FIIT STU

Ilkovičova 2, 842 16 Karlova Ves

Zadanie 4

Klastrovanie

Autor: Martin Pirkovský

2020/2021

Obsah

1) Zadanie

Mojou úlohou v tomto zadaní bolo vygenerovať náhodne 20 bodov, z ktorých som mal následne vybrať 1 a v jeho okolí s určitým offsetom vygenerovať nový bod. Takýchto bodov som mal vygenerovať určitý počet, ktorý je premenlivý počas testovania. Tieto body nám po vygenerovaní vytvoria zhluky, ktoré som mal ohodnotiť za pomoci zhlukovača, ktorý používa jeden zo štyroch algoritmov na ohodnotenie plochy. Použité algoritmy sú k-means, kde stred je centroid, k-means, kde stred je medoid, divízne zhlukovanie, kde stred je centroid a aglomeratívne zhlukovanie, kde stred je centroid. Následne máme vyhodnotiť zhlukovač ako úspešný/neúspešný ak je jeho priemerná vzdialenosť bodov v rámci klastra od jeho stredu menšia ako 500. Ďalšiu vec, ktorú máme implementovať je vizualizácia zhlukovača, kde výsledné klastre budú označené napríklad rovnakou farbou.

2) Opis použitých algoritmov

K-means

Pri k-means ako prvé vyberiem 20 jedinečných bodov, ktoré určím ako stredy zhlukov. Následne priradím každý bod najbližšiemu klastru, čim si vytvorím prvé ohodnotenie bodov. Po ohodnotení bodov vyrátam nový stred klastra. Ak má byť stred centroid, tak ho vyrátam ako priemernú vzdialenosť všetkých x-ových súradníc, čím dostanem x-ovú súradnicu centroidu a rovnako vyrátam priemer všetkých y-ových súradníc, a tak získam y-ovú súradnicu centroidu. Centroid je iba fiktívny stred, tento bod reálne vykreslený nie je. Naopak ak má byť stred medoid, tak toto je už reálny bod, ktorý môžeme povedať, že to je najbližší bod k všetkým bodom, teda bod s najmenším súčtom euklidovských vzdialeností k všetkým ostatným bodom. Keď už mám aktualizovaný stred klastra, tak znova priradím všetky body k danému klastru. Tento cyklus priraďovania bodov a vyrátavania nového stredu vykonávam až pokiaľ sa pri prechode jednou iteráciou nič nezmení, čo znamená, že už máme finálne ohodnotenie v danom experimente.

Divízne zhlukovanie

V divíznom zhlukovaní využívam k-means algoritmus so stredom ako centroid. Na začiatku mám 1 klaster, ktorý rozdelím na 2 klastre. Následne ak ešte nemám požadovaný počet klastrov, tak vyberiem klaster, ktorý má najväčší priemer vzdialeností v rámci klastra a ten znova rozdelím na 2 klastre. Tento cyklus sa opakuje až pokiaľ nedosiahnem zadaný počet klastrov.

Aglomeratívne zhlukovanie

Aglomeratívne zhlukovanie funguje naopak ako divízne, keďže teraz máme na začiatku n-bodov, pričom každý bod tvorí samostatný klaster. Na začiatku si teda vytvorím n-klastrov, a následne si vytvorím maticu vzdialeností medzi dvoma klastrami. Následne nájdem klastre,

ktoré sú k sebe najbližšie a tie spojím do jedného. Pôvodné klastre odstránim ako z matice vzdialeností, tak aj z poľa, kde mám uložené všetky klastre. Nový klaster, ktorý vznikol spojením, pridám na koniec matice a na koniec poľa klastrov. Tento cyklus spájania vykonávam pokiaľ nemám počet klastrov rovný požadovanému počtu klastrov.

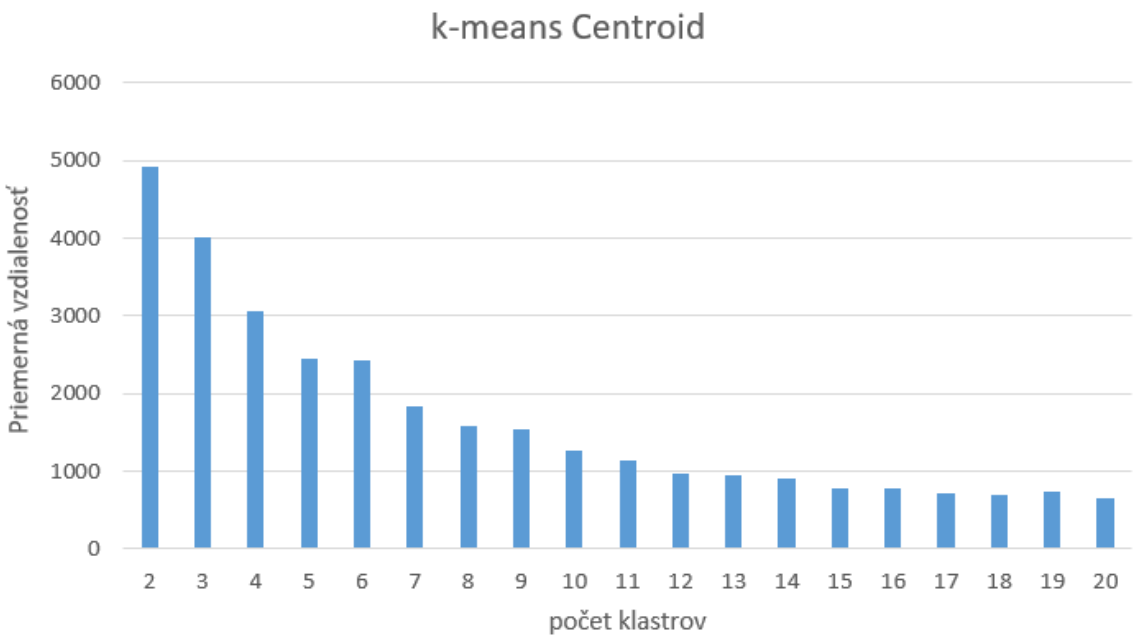
3) Vylepšenia oproti zadaniu

V zadaní máme generovať body na základe akéhosi náhodného offsetu v intervale od -100 po 100. Takéto generovanie však vytváralo štvorce, ktoré sa na prvý pohľad veľmi nepodobali na klasický zhluk bodov, ktorý je v strede hustejší, po krajoch redší, a vytvára akýsi kruh. Preto som sa rozhodol namiesto toho využiť na generovanie nových bodov funkciu gauss(bod, odchýlka) z knižnice random, ktorej ako argumenty dám súradnicu bodu a akceptovateľnú odchýlku. Keďže v zadaní je ako akceptovateľné ak je priemer v rámci klastra menší ako 500, čo je 5 násobok offsetu, tak nakoľko si používateľ môže zvoliť odchýlku akú chce, tak som sa ako úspešný zhlukovač rozhodol označiť taký, ktorého vzdialenosť v rámci klastra je menšia ako 5 násobok odchýlky. Ďalej máme v zadaní na konci ohodnotenia vizualizovať konečný stav zhlukovača. Ja som sa rozhodol, že vykreslím celý priebeh zhlukovania, a teda vytváram z neho ako keby takú animáciu ako sa počas iterácií klastre menia a prispôsobujú novým stredom.

4) Testovanie

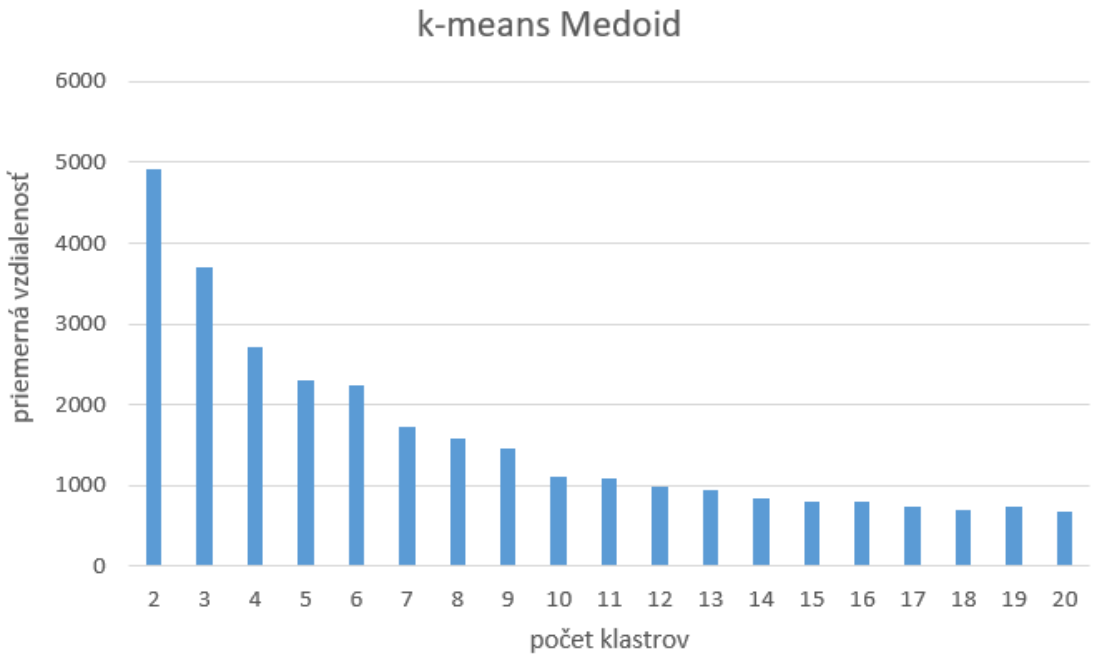
Počas testovania som sa rozhodol porovnávať priemerné vzdialeností v rámci klastra a zároveň spriemerovať tieto priemery do jedného čísla, ktoré považujem ako finálne ohodnotenie zhlukovača v danom experimente. Ako testovací pomer som si zvolil interval generovaných hodnôt od -5000 po 5000 ako je v zadaní. Nakoľko som použil funkciu random.gauss(), ako som spomenul vyššie, tak ako odchýlku som si zvolil 500, nakoľko takýto pomer intervalu a odchýlky mi dával pekne tvarované klastre. Za úspešný zhlukovač som teda považoval ten, ktorého všetky klastre mali priemer v rámci klastra menší ako 5 násobok odchýlky, teda menší ako 2500.

Ako prvé by som porovnal závislosť priemernej vzdialenosti od rastúceho „k“, pri jednotlivých zhlukovacích algoritmoch. Vývoj algoritmu k-means, kde stred klastra je centroid, môžeme vidieť na grafe na obrázku 1. Výsledný graf je zostavený z piatich spriemerovaných vzoriek. Prvú vec, ktorú si všimneme hneď, a ktorú by sme aj očakávali je, že so zvyšujúcim sa „k“ sa znižuje aj priemerná vzdialenosť v rámci klastrov. Ďalšiu vec, ktorú si môžeme všimnúť, že zo zvyšujúcim sa „k“ nám na začiatku klesá priemer vo veľkej miere, avšak približne od 8 nám začne klesať „k“ už miernejšie, a teda hodnotu 8, môžeme v danom grafe označiť za takzvaný lakeť grafu. Lakeť grafu je hodnota, kedy zmena hodnoty priemeru začne byť miernejšia, a menej výrazná. Táto hodnota je dôležitá, nakoľko sa považuje za primerane optimálne riešenie bez toho aby sme mali vysoký počet klastrov.



Obrázok 1

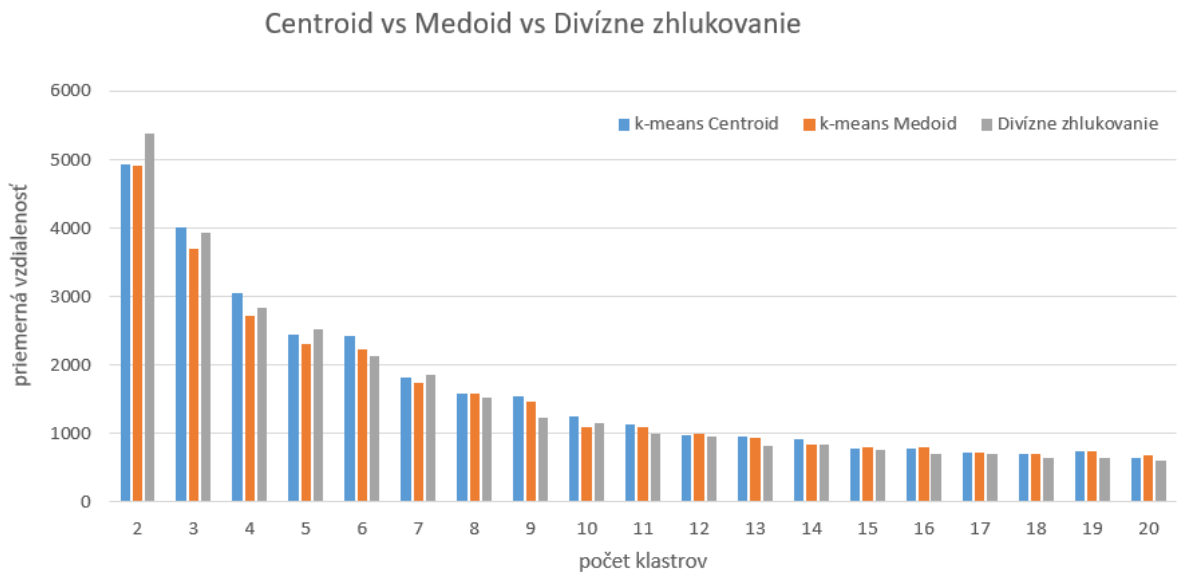
Obrázok 2 nám zobrazuje vývoj k-means algoritmu so stredom reprezentovaným ako medoid. Experiment bol vykonaný tiež na 5-tich vzorkách, ktoré sú spriemerované. Opäť môžeme vidieť, že so zvyšujúcim sa počtom klastrov blížiacim sa k 20 sa nám znižuje aj priemerná vzdialenosť v rámci klastrov. Nakoľko to je rovnaký algoritmus, iba so zmenou pri určovaní stredu klastra, tak aj tu môžeme považovať za lakeť grafu približne hodnotu 8.



Obrázok 2

Na rovnakej množine bodov ako pri predošlých dvoch testoch som vykonal aj experiment s divíznym zhlukovačom. Na samotnom grafe však nebolo vidieť nič nového, a teda som sa rozhodol samostatný graf do dokumentácie nedávať. Výsledky som ale zahrnul do spojeného grafu na obrázku 3, kde som navzájom porovnal k-means s centroidom, medoidom a divízne zhlukovanie.

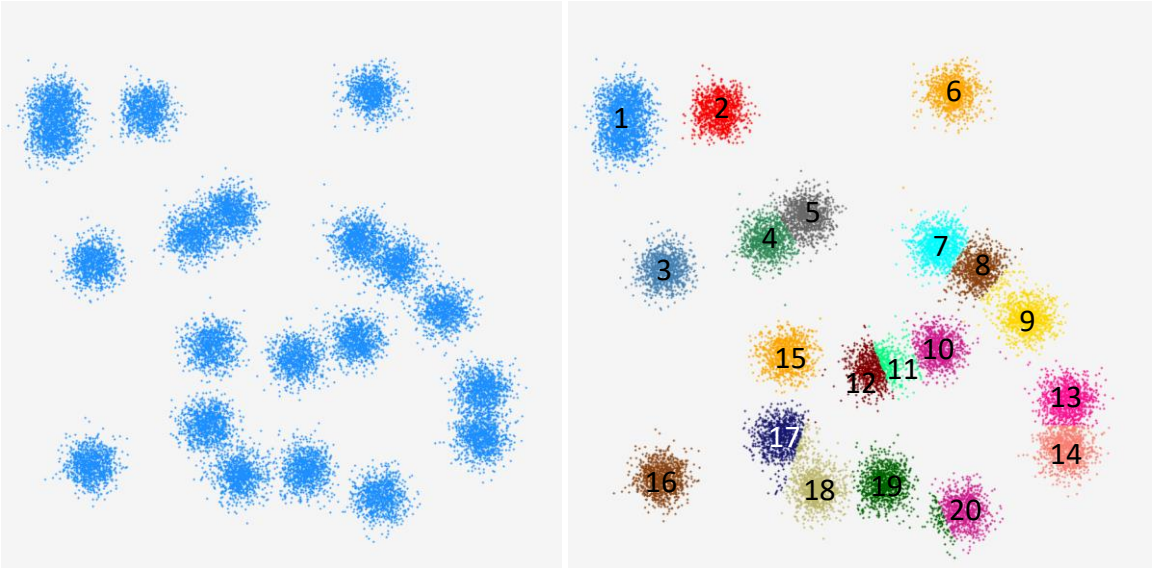
Nakoľko prvé 2 algoritmy sa líšia iba v určovaní stredu, tak aj ich výsledky sú veľmi podobné. Priemerne však dosahoval medoid o trochu lepšie výsledky, keďže ak klaster zahrňoval aj nejakých „outlierov“, tak títo mali oveľa väčší dopad na súradnice centroidu ako medoidu. Pri väčšom počte klastrov však môžeme vidieť, že už aj tieto malé rozdiely sa stratili a priemery sú približne rovnaké. Dôvodom, prečo sú pri väčšom počte klastrov rozdiely už rovnaké je, že klastre sú už menšie, a teda aj outlierov je tam menej, respektíve sú bližšie ako napríklad pri počte klastrov rovný 3. Avšak aj keď nám medoid dával lepšie výsledky, časovo bol na tom horšie nakoľko sme tam stred hľadali ako bod najbližšie ku všetkým bodom, čo má časovú zložitosť $O(n^2)$, kde „n“ je počet prvkov v klastri. To mi pri 20020 prvkoch trvalo približne 5-6 minút. Naopak ak je stred centroid, tak časová zložitosť pre nájdenie nového stredu klastru je iba O(n), pretože iba raz prejdem celý klaster a zrátam súradnice a následne ich spriemerujem, čo mi trvalo približne 6 sekúnd. Divízne zhlukovanie v porovnaní s k-means dosahovalo zo začiatku mierne horšie výsledky. Od hodnoty „k“ rovnej 8 sa však tento pomer otáča, a divízne zhlukovanie začína mať nižšiu hodnotu priemernej vzdialenosti v rámci klastrov ako k-means. Tieto rozdiely však sú veľmi nízke až zanedbateľné, a teda je ťažké určiť dôvod rozdielov.



Obrázok 3

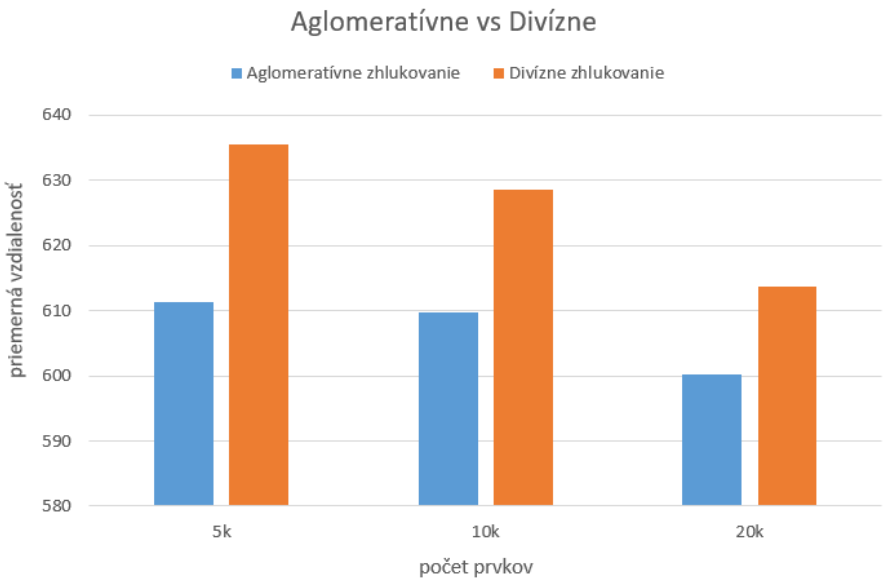
Na obrázku 4 môžeme vidieť stav pred a po spustení divízneho zhlukovača, kedy nám vytvorí 20 klastrov. Môžeme si všimnúť, že klaster číslo 1 sa skladá v podstate z dvoch klastrov, avšak keďže sú tieto zhluky veľmi blízko seba, tak ich vyhodnotí ako 1 klaster. To má za následok, nakoľko sme si určili, že chceme 20 klastrov, že jeden klaster rozdelil na 2 polovice a to na klastre číslo 11 a 12. Na tomto príklade teda vidíme, že divízny zhlukovač nám pri počte klastrov 20 skoro ohodnotil body ako boli pôvodne vygenerované, s malými výchylkami aké

môžeme vidieť pri klastri 20, kde nám malú časť klastra označil ako zelenú, aj keď by sme povedali, že by mala byť ružová celá.



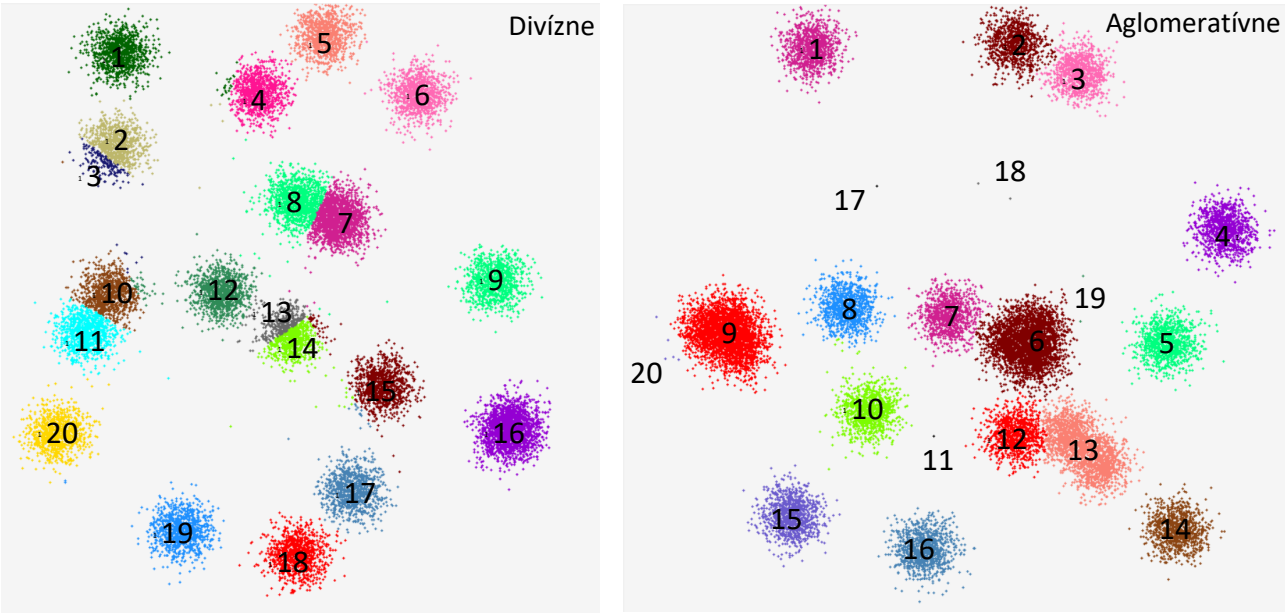
Obrázok 4

Ako posledné by som porovnal divízne zhlukovanie s aglomeratívnym. Oba sú to hierarchické zhlukovacie algoritmy, každý však ohodnocuje z opačného smeru. Divízne má na začiatku 1 klaster, ktorý rozdelí na 2 a následne si vyberie z tých dvoch ten, ktorý má väčšiu priemernú vzdialenosť v rámci klastra. Aglomeratívne naopak považuje na začiatku každý bod za samostatný klaster a spojí 2 najbližšie. Porovnanie pre 5020, 10020 a 20020 prvkov môžeme vidieť na grafe na obrázku 5. Pri všetkých troch meraniach si môžeme všimnúť, že aglomeratívne zhlukovanie má pri 20 klastroch vždy o niečo lepšie priemerné vzdialenosti.



Obrázok 5

Dôvod prečo obišlo v tomto experimente lepšie aglomeratívne si ukážeme na príklade, ktorý je na obrázku 6. Môžeme si všimnúť, že pri divíznom zhlukovaní má každý klaster určitý počet prvkov, a žiadny neobsahuje menej ako 10 prvkov. Naopak pri aglomeratívnom si môžeme všimnúť, že máme dokonca 5 klastrov (11, 17, 18, 19, 20), ktoré majú menej ako 6 prvkov. To nám zabezpečí, že „outlieri“ nebudú kaziť priemer nejakého klastra, ale ostanú sami. Napríklad na rozložení bodov naľavo, by aglomeratívne zhlukovanie pravdepodobne nepridelilo klastru 19 aj modrý bod pri žltom klastri číslo 20.

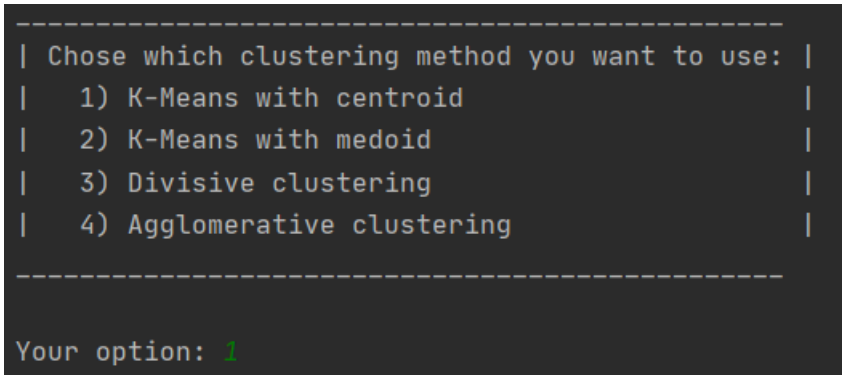


Obrázok 6

Avšak napriek tomu, že aglomeratívne zhlukovanie dávalo trochu lepšie výsledky, jeho časová zložitosť je privysoká, konkrétne $O(n^3)$, nakoľko využívam iba maticu susednosti a nie aj haldu, ktorá by mala usporiadané vzdialenosti medzi klastrami. Z tohto dôvodu mi ohodnotenie bodov, spolu s vykreslením pri aglomeratívnom zhlukovaní pri 10020 bodoch trvalo približne 61 minút a pri 20020 bodoch trvalo 8 hodín aj 16 minút. Výsledný počet klastrov pri týchto meraniach bol rovný 20.

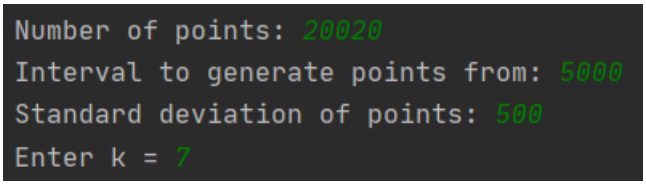
5) Používateľské prostredie

Používateľ si ako prvé na začiatku vyberie, ktorý algoritmus chce spustiť. Na obrázku 7 môžeme vidieť, ktoré číslo má napísať pre ktorý zhlukovací algoritmus.



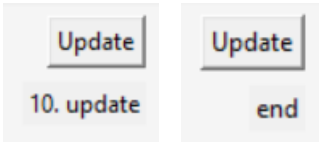
Obrázok 7

Následne si môže používateľ určiť aký chce testovací pomer. Ako vidíme na obrázku 8, tak je potrebné aby zadal koľko bodov chce vygenerovať, aký má byť interval generovania bodov a aká má byť štandardná odchýlka. Interval z obrázka 8 by bol napríklad od -5000 po 5000. Takže zadávame ako keby jeho vzdialenosť od 0, pričom sa vygeneruje od „-číslo“ po „číslo“. Poslednú vec, ktorú používateľ zadá je hodnota „k“, čiže počet klastrov na konci.



Obrázok 8

Po úspešnom zbehnutí programu sa otvorí okno s vykreslenými bodmi. Ak si používateľ zvolil na začiatku vykreslenie jedného z prvých troch algoritmov, tak má v okne aj funkčné tlačidlo na „update“ okna, ktoré po stlačení vykreslí rozloženie bodov po jednej iterácii daného algoritmu (obrázok 9 naľavo). Ak už je vykreslená posledná iterácia, tak sa pod tlačidlom zmení nadpis na „end“, čo nám hovorí, že sme na konci a môžeme už zavrieť oknom do ktorého boli body vykresľované.



Obrázok 9

6) Záver

Na záver by som zhodnotil moje pozorovania, ktoré som obdržal počas experimentov. Ak si porovnáme k-means so stredom ako centroid a so stredom ako medoid, tak zistíme, že keď má daný klaster viac outlierov, čo sa pri generovaní bodov za pomoci funkcie gauss() stane, tak títo outlieri viac ovplyvnia centroid ako medoid, a tým nakoľko ho posunú tak ovplyvnia aj celkový priemer a teda mal medoid vo väčšine prípadov o trochu lepšie výsledne priemery. Nevýhodou k-means však je, že vo veľkej miere záleží od počiatočného stavu, ako sú určené prvé klastre, a tak koľko-krát pustíme algoritmus, toľko rôznych výsledkov dostaneme. Výhodou ale je jeho časová zložitosť, ktorá záleží od počtu klastrov, bodov a času potrebného na výpočet vzdialenosti dvoch bodov. To vieme zjednodušiť v podstate na O(n) a teda je to o dosť rýchlejšie ako napríklad moja implementácia aglomeratívneho zhlukovania, ktorý má časovú zložitosť $O(n^3)$. Aglomeratívne zhlukovanie, mi napriek jeho časovej zložitosti vyšlo ako najlepší algoritmus, ktorý má najmenšiu hodnotu priemernej vzdialeností v rámci klastra. Jeho výsledky boli síce najlepšie, avšak je za ne vysoká časová daň, ktorú som už spomínal. Ďalším zistením je, že ak aglomeratívny zhlukovač má medzi bodmi experimentu aj nejakých outlierov, tak ak je to možné, tak ich nechá osamote, čo sa mi aj stalo pri situácií, ktorú môžeme vidieť na obrázku 6. Ak si porovnáme časovú zložitosť, tak tam mi vyšiel najrýchlejší divízny zhlukovač, nakoľko aj keď využíva k-means algoritmus, tak vždy ho využíva v podstate iba na počet klastrov rovný 2 a zároveň každým delením tam posielame čím ďalej tým menšie klastre. To má vo finále za následok, že je rýchli a zároveň dáva aj pomerne dobré výsledky. Jedinou jeho nevýhodou je, že ak nejaký bod zaradíme do jedného klastra na začiatku, už nie je možnosť ho zaradiť do iného klastra ako do toho, ktorý môže vzniknúť rozdelením klastra do ktorého je zaradený.

V mojich implementáciách vidím aj priestor na zlepšenie a zefektívnenie daných algoritmov. Napríklad pri aglomeratívnom by sme časovú zložitosť vedeli znížiť z $O(n^3)$ na $O(n^2 * \log(n))$, ak by som si dával vzdialenosti medzi klastrami do minimálnej haldy, a z nej vždy spájal 2 najbližšie prvky. Toto vylepšenie by nás však stalo nejakú tú pamäť a teda by to nebolo zadarmo. Ďalším vylepšením by mohlo byť pri k-means so stredom ako medoid, ak by som vzdialenosti nepočítal vždy nanovo ale by som si ich pamätal v matici, podobne ako pri aglomeratívnom zhlukovaní, čo by mi určite urýchlilo program. Posledným vylepšením, ktoré mi ešte napadlo je označovať si jednotlivé klastre trochu inač, aby som na konci pri vykreslení im iba na základe označenia pridelil farbu, čím by nevznikal mierny chaos vo farbách, ktorý momentálne môže nastať ak je náhodou pre klastre vedľa seba vybraná rovnaká farba z poľa farieb, ktoré mám staticky zadefinované na začiatku programu.