Umelá inteligencia – Zadanie 4b

Klastrovanie
Dokumentácia
2021/2022

Meno: Jakub Šimko

AIS ID: 103146

Cvičiaci: Ing. Ivan Kapustík

Čas cvičení: Streda 13:00

Meno: Jakub Šimko
AIS ID: 103146

Obsah

Meno: Jakub Šimko
AIS ID: 103146

1. Zadanie

Zadanie 4b – klastrovanie

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 20000 bodov, avšak tieto body nebudú generované úplne náhodne, ale nasledovným spôsobom:

1. Náhodne vyberte jeden zo **všetkých** doteraz vytvorených bodov v 2D priestore.
   Ak je bod príliš blízko okraju, tak zredukujete príslušný interval v nasledujúcich dvoch krokoch.
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.

Dokumentácia musí obsahovať opis konkrétne použitých algoritmov a reprezentácie údajov. V závere zhodnoťte dosiahnuté výsledky ich porovnaním.

Poznámka: Je vhodné použiť rôzne optimalizácie pre dostatočne efektívnu prácu Vášho zhlukovača. Napríklad pre aglomeratívne zhlukovanie je možné použiť 2-rozmernú maticu vzdialenosti dvojíc bodov. Naplnenie takejto matice má kvadratickú zložitosť. Potom sa hľadá najbližšia dvojica (najmenšie číslo v matici), to má opäť kvadratickú zložitosť, ale nenásobia sa tie časy, ale sčítavajú. Po výbere najbližšej dvojice túto dvojicu treba zlúčiť a tým sa zníži veľkosť matice o 1 (lebo sa zníži počet zhlukov o 1) Pri tom sa aktualizujú len vzdialenosti pre tento nový zhluk (len jeden stĺpec/riadok, zvyšok matice ostáva nezmenený).

Obrázok 1: Zadanie 4b

Meno: Jakub Šimko
AIS ID: 103146

2. Úvod

Cieľom dokumentácie je vysvetliť fungovanie implementovaného zhlukovača pre metódy klastrovania zo zadania.

3. Optimalizácie a vylepšenia

Pre obidve metódy aglomeratívneho klastrovania je využívana matica vzdialeností.

Pre zvýšenie rýchlosti je využívaný interpreter pypy a nie python.

Vymyslel som aj menšie vylepšenia pre algortimy divizívneho a aglomeratívneho klastrovania, ktoré sú uvedené v daných kapitolách. Taktiež jedno vylepšenie pri testovaní na určovanie potencionálne optimálneho k pre k-means.

4. Centroid

Funkcia **calc_centroid()** v alghoritms.py je zodpovedná za výpočet centroidu.

5. Medoid

Funkcia **calc_medoid()** v algorithms.py je zodpovedná za výpočet medoidu.

Let $\mathcal{X} := \{x_1, x_2, \ldots, x_n\}$ be a set of n points in a space with a distance function d. Medoid is defined as

$$x_{\text{medoid}} = \arg\min_{y \in \mathcal{X}} \sum_{i=1}^{n} d(y, x_i).$$

Obrázok 2: Výpočet medoidu

Skúšal som vytvoriť aj optimalizáciu, ktorá by využívala maticu vzdialeností aby nebolo nutné počítať vzdialenosť z bodu A do bodu B a potom pri ďalšom výpočte zase vzdialenosť z bodu B do bodu A. Táto funkcia bola pre môj notebook príliš pamäťovo náročná a nakoniec som ju nevyužil. Ostala zakomentovaná pod funkciou **calc_medoid()**.

Meno: Jakub Šimko
AIS ID: 103146

6. K-means klastrovanie

Pri K-means je nutné určiť k a poslať ho do funkcie ako argument. Na vykonanie k-means klastrovania sa využívajú okrem funkcií na výpočet centroidu a medoidu ešte funkcie:

k_means() – hlavná funkcia klastrovanie – vykonáva celý algoritmus

Funkcia funguje následovne:

1. Vyber k začiatočných bodov (funkcia k_means_initial())
2. Priraď body do klastrov (funkcia assign_to_clusters())
3. Prepočítaj medoidy/centroidy klastrov
4. Opakuj body 2 a 3 dokedy sa centroidy/medoidy nerovnajú tým z prechádzajúcej iterácie (žiadna zmena)

k_means_initial() – vyberie k začiatočných bodov

assign_to_clusters() – priradzuje body do klastrov na základe vzdialeností bodu od centroidu/medoidu.

Meno: Jakub Šimko
AIS ID: 103146

7. Divízne klastrovanie – centroid

Divizívne klastrovanie som implementoval následovne:

1. Rozdeľ klaster obsahujúci všetky body na polovicu a vypočítaj centroidy pomocou k-means
2. Ak klaster spĺňa podmienku úspešnosti tak ho priraď do poľa výsledkových klastrov.
3. Ak klaster túto pomienku nespĺňa tak ho rozdeľ na polovičku a vypočítaj centroidy pomocou k-means
4. Opakuj body 2 a 3 dokedy nie sú všetky klastre v poli výsledkových klastrov a teda všetky klastre majú priemernú vzdialenosť bodov menšiu ako 500 (úspešny zhlukovač)

Dôvod prečo som divizívne klastrovanie implementoval takto narozdiel od klasického riešenia je ten, že ak chceme dosiahnuť 100% úspešnosť tak nam stačí rozdeľovať klastre dokedy nebudú všetky splnať zadanú podmienku. Nemá teda zmysel vykonávať ďalšie rozdeľovanie ak daný klaster túto podmienku už splňa. Týmto vylepšením sa dokáže program vyhnúť nadbytočným výpočtom čo nám ušetrí čas. Zároveň pre nás nie je nutné určovať počet výsledných klastrov, program na optimálny počet príde sám.

Divizívne klastrovanie je implementované vo funkcii **divisive()**.

8. Aglomeratívne klastrovanie – centroid

Pri tejto implementácii aglomeratívneho klastrovania je nutné určiť k (počet výslednych klastrov) a poslať ho do funkcie ako argument. Na základe k program vie kedy už nemá klastre ďalej spájať.

Aglomeratívne klastrovanie (obyčajné) som implemetnoval následovne:

1. Vytvor maticu vzdialeností a priraď každý bod do samostatného klastra
2. Najdi najmenšiu vzdialenosť v matici
3. Spoj klastre s najmenšou vzdialenosťou
4. Aktualizuj centroid tohto spojeného klastra
5. Aktualizuj hodnoty matice pre riadok a stlpec klastra do ktorého sa druhý klaster pripojil
6. Vymaž z matice stlpec a riadok klastra ktory sa pripojil
7. Opakuj body 2 až 6 dokedy nie je počet klastrov rovný k

Aglomeratívne klastrovanie je implementované vo funkcii **aglomerative().**

9. Aglomeratívne klastrovanie – centroid (vylepšené)

Pri tejto implementácií nie je nutné určiť k (počet výslednych klastrov). Myšlienka za mojim vylepšením je taká, že nepotrebujeme vopred určiť/poznať optimálne k ak chceme dosiahnuť 100% úspešnosť. Jedine čo nám stačí robiť je upraviť pôvodny algoritmus tak aby vykonával spájanie pre najbližšie klastre iba vtedy ak týmto spojením nedôjde k presiahnutiu maximálnej vzdialenosti pre úspešny zhlukovač (500). Dôvod je taký, že ak spojením klastrov, ktoré sú k sebe najbližšie bude prekonaná maximálna vzdialenosť úspešnosti tak neexistuje taký klaster ktorým spojením by podmienka úspešnosti nebola porušená.

Aglomeratívne klastrovanie (vylepšené) som implemetnoval následovne:

1. Vytvor maticu vzdialeností a priraď každý bod do samostatného klastra
2. Najdi najmenšiu vzdialenosť v matici
3. Ak spojením týchto dvoch klastrov bude porušená podmienka úspešnosti tak ich nespájaj ale presuň ich do poľa výsledkových klastrov. Vymaž riadky a stlpce týchto klastrov z matice vzdialeností a vráť sa na bod 2.
4. Ak spojením týchto dvoch klastrov nebude porušená podmienka úspešnosti tak ich spoj.
5. Aktualizuj hodnoty matice pre riadok a stlpec klastra do ktorého sa druhý klaster pripojil
6. Vymaž z matice stlpec a riadok klastra ktory sa pripojil
7. Opakuj body 2 až 6 dokedy nie sú všetky klastre v poli výsledkových klastrov – nejde už vykonať žiadne spojenie, ktorým by sa neprekročila maximálna vzialenosť úspešnosti.

Aglomeratívne klastrovanie (vylepšené) je implementované vo funkcii **aglomerative2()**

10.Testovanie

Testovanie bolo realizované tak, že sa na začiatku daného testovacieho kola vygeneroval dataset 20 020 bodov a následne sa nad týmto datasetom vykonal každý algoritmus. Testovacie kolo sa opakovalo 10 krát (vždy s novým datasetom) a bola určená úspešnosť zhlukovača.

Výsledky testovania sa ukládajú do priečinku GRAFY kde sú rozdelené na základe spustenia daného testu. Ak sa vykoná testovanie rovnaké tomu v dokumentácií tak sa v priečinku GRAFY/<čas_začiatku_vykonania_testu>/ budú nachádzať vytvorené vizualizácie výsledkov klastrovania pre každý algoritmus daného testovacieho kola.

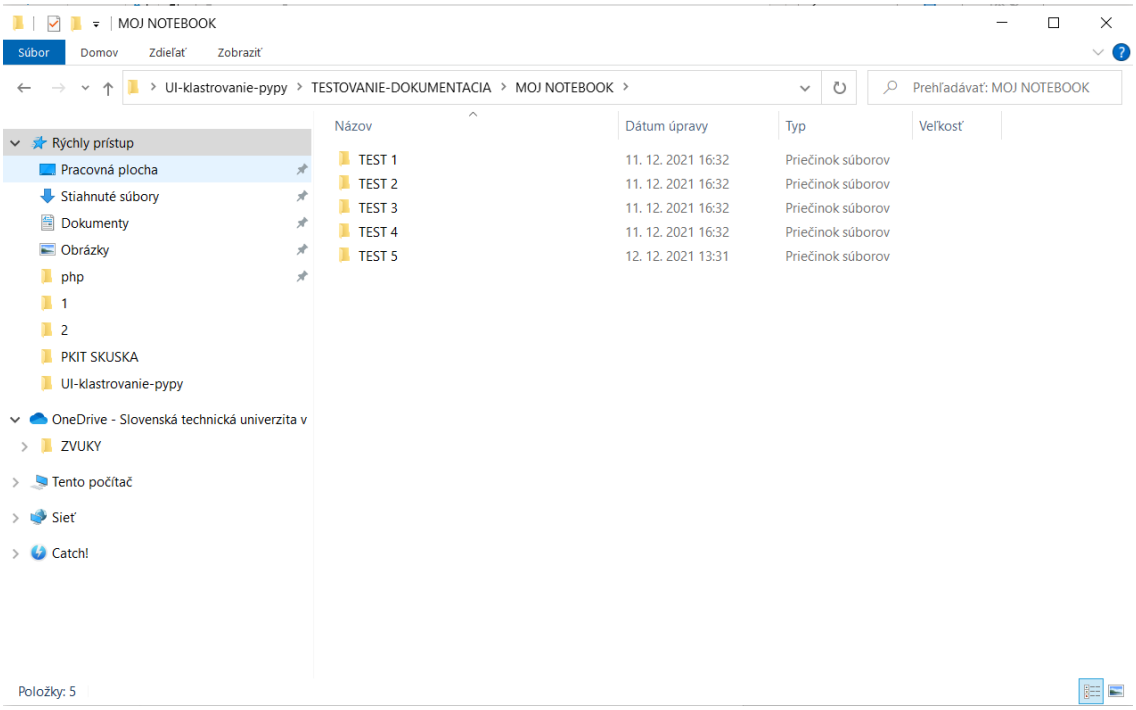
Ak by som testovanie vykonával len na svojom notebooku tak by zabralo cca 30-33 hodín. Je tomu tak pretože mám v programe 2 aglomeratívne algoritmy, ktoré dokopy bežia 160minút. Testovanie som musel teda kvôli časovej náročnosti vykonávať na 2 notebookoch. Na obidvoch notebookoch bolo dokopy vykonaných 10 testovacích kôl. 5 na mojom, 5 na výkonnejšom. Pri priemerných časoch je spomenutá priemerná dĺžka vykonávania pre obidve notebooky. Parametre notebookov:
Notebook 1 (môj): CPU: Intel Core i5-6300HQ, RAM: 8GB DDR4 2133MHz, HDD 1TB, GPU: GTX NVIDIA GeForce 950M 4GB
Notebook 2 (výkonnejší): CPU: Intel Core i7-8750H, RAM: 16GB DDR4 2667MHz, SSD 1TB, GPU: NVIDIA GeForce RTX 2060
Dôvod prečo uvidíme, že môj notebook bol v niektorých prípadoch rýchlejší ako výkonnejší notebook je ten, že môj notebook bol absolútne nepoužiteľný počas testovania a program mal teda 100% výkonu len pre seba zatiaľ čo druhý notebook sa mohol v poriadku používať na iné aktivity čo sa teda prejavilo na niektorých časoch z testovania.

Testovanie pre k-means som vykonával s k, ktoré bolo určené počtom klastrov výsledku divizivného klastrovania. Ak by sme chceli nájsť optimálne k pre každý jeden 20 020 dataset tak by bolo nutné vykonávať hľadanie tzv. kolena/lakťa v grafe pre jednotlivé k-means. Preto si myslím, že toto je elegantné vylepšenie/spôsob ako sa priblížiť k optimálnemu k s nízkou časovou náročnosťou. K-means ale ajtak nedosahuje vysokú úspešnosť. Dôvod prečo sa k-means nedarí dosahovať vďaka takémuto prístupu vyššiu úspešnosť je ten, že môj nápad sa mi nepodarilo

dotiahnuť do konca. Musel som už začať testovať (na testovanie som potreboval cca 15 hodín [s 2 notebookmi bežiacimi zároveň] a na vyhodnotenie ďalších pár hodín...). Síce sa nám podarilo určiť potencionálne optimálne k ale na začiatku k-means sa vyberajú body náhodne a to vlastne ničí výslednú úspešnosť. Ak by testovanie nebolo tak časovo náročné tak by bol priestor dokončiť túto myšlienku a teda posielať do k-means začiatočné body určené na základe najbližších bodov k centroidom z divizívneho klastrovania. Nemám to síce overené ale myslím si, že tým by sa podarilo to aby k-means dosahovalo vysokú úspešnosť. Časť nápadu som tam ale zanechal a určite ho dotiahnem do konca počas voľného času.

Pri testovaní aglomeratívneho klastrovania (obyčajného) som sa rozhodol, že je vhodné aby počet klastrov, ktorý chceme klastrovaním dosiahnuť bol určený počtom klastrov z výpočtu vylepšeného aglomeratívneho klastrovania. Týmto sa nám podarí dosiahnuť OK úspešnosť.

K výsledkom pre dané algoritmy som pridával len jednu z 10 vizuálizacií s podrobnejšími výsledkami z daného testovania algoritmu. Pri vizualizácií sa čislo klastra dá zistiť priblížením na jeho medoid/centroid. Na ostatné výsledky a vizualizácie z vykonaného testovania sa dá pozrieť v priečinku TESTOVANIE-DOKUMENTACIA.



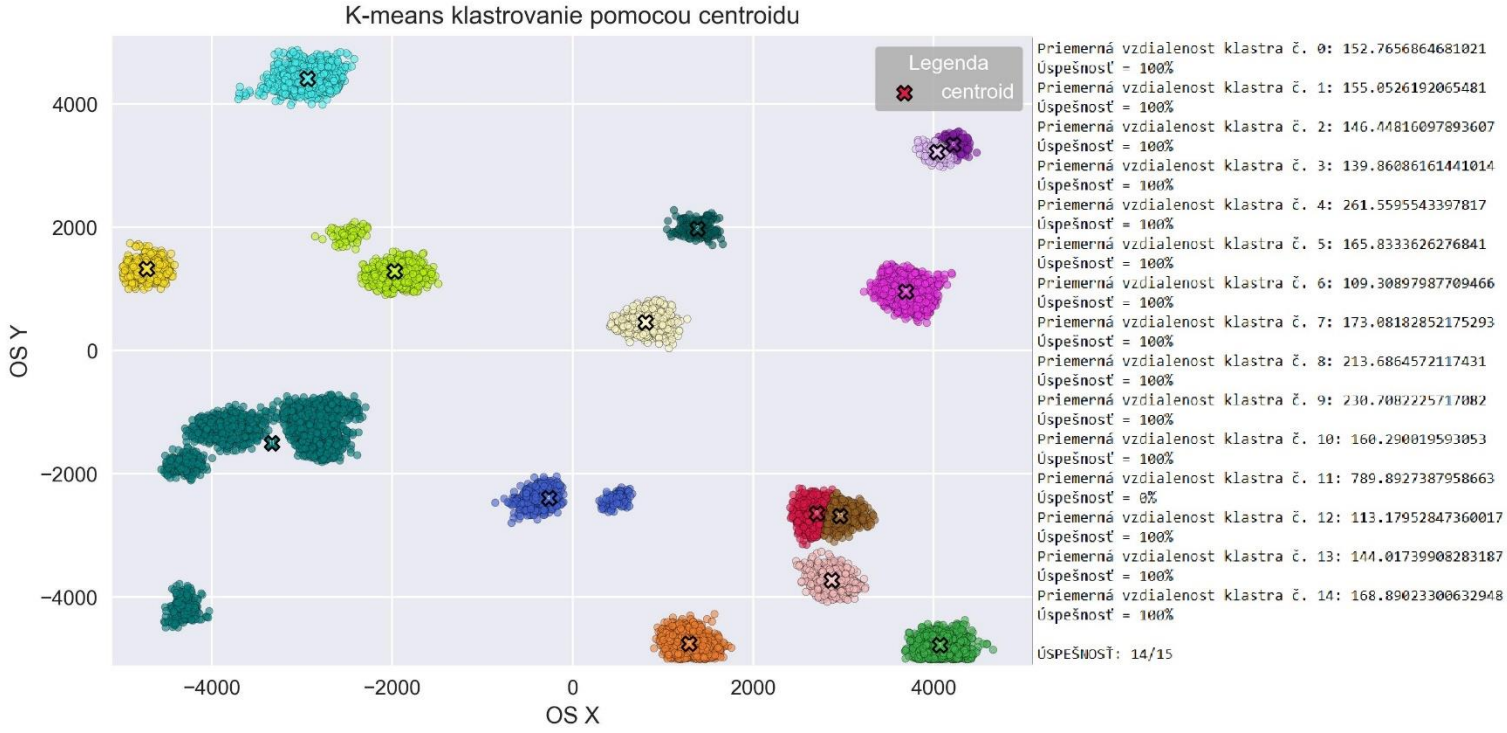
Meno: Jakub Šimko
AIS ID: 103146

10.1. K-means klastrovanie (centroid) - výsledky

Úspešnosť: 0/10

Priemerná dĺžka vykonávania (Môj notebook): 1,3s

Priemerná dĺžka vykonávania (Notebook 2): 1,8s



Obrázok 3: Vizualizácia a úspešnosť k-means (centroid) z testovacieho kola č. 2

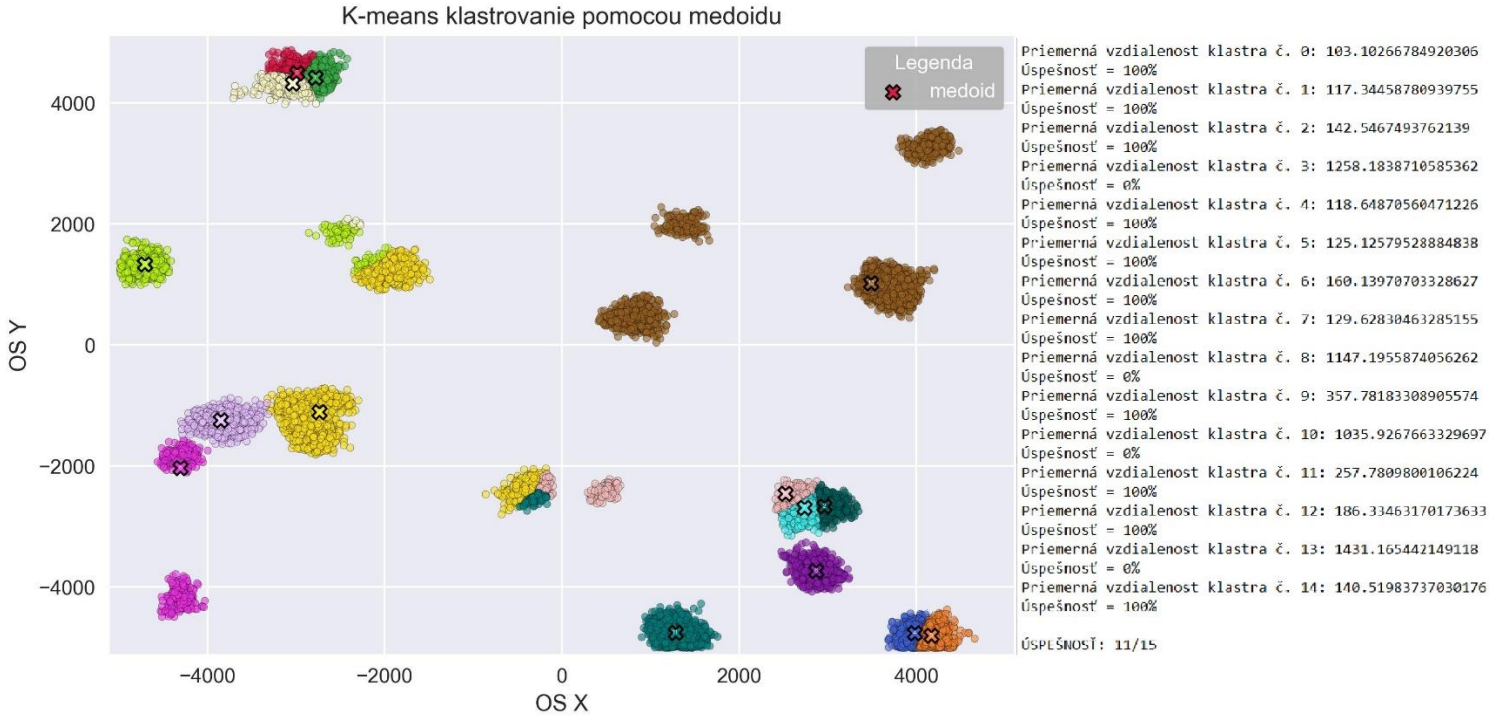
Meno: Jakub Šimko
AIS ID: 103146

10.2. K-means klastrovanie (medoid) - výsledky

Úspešnosť: 0/10

Priemerná dĺžka vykonávania (Môj notebook): 103s

Priemerná dĺžka vykonávania (Notebook 2): 74,6s



Obrázok 4: Vizualizácia a úspešnosť k-means (medoid) z testovacieho kola č. 2

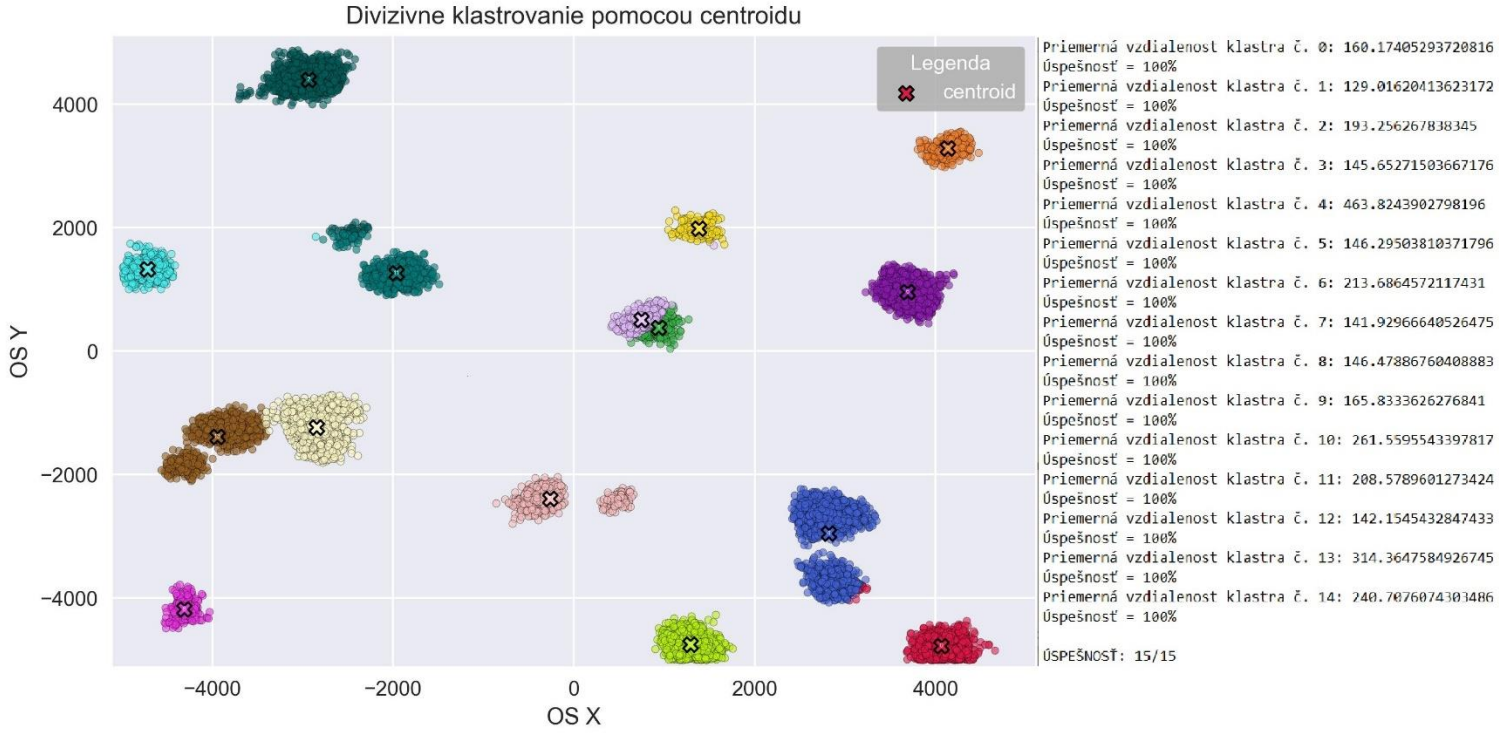
Meno: Jakub Šimko
AIS ID: 103146

10.3. Divizívne klastrovanie - výsledky

Úspešnosť: 10/10

Priemerná dĺžka vykonávania (Môj notebook): 0,3s

Priemerná dĺžka vykonávania (Notebook 2): 0,24s



Obrázok 5: Vizualizácia a úspešnosť divizivneho klastrovania (centroid) z testovacieho kola č. 2

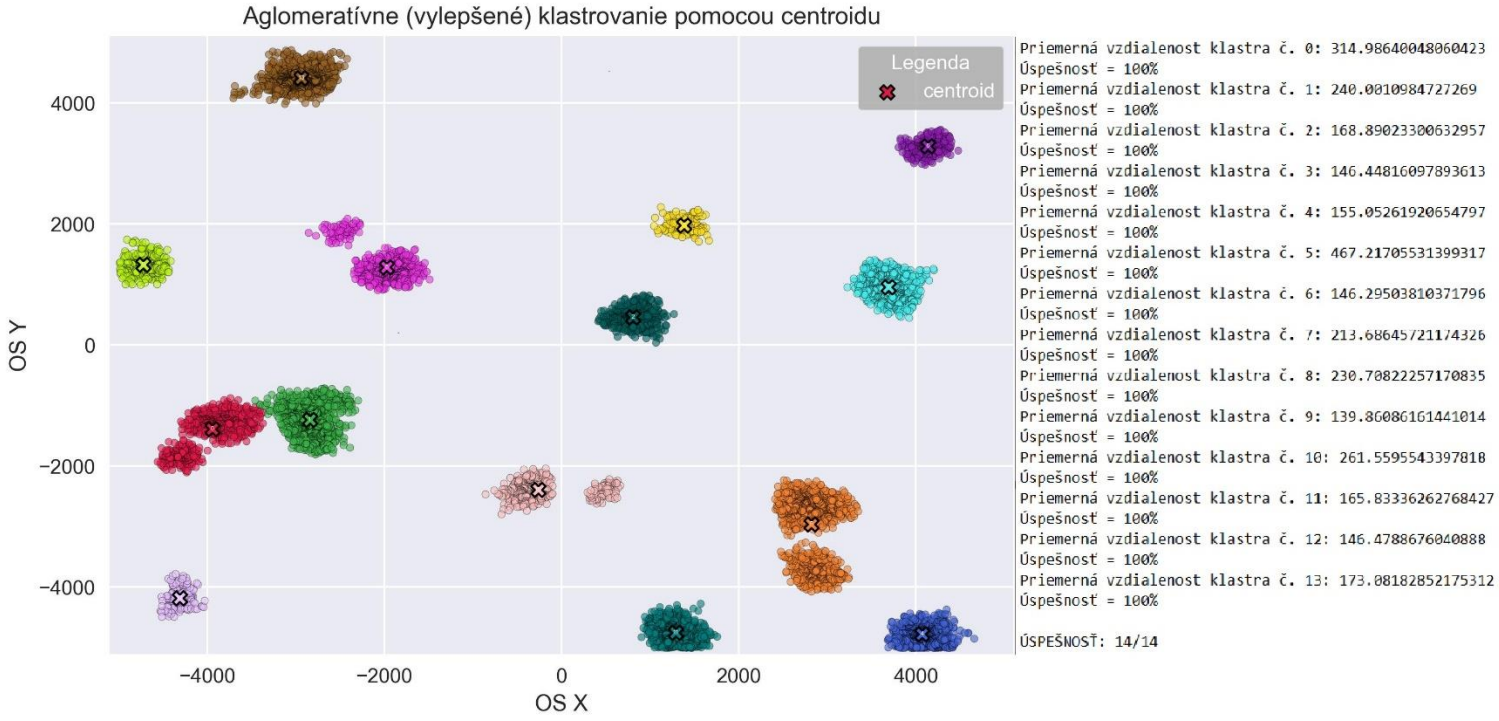
Meno: Jakub Šimko
AIS ID: 103146

10.4. Aglomeratívne klastrovanie (vylepšené) - výsledky

Úspešnosť: 10/10

Priemerná dĺžka vykonávania (Môj notebook): 4855s

Priemerná dĺžka vykonávania (Notebook 2): 5177s



Obrázok 6: Vizualizácia a úspešnosť vylepšeného aglomeratívneho klastrovania (centroid) z testovacieho kola č. 2

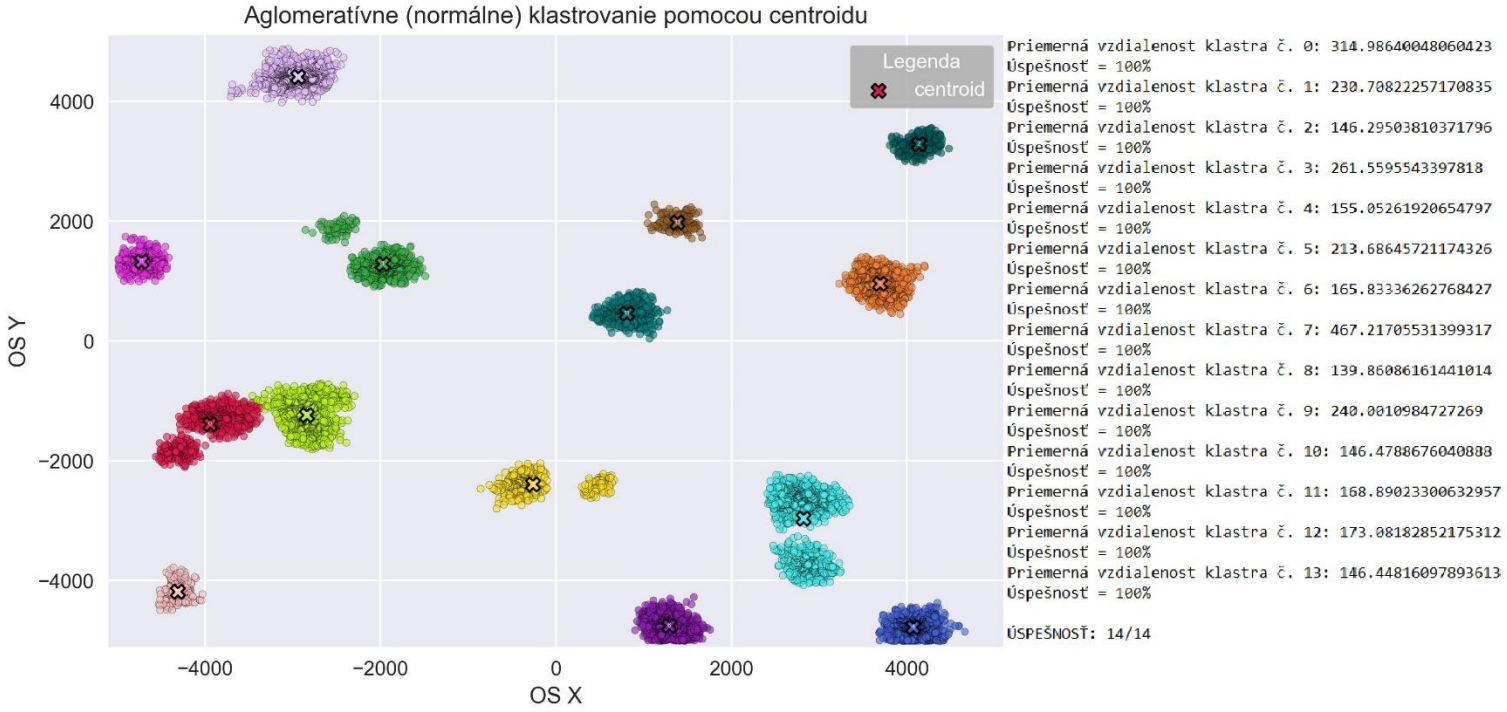
Meno: Jakub Šimko
AIS ID: 103146

10.5. Aglomeratívne klastrovanie (obyčajné) - výsledky

Úspešnosť: 2/10

Priemerná dĺžka vykonávania (Môj notebook): 4859s

Priemerná dĺžka vykonávania (Notebook 2): 5269s



Obrázok 7: Vizualizácia a úspešnosť normálneho aglomeratívneho klastrovania (centroid) z testovacieho kola č. 2

11. Testovanie – porovnanie

K-means má nepresné výsledky, nakoľko berie na začiatku náhodné body (to som spomínal už na začiatku testovania). Je ale rýchle ak využívame na výpočet centroid. Pre počítanie s medoidmi rastie časová zložitosť kvadraticky.

Divizívne klastrovanie má očakavanú 100% úspešnosť, pretože som ho tak navrhol a implementoval. Zároveň je aj veľmi rýchle.

Aglomeratívne zhlukovanie (obyčajné) má OK výsledky. Po implementovaní môjho vylepšenia dosahuje aglomeratívne zhlukovanie očakávanú 100% úspešnosť. Časová zložitosť je ale v obidvoch prípadoch príliš vysoká, pretože narastá kubicky.

Došiel som teda k záveru, že v konečnom dôsledku je najlepšie divizívne klastrovanie. Má 100% úspešnosť a zároveň aj výbornú časovú zložitosť.

12.Vizualizácia

Pri vizualizácií som sa snažil nájsť farby, ktoré sa dajú jednoducho rozlíšiť. Preto je možne vykresliť len 30 rôznych farieb (klastrov). Ak by sme chceli vykresľovať väčšie množstvo klastrov tak treba pridať ďalšie farby.

```
# 30 farieb
colors = ['#e6194b', '#3cb44b', "#00615f", '#ffe119', '#4363d8', '#f58231', '#911eb4', '#46f0f0', '#f032e6',
          '#bcf60c', '#fabebe', '#008080', '#e6beff', '#9a6324', '#fffac8', '#800000', '#aaffc3', '#808000',
          '#614931', '#000075', '#808080', '#9ACD32', '#7B68EE', '#FF7F50', '#BC8F8F', '#B8860B', '#4169E1',
          '#8A2BE2', '#B22222', '#FFA500']
```

13.Záver

Každý študent, ktorý dostal vypracovať zadanie B bol značne znevýhodnený oproti študentom, ktorý dostali na vypracovanie zadanie A. Zatiaľ čo vypracovanie algoritmov k-means a divizivneho klastrovania bolo časovo nenáročné, tak implementácia aglomeratívneho klastrovania zabrala príliš veľa úsilia v podobe nutnosti optimalizácie (ktoré pravdepodobne ani neni hodnotené) a času na testovanie, ktorý je neuveriteľne dlhý. Taktiež stojí za zmienku, že ja osobne nemám veľmi výkonný notebook čo mi testovanie tohto zadania veľmi výrazne predĺžilo. Notebook bol pre mňa počas testovania viac menej nepoužiteľný, kvôli nedostatku RAM na vykonávanie inných činností.

Celkovo sa mi ale zadanie veľmi páčilo, nadobudol som veľa nových vedomostí vrámci klastrovania a umelej inteligencie. Dokonca sa mi podarilo vymyslieť rôzne vylepšenia aj keď nie všetky nakoniec ostali v programe a boli vhodné/funkčné. Vďaka tomu sa mi ale podarilo zamyslieť sa nad danou problematikou o niečo viac. V konečnom dôsledku ale dúfam, že som zadanie vypracoval správne a vylepšeniami som si neuškodil.

Na záver by som rád povedal, že by bolo vhodné vykonať korekciu rozdielu náročnosti zadaní A a B alebo dať vypracovanie aglomeratívneho algoritmu za bonusové body.