



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	Информатика и системы управления
КАФЕДРА	Информационная безопасность (ИУ8)

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Лабораторная работа №9 на тему:
«Алгоритмы кластерного анализа данных»

Вариант 8

Выполнил: Песоцкий А. А.,
студент группы ИУ8-61

Проверила: Коннова Н.С.,
доцент каф. ИУ8

Москва, 2020 г.

Цель работы

Исследовать применение основных алгоритмов кластерного анализа, включая их модификации, на примере различных типов данных.

Постановка задачи

Выполнить разбиение колледжей г. Москвы с помощью НС Кохонена с использованием метрики принадлежности округу Москвы (Евклидово расстояние до координат центра округа).

Кластерный анализ — процедура, заключающаяся в сборе данных, содержащих информацию о выборке объектов, и последующем упорядочивании объектов в сравнительно однородные группы на основе какого-либо признака(ов). Формально: пусть X — множество объектов, Y — множество кластеров. Задана функция расстояния между объектами $\rho(x, x')$. Имеется конечная обучающая выборка объектов $X^m = \{x_1, \dots, x_m\} \subset X$. Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике ρ , а объекты разных кластеров существенно отличались. При этом каждому объекту $x_i \in X^m$ приписывается номер кластера y_i . *Алгоритм кластеризации* — это функция $\varphi: X \rightarrow Y$, которая любому объекту $x \in X$ ставит в соответствие номер кластера $y \in Y$. Множество Y в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров с точки зрения того или иного критерия качества кластеризации.

НС Кохонена — класс НС, основным элементом которых является слой Кохонена, состоящий из k адаптивных линейных сумматоров. Они имеют одинаковое число входов m и получают на свои входы вектор входных сигналов $\mathbf{x} = (x_1, \dots, x_m)$. На выходе j -го линейного элемента имеем сигнал

$$y_j = w_{j0} + \sum_{i=1}^m w_{ji} x_i$$

где j — номер нейрона; w_{j0} — пороговый коэффициент; i — номер входа; w_{ji} — весовой коэффициент i -го входа j -го нейрона.


Выходные сигналы слоя Кохонена обрабатываются по правилу «победитель получает всё»: наибольший сигнал превращается в единичный, остальные обращаются в нуль. Таким образом, применительно к задаче кластеризации каждому j -му нейрону ставятся в соответствие точки-центры кластеров, для входного вектора $\mathbf{x} = (x_1, \dots, x_m)$ вычисляются расстояния $\rho_j(\mathbf{x})$, и тот нейрон, до которого это расстояние минимально, выдает единицу, остальные — нуль.

Ход работы

В качестве данных для кластеризации возьмём выборку платных парковок города Москвы размером 200 объектов. Критерием является вместительность парковок.

Для этого предварительно получим через API mos.ru данные по парковкам в формате json:

(https://apidata.mos.ru/v1/datasets/623/features?api_key=e7001de51ff1e03bfe9a57548d25075b)



```
{
  "features": [
    {
      "geometry": {
        "coordinates": [
          [
            [
              37.5948822,
              55.7154477
            ],
            [
              37.595188,
              55.7157046
            ]
          ]
        ],
        "type": "MultiLineString"
      },
      "properties": {
        "DatasetId": 623,
        "VersionNumber": 9,
        "ReleaseNumber": 4,
        "RowId": null,
        "Attributes": {
          "ParkingName": "Парковка №3020 (48)",
          "ParkingZoneNumber": "3020",
          "global_id": 64189573,
          "AdmArea": "Южный административный округ",
          "District": "Донской район",
          "Address": "город Москва, Малая Калужская улица, дом 27",
          "CarCapacity": 4,
          "CarCapacityDisabled": 2,
          "Tariffs": [
            {
              "HourPrice": 40,
              "TimeRange": "круглосуточно",
              "FirstHalfHour": null,
              "SecondHalfHour": null
            }
          ]
        }
      }
    }
  ]
}
```

Рисунок 1. Полученный файл parking.json

Выберем 200 случайных парковок и приступим к кластеризации.

Центрами кластеров будем считать набор вместимостей:

$$W = [1, 5, 10, 15, 20, 25, 30, 35]$$

Полученные вместимости (200 объектов):

[4, 18, 4, 9, 10, 3, 3, 14, 14, 8, 7, 24, 16, 20, 22, 21, 15, 5, 1, 7, 3, 6, 6, 9, 20, 29, 4, 4, 8, 7, 6, 4, 13, 5, 5, 6, 2, 10, 19, 4, 1, 3, 21, 3, 4, 26, 12, 15, 7, 7, 8, 3, 6, 8, 4, 4, 15, 23, 14, 33, 24, 12, 11, 9, 28, 13, 21, 16, 9, 7, 8, 11, 6, 7, 7, 5, 2, 13, 2, 9, 7, 12, 7, 10, 15, 17, 14, 8, 7, 11, 16, 5, 10, 7, 5, 19, 6, 10, 13, 5, 3, 4, 3, 1, 4, 3, 13, 24, 14, 17, 4, 11, 3, 4, 8, 5, 2, 6, 4, 10, 2, 6, 10, 4, 10, 3, 3, 1, 10, 2, 1, 20, 10, 17, 6, 4, 22, 8, 12, 9, 24, 17, 23, 3, 16, 25, 5, 9, 16, 2, 3, 31, 2, 10, 10, 3, 2, 8, 9, 21, 14, 7, 3, 6, 5, 9, 19, 17, 7, 4, 12, 8, 11, 3, 5, 3, 7, 7, 9, 10, 5, 10, 2, 7, 29, 19, 4, 7, 12, 7, 6, 6, 6, 3, 2, 5, 2, 3, 3, 2]

В качестве алгоритма возьмём НС Кохонена. Для каждого объекта (вместимости парковки) вычисляем расстояние до каждого кластера, выбираем минимальное и относим объект к этому кластеру. Для НС Кохонена веса W — центры кластеров, а входные векторы — координаты точек. Нейрон-победитель, расстояние до которого минимально, обращается в единицу, остальные нейроны — в ноль. Таким образом, результатом работы НС является матрица размером 200×8 (размер выборки на число кластеров):

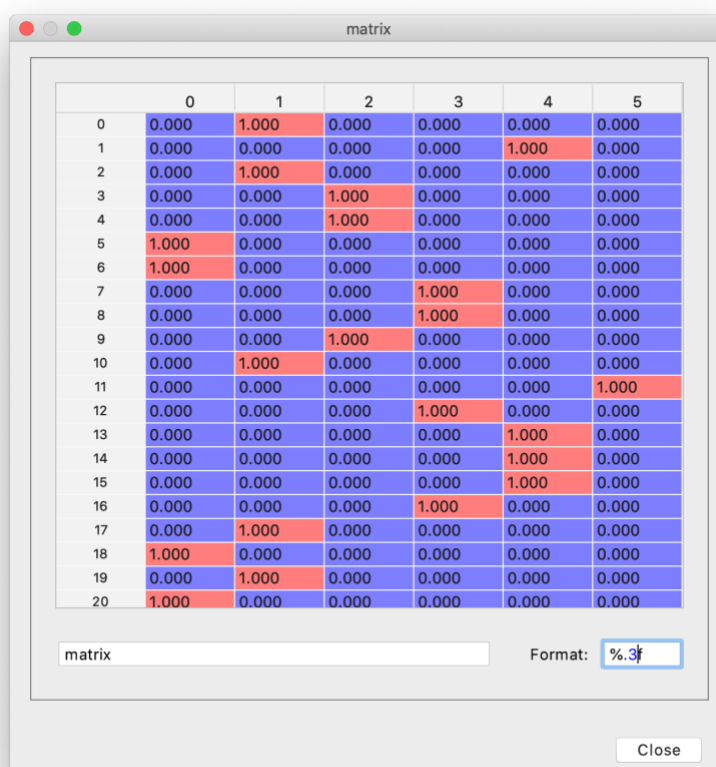


Рисунок 2. Часть получившейся матрицы

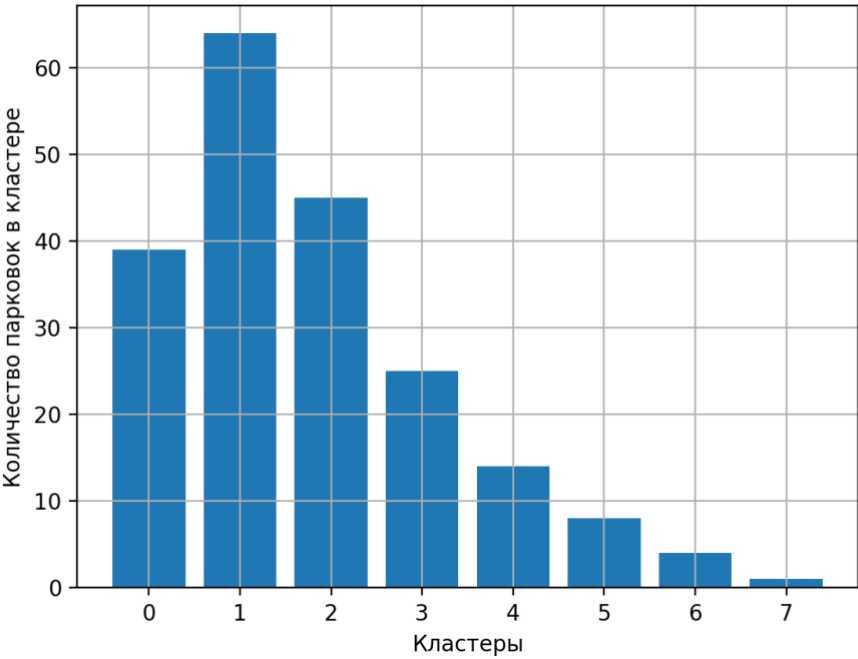
В результате работы алгоритма получаем таблицу принадлежности парковки (её вместительности) кластеру:

№ кластера	Центр	Вместимости парковок
0	1	[3, 3, 1, 3, 2, 1, 3, 3, 3, 2, 2, 3, 3, 1, 3, 3, 2, 2, 3, 3, 1, 2, 1, 3, 2, 3, 2, 3, 2, 3, 3, 3, 2, 3, 2, 2, 3, 3, 2]
1	5	[4, 4, 7, 5, 7, 6, 6, 4, 4, 7, 6, 4, 5, 5, 6, 4, 4, 7, 7, 6, 4, 4, 7, 6, 7, 7, 5, 7, 7, 7, 5, 7, 5, 6, 5, 4, 4, 4, 4, 5, 6, 4, 6, 4, 6, 4, 5, 7, 6, 5, 7, 4, 5, 7, 7, 5, 7, 4, 7, 7, 6, 6, 6, 5]
2	10	[9, 10, 8, 9, 8, 10, 12, 8, 8, 12, 11, 9, 9, 8, 11, 9, 12, 10, 8, 11, 10, 10, 11, 8, 10, 10, 10, 10, 10, 8, 12, 9, 9, 10, 10, 8, 9, 9, 12, 8, 11, 9, 10, 10, 12]
3	15	[14, 14, 16, 15, 13, 15, 15, 14, 13, 16, 13, 15, 17, 14, 16, 13, 13, 14, 17, 17, 17, 16, 16, 14, 17]
4	20	[18, 20, 22, 21, 20, 19, 21, 21, 19, 20, 22, 21, 19, 19]
5	25	[24, 26, 23, 24, 24, 24, 23, 25]
6	30	[29, 28, 31, 29]
7	35	[33]

```
/Users/asymmetriq/Documents/PyCharm/lab_6/venv/bin/python /Users/asymmetriq/Documents/PyCharm/lab_6/cluster_analysis.py
| № кластера | Центр |
| 0 | 1 | [3, 3, 1, 3, 2, 1, 3, 3, 3, 2, 2, 3, 3, 1, 3, 3, 2, 2, 3, 3, 1, 2, 1, 3, 2, 3, 2, 3, 2, 3, 3, 3, 2, 3, 2, 2, 3, 3, 2]
| 1 | 5 | [4, 4, 7, 5, 7, 6, 6, 4, 4, 7, 6, 4, 5, 5, 6, 4, 4, 7, 7, 6, 4, 4, 7, 6, 7, 7, 5, 7, 7, 7, 5, 7, 5, 6, 5, 4, 4, 4, 4, 5, 6, 4, 6, 4, 6, 4, 5, 7, 6,
| 2 | 10 | [9, 10, 8, 9, 8, 10, 12, 8, 8, 12, 11, 9, 9, 8, 11, 9, 12, 10, 8, 11, 10, 10, 11, 8, 10, 10, 10, 10, 10, 8, 12, 9, 9, 10, 10, 8, 9, 9, 12, 8, 11, 9, 10, 10, 8, 9,
| 3 | 15 | [14, 14, 16, 15, 13, 15, 15, 14, 13, 16, 13, 15, 17, 14, 16, 13, 13, 14, 17, 17, 17, 16, 16, 14, 17]
| 4 | 20 | [18, 20, 22, 21, 20, 19, 21, 21, 19, 20, 22, 21, 19, 19]
| 5 | 25 | [24, 26, 23, 24, 24, 24, 23, 25]
| 6 | 30 | [29, 28, 31, 29]
| 7 | 35 | [33]
```

Рисунок 3. В программе эта таблица представлена в ASCII.

Также для наглядности построим диаграмму количества парковок в каждом кластере:



Выводы

В ходе выполнения работы был изучен алгоритм кластерного анализа данных с помощью НС Кохонена.

В качестве данных для кластеризации использовали выборку платных парковок города Москвы размером 200 объектов. Критерием являлась вместительность парковок.

Приложение А.

Файл 'cluster_analysis.py'.

```
from computation import *
from prettytable import PrettyTable

"""
Основная функция кластеризации
:param capacities: вместительности парковок
:param centers: центры кластеров
:param return: матрица весов, матрица распределения на кластеры
"""

def clusterize(capacities, centers):
    matrix = np.zeros((len(capacities), len(centers)))
    for i, value in enumerate(capacities):
        for j, center in enumerate(centers):
            matrix[i][j] = np.abs(value - center)
    clusterized = [[] for x in range(len(centers))]
    for i in range(len(matrix)):
        index = np.argmin(matrix[i])
        clusterized[index].append(capacities[i])
        render(matrix[i], index)
    return matrix, clusterized

if __name__ == "__main__":
    # get_json("https://apidata.mos.ru/v1/datasets/623/features?api_key=e7001de51ff1e03bfe9a57548d25075b")
    data = load_json('parking.json')
    capacities = get_capacities(200, data)
    centers = [1, 5, 10, 15, 20, 25, 30, 35]
    matrix, result = clusterize(capacities, centers)

    table = PrettyTable(['№ кластера', 'Центр', 'Вместимости парковок'])
    for index, (line, weight) in enumerate(zip(result, centers)):
        table.add_row([index, np.round(weight, 4), line])
    print(table)
    get_diagram(result)
```

Файл 'computation.py'.

```

import requests
import json
import numpy as np
import matplotlib.pyplot as plt
from itertools import islice

"""
Функция получает и записывает json-файл
:param url: ссылка
"""
def get_json(url):
    items = requests.get(url)
    data = items.json()
    with open('parking.json', 'w', encoding='utf-8') as write_file:
        json.dump(data, write_file, ensure_ascii=False, indent=4)

"""
Функция загружает сохранённый json-файл
:param filename: имя файла
:param return: данные
"""
def load_json(filename):
    with open(filename, 'r') as read_file:
        data = json.load(read_file)
    return data

"""
Функция выдаёт выборку вместительностей
:param n: число объектов
:param json_data: данные
:param return: массив вместительностей
"""
def get_capacities(n, json_data):
    prk = json_data['features']
    n_elements = list(islice(prk, n))

    capacity_list = list()
    for parking in n_elements:
        capacity_list.append(parking['properties']['Attributes']['CarCapacity'])
    return capacity_list

```



```

"""
Функция вычисляет расстояние между объектом и центром
:param weight: вес
:param value: значение вместительности
:param return: расстояние (разница)
"""

def get_diff(weight, value):
    return value - weight

```

```

"""
Функция обработки матрицы (победитель получает всё)
:param array: строка матрицы
:param index: минимальный индекс
"""

def render(array, index):
    for i in range(0, index):
        array[i] = 0
    array[index] = 1
    for i in range(index+1, len(array)):
        array[i] = 0

```

```

"""
Функция построения диаграммы по матрице распределения
:param matrix: матрица распределения
"""

def get_diagram(matrix):
    sizes = list()
    indexes = list()
    for index, line in enumerate(matrix):
        indexes.append(index)
        sizes.append(len(line))
    plt.grid()
    plt.bar(indexes, sizes)
    plt.xlabel('Кластеры')
    plt.ylabel('Количество парковок в кластере')
    plt.show()

```

