Johanna Tilešová AIS ID: 127293

Dokumentácia

Aglomeratívne zhlukovanie v 2D priestore
Zadanie 2c

Obsah

Zadanie

Úloha spočíva v naprogramovaní zhlukovača, ktorý analyzuje 2D priestor s rozmermi od -5000 do +5000 v osiach X a Y. Program najprv vygeneruje 20 náhodných bodov s unikátnymi súradnicami. Následne vytvorí ďalších 20 000 bodov, ktoré sú umiestnené v blízkosti už existujúcich bodov, a to pomocou náhodného posunu v intervale od -100 do +100. Cieľom je tieto body rozdeliť do zhlukov (klastrov) pomocou dvoch verzií aglomeratívneho algoritmu - klastrovanie na základe centroidov a klastrovanie na základe medoidov.

Cieľ zadania

Cieľom projektu je vytvoriť funkčný a efektívny zhlukovač pre náhodne generované body v 2D priestore. Program musí generovať body podľa stanovených kritérií a následne ich rozdeliť do zhlukov tak, aby splnil podmienky úspešnosti. Úspešnosť zhlukovača sa hodnotí podľa priemernej vzdialenosti bodov od stredového bodu (centroidu alebo medoidu) v každom zhluku.

Očakávaný výstup

Výsledkom by malo byť rozdelenie bodov do zhlukov v 2D priestore, pričom každý zhluk bude reprezentovaný **centroidom** (priemer súradníc bodov v zhluku) alebo **meloidom** (bod s najmenšou celkovou vzdialenosťou od ostatných bodov v zhluku). Výsledky zhlukovania by mali byť vizualizované v 2D priestore s farebným odlíšením pre jednotlivé zhluky. Program tiež vyhodnotí úspešnosť zhlukovača tým, že porovná úspešné zhluky s počtom všetkých vytvorených zhlukov.

Riešenie

Generovanie bodov

Na generovanie všetkých bodov som použila funkciu **generuj_bodiky**, ktorá najprv vygeneruje 20 počiatočných bodov s náhodnými unikátnymi súradnicami v intervale od -5000 do +5000 na osi X a Y. Následne vygenerujeme ďalších 20000 extra bodov s posunom. Každý z týchto bodov vznikne výberom existujúceho bodu a pripočítaním náhodného posunu v intervale -100 až +100 k jeho súradniciam. Týmto spôsobom vznikne zhustená distribúcia bodov okolo počiatočných súradníc.

```python
# Generovanie počiatočných bodov a bodov s posunom
def generuj_bodiky(pociatocne_body, extra_body, x_min, x_max, y_min, y_max, limit_posunu):  1 usage
    print(f"Generujem {pociatocne_body + extra_body} bodov...")

    vsetky_body = set()
    while len(vsetky_body) < pociatocne_body:
        x, y = random.randint(x_min, x_max), random.randint(y_min, y_max)
        vsetky_body.add((x, y))

    vsetky_body = list(vsetky_body)

    for i in range(extra_body):
        zaklad_x, zaklad_y = random.choice(vsetky_body)
        x_posun = random.randint(max(-limit_posunu, x_min - zaklad_x), min(limit_posunu, x_max - zaklad_x))
        y_posun = random.randint(max(-limit_posunu, y_min - zaklad_y), min(limit_posunu, y_max - zaklad_y))
        novy_bod = (zaklad_x + x_posun, zaklad_y + y_posun)
        vsetky_body.append(novy_bod)
    return np.array(vsetky_body)
```

Výpočet vzdialenosti

Na výpočet vzdialenosti medzi dvoma bodmi som použila funkciu **euklidovska_vzdialenost**, ktorá počíta priamu vzdialenosť medzi dvoma bodmi v 2D priestore. Táto vzdialenosť sa vypočíta ako odmocnina zo súčtu druhých mocnín rozdielov medzi súradnicami bodov.

Funkcia **vytvor_maticu_vzdialenosti** slúži na vytvorenie matice vzdialeností medzi všetkými bodmi v poli bodiky. Táto matica efektívne vyhľadáva najbližšie dvojice bodov.

Pre tento prípad využívam operácie knižnice NumPy na rýchly výpočet vzdialeností pre všetky body súčasne, aby sa optimalizoval výpočet vzdialeností pri procese zhlukovania.

```python
#  Výpočet euklidovskej vzdialenosti
def euklidovska_vzdialenost(jeden_bod, druhy_bod):  5 usages
    return np.linalg.norm(jeden_bod - druhy_bod)

# Vytvorenie matice vzdialeností pomocou efektívnych NumPy operácií
def vytvor_maticu_vzdialenosti(bodiky):  2 usages
    rozdiel = bodiky[:, np.newaxis, :] - bodiky[np.newaxis, :, :]
    vzdialenost = np.sqrt(np.sum(rozdiel**2, axis=-1))
    np.fill_diagonal(vzdialenost, np.inf)
    return vzdialenost
```

Vyhodnotenie úspešnosti zhlukovača

Funkcia **uspesnost_zhlukovania** slúži na vyhodnotenie úspešnosti zhlukovania na základe priemernej vzdialenosti bodov od stredového bodu (medoidu) v každom zhluku. Za úspešný zhluk sa považuje taký, v ktorom priemerná vzdialenosť bodov od medoidu nepresahuje stanovený limit - **vzdialenost_od_stredu** je predvolená hodnota je 500.

```python
# Vyhodnotenie úspešnosti zhlukovania na základe stredovej vzdialenosti
def uspesnost_zhlukovania(klastre, vzdialenost_od_stredu=500):  2 usages
    uspesne_klastre = 0  # Počítadlo úspešných zhlukov

    for zhluk in klastre:
        # najdenie medoidu
        medoid = min(zhluk, key=lambda p: sum(euklidovska_vzdialenost(p, dalsi) for dalsi in zhluk))
        # Výpočet priemernej vzdialenosti všetkých bodov od medoidu
        priemerna_vzdialenost = np.mean([euklidovska_vzdialenost(bod, medoid) for bod in zhluk])

        # Kontrola, či priemerná vzdialenosť nepresahuje limit
        if priemerna_vzdialenost <= vzdialenost_od_stredu:
            uspesne_klastre += 1

    # Výpočet percentuálnej úspešnosti zhlukovania
    uspesnost = (uspesne_klastre / len(klastre)) * 100
    print(f"\nÚspešnosť zhlukovania: {uspesnost:.2f}%")

    return uspesnost
```

Zhlukovanie pomocou centroidov

Reprezentácia údajov

Body sú reprezentované ako dvojrozmerné numpy pole, ktoré obsahuje súradnice jednotlivých bodov. Zhluky sú implementované ako zoznam zoznamov, kde každý vnútorný zoznam predstavuje jeden zhluk a obsahuje body patriace do tohto zhluku.

Implementácia algoritmu

Na začiatku algoritmu je každý bod považovaný za samostatný zhluk. Následne sa vytvorí matica vzdialeností medzi všetkými bodmi, pričom diagonálne hodnoty tejto matice sú nastavené na nekonečno, aby sa zabránilo výberu rovnakého bodu ako najbližšieho. Algoritmus potom iteratívne zlúči dva najbližšie zhluky do jedného. Po zlúčení sa pre nový zhluk vypočíta centroid ako priemer súradníc všetkých bodov v tomto zhluku. Aktualizuje sa tiež matica vzdialeností iba pre nový zhluk, čo optimalizuje počet výpočtov. Algoritmus pokračuje, kým vzdialenosť medzi najbližšími zhlukmi nepresiahne limit 500, čím sa dosiahne požadované rozdelenie priestoru do zhlukov.

```python
def zhlukovanie_centroidy(bodiky, vzdialenost_od_stredu=500):  1 usage
    klastre = [[bod] for bod in bodiky] # pociatocne zhluky s jednotlivymi bodmi
    matica_vzdialenosti = vytvor_maticu_vzdialenosti(bodiky)
    centroidy = [np.mean(zhluk, axis=0) for zhluk in klastre] # vypocet pociatocnych centroidov

    cas_na_zaciatku = time.time()
    celkove_klastre = len(klastre)

    iteracia = 0
    while True:
        minimalna_vzdialenost = np.min(matica_vzdialenosti)
        if minimalna_vzdialenost > vzdialenost_od_stredu:
            break

        # zlucovanie dvoj najblizsich zhlukov
        i, j = np.unravel_index(np.argmin(matica_vzdialenosti), matica_vzdialenosti.shape)
        klastre[i].extend(klastre[j]) #pridanie bodov z klastu j do klastru i
        del klastre[j] #vymyzanie klastru j
        matica_vzdialenosti = np.delete(matica_vzdialenosti, j, axis=0)
        matica_vzdialenosti = np.delete(matica_vzdialenosti, j, axis=1)

        # aktualizuje sa centroid pre zluceny klaster i
        centroidy[i] = np.mean(klastre[i], axis=0)
        del centroidy[j]

        # prepocitanie vzdialenosti medzi zhlukmi
        for k in range(len(klastre)):
            if k != i:
                matica_vzdialenosti[i, k] = matica_vzdialenosti[k, i] = euklidovska_vzdialenost(centroidy[i], centroidy[k])

        iteracia += 1
        if iteracia % 100 == 0:
            percenta_hotovo = 100 * (celkove_klastre - len(klastre)) / celkove_klastre
            aktualny_cas = time.time() - cas_na_zaciatku
            print(f"Iterácia {iteracia}: {percenta_hotovo:.2f}% hotovo "
                  f"Čas = {aktualny_cas:.2f} sekúnd")

    cas_na_konci = time.time() - cas_na_zaciatku
    hodiny, zostatok = divmod(cas_na_konci, 3600)
    minuty, sekundy = divmod(zostatok, 60)
    print(f"\nZhlukovanie na základe centroidov dokončené za {int(hodiny)} hodín, {int(minuty)} minút a {sekundy:.2f} sekúnd")

    uspesnost = uspesnost_zhlukovania(klastre, vzdialenost_od_stredu)
    print(f"Zhlukovanie na základe centroidov pre {len(bodiky)} bodov - Úspešnosť: {uspesnost:.2f}%\n")

    popisky = np.full(len(bodiky), -1, dtype=int)
    for idx, zhluk in enumerate(klastre):
        for bod in zhluk:
            idx_bodu = np.where((bodiky == bod).all(axis=1))[0]
            if idx_bodu.size > 0:
                popisky[idx_bodu[0]] = idx

    return popisky, klastre
```

Zhlukovanie pomocou medoidov

Reprezentácia údajov

Údaje sú reprezentované podobne ako v algoritme s centroidmi – body sú uložené v numpy poli a zhluky ako zoznam zoznamov, kde každý vnútorný zoznam predstavuje jeden zhluk.

Implementácia algoritmu

Na začiatku algoritmus funguje rovnako ako zhlukovanie pomocou centroidov – každý bod je samostatným zhlukom a vytvorí sa matica vzdialeností. Zmena nastáva v každej iterácii, kde sa zlúčia dva najbližšie zhluky a pre nový zhluk sa vyberie medoid, teda bod, ktorý minimalizuje súčtovú vzdialenosť k ostatným bodom v zhluku. Matica vzdialeností sa potom aktualizuje len pre nový medoid, čo znižuje počet potrebných výpočtov a zvyšuje efektivitu algoritmu. Zhlukovanie pokračuje, až kým vzdialenosť medzi najbližšími zhlukmi neprekročí limit 500.

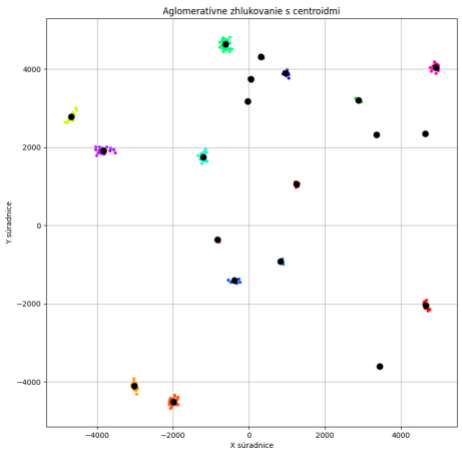
```python
def zhlukovanie_medoidy(bodiky, vzdialenost_od_stredu=500):  1 usage
    klastre = [[bod] for bod in bodiky] # pociatocne zhluky
    matica_vzdialenosti = vytvor_maticu_vzdialenosti(bodiky) # vytvorenie matice medzi vsetkymi bodmi
    cas_na_zaciatku = time.time()
    celkove_klastre = len(klastre)

    iteracia = 0
    while True:
        minimalna_vzdialenost = np.min(matica_vzdialenosti) # najdenie minimalnej vzdialenosti medzi dvoma zhlukmi
        # ak je minimálna vzdialenosť väčšia ako vzdialenost od stredu, ukonci sa cyklus
        if minimalna_vzdialenost > vzdialenost_od_stredu:
            break

        # ziskanie indexov dvoch zhlukov, ktore maju najmensiu vzdialenost
        i, j = np.unravel_index(np.argmin(matica_vzdialenosti), matica_vzdialenosti.shape)
        klastre[i].extend(klastre[j]) # zlucenie zhluku i a j
        del klastre[j]

        # aktualizacia matice
        matica_vzdialenosti = np.delete(matica_vzdialenosti, j, axis=0)
        matica_vzdialenosti = np.delete(matica_vzdialenosti, j, axis=1)

        # vypocet noveho medoidu
        medoid = min(klastre[i], key=lambda p: sum(euklidovska_vzdialenost(p, dalsi) for dalsi in klastre[i]))

        for k in range(len(klastre)):
            if k != i:
                matica_vzdialenosti[i, k] = matica_vzdialenosti[k, i] = euklidovska_vzdialenost(medoid, klastre[k][0])

        iteracia += 1
        if iteracia % 100 == 0:
            percenta_hotovo = 100 * (celkove_klastre - len(klastre)) / celkove_klastre
            aktualny_cas = time.time() - cas_na_zaciatku
            print(f"Iterácia {iteracia}: {percenta_hotovo:.2f}% hotovo "
                  f"Čas = {aktualny_cas:.2f} sekúnd")

    cas_na_konci = time.time() - cas_na_zaciatku
    hodiny, zostatok = divmod(cas_na_konci, 3600)
    minuty, sekundy = divmod(zostatok, 60)
    print(f"\nZhlukovanie na báze medoidov dokončené za {int(hodiny)} hodín, {int(minuty)} minút a {sekundy:.2f} sekúnd")

    uspesnost = uspesnost_zhlukovania(klastre, vzdialenost_od_stredu)
    print(f"Zhlukovanie na báze medoidov pre {len(bodiky)} bodov - Úspešnosť: {uspesnost:.2f}%")

    popisky = np.full(len(bodiky), -1, dtype=int)
    for idx, zhluk in enumerate(klastre):
        for bod in zhluk:
            idx_bodu = np.where((bodiky == bod).all(axis=1))[0]
            if idx_bodu.size > 0:
                popisky[idx_bodu[0]] = idx

    return popisky, klastre
```

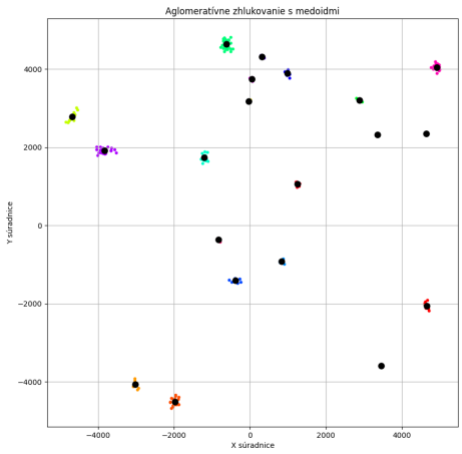
Testovanie

Pri testovaní kódu som postupne skúšala ako algoritmus funguje na rôznych počtoch bodov – najprv na menšom množstve napríklad 200 bodov, potom na 2000 bodoch a nakoniec na 10 000 bodov. Cieľom testovania je zistiť, aká úspešná je metóda zhlukovania – koľko percent zhlukov má priemernú vzdialenosť bodov od stredového bodu (centroidu alebo medoidu) menšiu alebo rovnakú ako 500.

Po skončení testovania sa zobrazia výsledky na 2D vizualizácií, kde každý zhluk má svoju farbu a stredový bod je označený čiernym bodom.

Testovanie na 200 náhodných extra bodov:

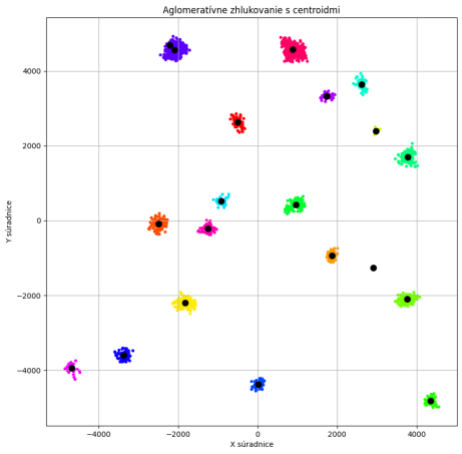


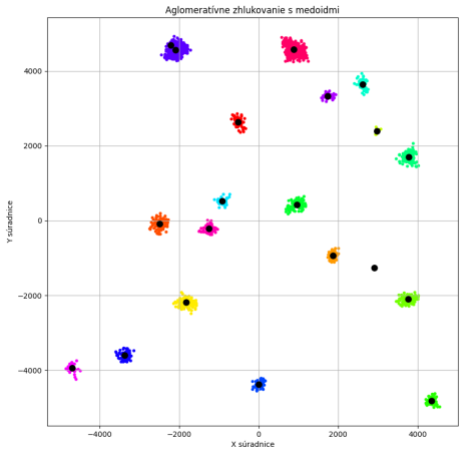



```
Zhlukovanie na základe centroidov dokončené za 0 hodín, 0 minút a 0.48 sekúnd

Úspešnosť zhlukovania: 100.00%
Zhlukovanie na základe centroidov pre 220 bodov - Úspešnosť: 100.00%
```

```
Zhlukovanie na báze medoidov dokončené za 0 hodín, 0 minút a 0.37 sekúnd

Úspešnosť zhlukovania: 100.00%
Zhlukovanie na báze medoidov pre 200 bodov - Úspešnosť: 100.00%
```

Testovanie na 2000 náhodných extra bodov:

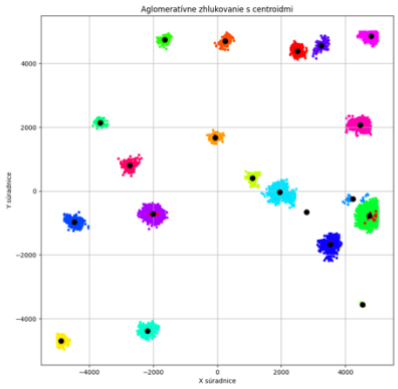




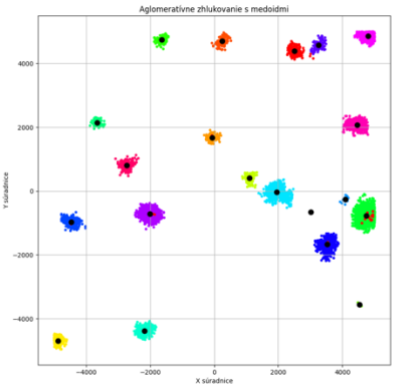

```
Zhlukovanie na základe centroidov dokončené za 0 hodín, 0 minút a 39.88 sekúnd

Úspešnosť zhlukovania: 100.00%
Zhlukovanie na základe centroidov pre 2020 bodov - Úspešnosť: 100.00%
```

```
Zhlukovanie na báze medoidov dokončené za 0 hodín, 1 minút a 2.00 sekúnd

Úspešnosť zhlukovania: 100.00%
Zhlukovanie na báze medoidov pre 2000 bodov - Úspešnosť: 100.00%
```

Testovanie na 10 000 náhodných extra bodov:




```
Zhlukovanie na základe centroidov dokoncene za 0 hodin, 29 minut a 47.90 sekund

Uspesne klastrovanie: 100.00%
Klastrovanie bolo uspesne na 100%
Zhlukovanie na základe centroidov pre 10020 bodov - Uspesnost: 100.00%
```




```
Zhlukovanie na báze medoidov dokončené za 0 hodin, 33 minut a 25.10 sekund

Uspesne klastrovanie: 100.00%
Klastrovanie bolo uspesne na 100%
Zhlukovanie na báze medoidov pre 10000 bodov - Uspesnost: 100.00%
```

Zhodnotenie a záver

Na záver porovnávam úspešnosť oboch verzií algoritmu primárne na základe vizualizácií.

Úspešnosť zhlukovača sa síce vypíše, ale stále bola podozrivo 100%, čo pri menších počtoch bodov, ako je 200 alebo 2000 je možné, ale pri vyšších počtoch by to nemuselo počítať správne. Preto je možné, že sa v kóde nachádza chyba, pretože pri väčších počtoch bodov by výsledky nemuseli byť vždy také ideálne.

Výsledky ukazujú ako dobre algoritmus rozdelil body do logických zhlukov s požadovanou vzdialenosťou od stredu, a umožnia nám rozhodnúť, ktorá verzia algoritmu je vhodnejšia pre dané použitie. Aj keď mám pocit, že pri týchto menších počtoch sme dostali skoro identické výsledky.