Zadanie č. 3: Aglomeratívne zhlukovanie

UMELÁ INTELIGENCIA

Matúš Makay

STU FIIIT

Cvičenie: *Pondelok 11:00*

Cvičiaci: *Ing. Martin Komák, PhD.*

Aglomeratívne zhlukovanie

Cieľom tohto zadania bolo naprogramovať systém ktorý bude z bodov ktoré náhodne vygenerujem na mape vytvoriť skupiny bodov takzvané clustre. Cluster je skupina bodov ktoré majú čo najmenšie rozdiely. V mojom zadaní rozdiel medzi bodmi reprezentuje ich euclidova vzdialenosť.

V programe vyberiem náhodný bod na mape a porovnám vzdialenosti so všetkými ostatnými bodmi na mape a spojím ho s bodom ktorý má čo najmenšiu euclidovu vzdialenosť s náhodne vygenerovaným bodom a takýmto spôsobom vznikne cluster.

Cluster reprezentujem buď medoidom alebo centroidom:

A. **Centroid** je bod so súradnicami **x** a **y** ktorý vznikne následovným spôsobom. **X-ová súradnica** vznikne súčtom všetkých x-ových suradníc bodov ktoré cluster obsahuje vydelených počtom bodov v clusteri.
B. **Medoid** je bod so súradnicami **x** a **y** ktorý vznikne následovným spôsobom. Vyberiem si bod zo zoznamu bodov ktoré cluster obsahuje. Vypočítam **euclidovu vzdialenost** od každého boda v clustri a **tieto vzdialenosti nasumujem**. Túto operáciu zopakujem pre každý bod a bod ktorý bude mať **najmenšiu sumu vzdialeností označím za medoid**

Reprezentácia údajov

Cluster

```
class Cluster:

    def __init__(self, x, y, repr, type):
        self.x = x
        self.y = y

        #   CENTROID, MEDOID
        self.type = type
        self.repr = repr
        self.color = None

        # tuples
        self.list_points = []
        self.list_points.append((x, y)
```

- Cluster reprezentujem súradnicami **x, y**.
- **Type** rozhoduje či sa reprezentatívne súradnice budú počítať ako medoid alebo ako centroid
- **Repr** je číslo clustra v poradí ako bol vložený do mapy
- **Color** je atribut ktorý slúži na rozlíšenie klustor v plote
- **List points** je list bodov ktoré obsahuje cluster

Popis dôležitých metód clustra

Spájanie clustrov

```
def add_cluster_points(self, points):
        """
        Method has to get cluster which is going to be removed from a map
        Append point in list of points and calculate new cord representation for cluster.
        Removes new point from map and set on map new representative coordinates
        """
        old_cordinates = self.get_cordinates()

        for point in points:
            self.list_points.append(point)

        new_x, new_y = self.calc_new_repr_cord()

        return old_cordinates[0], old_cordinates[1], new_x, new_y
```

- Túto metódu využívam pri spájaní dvoch clustrov
- Vstupný parameter sú body druhého clustra ktorý zanikne

Popis metódy v krokoch

1. Uložím si staré súradnice ktoré po vykonaní funkcie vrátim a vymažem ich z mapy
2. Body ktoré som poslal pridám do listu a následne vypočítam nové reprezentatívne súradnice ktoré vrátim a setnem daný cluster na nové suradnice na mapu

Počítanie nových reprezentativnych súradnic clustra po pridaní nových bodov

```
def calc_new_repr_cord(self,):
        """
        Calculate and returns new representative coordinates for MEDOID or CENTROID
        """
        if self.type == "CENTROID":
            self.x, self.y = self.find_centroid()
        else:
            self.x, self.y = self.find_medoid()

        return self.x, self.y
```

Počítanie súradníc centroidu

```
def find_centroid(self, ):
        sum_x = 0
        sum_y = 0
        count_points = len(self.list_points)

        for cordinates in self.list_points:
            sum_x += cordinates[0]
            sum_y += cordinates[1]

        centroid_x = round(sum_x / count_points)
        centroid_y = round(sum_y / count_points)

        return centroid_x, centroid_y
```

- Centroid počítam ako som popísal v úvode
- Nasumujem si všetky hodnoty súradníc bodov x a samostatne y ktoré predelím počtom bodov v clustri
- Nové súradnice nastavím ako reprezentatívne súradnice clustra

Počítanie medoidu

```
def find_medoid(self):
        """
        Vyuzivam pre najdenie najblizsie bodu pri spajani klustrov
        """
        x, y = self.find_centroid()
        centroid = (x, y)

        min_len = math.dist(centroid, self.list_points[0])
        min_cord = (self.list_points[0])

        for cordinates in self.list_points:
            if min_cord != cordinates:
                euc_distance = math.dist(centroid, cordinates)

                if euc_distance < min_len:
                    min_len = euc_distance
                    min_cord = cordinates

        return min_cord[0], min_cord[1]
```

- Medoid nachádzam takým spôsobom že nájdem centroid z bodov ktoré sa nachadzajú v clustri a ako medoid označím bod ktorý je najbližšie k centroidu.
- Takýto spôsob hľadania medoidu značne ovplyvnil rýchlosť vykonávania programu nakoľko nemusím pre každý bod v clustri počítať vzdialenosti od všetkých bodov
- Toto riešenie mi odporučil cvičiaci Kapustík

Popis Main funkcie

```
if len(sys.argv) != 3:
    exit("Send args: num_cluster M/C")

n = int(sys.argv[1])
t = sys.argv[2]

MAX_SIZE_MAP = 10000
NUM_POINTS = 5000
NUM_KLUSTER = n

if t == "M" or t == "m":
    CLUSTER_TYPE = "MEDOID"

else:
    CLUSTER_TYPE = "CENTROID"
list_clusters = []
board = create_board_fill_points(MAX_SIZE_MAP, NUM_POINTS, CLUSTER_TYPE, list_clusters)
```

- Na začiatku si nastavím konštanty ako
    1. Rozmer mapy
    2. Pocet bodov
    3. Pocet klustrov ktoré chcem mať
    4. Typ clustra
- Mapu reprezentujem 2D poľom. Na mape držím všetky clustre. Na začiatku je každý bod cluster

Main loop

```
while(len(list_clusters) != NUM_KLUSTER):
    random_idx = randint(0, len(list_clusters) - 1)
    pick_cordinates = list_clusters[random_idx]
    x, y = pick_cordinates[0], pick_cordinates[1]
    picked_cluster = board[x][y]
    min_cord=()

    i = 0
    min_cord = list_clusters[i]

    #ak sa min_cord rovna tomu co som vylosoval posun sa dalej v poli a zmen min_cord
    while min_cord == pick_cordinates:
        i+=1
        min_cord = list_clusters[i]

    min_dis = math.dist(pick_cordinates, min_cord)

    for cordinates in list_clusters:
        if cordinates != pick_cordinates and cordinates != min_cord:
            euc_dis = math.dist(pick_cordinates, cordinates)
            if euc_dis < min_dis:
                min_dis = euc_dis
                min_cord = cordinates

    x, y = min_cord[0], min_cord[1]
    closest_cluster = board[x][y]

    final_cluster = {}

    if len(picked_cluster.list_points) > len(closest_cluster.list_points):
         source_x, source_y, old_dest_x, old_dest_y, new_dest_x, new_dest_y =
join_clusters(closest_cluster, picked_cluster)
         final_cluster = picked_cluster
    else:
        source_x, source_y, old_dest_x, old_dest_y, new_dest_x, new_dest_y =
join_clusters(picked_cluster, closest_cluster)
        final_cluster = closest_cluster

    board[source_x][source_y] = 0
    board[old_dest_x][old_dest_y] = 0
    board[new_dest_x][new_dest_y] = final_cluster
```

```
    list_clusters.remove((source_x, source_y))

    if (source_x, source_y) != (old_dest_x, old_dest_y):
        list_clusters.remove((old_dest_x, old_dest_y))

    list_clusters.append((new_dest_x, new_dest_y))
```

- Popis jednej iterácie main loopu:
    1. Vyberiem si náhodný cluster zo zoznamu clustrov
    2. K vybratému clustru nájdem v zozname clustrov cluster ktorý je k nemu najbližšie
    3. Následne spojím clustre spôsobom že cluster s menším počtom bodov pridám ku clustru s vačším počtom bodov. Toto by malo zabezpečít vačšiu rýchlosť program nakoľko pridávam menší počet bodov
    4. **Z mapy a zoznamu clustrov odstránim** súradnice menšieho clustra a stare súradnice vačšieho clustra a následne do **mapy a zoznamu clustrov zapíšem** nové súradnice väčšieho clustra

Spájanie clustrov

```
def join_clusters(source, destination):
    """
    Join clusters, append points from first argument to second argument
    Return source.xy, destination_old_xy, destination_new_xy
    """
    old_dest_x, old_dest_y, new_dest_x, new_dest_y =
destination.add_cluster_points(source.list_points)
    source_cord = source.get_cordinates()

    return source_cord[0], source_cord[1], old_dest_x, old_dest_y, new_dest_x, new_dest_y
```

- Táto metóda pridá body zo source clustra do destination clustra
- Vráti súradnice source clustra, staré súradnice destination clustra a nové súradnice destination clustra
- Návratové hodnoty slúžia na updatnutie mapy v **main loope**

Testovanie

Návod na spustenie

Program spúštam zo vstupnými argumentami v poradí

1. Počet bodov ktoré chcem vygenerovat
2. Počet clustrov ktoré chceme na konci vypisu vidiet na mape
3. Typ clustra, buď medoid alebo centroid

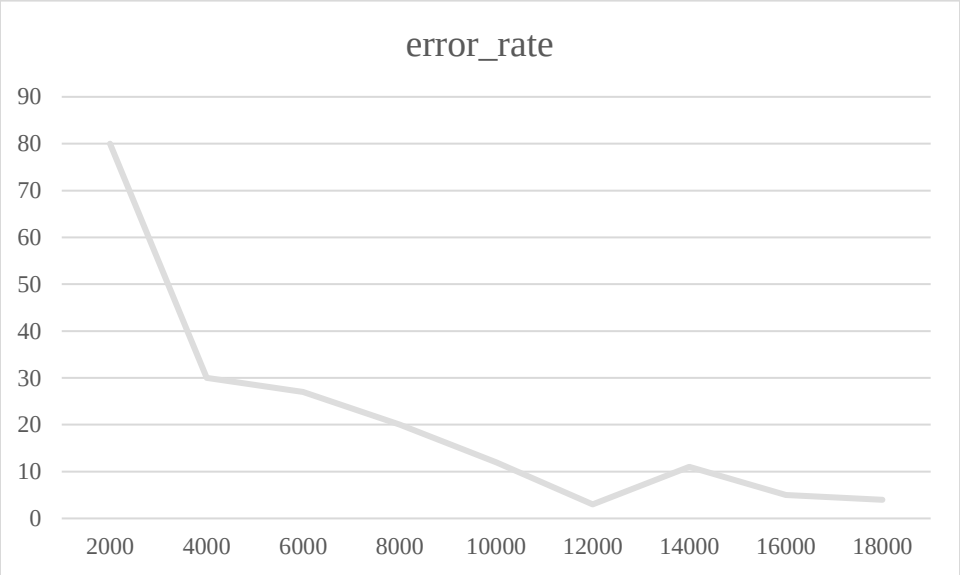
```
PS C:\Users\matus\ui_zad3> python .\main.py
Send args: max_points num_cluster M/C
```

- Ak program spustíte bez vstupných argumentov vypíše sa táto chybová hláška

Testovanie Centroidov

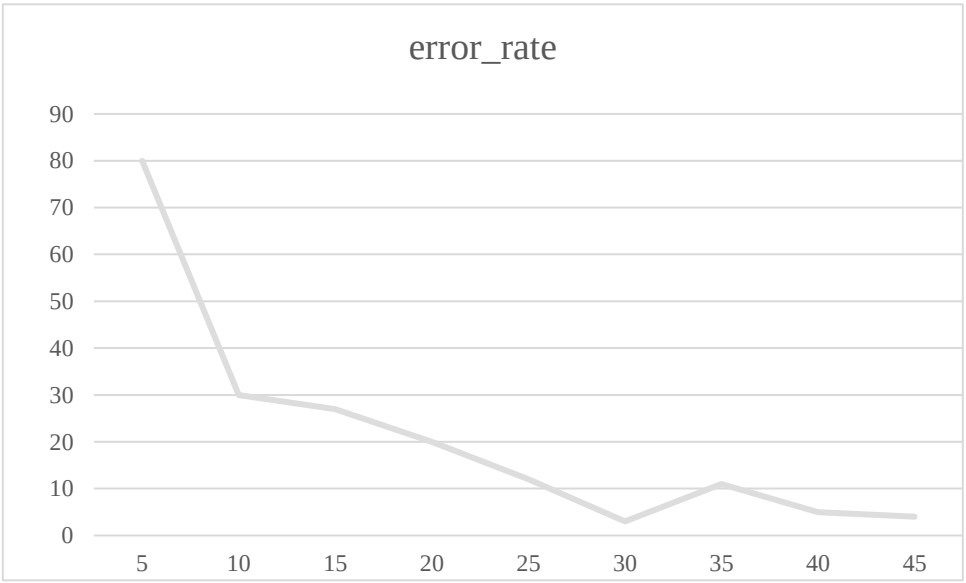
Závislosť chyby zhlukovača na počte bodov na mape

num_points	2000	4000	6000	8000	10000	12000	14000	16000	18000
error_rate	80	30	27	20	12	3	11	5	4

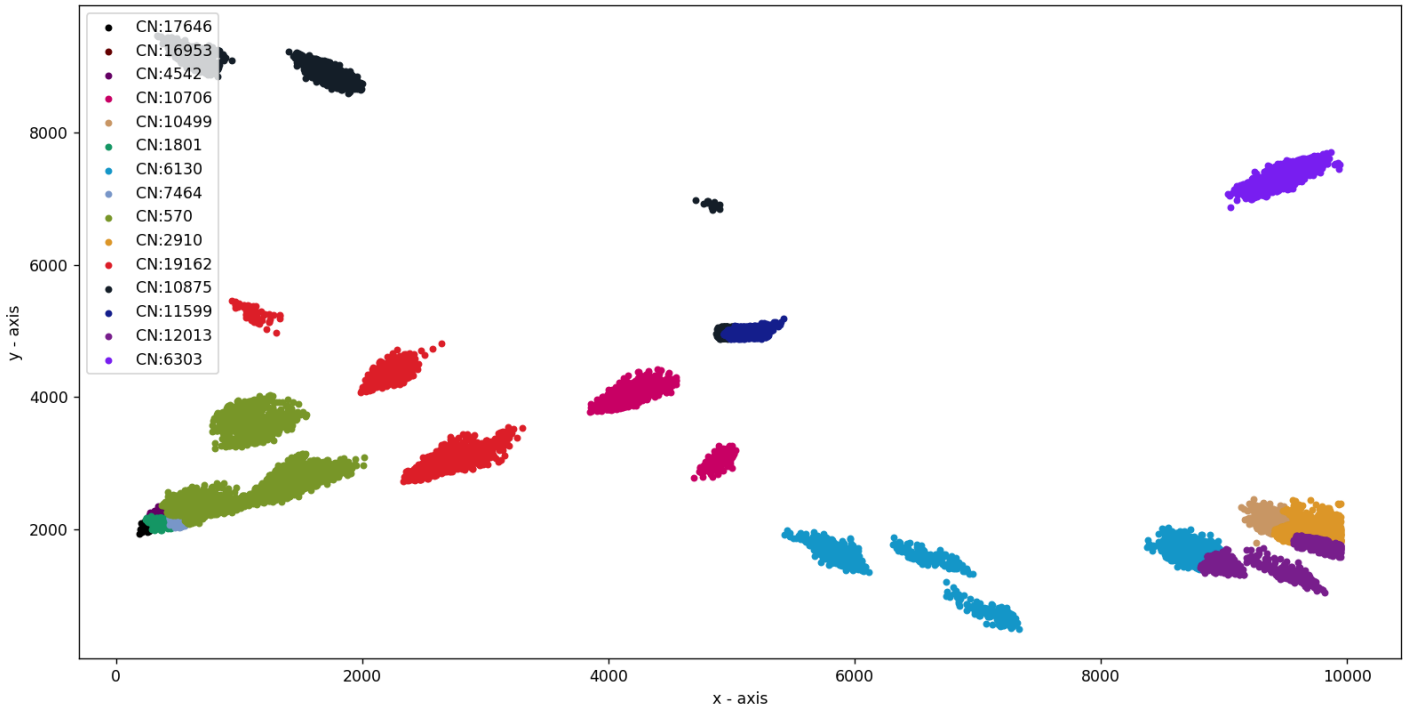


Závislosť chyby zhlukovača na počte konečných klustrov

num_cluster	5	10	15	20	25	30	35	40	45
error_rate	80	30	27	20	12	3	11	5	4



Testovanie pre 20000 bodov



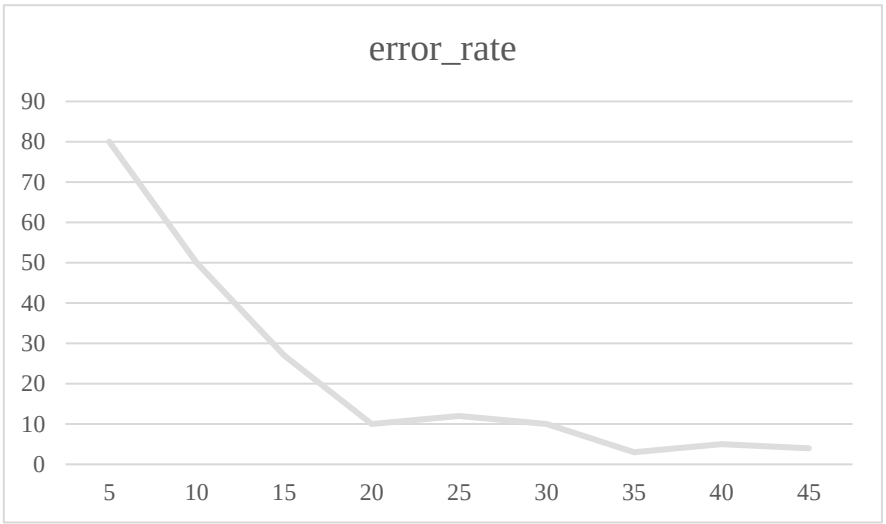
Testovanie Medoidov

Závislosť chyby zhlukovača na počte bodov na mape

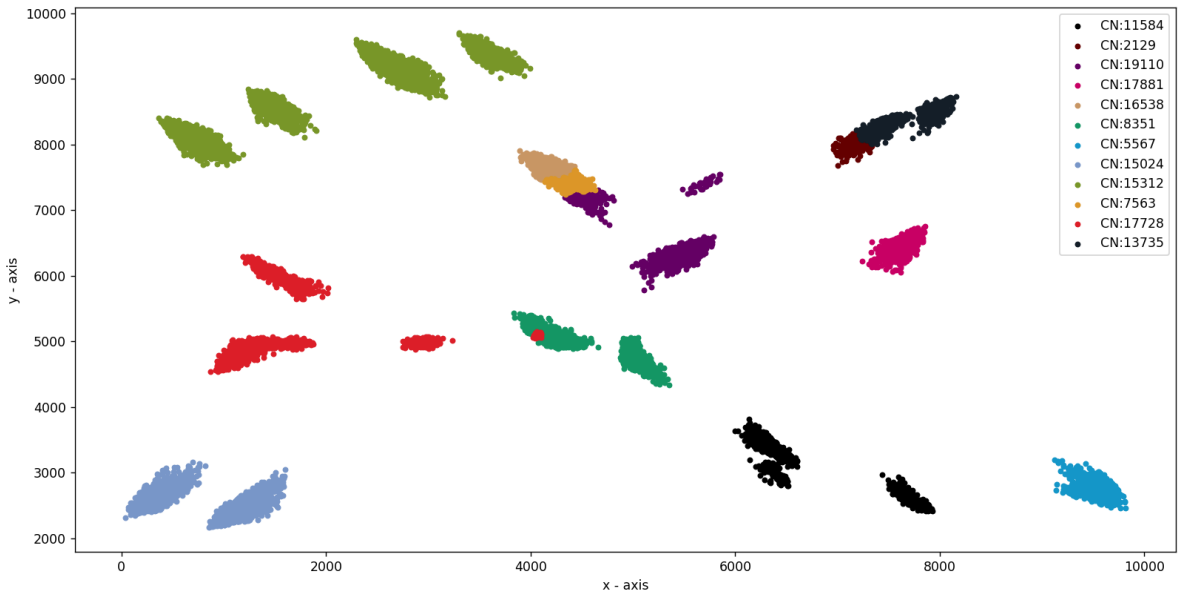
num_points	2000	4000	6000	8000	10000	12000	14000	16000	18000
error_rate	80	50	27	10	12	10	3	5	4

Závislosť chyby zhlukovača na počte konečných clustrov

num_cluster	5	10	15	20	25	30	35	40	45
error_rate	80	50	27	10	12	10	3	5	4



Testovanie pre 20000 bodov



Záver k testovaniu

Pri testovaní medoidov a centroidov som nebol schopný stanoviť jasný záver nakoľko ani jeden spôsob reprezentácie klustrov nevykazoval tendenciu byť vo všetkých prípadoch lepší. Takže to vyzerá že oba spôsoby sú skoro rovnako dobré a výskyt chyby klustra je pri obidvoch spôsoboch cca rovnaký. Oba algoritmy vykazujú veľkú chybovosť pri nízkom počte bodov a nízkom počte clustrov. Toto správanie sa so zvyšujúcim počtom bodov a clustrov znižuje až na zanedbatelnú mieru. Časovú zložitosť mal algortimus rýchlejšiu pri reprezentovaní clustra centroidom.