SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Zadanie 4 - Klastrovanie

Matúš Krajčovič

Kód predmetu: **UI_B**
Názov predmetu: **Umelá inteligencia**
Študijný program: Informatika
Študijný odbor: Informatika
Garant predmetu: : **Ing. Lukáš Kohútka, PhD.**
Vyučujúci: Ing. Lukáš Kohútka, PhD.
Cvičiaci: Ing. Ivan Kapustík
Úloha: **Zadanie 4, úloha B - Klastrovanie**
AIS ID: 103003
Dátum: 13. 12. 2020

Zadanie úlohy

Máme 2D priestor, ktorý má rozmery X a Y, v intervaloch od 5000 do +5000. Tento 2D priestor vyplňte 20 bodmi, pričom každý bod má náhodne zvolenú polohu pomocou súradníc X a Y. Každý bod má unikátne súradnice (t.j. nemalo by byť viacej bodov na presne tom istom mieste). Po vygenerovaní 20 náhodných bodov vygenerujte ďalších 40000 bodov, avšak tieto body nebudú generované úplne náhodne, ale nasledovným spôsobom:

1. Náhodne vyberte jeden z existujúcich bodov v 2D priestore
2. Vygenerujte náhodné číslo X_offset v intervale od -100 do +100
3. Vygenerujte náhodné číslo Y_offset v intervale od -100 do +100
4. Pridajte nový bod do 2D priestoru, ktorý bude mať súradnice ako náhodne vybraný bod v kroku 1, pričom tieto súradnice budú posunuté o X_offset a Y_offset

Vašou úlohou je naprogramovať zhlukovač pre 2D priestor, ktorý zanalyzuje 2D priestor so všetkými jeho bodmi a rozdelí tento priestor na k zhlukov (klastrov). Implementujte rôzne verzie zhlukovača, konkrétne týmito algoritmami:

- k-means, kde stred je centroid
- k-means, kde stred je medoid
- aglomeratívne zhlukovanie, kde stred je centroid
- divízne zhlukovanie, kde stred je centroid

Vyhodnocujte úspešnosť/chybovosť vášho zhlukovača. Za úspešný zhlukovač považujeme taký, v ktorom žiaden klaster nemá priemernú vzdialenosť bodov od stredu viac ako 500.

Vizualizácia: pre každý z týchto experimentov vykreslite výslednú 2D plochu tak, že označkujete (napr. vyfarbíte, očíslujete, zakrúžkujete) výsledné klastre. V závere zhodnoťte dosiahnuté výsledky ich porovnaním.

1 Algoritmy

1.1 K-means algoritmus (rozdeľovací)

V k-means algoritme sú vstupnými údajmi množina bodov a počet výsledných clusterov = parameter k. Na začiatku vygenerujeme k náhodných bodov (centroidov/medoidov) zo vstupnej množiny a všetky body množiny rozdelíme do skupín podľa toho, ku ktorému z týchto náhodných bodov sú najbližšie. Po zatriedení do skupín optimalizujeme centroidy alebo medoidy, čo je vysvetlené nižšie. Priraďovanie do skupín a optimalizácia sa cyklicky opakujú, až pokým sa už množina centroidov/medoidov nemení - konvergencia ku riešeniu.

1.1.1 Centroidy

Pri optimalizácii centroidov spočítame x a y súradnice všetkých bodov v clusteri a vydelíme ich počtom bodov v clusteri. Výsledné súradnice nám dávajú nový centroid. Toto sa v každom cykle zopakuje pri všetkých clusteroch.

1.1.2 Medoidy

Pri medoidoch vždy hľadáme bod v clusteri, od ktorého je súčet vzdialeností ku ostatným bodom čo najmenší. Pre každý bod teda robíme súčet vzdialeností, a hladáme ten s najmenším, ktorý sa stane novým medoidom daného clustera.

1.2 Hierarchické algoritmy

1.2.1 Algomeratívny algoritmus

Pri aglomeratívnom algoritme začímane s N clustermi, pričom N je počet bodov celej množiny. Postupne v každom cykle spojíme dva clustery, ktoré sú k sebe najbližšie (ich centroidy, ktoré po každom spojení clusterov prepočítavame). Takto pokračujeme, až pokým nedostaneme výsledný počet clusterov.

1.2.2 Divizívny algoritmus

Pri divíznom algoritme začíname s jedným clusterom so všetkými bodmi množiny. Postupne v cykle rozdeľujeme "najhorší cluster" na dva menšie (napr. pomocou k-means algoritmu s centroidmi). Pri každej iterácii teda nájdeme cluster, ktorého súčet vzdialeností bodov od centroidu je najväčší, a ten rozdelíme. Rozdeľujeme až pokým nedostaneme žiadaný počet clusterov.

2 Zdrojový kód

Programoval som v jazyku C++, kód je rozdelený do dvoch priečinkov - header súbory (include) a samotné zdrojové súbory (files).

Využívam klasické knižnice <iostream> a <fstream> na prácu so vstupom a výstupom, <vector>, <string> a tiež <cmath>. Neskôr som pridal okrem knižnice <random> aj <stdlib.h> a <time.h>.

2.1 point, plane

Trieda **point** je jednoduchá trieda obsahujúca dve súradnice pre x a y. Je poskytnutých niekoľko konštruktorov, pomocou ktorých môžeme vytvoriť bod s dvomi súradnicami, náhodný bod v nejakom rozmedzí alebo náhodný bod s offsetom od iného, zadaného bodu. Obsahuje základné get a set funkcie.

Trieda **plane** obsahuje vektor bodov, pričom môže byť vytvorená s parametrami (počet iniciálnych bodov, pošet bodov s offsetom, max. a min. hodnota súradníc, offset). Rovnako môže byť vytvorená s už existujúcim vektorom bodov. Ponúka jednoduché funkcie na zistenie počtu bodov a ich získanie.

2.2 Abstraktné triedy

Trieda **clustering** je abstraktná trieda s virtuálnymi funkciami a funkciou distance používanou všetkými implementáciami.

Ďalšia abstraktná trieda **partitional** pre rozdeľujúce algoritmy, ktorá obsahuje funkcie **assign_groups()** pre zadelenie bodov do clusterov, **init_random()** sa náhodnú inicializáciu začiatočných centrálnych bodov, **converged()** pre zistenie, či algoritmus skonvergoval, **contains()** sa používa pri začiatočnej inicializácii, aby sme nedostali dva rovnaké centrálne body, a napokon **print()** na vypísanie výsledku do súboru. Okrem toho obsahuje premenné používané rozdeľovacími algoritmami, napr. pre uchovávanie clusterov alebo centrálnych bodov.

2.3 Triedy s implementáciami

Triedy **k_means_centroids** a **k_means_medoids** dedia od triedy partitional, pričom definujú hlavne metódu **launch()**, ktorá obsahuje kód samotného algoritmu. Okrem toho obsahuje trieda k_means_clustering niekoľko funkcii na získanie clusterov a centroidov využívaných v triede divisive.

Triedy **aglomeragive** a **divisive** dedia z hlavnej triedy clustering, a rovnako implementujú hlavne metódu **launch()** a **print()**.

Všetky tieto triedy implementujú aj funkciu **test()**, ktorou meriame priemernú vzdialenosť bodov v clusteroch od ich centier.

2.4 Vizualizácia

Vizualizáciu robím v jazyku Python, pričom využívam knižnicu matplotlib. Samotný kód je v subore visualizer.py. Jednoducho načítam zdrojový súbor, z ktorého dostanem počet clusterov a jednotlivé body v nich, ktoré sa vykreslia po spustení skriptu.

Skript načíta počet clusterov, ich centroidy a samotné body, ktoré postupne vykresľuje na plochu.

Pre vizualizáciu je potrebný výstupný súbor output.txt, ktorý dostaneme pri spustení jedného z typov clusterovania v našom hlavnom programe. Po každom spustení clusterovania môj program tento skript zavolá a spustí sa. Preto musí byť v rovnakom priečinku ako samotný program.

2.5 Implementácia algoritmov

Vo funkciách launch() všetkých tried mám naimplementované algoritmy nasledovne:

Všetky triedy majú premennú **m_plane** so všetkými bodmi (vektor tried point) a počet výsledných clusterov v premennej **m_cluster_count**.

Obe triedy pre k-means algoritmy majú premenné **m_centers** pre controidy/medoidy (vektor tried point) a **m_groups** pre samotné clustre (vektor vektorov s indexami z m_plane). Rovnako algoritmy môžem spúšťať viac krát, pričom pri každom spustení počítam celkovú variáciu. Riešenie s tou najmenšou zapíšem do premenných **m_best_centers** a **m_best_groups**. V oboch algoritmoch vždy na začiatku nainicializujem náhodných k bodov, ktoré zvolím ako centroidy/medoidy. V každej z iterácií pridelím indexy bodov do skupín a vypočítam nové body - pri centroidoch priemery súradníc v skupinách, pri medoidoch je to bod, ktorý má od ostatných najmenší súčet vzdialeností. Nakoniec zistím, či sa predchádzajúce centrálne body rovnajú s aktuálnymi, ak áno, ukončím.

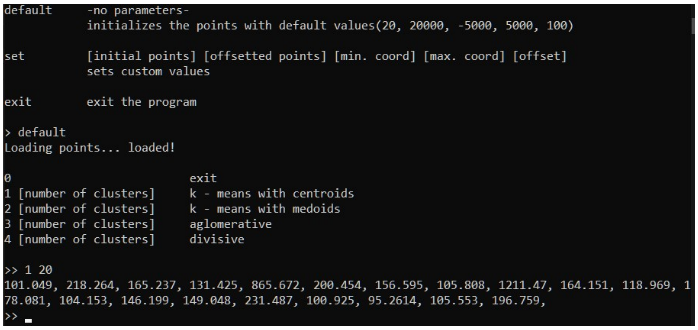
Pri aglomeratívnom algoritme mám k dispozícii **m_clusters** (vektor párov s centroidom a vektorom indexov), pričom na začiatku algoritmu ho naplním plným počtom clusterov s jedným indexom a centroidom, ktorý je zároveň bod na danom indexe. Vytvorím si aj maticu vzdialeností medzi jednotlivými clustermi. V každej z iterácii prehľadám maticu, nájdem dva clustre, ktoré sú k sebe najbližšie a spojím ich do prvého, pričom druhý vymažem (z m_clusters aj z matice). Prepočítam nové vzdialenosti iba tam, kde nastala zmena. Opakujem, pokým nie je výsledný počet clusterov.

Pri divizívnom clusterovaní pri každej iterácii nájdem skupinu, v ktorej je súčet vzdialeností od centroidy najväčší, a ten algoritmom k-means rozdelím na dve časti. Z premennej **m_planes** (vektor párov s triedou point a plane) vymažem pôvodný cluster a pridám nové dva. Opakujem, pokým nie je výsledný počet clusterov.

3 Používateľské prostredie

Prostredie je veľmi jednoduché, na začiatku programu sa spustí návod.

Môžeme inicializovať 20020 bodov pre 20 clusterov s preddefinovanými hranicami príkazom **default**, alebo môžeme nastaviť vlastné príkazom **set**. Po načítaní bodov môžeme spúšťať jednotlivé algoritmy, číslami 1 až 4, pričom vždy zadefinujeme aj počet clusterov, ktoré chceme dosiahnuť.



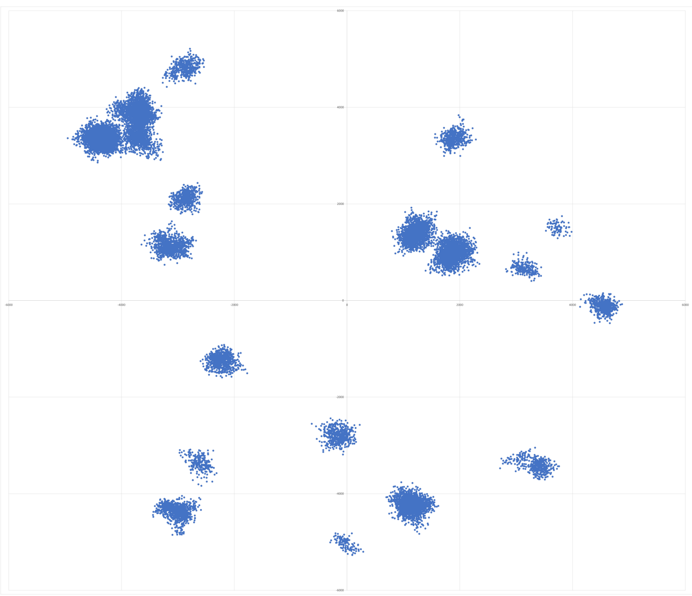
```
default     -no parameters-
            initializes the points with default values(20, 20000, -5000, 5000, 100)

set         [initial points] [offsetted points] [min. coord] [max. coord] [offset]
            sets custom values

exit        exit the program

> default
Loading points... loaded!

0                          exit
1 [number of clusters]     k - means with centroids
2 [number of clusters]     k - means with medoids
3 [number of clusters]     aglomerative
4 [number of clusters]     divisive

>> 1 20
101.049, 218.264, 165.237, 131.425, 865.672, 200.454, 156.595, 105.808, 1211.47, 164.151, 118.969, 1
78.081, 104.153, 146.199, 149.048, 231.487, 100.925, 95.2614, 105.553, 196.759,
>> 
```

Po spustení niektorého z algoritmov sa údaje zapíšu do súboru output.txt, a automaticky sa spustí vizualizácia a do konzoly sa vypíšu priemerné vzdialenosti ku centru pre každý jeden cluster.

4 Testovanie

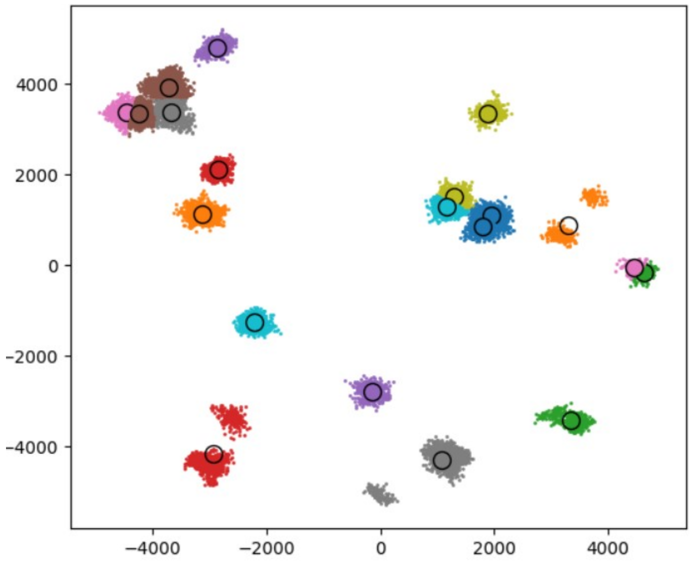
Výsledky testovania sú zhrnuté v zhodnotení.

Pri testovaní som využíval niekoľko náhodne vygenerovaných rozložení clusterov, pre algoritmy som skúšal rôzne počty cieľových clusterov a zaznamenával hodnoty. Ako prvé som porovnával jednotlivé algoritmy na 20020 bodoch:



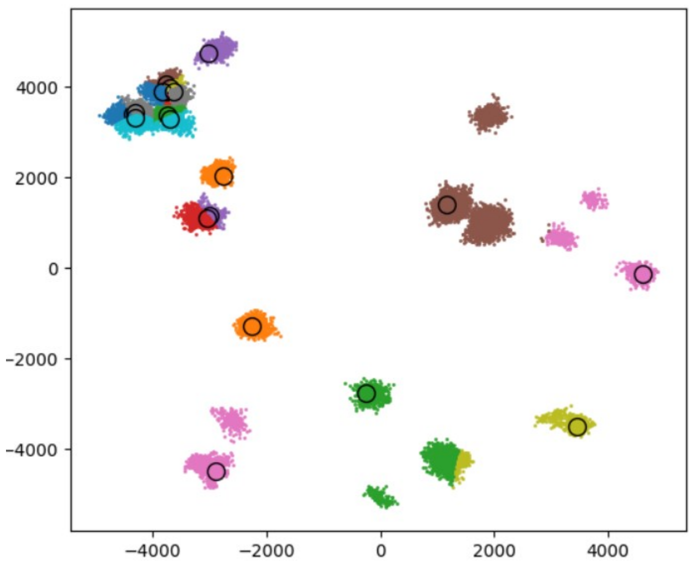
4.1 K-means algoritmus s centroidmi

Čas: približne 1 sec.
Zaokrúhlené priemerné vzdialenosti od stredu:
152, 179, 111, 399, 172, 185, 140, 261, 156, 156, 146, 428, 187, 149, 169, 134, 113, 188, 132, 129



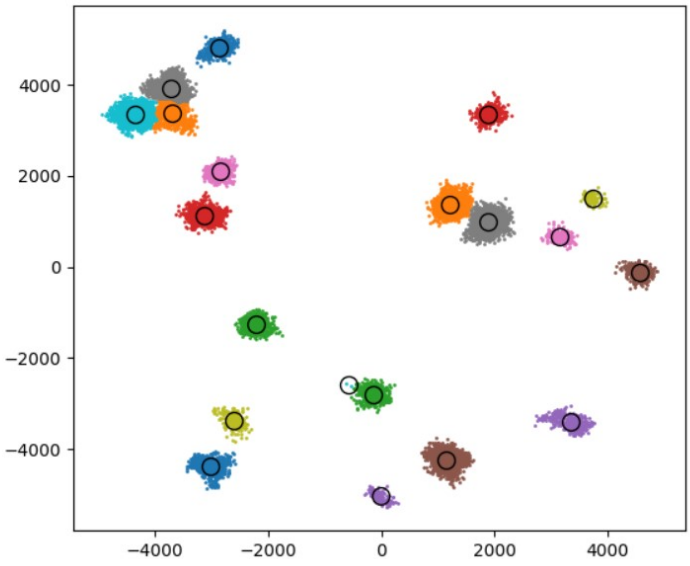
4.2 K-means algoritmus s medoidmi

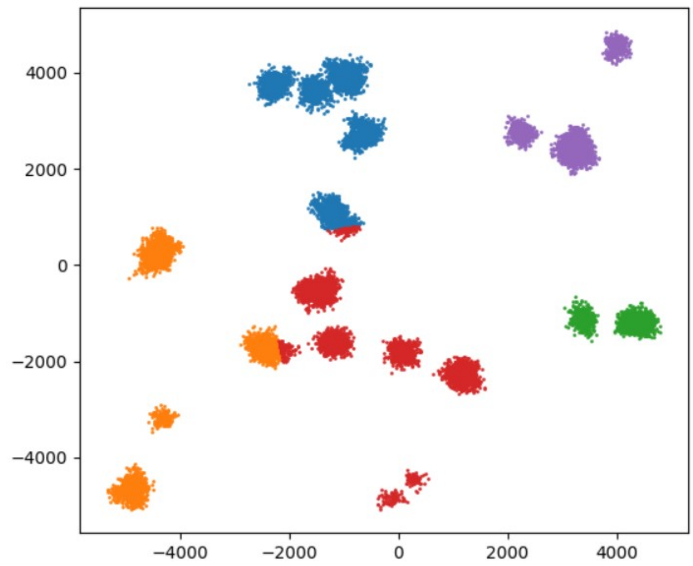
Čas: približne 5 sec.
Zaokrúhlené priemerné vzdialenosti od stredu:
150, 167, 161, 141, 392, 119, 465, 151, 180, 174, 424, 134, 1636, 194, 156, 111, 155, 147, 227, 696



4.3 Aglomeratívny algoritmus

Čas: približne 40 min.
Zaokrúhlené priemerné vzdialenosti od stredu:
180, 170, 156, 156, 187, 142, 142, 191, 123, 169, 172, 197, 168, 179, 139, 181, 149, 185, 165, 46

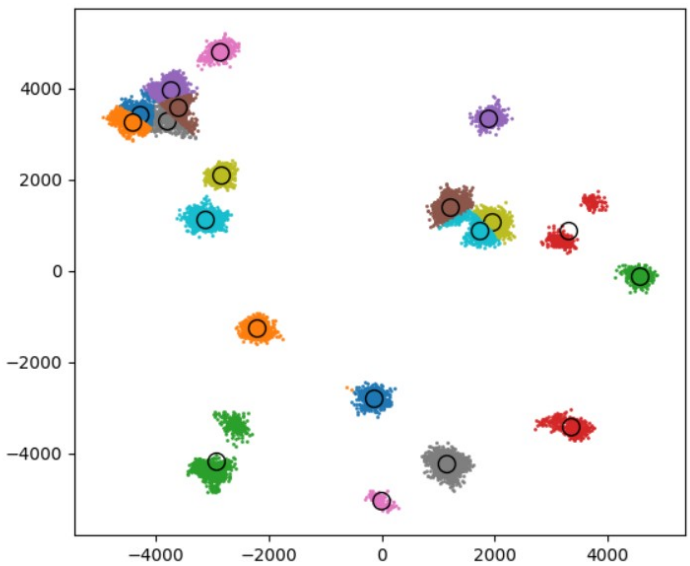


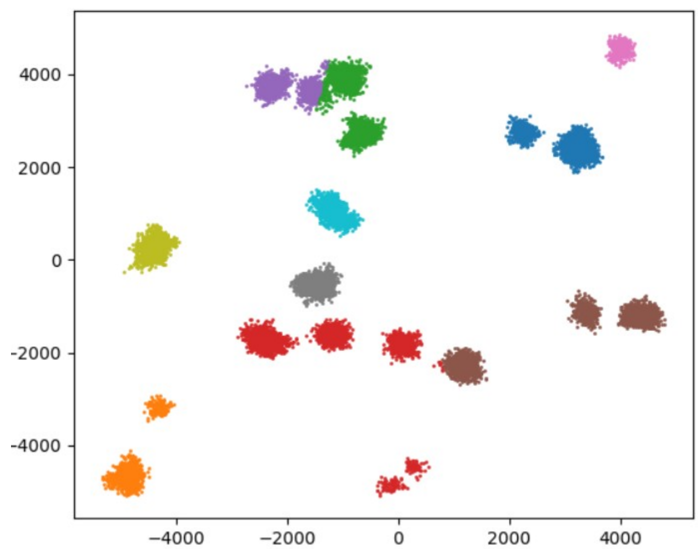


4.4 Divizívny algoritmus

Čas: približne 1 sec.
Zaokrúhlené priemerné vzdialenosti od stredu:
168, 162, 399, 187, 156, 165, 172, 186, 149, 179, 131, 145, 142, 428, 172, 182, 139, 181, 154, 182

k=10:
413, 398, 594, 1037, 375, 1519, 144, 181, 193, 235



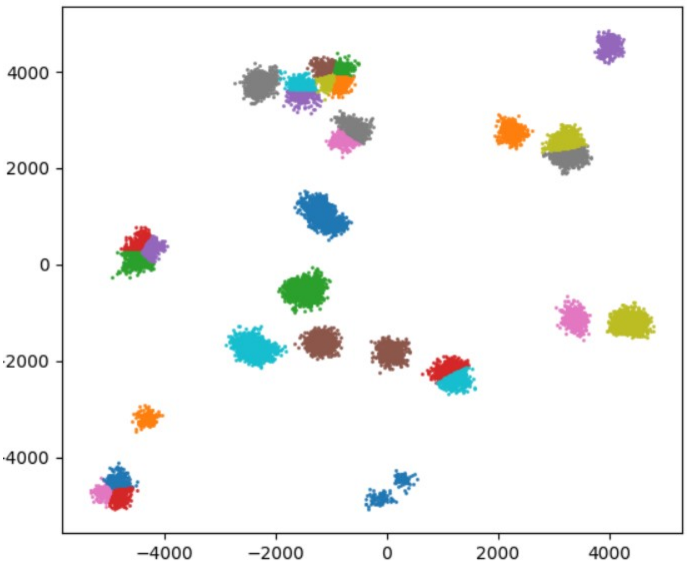


4.5 Ostatné testy

Ďalej som testoval rôzne hodnoty k:

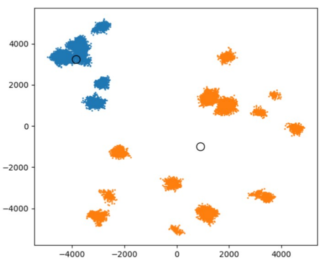
Pre K-means s centroidmi som algoritmus otestoval pre rôzne k:
k=5:
1105, 2138, 383, 1355, 678

k=30:
235, 129, 112, 136, 132, 161, 109, 144, 163, 126, 307, 100, 181, 127, 144, 148, 131, 151, 106, 175, 118, 115, 138, 110, 120, 121, 145, 169, 142, 125

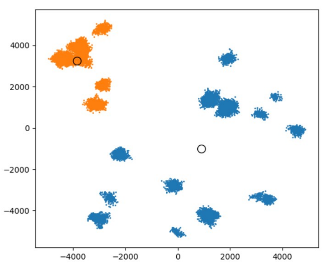


Výsledky pre ostatné metódy boli porovnateľné.

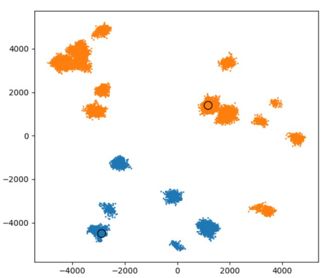
Testy pre počet clusterov 2 (stále s inicializáciou 20 clusterov, netestovali sme aglomeratívne):



Obr. 1: centroidy

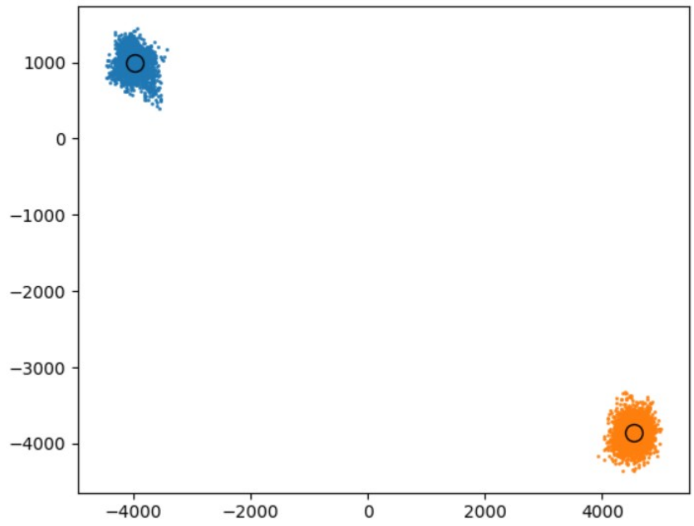


Obr. 3: divizivne

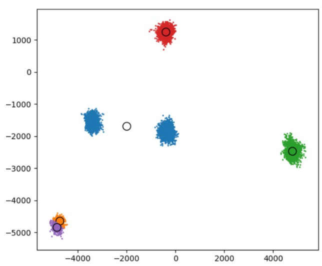


Obr. 2: medoidy

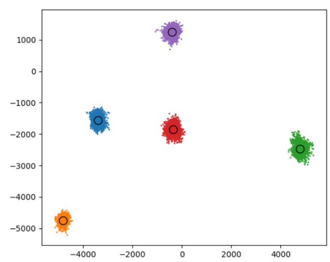
Testy pre iniciálny počet 2 a počet výsledných clusterov tiež 2 dopadli porovnateľne rovnako pre všetky implementácie:



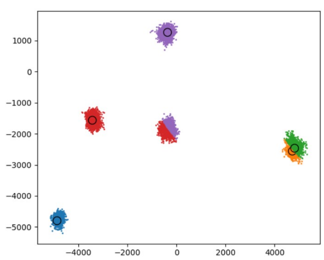
Testy pre iniciálny počet 5 a výsledný tiež 5, pre 10000 bodov:



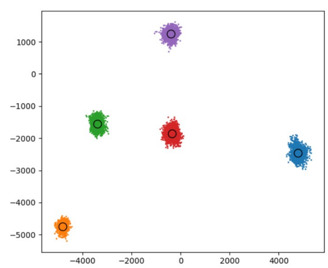
Obr. 4: centroidy



Obr. 6: aglom.

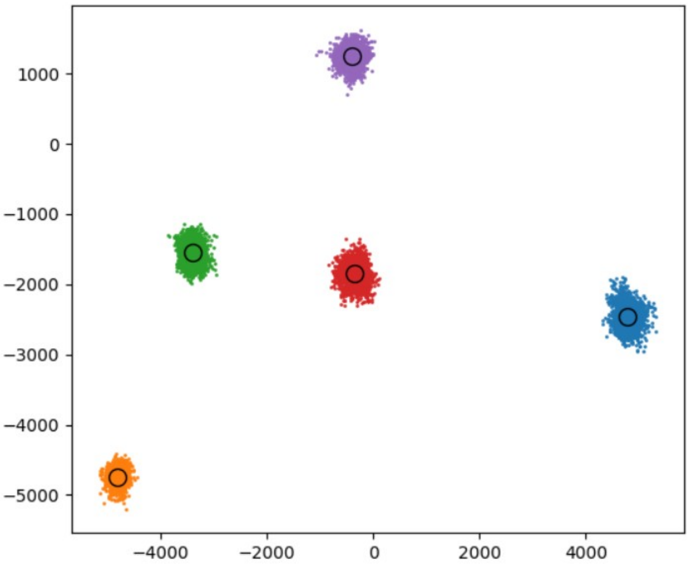


Obr. 5: medoidy



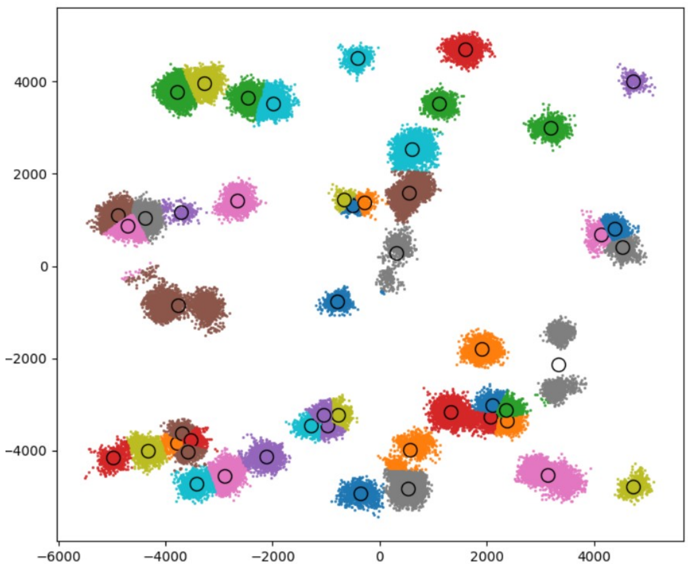
Obr. 7: div.

Keď k-means algoritmus s centroidmi spustím viac krát, a vyberiem najlepšie riešenie:

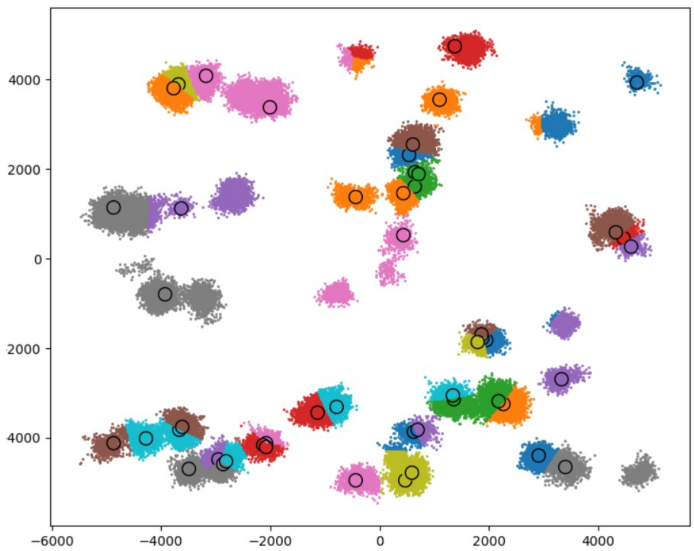


Vyskúšal som aj 100 000 bodov a 50 clusterov, okrem aglomeratívneho clusterovania. K-means s centroidmi a divizívne to zvládlo za pár sekúnd, medoidy asi pol minúty:

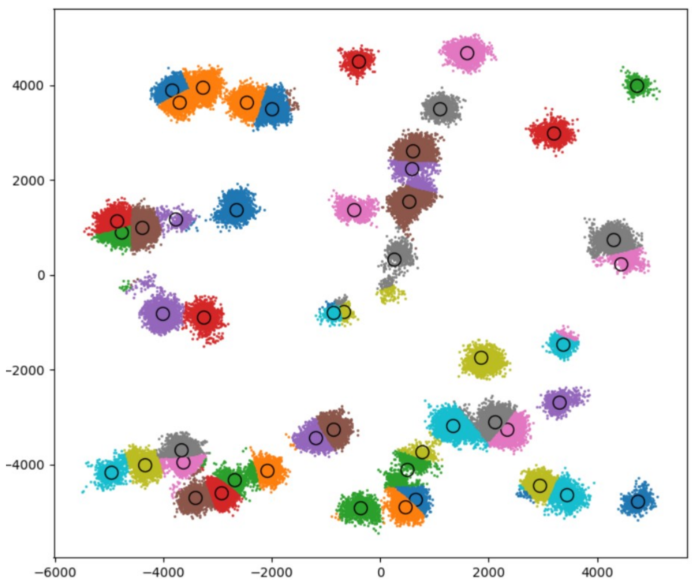
K-means s centroidmi:
98, 111, 191, 179, 126, 111, 160, 322, 193, 134, 142, 167, 188, 106, 174, 165, 186, 166, 166, 167, 189, 124, 154, 134, 169, 383, 181, 167, 131, 201, 124, 125, 185, 172, 119, 118, 149, 184, 179, 192, 144, 274, 121, 172, 138, 230, 295, 621, 112, 150



K-means s medoidmi:
163, 518, 188, 176, 880, 216, 136, 171, 42, 150, 248, 175, 174, 142, 128, 107, 202, 378, 138, 170, 173, 176, 127, 456, 166, 155, 1141, 510, 177, 168, 1344, 192, 129, 151, 143, 238, 384, 151, 175, 174, 140, 150, 138, 129, 614, 177, 269, 308, 121, 150



Divizívne:
213, 160, 185, 150, 166, 210, 172, 152, 99, 102, 166, 193, 283, 174, 192, 179, 161, 160, 246, 147, 198, 188, 144, 159, 225, 196, 173, 330, 161, 181, 169, 148, 138, 185, 165, 161, 339, 198, 151, 190, 147, 140, 196, 146, 199, 180, 153, 146, 176, 172



5 Zhodnotenie

5.1 Porovnanie algoritmov - čas a priestor

K-means algoritmus s centroidmi je najrýchlejší, za čo môže jeho nízka časová zložitosť O(n), keďže v každom cykle iba priraďujeme všetky body ku centroidom a prepočítame ich. Aj pri 40000 bodoch bol čas menej ako sekundu. K-means s medoidmi je už časovo náročnejší, keďže v každej z iterácií musíme raz priradiť body do clusterov, potom prejsť všetkými bodmi v každom z nich a nájsť tenm s najmenším súčtom vzdialeností. Aj pri 40000 bodoch mi algoritmus zbehne do jednotiek sekúnd. Aglomeratívne clusterovanie bola najväčšia výzva. Optimalizoval som ho ja pomocou matice vzdialeností, čo mi trochu pomohlo, no aj pri 20000 bodoch mi výpočet trval asi hodinu. Je to dané jeho veľkou zložisťou $O(n^3)$, keďže musíme prejsť celou maticou, a to je ešte vnorené v jednom cykle. Divizívny algoritmus mi tiež zbehol rýchlo, keďže iba 20 krát spúšťam k-means pre k = 2, čo nie je veľmi náročné. Vykonáva sa o trochu pomalšie ako samotný k-means s centroidmi.

Priestorovo sú algoritmy zanedbateľne zložité, okrem aglomeratívneho algoritmu, ktorého matica zaberá pri 20000 bodoch okolo 1.5 GB pamäte RAM pri využití typu float.

5.2 Vylepšenia a optimalizácie

Pri k-means algoritmoch veľmi záleží na začiatočnej náhodnej inicializácii, pričom pri každom spustení sú výsledky dosť odlišné, preto som testoval aj možnosť viacnásobného spustenia algoritmu, pričom uchovávam iba to s najmenšími súčtami vzialeností od centrálnych bodov. Testoval som to najmä na centroidoch, kvôli krátkemu času, a pri menčích počtoch clusterov sú výsledky porovnateľné s aglomeratívnym. Pri väčšom sú výsledky určite lepšie, no na aglomeratívne clusterovanie to nemá.

Ako som už spomínal, pri aglomeratívnom clusterovaní využívam maticu vzdialeností, pri ktorej teda znižuje časovú zložitosť niekoľkonásobne, no zvyšujem zas tú priestorovú.

5.3 Porovnanie počtu clusterov

Pri všetkých algoritmoch pri príliš malom počte clusterov (za predpokladu inicializácie 20 clusterov) sú priemerné vzdialenosti veľmi veľké, keďže v každom clusteri zahŕňame rôznorodé body z rôznych skupín. Pri veľmi veľkom počte sú zas vzdialenosti malé, výrazne menšie ako 500, no začína sa strácať význam clusterovania, keďže každá skupinka je rozdelené na niekoľko menších.

Pri zvolení malého počtu clusterov sú implementácie podobné, pri zvolení veľkého je aglomeratívne stále najlepšie, lebo aj pri veľkom počte malých clusterov ich rozdeľuje rovnomerne. Nie je závislé od náhodnej začiatočnej inicalizácie.

Najviac sa mi osvedčilo zvoliť číslo podobné tomu, ktoré sme použili pri vytváraní priestoru bodov.

5.4 Úspešnosť

Pri vstupných hodnotách zo zadania bol najúspešnejší algoritmus aglomeratívny, pri ktorom boli priemerné vzdialenosti od stredu hlboko pod 500. Pri divízívnom to dopadlo podobne dobre, niekedy jeden cluster vyskočil nad 500.

Horšie sú na tom k-means algoritmy, hlavne kvôli začiatočnej náhodnej inicializácii. Často sú jeden, dva clustery so vzdialenosťou väčšou ako 500, niekedy dokonca viac. Riešenia boli lepšie pri viacnásobnom spustení algoritmu a hľadaní najlepšieho riešenia, často som našiel riešenia so všetkými clustermi pod 500.