МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА
Факультет информатики и систем управления
Кафедра теоретической информатики и компьютерных технологий

Лабораторная работа №14
по курсу «Информационный поиск»
«Агломеративная кластеризация»

Лабораторная работа №15
по курсу «Информационный поиск»
«K-means»

Выполнил:
студент группы ИУ9-21М
Беляев А. В.

Проверила:
Лукашевич Н. В.

Москва 2019

1 Цель работы

Для заданного набора точек построить агломеративную кластеризацию по методу single-link и complete-link.

Для этого же набора точек и заданных двух изначальных точек построить кластеризацию методом K-means, выбрав зв начальные кластера 2 указанные ранее точки.

Сравнить методы.

2 Агломеративная кластеризация

2.1 Ход работы

```python
import matplotlib.pyplot as plot
import numpy as np
import scipy.cluster.hierarchy as sch


POINTS = [
    (0.6, 1.9),
    (1.8, 1.6),
    (2.7, 2.0),
    (3.0, 2.1),
    (3.0, 2.6),
    (3.1, 4.5),
    (3.8, 0.6),
    (4.2, 2.7)
]


class Cluster:
    def __init__(self):
        self.points = []

    def add_point(self, point: tuple):
        self.points.append(point)

    def coordinates(self):
        return list(point[0] for point in self.points), list(point[1]
        ↪  for point in self.points)

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return f'Cluster {self.points}'

```

```
34
35 def single_link(distance_function):
36     linkage = sch.linkage(distance_function, method='single')
37     return sch.fcluster(linkage, criterion='maxclust', t=2)
38
39
40 def complete_link(distance_function):
41     linkage = sch.linkage(distance_function, method='complete')
42     return sch.fcluster(linkage, criterion='maxclust', t=2)
43
44
45 def main():
46     distance_function = sch.distance.pdist(np.array(POINTS),
       ↪  metric='cosine')
47     clusterized = single_link(distance_function)
48     # clusterized = complete_link(distance_function)
49
50     # (-1) for each index since numeration after `fclutser` starts from
       ↪  1
51     clusterized = [cluster_indx - 1 for cluster_indx in clusterized]
52
53     clusters_num = max(clusterized) + 1
54     clusters = [Cluster() for _ in range(clusters_num)]
55
56     i = 0
57     for cluster_index in clusterized:
58         clusters[cluster_index].add_point(POINTS[i])
59         i += 1
60
61     for c in clusters:
62         xs, ys = c.coordinates()
63         plot.scatter(xs, ys)
64     plot.show()
65
66
67 if __name__ == '__main__':
68     main()
```

Рузльтаты кластеризации по методу single-link предаствалены на Рисунке 1. Complete-link – на Рисунке 2. Используемая мера близости - косинусная мера. Функция `flsuter` пакета scipy позволяет разбить данное множество на t кластеров. В данном случае t = 2.

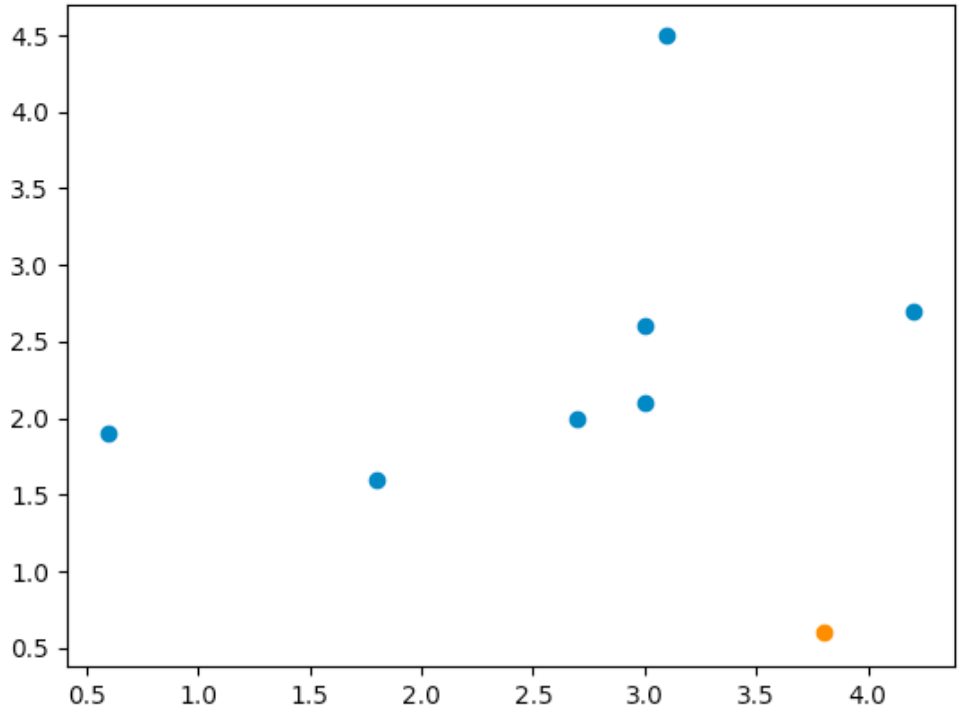

Рис. 1: Single-link

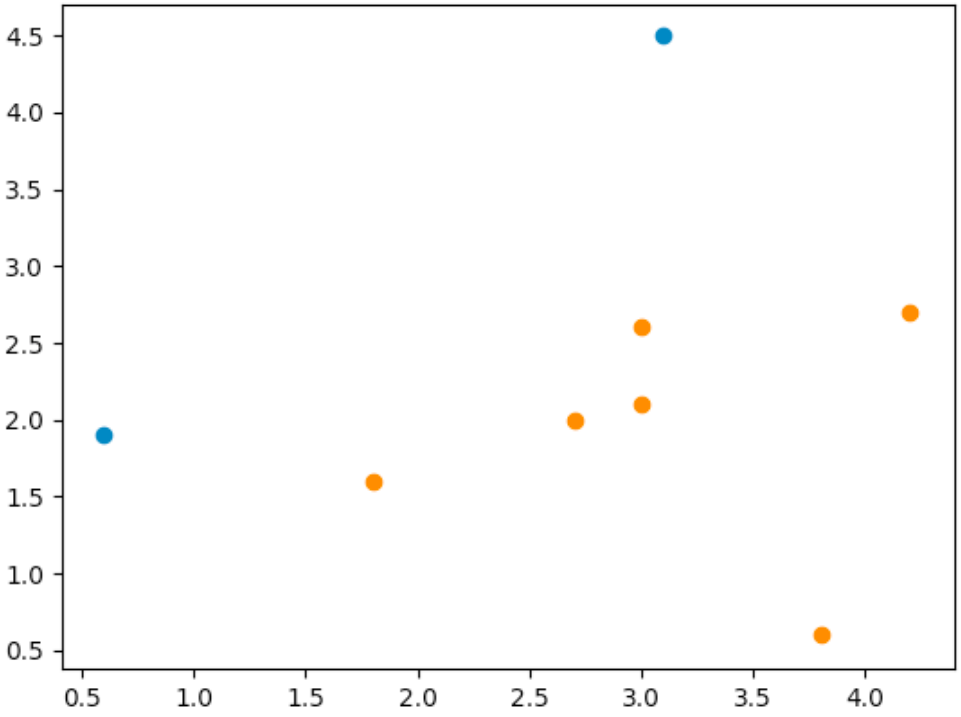

Рис. 2: Complete-link

3 Кластеризация K-means

3.1 Ход работы

```python
import math
import matplotlib.pyplot as plot
from sklearn.metrics.pairwise import cosine_similarity


POINTS = [
    (0.6, 1.9),
    (1.8, 1.6),
    (2.7, 2.0),
    (3.0, 2.1),
    (3.0, 2.6),
    (3.1, 4.5),
    (3.8, 0.6),
    (4.2, 2.7)
]

INIT_CLUSTER_CENTERS = [
    (1.8, 1.6),
    (3.0, 2.6)
]


class Point:
    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y
        self.cluster_index = None

    def to_vector(self):
        return [self.x, self.y]

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return f'[{self.x} {self.y}]'


class Cluster:
    def __init__(self, center: Point):
        self.points = []
```

```
42          self.center = center
43
44      def add_point(self, p: Point):
45          self.points.append(p)
46
47      def remove_point(self, p: Point):
48          if p in self.points:
49              self.points.remove(p)
50
51      def update_center(self):
52          if 0 != len(self.points):
53              x_avg = sum(p.x for p in self.points) / len(self.points)
54              y_avg = sum(p.y for p in self.points) / len(self.points)
55              self.center = Point(x_avg, y_avg)
56
57      def coordinates(self):
58          return list(point.x for point in self.points), list(point.y for
            ↪  point in self.points)
59
60      def __repr__(self):
61          return self.__str__()
62
63      def __str__(self):
64          return f'Cluster {len(self.points)}: {self.center}'
65
66
67  def reshuffle(points: list, clusters: list) -> bool:
68      def cosine_dist(a: Point, b: Point):
69          return cosine_similarity([a.to_vector()], [b.to_vector()])
70
71      # not used in favor of cosine_sim
72      def euclidean_dist(a: Point, b: Point):
73          x = math.fabs(a.x - b.x)
74          y = math.fabs(a.y - b.y)
75          return math.sqrt(x**2 + y**2)
76
77      # algorithm converges when there are no more changes in clusters
78      smth_has_changed = False
79      for p in points:
80
81          min_dist = float('inf')
82          new_cluster_index = 0
83
84          # which cluster the point should be added to
85          for j in range(len(clusters)):
```

```
            curr_center = clusters[j].center
            distance = cosine_dist(p, curr_center)
            if distance <= min_dist:
                new_cluster_index = j
                min_dist = distance

        if p.cluster_index != new_cluster_index:
            # remove point from old cluster
            if p.cluster_index is not None:
                clusters[p.cluster_index].remove_point(p)

            # append point to new cluster
            clusters[new_cluster_index].add_point(p)
            p.cluster_index = new_cluster_index
            smth_has_changed = True

    return smth_has_changed


def k_means_clusterize(points: list, initial_centers: list):
    clusters = [Cluster(initial_center) for initial_center in
     ↪  initial_centers]

    not_converged = True
    while not_converged:
        # assign points to nearest clusters
        not_converged = reshuffle(points, clusters)

        # update cluster centers
        [c.update_center() for c in clusters]
        print(clusters)

    return clusters


def main():
    points = [Point(p[0], p[1]) for p in POINTS]
    centers = [Point(p[0], p[1]) for p in INIT_CLUSTER_CENTERS]

    clusters = k_means_clusterize(points, centers)

    # draw clusters
    for c in clusters:
        xs, ys = c.coordinates()
        plot.scatter(xs, ys)
```

```
130
131     # draw cluster centers
132     cxs, cys = list(c.center.x for c in clusters), list(c.center.y for c
        ↪  in clusters)
133     plot.scatter(cxs, cys, color='black')
134     plot.show()
135
136
137 if __name__ == '__main__':
138     main()
```

Метод сошелся за 2 итерации. Результаты кластеризации представлены на Рисунке 3. Цетры кластеров обозначены черным цветом.

Используемая мера близости - косинусная мера.

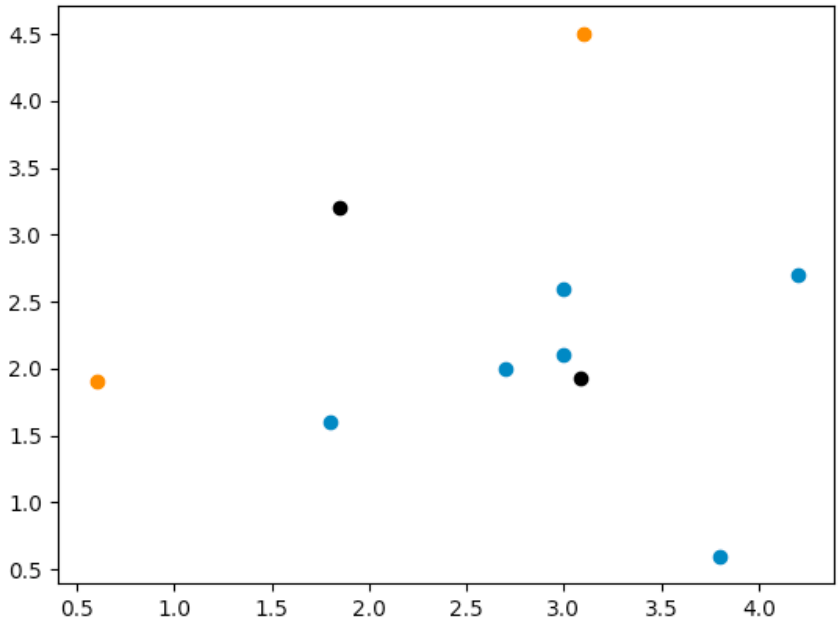

Рис. 3: K-means

4 Выводы

Были реализованы 2 способа кластеризации – агломеративная (с помощью single-link и complete-link) и K-means.

Метод кластеризации K-means предельно прост и понятен и может быть реализован вручную за адекватное время.

В то же время агломеративная кластеризация треубет больших усилий для ручной реализации. Поэтому был использован пакет scipy. Главным недостатком подобных пакетов (scipy, scikit и прочие) является то, что реализация метода скрыта

глубоко в коде (в данном случае пакет представляет лишь обертку над реализацией на языке C) и не может быть изучена, а управление методами реализовано посредством большого количества параметров. Таким образом, для простейшего использования метода необходимо изучить все параметры, а затем, ввиду их большого количества, подбирать их практически случайным образом. При этом, параметры позволяют лишь «подогнать» получаемый результат под ожидаемый в конкретном случае результат и, с большой долей вероятности, на другом наборе данных работать не будут.

После определенного количества попыток, подобные параметры для методов класетризации из пкета scipy были подобраны и кластеризация прошла успешно. Тенденция к «широким» и «узким» кластерам отражена на соответствующих рисунках агломеративнйо кластеризации. Она некоторм образом кореллирует с кластреизацией K-means.