**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ИС**

**ОТЧЁТ**

**по практической работе № 6**

**по дисциплине «Статистический анализ»**

**Тема: Кластерный анализ. Метод k-средних**

**Вариант № 8**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 9372 | |  | Иванов Р.С. |
| Преподаватель |  |  | Сучков А.И. |

Санкт-Петербург

2021

**Цель работы.**

Освоение основных понятий и некоторых методов кластерного анализа, в частности, метода k-means.

**Основные теоретические положения.**

Формулы для формирования интервального вариационного ряда:

- количество интервалов, где n - объем выборки.

R = xmax - xmin - размах выборки

- длина интервала

– нормализация

Алгоритм k-means

**Шаги алгоритма:**

1. **Начальный шаг: инициализация кластеров**

Выбирается произвольное множество точек μi, i=1,...,k, рассматриваемых как начальные центры кластеров: μ(0)i=μi, i=1,...,k

1. **Распределение векторов по кластерам**

**Шаг** t: ∀xi ∈ X, i=1...n : xi ∈ Sj ⟺ j = argminkρ(xi,μ(t−1)k)2

1. **Пересчет центров кластеров**

**Шаг**t: ∀I =1,...,k : μ(t)i= ∑x ∈Si x

1. **Проверка условия останова:**
   * **if**  ∃i ∈ : μ(t)i≠μ(t−1)i **then**
     + t=t+1;
     + goto 2;
   * **else**
     + **stop**

**Постановка задачи.**

Дано конечное множество из объектов, представленных двумя признаками (в качестве этого множества принимаем исходную двумерную выборку, сформированную ранее в практической работе №4). Выполнить разбиение исходного множества объектов на конечное число подмножеств (кластеров) с использованием метода k-means. Полученные результаты содержательно проинтерпретировать.

**Выполнение работы.**

Во время выполнения работы был написан код на языке Python, выполняющий поставленную задачу. Реализовано чтение и дальнейшая обработка данных из Price\_Mileage.csv, одобренного преподавателем. Для этого была использована библиотека csv.

***Пункт 1***

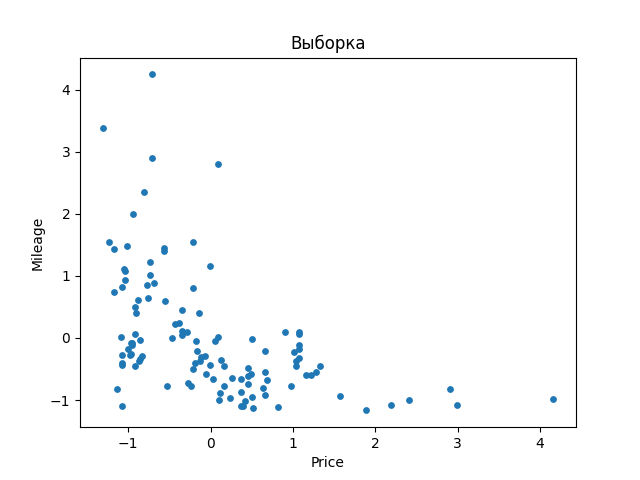
 Нормализовано множество точек из предыдущего раздела, на графике отображено полученное множество.

Рисунок 1 (Выборка)

***Пункт 2***

Определена грубая верхняя оценка количества кластеров.

***Пункт 3***

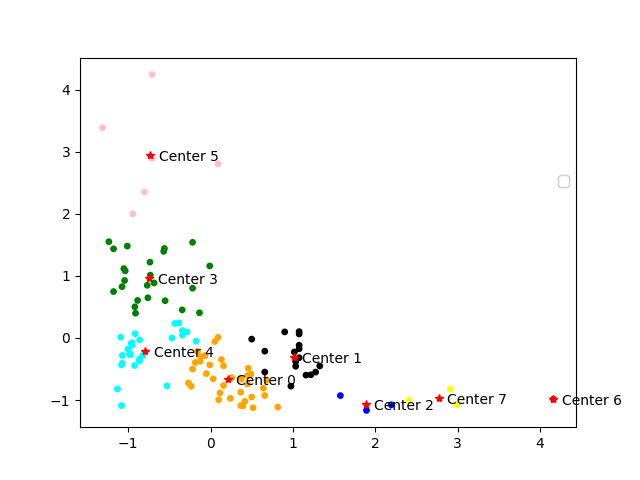
 Реализован алгоритм k-means. Отображены полученные кластеры, выделен каждый кластер разным цветом, отмечены центроиды.

Рисунок 2 (K-means)

***Пункт 4***

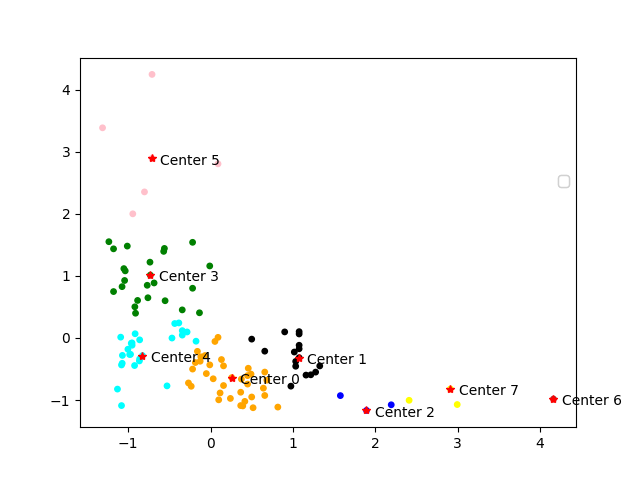
 Реализован алгоритм kmedoids (для чётных вариантов). Отображены полученные кластеры, выделен каждый кластер разным цветом, отмечены центроиды.

Рисунок 3 (K-medoids)

***Пункт 5***

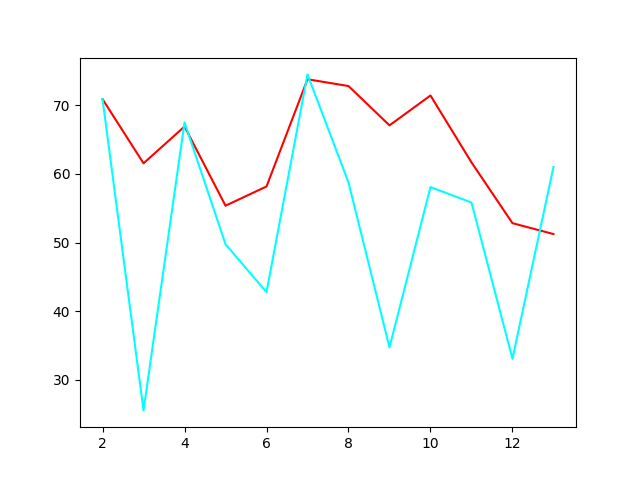
 Построен график для определения оптимального количества кластеров с помощью метода силуэтов.

Рисунок 4 (Метод силуэтов)

Красная линия на графике — это применение метода силуэтов для k-means, синяя – k-medoids. По графику можно сказать, что для обоих методов оптимальным количеством кластеров является k = 7.

***Пункт 6***

Были оценены методы разбиения с помощью получения суммарного внутрикластерного расстояния между точками. Были получены следующие величины:

|  |  |
| --- | --- |
| K-means | K-medoids |
| 21.94 | 35.31 |

По полученным данным можно сказать, что на данной выборке лучше работает алгоритм k-means. После его применение внутрикластерное расстояние меньше, чем при применении алгоритма k-medoids.

***Пункт 7***

Алгоритмы k-means и k-medoids очень похожи по своей реализации, единственное отличие отличается в том методе пересчёта центройдов, в одном случае это точка(не обязательно присутствующая в кластере) расстояние до которой минимально для каждого члена кластера, в другом обязательно лежащая в кластере точка.

**Вывод.**

В процессе выполнения практической работы были реализованы такие алгоритмы разбиения на кластеры, как k-means , k-medians. Выполнена верхняя оценка количества кластеров и оценка оптимального количества кластеров методом локтя. Заметили, что в алгоритмах k-means и k-medians первичные центроиды выбираются случайно из множества всех точек, таким образом, эти алгоритмы очень чувствительны к их выбору.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

import numpy as np  
import matplotlib.pyplot as plt  
import math  
import csv  
# import scipy  
import random  
import sklearn  
from openpyxl import Workbook  
  
# from openpyxl.drawing.image import Image  
from sklearn.cluster import KMeans  
  
  
def calc\_distance(coordinate\_1, coordinate\_2):  
 return (coordinate\_1[0] - coordinate\_2[0]) \*\* 2 + (coordinate\_1[1] - coordinate\_2[1]) \*\* 2  
  
  
def recalc\_centroids(cluster):  
 centroid = []  
 temp\_1 = 0  
 temp\_2 = 0  
 len\_cluster = len(cluster)  
 for i in range(len\_cluster):  
 temp\_1 += cluster[i][0]  
 temp\_2 += cluster[i][1]  
 centroid.append(temp\_1 / len\_cluster)  
 centroid.append(temp\_2 / len\_cluster)  
 return centroid  
  
  
def recalc\_medoids(s, clu, medoid):  
 m = 10 \*\* 10  
 ans = -1  
 for i in range(len(clu)):  
 med = medoid  
 cluL = clu  
 med = s[cluL[i]]  
 rang = all\_ranges(s, clu, med)  
 if rang < m:  
 m = rang  
 ans = i  
 if ans != -1:  
 medoid = s[clu[ans]]  
 return clu, medoid  
  
  
def all\_ranges(s, clu, medoid):  
 ans = 0  
 for i in range(len(clu)):  
 ans += calc\_distance(medoid, s[clu[i]])  
 return ans  
  
  
def calc\_distance(coordinate\_1, coordinate\_2):  
 return (coordinate\_1[0] - coordinate\_2[0]) \*\* 2 + (coordinate\_1[1] - coordinate\_2[1]) \*\* 2  
  
  
def cluster\_index(D, cluster):  
 return [D.index(coordinate) for coordinate in cluster]  
  
  
def k\_means(centers, s):  
 centers\_new = centers  
 ku = len(centers)  
 for max in range(200):  
 thr = []  
 clu = []  
 clus = []  
 for i in range(ku):  
 clus.append([])  
  
 for point in s:  
 distance = []  
 min = 1000  
 l = 0  
 for j in range(ku):  
 distance.append(calc\_distance(centers\_new[j], point))  
  
 if distance[j] < min:  
 min = distance[j]  
 l = j  
 clus[l].append(point)  
 centers\_old = centers\_new.copy()  
 for i in range(ku):  
 centers\_new[i] = recalc\_centroids(clus[i])  
 for j in range(ku):  
 thr.append(bool(calc\_distance(centers\_old[j], centers\_new[j])))  
 for l in range(ku):  
 clu.append(cluster\_index(s, clus[l]))  
 if not any(thr):  
 break  
 return centers\_new, clu  
  
  
def k\_meds(meds, s):  
 med\_new = meds  
 ku = len(meds)  
 for mx in range(200):  
 thr = []  
 clu = []  
 clus = []  
 for i in range(ku):  
 clus.append([])  
  
 for point in s:  
 distance = []  
 min = 1000  
 l = 0  
 for j in range(ku):  
 distance.append(calc\_distance(med\_new[j], point))  
 if distance[j] < min:  
 min = distance[j]  
 l = j  
 clus[l].append(point)  
 meds\_old = med\_new.copy()  
 for l in range(ku):  
 clu.append(cluster\_index(s, clus[l]))  
 for i in range(ku):  
 clu[i], med\_new[i] = recalc\_medoids(s, clu[i], meds[i])  
 for j in range(ku):  
 thr.append(calc\_distance(meds\_old[j], med\_new[j]))  
 if not any(thr):  
 break  
 return med\_new, clu  
  
  
def make\_plot(centers, sample, clusters):  
 mycolors = ['orange', 'black', 'blue', 'green', 'cyan', 'pink', 'brown', 'yellow']  
 sample = np.array(sample)  
 fig, ax = plt.subplots()  
 for i, coordinate in enumerate(centers):  
 ax.plot(coordinate[0], coordinate[1], 'r\*')  
 ax.text(coordinate[0] + 0.1, coordinate[1] - 0.1, 'Center ' + str(i))  
 labels\_ = [0] \* len(sample)  
 sum =0  
 for i in range(len(clusters)):  
 for index in range(len(clusters[i])):  
 labels\_[clusters[i][index]] = mycolors[i]  
  
 # for cluster in clusters:  
 # for index in cluster:  
 # ax.text(D[index, 0] + 0.01, D[index, 1] + 0.1, index)  
 scatter = ax.scatter(sample[:, 0], sample[:, 1], s=15, c=labels\_, cmap='rainbow')  
 legend = ax.legend(\*scatter.legend\_elements(fmt='Cluster {x:.0f}'), bbox\_to\_anchor=(1, 0.7))  
 ax.add\_artist(legend)  
 plt.show()  
  
  
sample = []  
  
with open('Price-Mileage.csv') as csv\_file: # Читаем выборку из файла 0 работы  
 spam\_reader = csv.reader(csv\_file, quotechar='|')  
 for row in spam\_reader:  
 x, y = row[0].split(';')  
 if y.isdigit() and x.isdigit():  
 sample.append([int(x), int(y)])  
  
n = len(sample)  
k = math.ceil(math.sqrt(n / 2))  
  
Xs = 0 # Выброчное среднее по x  
Ys = 0 # По y  
  
Dsx = 0 # Дисперсия по x  
Dsy = 0 # По s  
  
sy = 0 # Исправленное СКО по y  
sx = 0 # По x  
  
for i in range(n):  
 Xs += sample[i][0]  
Xs = Xs / n  
  
for i in range(n):  
 Ys += sample[i][1]  
Ys = Ys / n  
  
for i in range(n):  
 Dsx += (sample[i][0] - Xs) \*\* 2  
Dsx = Dsx / n  
  
for i in range(n):  
 Dsy += (sample[i][1] - Ys) \*\* 2  
Dsy = Dsy / n  
  
sx = math.sqrt(Dsx) \* n / (n - 1)  
sy = math.sqrt(Dsy) \* n / (n - 1)  
  
for i in range(n): # Нормализация точек  
 sample[i][0] = (sample[i][0] - Xs) / sx  
 sample[i][1] = (sample[i][1] - Ys) / sy  
  
X = [sample[i][0] for i in range(n)]  
Y = [sample[i][1] for i in range(n)]  
  
plt.scatter(x=X, y=Y, s=15)  
ax = plt.gca()  
ax.set(title='Выборка')  
# Добавляем подписи к осям:  
ax.set\_xlabel('Price')  
ax.set\_ylabel('Mileage')  
plt.show()  
  
e = 0.01  
means\_init = random.sample(sample, k)  
  
U, clusters = k\_means(means\_init, sample)  
make\_plot(U, sample, clusters)  
  
meds\_init = random.choices(sample, k = k)  
meds\_index = []  
# for i in range(k):  
# for j in range(len(sample)):  
# if meds\_init[i] == sample[j]:  
# meds\_index.append(j)  
# break  
# samp = sample  
# samp = [x for x in samp if x not in meds\_init]  
  
meds\_init, clusters = k\_meds(means\_init, sample)  
make\_plot(meds\_init, sample, clusters)  
  
def silhouette(centers, clus, samp):  
 for i in range(len(samp)):  
 for j in range(len(clus)):  
 for l in range(len(clus[j])):  
 if i == clus[j][l]:  
 samp[i].append(j)  
 break  
 if len(samp[i]) == 3:  
 break  
  
 def min\_rang(point, cens, sam):  
 mi = 10\*10  
 for i in range(len(cens)):  
 if i != point[2]:  
 ze = calc\_distance(point, cens[i])  
 if ze < mi:  
 mi = ze  
 ans = i  
 return(ans)  
  
 ans = 0  
 for i in range(len(samp)):  
 a = all\_ranges(samp, clus[samp[i][2]], samp[i])  
 b = all\_ranges(samp, clus[min\_rang(samp[i], centers, samp)], samp[i])  
 ans += (b-a)/max(a, b)  
 samp[i].pop(2)  
 return ans  
  
silhous\_means = []  
silhous\_meds = []  
for i in range(2, 14):  
 means\_init = random.sample(sample, i)  
 means\_init, clusters1 = k\_means(means\_init, sample)  
 silhous\_means.append(silhouette(means\_init, clusters1, sample))  
  
  
 meds\_init = random.choices(sample, k=i)  
 meds\_init, clusters2 = k\_meds(meds\_init, sample)  
 silhous\_meds.append(silhouette(meds\_init, clusters2, sample))  
  
 #make\_plot(means\_init, sample, clusters1)  
 #make\_plot(meds\_init, sample, clusters2)  
  
X = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]  
plt.plot(X, silhous\_means, color='red')  
plt.plot(X, silhous\_meds, color='cyan')  
plt.show()  
  
meds\_init = random.choices(sample, k=7)  
meds\_init, clusters2 = k\_meds(meds\_init, sample)  
  
means\_init = random.sample(sample, 7)  
means\_init, clusters1 = k\_means(means\_init, sample)  
  
def ranges\_clusters(clus, centrs, samp):  
 ans = 0  
 for i in range(len(clus)):  
 ans += all\_ranges(samp,clus[i],centrs[i])  
 return ans  
  
qual\_mean = ranges\_clusters(clusters1, means\_init, sample)  
qual\_meds = ranges\_clusters(clusters2, meds\_init, sample)  
  
make\_plot(means\_init, sample, clusters1)  
make\_plot(meds\_init, sample, clusters2)  
  
print(qual\_mean)  
print(qual\_meds)