«Внутренние» критерии качества

 «Внутренние (собственные)» параметры качества кластеризации зависят лишь от собственных свойств набора данных и не требуют дополнительной информации такой как обучающий набор данных.

 Таким параметром качества могут служить внутри кластерные и меж кластерные расстояния.

 Исходя из определения, кластеризация данных сводится к минимизации меж кластерных расстояний:

 Алгоритм кластеризации K-means использует аналогичную формулу:

- разброс точек от центра кластера.

 Оптимальное (минимальное) значение величины меняется с заданным числом кластеров и может быть использовано для определения числа кластеров.

 Пример:



Исходный датасет

Кластеризация для разного числа кластеров:



Величина внутри кластерного разброса:

= 9.832727169934977

= 8.391418849961259

= 6.556106062807646

= 7.387319955728708

= 7.61211507101378

= 7.987895612302369

Аналогично для Ирисов Фишера:



= 1.814964552432165

= 1.5823754361487337

= 1.5823754361487337

= 1.574988314800791

= 1.6013994087896355

= 1.6139755244745022

from sklearn import datasets

import pandas as pd

import numpy as np

from sklearn import preprocessing

import matplotlib.pyplot as plt

import random

from math import sqrt

iris = pd.read\_csv('../iris.csv')

print(iris.info(),"\n")

DIM\_N = 4

POINT\_N = iris.shape[0]

CLUST\_N = 7

targ = np.array(iris)[:, DIM\_N]

targ = preprocessing.LabelEncoder().fit\_transform(targ)

data = np.array(iris)[:, 0:DIM\_N]

"""

DIM\_N = 2

POINT\_N = 300

CLUST\_N = 7

centers = [[0.0, 0.0], [3.5, 0.0], [0.0, 3.5], [3.5, 3.5]]

data, targ = datasets.make\_blobs(n\_samples=POINT\_N, centers=centers, shuffle=False, random\_state=42)

for p in data:

 plt.scatter(p[0], p[1], c='black', s=20)

plt.show()

"""

class POINT:

 def \_\_init\_\_(self, cl, x):

 self.clust = cl

 self.X = x

class CLUSTER(POINT):

 def \_\_init\_\_(self, cl, x):

 POINT.\_\_init\_\_(self, cl, x)

 self.X2 = np.empty(DIM\_N)

 self.N = 0

 def Dist(self, p):

 d2 = 0.0

 for i in range(DIM\_N):

 d2 += (self.X[i] - p.X[i])\*\*2

 return sqrt( d2 )

 def Eval\_Center(self, P):

 self.N = 0

 self.X = [0.0 for i in range(DIM\_N)]

 self.X2 = [0.0 for i in range(DIM\_N)]

 for p in P:

 if p.clust == self.clust:

 self.N += 1

 for i in range (DIM\_N):

 self.X[i] += p.X[i]

 self.X2[i] += p.X[i]\*p.X[i]

 for i in range(DIM\_N):

 if self.N > 0:

 self.X[i] /= self.N

 self.X2[i] = self.X2[i]/self.N - self.X[i]\*self.X[i]

#random.seed(5)

random.seed(0)

for cl\_n in range(2,CLUST\_N+1):

 Cl = [CLUSTER(i, [4.0\*random.random()+2.0 for j in range(DIM\_N)]) for i in range(cl\_n)]

 PP = [POINT(random.randint(0,cl\_n), data[i,:]) for i in range(POINT\_N)]

 while (1):

 CC = [POINT(0, Cl[i].X) for i in range(cl\_n)]

 for cl in Cl:

 cl.Eval\_Center(PP)

 for p in PP:

 dd = [Cl[i].Dist(p) for i in range(cl\_n)]

 p.clust = np.argmin (dd)

 d = 0.0

 for i in range(cl\_n):

 d += Cl[i].Dist(CC[i])

 if d < 0.01\*cl\_n:

 break

 d = 0.0

 for cl in Cl:

 for i in range(DIM\_N):

 d += cl.X2[i]

 print (cl\_n, d)

 colors = ['blue', 'green', 'magenta', 'cyan', 'brown', 'orange', 'black']

 for p in PP:

 plt.scatter(p.X[0], p.X[1], c=colors[p.clust], s=20)

# for cl in Cl:

# plt.scatter(cl.X[0], cl.X[1], c='red', marker="\*")

 plt.show()