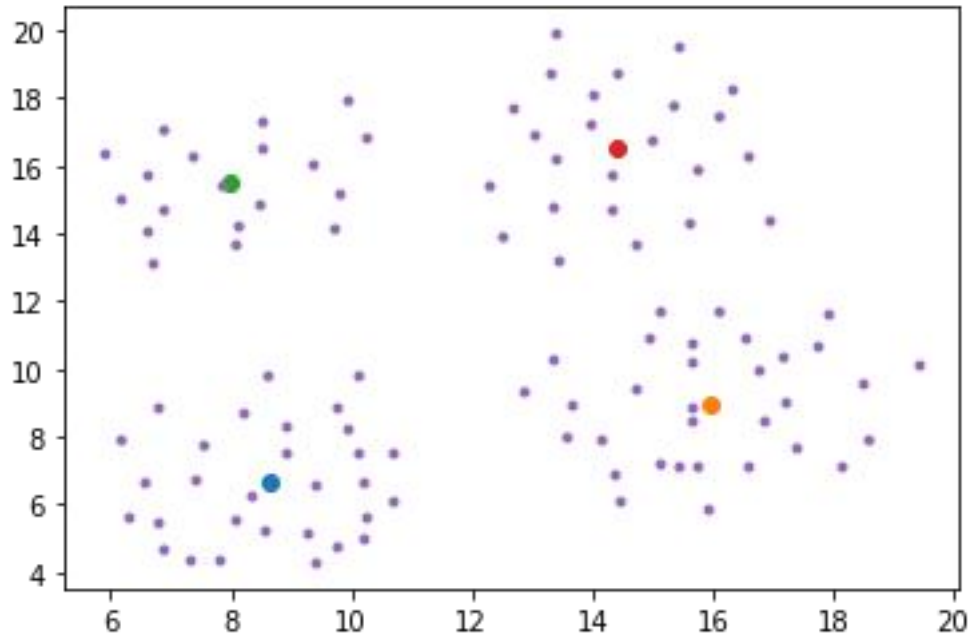


图 3 分类结果

手写 KMeans 同时展现了迭代过程，如图 4 所示

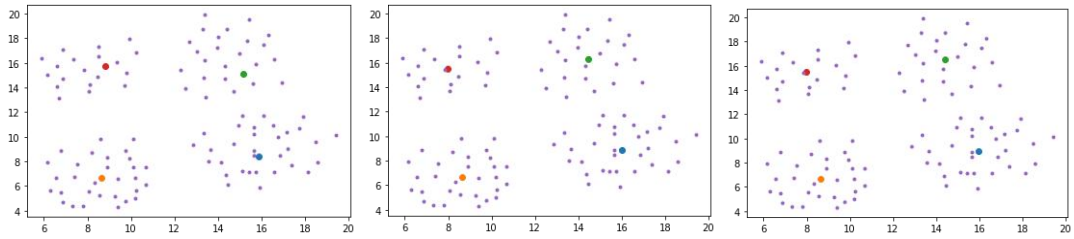


图 4 手写 KMeans 迭代过程

最终簇中心的 x，y 值以及每个簇中对应的点，如表 1 所示，

表 1 分类结果统计

	x	y	簇中点的个数
A	15.9625	8.9859	32
B	8.6266	6.6817	30
C	14.425	16.496	24
D	7.9737	15.5052	19

四、收获与体会

本次实验中，主要学习到了机器学习中比较基础且常用的 KMeans 算法，用于处理聚类问题，不仅通过 *scilearn* 库快捷地对其进行了验证和使用，也学习了 KMeans 的原理，并根据原理手写代码进行了实现，对 KMeans 的算法过程有了更进一步的理解，同时也提升了 Python 的代码能力。对于聚类的结果，也通过使用 *matplotlib* 库进行了可视化展示，得到了较好的结果。

更重要的是，学习到了对各类官方文档的查阅，库和 API 永远不是一成不变的，在使用时需要勤查官方文档，才能更好地掌握最新、更便捷的工具。

附：数据 dataforkmeans - .txt

5.9 16.35
6.85 17.05
7.35 16.3
6.85 14.7
6.6 15.7
6.15 15.05
6.6 14.1
7.85 15.45
8.5 17.35
8.45 14.85
8.1 14.25
9.35 16.05
9.95 17.95
8.5 16.5
9.8 15.2
10.25 16.85
9.7 14.15
8.05 13.65
6.7 13.1
12.7 17.7
14.4 18.75
14 18.1
13.05 16.95
12.3 15.45
15.6 14.35
15 16.75
13.4 16.2
13.35 14.8
14.3 15.7
16.3 18.3
16.1 17.5
16.6 16.3
16.95 14.4
14.7 13.7
14.3 14.7
15.75 15.9
13.95 17.25
13.3 18.75
13.4 19.9
15.45 19.55
15.35 17.75
12.5 13.9
13.45 13.25
6.15 7.9
6.8 8.85
8.6 9.8
8.9 8.35
7.55 7.8
6.3 5.65
6.8 5.5
7.4 6.75
8.35 6.3
8.05 5.55
8.55 5.25
9.4 6.6

8.9 7.55
9.75 8.9
9.95 8.2
10.1 7.55
10.25 5.65
9.25 5.2
9.4 4.3
6.85 4.7
7.3 4.35
7.8 4.4
6.55 6.7
8.2 8.75
10.1 9.8
10.7 7.5
10.7 6.1
10.2 6.7
9.75 4.8
10.2 5
14.7 9.45
15.65 10.75
15.65 10.2
15.65 8.9
15.65 8.45
16.85 8.45
17.15 10.35
17.9 11.6
18.5 9.55
18.6 7.9
18.15 7.1
15.9 5.85
14.35 6.9
12.85 9.35
13.35 10.3
15.1 11.75
16.1 11.75
14.95 10.9
13.65 8.95
13.55 8
14.15 7.95
15.1 7.25
15.45 7.1
15.75 7.1
16.6 7.15
16.75 9.95
16.55 10.95
17.2 9.05
17.4 7.65
17.75 10.7
19.4 10.15
14.45 6.15]