Развитие алгоритма

#  I. Имеются точки, не принадлежащие ни одному из кластеров:



 Для предотвращения включения «фоновых» точек в кластер ведём радиус кластера:

Для нормального распределения:

Прежний критерий принадлежности кластеру

заменяем на

***(Реализовать самостоятельно)***

#  II. Форма кластера отличается от сферы:

Алгоритм кластеризации Kmeans принадлежит к классу центроидных алгоритмов: кластер может быть представлен как центр и некоторая выпуклая область вокруг него.



Левый кластер - центроидный, правый нет.

 Для кластеров с формой близкой к эллипсоидной (левый кластер на картинке) вместо метрики Эвклида:

используется метрика Махаланобиса:

где – матрица, обратная матрице ковариации

здесь – координатные индексы,

*k* – номер точки в множестве.

 Диагональные элементы матрицы ковариаций совпадают с дисперсией данных вдоль соответствующих осей.

Недиагональные элементы матрицы зависимости (связи) соответствующих характеристик данных.

Пример, три кластера, нормальное распределение:



На левой картинке параметр , на правой: .

Нечёткая кластеризация (Fuzzy Classifier Means - FCM)

Для случая перекрывающихся кластеров можно использовать алгоритм FCM (Fuzzy Classifier Means) – нечёткой кластеризации. В этом случае для каждого его элемента вычисляется вероятность (степень) принадлежности кластеру.

В основе алгоритма FCM лежит использование матрицы , описывающей вероятность того, что i-й элемент принадлежит кластеру с номером k:

Параметр q задаёт "разброс" относительно центра кластера: чем больше значение q, тем больше вероятность включить в кластер далёкие точки.

 Центр кластера рассчитывается как взвешенное среднее координат точек, образующих кластер:

 Сам алгоритм аналогичен алгоритму К-средних, описанному ранее, с очевидной заменой критерия принадлежности объекта кластеру на 2-м шаге алгоритма. Принадлежность точки кластеру носит случайный характер с вероятностью задаваемой матрицей . Так, например, для случая двух кластеров точка принадлежит первому кластеру с вероятностью и второму с вероятностью .

 Программная реализация алгоритма:

import matplotlib.pyplot as plt

from sklearn import datasets

import random

from math import sqrt

from math import pow

DIM\_N = 2

POINT\_N = 300

CLUST\_N = 2

class POINT:

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

 self.clust = cl

 self.X = x

 self.Y = y

class CLUSTER(POINT):

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

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

 self.N = 0

 def Dist(self, p):

 return sqrt( (self.X - p.X)\*\*2 + (self.Y - p.Y)\*\*2 )

 def Eval\_Center(self, P, M):

 self.N = 0

 self.X = 0.0

 self.Y = 0.0

 a = 0.0

 for i in range(POINT\_N):

 if P[i].clust == self.clust:

 self.N += 1

 self.X += P[i].X\*M[i][self.clust]

 self.Y += P[i].Y\*M[i][self.clust]

 a = a + M[i][self.clust]

 self.X /= a

 self.Y /= a

centers = [(0, 0), (2, 2)]

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

for i in range(POINT\_N):

 if targ[i] == 0:

 plt.scatter(data[i][0], data[i][1], c='blue', s=20)

 else:

 plt.scatter(data[i][0], data[i][1], c='green', s=20)

plt.show()

Cl = [CLUSTER(0, 2.0\*random.random()-1.0, 2.0\*random.random()-1.0), CLUSTER(1, 2.0\*random.random()-1.0, 2.0\*random.random()-1.0)]

#Cl = [CLUSTER(0, 0, 0), CLUSTER(1, 2, 2)]

PP = [POINT(CLUST\_N, data[i,0], data[i,1]) for i in range(POINT\_N)]

QU = 0.3

for nn in range(10):

 CC = [POINT(0, Cl[0].X, Cl[0].Y), POINT(1, Cl[1].X, Cl[1].Y)]

 Mu = []

 for i in range(POINT\_N):

 Mu.append([])

 a = 0.0

 for k in range(CLUST\_N):

 Mu[i].append( pow(Cl[k].Dist(PP[i]), 1.0/QU) )

 a = a + Mu[i][k]

 for k in range(CLUST\_N):

 Mu[i][k] = Mu[i][k]/a

 for i in range(POINT\_N):

 r = random.random()

 if r < Mu[i][0]:

 PP[i].clust = 0

 else:

 PP[i].clust = 1

 for k in range(CLUST\_N):

 Cl[k].Eval\_Center(PP, Mu)

 for p in PP:

 if p.clust == 0:

 plt.scatter(p.X, p.Y, c='blue', s=20)

 else:

 plt.scatter(p.X, p.Y, c='green', s=20)

 for cl in Cl:

 plt.scatter(CC[cl.clust].X, CC[cl.clust].Y, c='cyan', marker="\*")

 plt.scatter(cl.X, cl.Y, c='red', marker="\*")

 plt.show()

Результат вычислений для параметра «жёсткости» *q*=0.3:







Для значения *q*=0.1 получаем более «чёткий» результат кластеризации:



#  II. Кластеры сложной формы. Метод ближайших соседей.



В определении кластеризации

Сумму расстояний вычислять не по всем парам точек, а лишь для соседних точек:

Мы приходим к алгоритму кластеризации – метод ближайших соседей.

 В приведённом ниже примере используется обучающий набор данных с известным распределением по кластерам. Приведённая программа реализует алгоритм классификации KNN – метод ближайших соседей.

import matplotlib.pyplot as plt

import numpy as np

import random

from math import sqrt

from math import cos

from math import sin

CLUST\_N = 2

POINT\_N = 300

KNBR\_N = 5

def DataMoons (A, N):

 for i in range(0, N):

 f = 3.14 \* random.random()

 r = 0.2 \* random.normalvariate(0.0, 0.4) + 0.9

 if random.random() < 0.5:

 A[i].clust = 0

 A[i].X = 0.5 + r \* cos(f)

 A[i].Y = -0.25 + r \* sin(f)

 else:

 A[i].clust = 1

 A[i].X = -0.5 + r \* cos(f)

 A[i].Y = 0.25 - r \* sin(f)

 return

class POINT:

 def \_\_init\_\_(self, X, Y):

 self.X = X

 self.Y = Y

 self.Kn = [0 for i in range(0, KNBR\_N)]

 self.clust = 0

 def Neighbors(self, A):

 l = [i for i in range(0, len(A))]

 for k in range(0, KNBR\_N):

 n = -1

 d = 1.0e+99

 for i in l:

 if d > (self.X-A[i].X)\*\*2 + (self.Y-A[i].Y)\*\*2:

 n = i

 d = (self.X-A[i].X)\*\*2 + (self.Y-A[i].Y)\*\*2

 self.Kn[k] = n

 l.remove(n)

PPtrn = [POINT(0.0, 0.0) for i in range(0, POINT\_N)]

DataMoons(PPtrn, POINT\_N)

PPtst = [POINT(0.0, 0.0) for i in range(0, POINT\_N)]

DataMoons(PPtst, POINT\_N)

for p in PPtrn:

 if p.clust == 0:

 plt.scatter(p.X, p.Y, c='blue', marker="+")

 else:

 plt.scatter(p.X, p.Y, c='green', marker="+")

for p in PPtst:

 if p.clust == 0:

 plt.scatter(p.X, p.Y, s=20, c='black')

 else:

 plt.scatter(p.X, p.Y, s=20, c='black')

plt.show()

for p in PPtrn:

 if p.clust == 0:

 plt.scatter(p.X, p.Y, c='blue', marker="+")

 else:

 plt.scatter(p.X, p.Y, c='green', marker="+")

for p in PPtst:

 p.Neighbors(PPtrn)

 cl = [0 for i in range(0, CLUST\_N)]

 for k in range(0, KNBR\_N):

 cl[PPtrn[p.Kn[k]].clust] += 1

 p.clust = np.argmax(cl, axis=0)

 if p.clust == 0:

 plt.scatter(p.X, p.Y, s=20.0, c='blue')

 else:

 plt.scatter(p.X, p.Y, s=20.0, c='green')

plt.show()



‘+’ обозначает обучающую последовательность, синий и зелёный цвета соответствуют кластерам, чёрные точки – тестовый набор данных.



Результат классификации (обучение с учителем) данных.

 С использованием библиотеки scikit-learn:

import matplotlib.pyplot as plt

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

import random

from math import sqrt

from math import cos

from math import sin

CLUST\_N = 2

POINT\_N = 200

KNBR\_N = 5

data = np.array([[0.0, 0.0] for i in range(2\*POINT\_N)])

targ = np.array([2 for i in range(2\*POINT\_N)])

for i in range(2\*POINT\_N):

 f = 3.14 \* random.random()

 r = 0.2 \* random.normalvariate(0.0, 0.4) + 0.9

 if random.random() < 0.5:

 targ[i] = 0

 data[i,0] = 0.5 + r \* cos(f)

 data[i,1] = -0.25 + r \* sin(f)

 else:

 targ[i] = 1

 data[i,0] = -0.5 + r \* cos(f)

 data[i,1] = 0.25 - r \* sin(f)

data\_train, data\_test, targ\_train, targ\_test = train\_test\_split(data, targ, test\_size=0.5, random\_state=0)

for i in range(POINT\_N):

 if targ\_train[i] == 0:

 plt.scatter(data\_train[i,0], data\_train[i,1], c='blue', marker="+")

 else:

 plt.scatter(data\_train[i,0], data\_train[i,1], c='green', marker="+")

 plt.scatter(data\_test[i,0], data\_test[i,1], s=20, c='black')

plt.show()

neigh = KNeighborsClassifier(n\_neighbors=5)

neigh.fit(data\_train, targ\_train)

res = neigh.predict(data\_test)

for i in range(POINT\_N):

 if res[i] == 0:

 plt.scatter(data\_test[i,0], data\_test[i,1], c='blue', s=20)

 else:

 plt.scatter(data\_test[i,0], data\_test[i,1], c='green', s=20)

plt.show()

Результат работы программы:





 Кластеризация, основанная на плотности, метод DBSCAN.

Идея метода: Если в заданную окрестность точки (eps=0.22) попадает более заданного числа точек (min\_samples=5), то все эти точки принадлежат одному кластеру.

import matplotlib.pyplot as plt

import numpy as np

from sklearn import metrics

from sklearn.cluster import DBSCAN

import random

from math import sqrt

from math import cos

from math import sin

CLUST\_N = 2

POINT\_N = 200

KNBR\_N = 5

data = np.array([[0.0, 0.0] for i in range(POINT\_N)])

targ = np.array([2 for i in range(POINT\_N)])

for i in range(POINT\_N):

 f = 3.14 \* random.random()

 r = 0.2 \* random.normalvariate(0.0, 0.4) + 0.9

 if random.random() < 0.5:

 targ[i] = 0

 data[i,0] = 0.5 + r \* cos(f)

 data[i,1] = -0.25 + r \* sin(f)

 else:

 targ[i] = 1

 data[i,0] = -0.5 + r \* cos(f)

 data[i,1] = 0.25 - r \* sin(f)

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

plt.show()

db = DBSCAN(eps=0.22, min\_samples=5).fit(data)

for i in range(POINT\_N):

 if db.labels\_[i] == 0:

 plt.scatter(data[i,0], data[i,1], c='blue', s=20)

 else:

 if db.labels\_[i] == 1:

 plt.scatter(data[i,0], data[i,1], c='green', s=20)

 else:

 plt.scatter(data[i,0], data[i,1], c='red', s=20)

plt.show()



В данном случае мы получили 2 кластера. Могло быть и по-другому:



Результат зависит от распределения точек и значений параметров eps, min\_samples.

 III. Иерархическая кластеризация.

 Кластеризация, основанная на построении иерархии кластеров. Существует два подхода:

* *Агломеративные методы* (*agglomerative*): новые кластеры создаются путем объединения более мелких кластеров;
* *Дивизивные* *методы* (*divisive*): новые кластеры создаются путем деления крупных кластеров на более мелкие.

Далее будем рассматривать агломеративные методы.

Используются различные принципы объединения кластеров: метод ближайшего соседа, метод дальнего соседа, центроидный метод, …, метод Уорда (*Ward’s method*).

Метод Уорда использует минимизация прироста суммы квадратов расстояний объектов до центра кластера (прирост дисперсии).

Алгоритм можно представить в виде след. шагов:

import matplotlib.pyplot as plt

import random

from math import sqrt

POINT\_N = 300

DIM\_N = 2

CLUST\_N = 3

class POINT:

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

 self.cl = cl

 self.X = x

 self.Y = y

class CLUSTER:

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

 self.point = [ind]

 self.clust = cl

 self.N = 1

 self.Sx = self.X = x

 self.Sy = self.Y = y

 self.Sxx = x\*x

 self.Syy = y\*y

 def Dist2(self, p):

 return (self.X - p.X)\*\*2 + (self.Y - p.Y)\*\*2

 def Merge(self, C):

 self.point.extend(C.point)

 self.N += C.N

 self.Sx += C.Sx

 self.Sy += C.Sy

 self.X = self.Sx/self.N

 self.Y = self.Sy/self.N

 self.Sxx += C.Sxx

 self.Syy += C.Syy

PP = [POINT(0, 0.0, 0.0) for i in range(0, POINT\_N)]

cl = [[0.0, 0.0], [4.0, 0.0], [0.0, 4.0]]

for i in range(0, POINT\_N):

 r = random.random()

 for c in range(0, CLUST\_N):

 if r <= (c+1)/CLUST\_N:

 PP[i].cl = c

 PP[i].X = random.normalvariate(cl[c][0], 1.0)

 PP[i].Y = random.normalvariate(cl[c][1], 1.0)

 break;

CC = [CLUSTER(i, i, PP[i].X, PP[i].Y) for i in range(0, POINT\_N)]

ClustN = POINT\_N

DD = []

for i in range(0, ClustN):

 dd = []

 for j in range(0, i):

# dd.append(sqrt(CC[i].Dist2(CC[j]))) # Centroid

 dd.append( CC[i].Dist2(CC[j]) \* CC[i].N\*CC[j].N/(CC[i].N+CC[j].N)/(CC[i].N+CC[j].N) ) # Word

 DD.append(dd)

while ClustN > CLUST\_N:

 i0 = 1

 j0 = 0

 d = DD[i0][j0]

 for i in range(1, ClustN):

 for j in range(0, i):

 if DD[i][j] < d:

 d = DD[i][j]

 i0 = i

 j0 = j

 CC[i0].Merge(CC[j0])

 for i in range(i0+1, ClustN):

# DD[i][i0] = sqrt(CC[i0].Dist2(CC[i])) # Centroid

 DD[i][i0] = CC[i0].Dist2(CC[i]) \* CC[i0].N\*CC[i].N/(CC[i0].N+CC[i].N)/(CC[i0].N+CC[i].N) # Ward s method

# DD[i][i0] = min(DD[i][i0], DD[i][j0]) # Nearest points

 for j in range(0, i0):

# DD[i0][j] = sqrt(CC[i0].Dist2(CC[j])) # Centroid

 DD[i0][j] = CC[i0].Dist2(CC[j]) \* CC[i0].N\*CC[j].N/(CC[i0].N+CC[j].N)/(CC[i0].N+CC[j].N) # Ward s method

 for i in range(j0+1, ClustN):

 DD[i].pop(j0)

 DD.pop(j0)

 CC.pop(j0)

 ClustN -= 1

colors = [plt.cm.tab10(float(n)/ClustN) for n in range(ClustN)]

for n in range(0, ClustN):

 for i in CC[n].point:

 plt.scatter(PP[i].X, PP[i].Y, color=colors[n], s=20)

plt.show()



 Тоже с использованием библиотеки sklearn:

import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import AgglomerativeClustering

from sklearn import datasets

POINT\_N = 300

CLUST\_N = 3

centers = [[0.0, 0.0], [3.0, 0.0], [0.0, 3.0]]

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

res = AgglomerativeClustering(n\_clusters=3, linkage='ward').fit(data)

colors = [plt.cm.tab10(float(n)/CLUST\_N) for n in range(CLUST\_N)]

for i in range(POINT\_N):

 plt.scatter(data[i,0], data[i,1], color=colors[res.labels\_[i]], s=20)

plt.show()

