­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Контрольна робота №1-2

з курсу «Теорія Алгоритмів»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 14

Виконав студент гр. КНз-2

Чалий Михайло

­­

Львів 2014

# Завдання №1

Здійснити розпізнавання образів із застосуванням евристичного алгоритму порогової величини. Порогова величина Т=2.

X1(1,2), X2(1,3), X3(1,4), X4(1,5), X5(2,2), X6(2,3), X7(2,4), X8(3,7), X9(3,8), X10(3,9)

## Рішення

import sys, math, random

class Thresold(object):

def \_\_init\_\_(self, points):

self.\_points = Points.from\_tuples(points)

self.\_centers = []

self.\_clusters = []

self.\_t = 0;

def cluster(self, t):

self.\_t = t

for next in self.\_points:

next\_handled = False

print("Process " + str(next))

for cluster\_index in range(len(self.\_clusters)):

center = self.\_centers[cluster\_index]

distance = center.distanceTo(next)

print("Distance from " + str(center) + ": " + str(distance))

if distance <= t:

print("Added to cluster " + str(cluster\_index))

# Create new cluster

self.\_clusters[cluster\_index].append(next)

next\_handled = True;

if next\_handled == False:

# Create new cluster

print("New cluster " + str(next))

self.\_centers.append(next)

self.\_clusters.append(Points([next]))

s = Thresold([

(1,2),

(1,3),

(1,4),

(1,5),

(2,2),

(2,3),

(2,4),

(3,7),

(3,8),

(3,9)

])

s.dump\_points()

s.cluster(2)

s.dump\_cluster()

s.plot\_cluster('k1\_1\_14');

## Результат

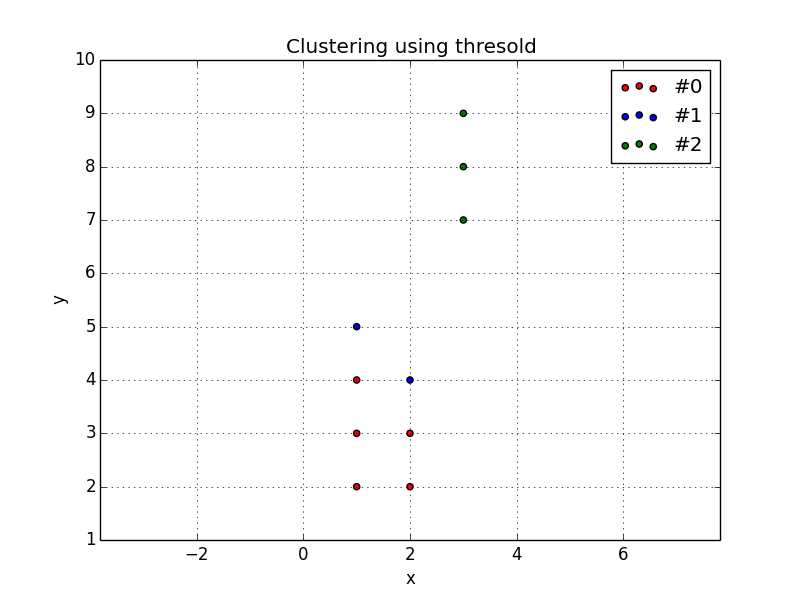
[(1; 2), (1; 3), (1; 4), (1; 5), (2; 2), (2; 3), (2; 4), (3; 7), (3; 8), (3; 9)]

Clusted with T=2 :

#1: (1; 2) [(1; 2), (1; 3), (1; 4), (2; 2), (2; 3)]

#2: (1; 5) [(1; 5), (2; 4)]

#3: (3; 7) [(3; 7), (3; 8), (3; 9)]



# Завдання №2

Здійснити розпізнавання образів із застосуванням максимінного алгоритму.

## Рішення

import sys, math, random

from point import \*

from pylab import \*

class MinMax(object):

def \_\_init\_\_(self, points):

self.\_points = [ Point(p[0], p[1]) for p in points ]

self.\_centers = []

def cluster(self):

self.\_centers = [self.\_points[0]]

max\_distances = []

while True:

other\_points = filter(lambda x: x not in self.\_centers, self.\_points)

def row\_min\_distance(other\_point):

min\_distance = 10000 # Should be max int

min\_point = None

for z\_point in self.\_centers:

item\_distance = other\_point.distanceTo(z\_point)

if item\_distance < min\_distance:

min\_distance = item\_distance

min\_point = other\_point

return (min\_point, min\_distance)

def col\_max\_distance(other\_points):

max\_distance = 0

max\_point = None

for other\_point in other\_points:

row\_distance = row\_min\_distance(other\_point)

item\_distance = row\_distance[1]

if item\_distance > max\_distance:

max\_distance = item\_distance

max\_point = row\_distance[0]

return (max\_point, max\_distance)

result = col\_max\_distance(other\_points)

max\_distance = result[1]

max\_point = result[0]

if len(max\_distances) == 0 or (max\_distance > min(max\_distances) / 2):

max\_distances.append(max\_distance)

self.\_centers.append(max\_point)

else:

return

s = MinMax([

(1,2),

(1,3),

(1,4),

(1,5),

(2,2),

(2,3),

(2,4),

(3,7),

(3,8),

(3,9)

])

s.dump\_points()

s.cluster()

s.dump\_cluster()

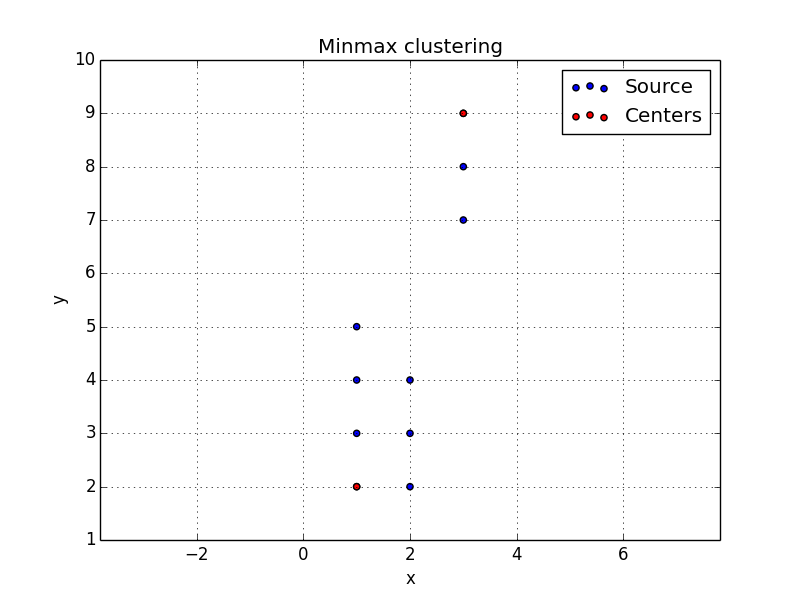
s.plot\_cluster('k1\_2\_14');

## Результат

[(1; 2), (1; 3), (1; 4), (1; 5), (2; 2), (2; 3), (2; 4), (3; 7), (3; 8), (3; 9)]

Cluster centers with 0.5 thresold:

[(1; 2), (3; 9)]



# Завдання №3

Здійснити розпізнавання образів із застосуванням алгоритму К-внутрішніх групових середніх. K= вибрати за кількістю кластерів отриманих 1м алгоритмом.

X1(1,2), X2(1,3), X3(1,4), X4(1,5), X5(2,2), X6(2,3), X7(2,4), X8(3,7), X9(3,8), X10(3,9)

## Рішення

import sys, math, random

from pylab import \*

from point import \*

INT\_MAX = 100000

class KMeans(object):

def \_\_init\_\_(self, points):

self.\_points = points

self.\_centers = []

self.\_clusters = []

self.\_k = 0

def cluster(self, K):

self.\_k = K

# Put first points as centers

self.\_centers = [self.\_points[i] for i in range(0, K)]

while True:

# Recluster against centers

self.\_clusters = [[] for i in range(0, K)]

for other\_point in self.\_points:

min\_z\_point\_index = 0

min\_distance = INT\_MAX

for z\_point\_index in range(len(self.\_centers)):

z\_point = self.\_centers[z\_point\_index]

item\_distance = other\_point.distanceTo(z\_point)

if item\_distance < min\_distance:

min\_distance = item\_distance

min\_z\_point\_index = z\_point\_index

self.\_clusters[min\_z\_point\_index].append(other\_point)

# New cluster centers

centers\_moved = False

for z\_point\_index in range(len(self.\_centers)):

prev\_center = self.\_centers[z\_point\_index]

cluster\_points = self.\_clusters[z\_point\_index]

new\_center = Point.find\_center(cluster\_points)

centers\_moved = centers\_moved or new\_center.distanceTo(prev\_center) > 0.01

self.\_centers[z\_point\_index] = new\_center

if not centers\_moved:

return

s = KMeans([

Point(1,2),

Point(1,3),

Point(1,4),

Point(1,5),

Point(2,2),

Point(2,3),

Point(2,4),

Point(3,7),

Point(3,8),

Point(3,9)

])

s.dump\_points()

s.cluster(3)

s.dump\_cluster()

s.plot\_cluster('k2\_1\_14');

## Результат

[(1; 2), (1; 3), (1; 4), (1; 5), (2; 2), (2; 3), (2; 4), (3; 7), (3; 8), (3; 9)]

Cluster with k-means. K=3:

#1: (1; 2) [(1; 2), (2; 2)]

#2: (1; 3) [(1; 3), (1; 4), (1; 5), (2; 3), (2; 4)]

#3: (3; 8) [(3; 7), (3; 8), (3; 9)]

