Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Umelá inteligencia

Zadanie 4 - Klastrovanie

Jakub Hrnčár
AIS ID: 110 800, ak. rok 2021/2022

1. Úvod do zadania

V zadaní som pracoval na jednoduchom zhlukovači, ktorí má za úlohu rozdeliť body v 2D boli do zhlukov - klastrov. Táto technika sa používa v dátovej vede veľmi často, preto je toto zadanie zaujímavé. Mali sme použiť a porovnať 4 algoritmy klastrovania: k-means-centroid, k-means-medoid, divizívne a aglomerativne klastrovanie. Každý algoritmus má svoje vlastné výhody a nevýhody, ktoré vieme porovnať na základe času a úspešnosti - koľko klastrov má priemernú vzdialenosť od stredu menšiu ako 500? .

Zadanie som vypracoval v jazyku Python 3.9, v prostredí PyCharm, na Windows 10. Aglomerativne klastrovanie som sa rozhodol vynechať.

2. Stručný opis algoritmov

Reprezentácia klastrov

Klaster reprezentujem ako niekoľko dimenzionálne pole vo formáte: [[body], [stred], priemerná vz. od stredu]. Tento prístup mi umožňuje mať všetky informácie na jednom mieste a ľahko dostupné.

Generovanie bodov

Body vygenerujem pred spustením algoritmov, aby sa všetky algoritmy vykonali na rovnakých bodoch. Najprv vygenerujem 20 bodov so súradnicami v rozmedzí <-4900, 4900>, a neskôr vygenerujem N bodov, ktoré sa offsetom líšia od niektorého z už vytvorených bodov. Tento prístup zabezpečuje rozdelenie dát do klastrov, ktoré môžem, neskôr skúmať.

Manhattanská vzdialenosť

Je to odľahčená forma klasickej Pytagorejskej vzdialenosti, ktorá je takmer rovnako presná a nepoužíva mocniny a odmocniny, preto je niekoľko krát rýchlejšia. Jej vzorec je |x1 - x2| + |y1 - y2|.

K-Means

Algoritmus som vypracoval v 2 podobách (stred centroid, stred medoid), avšak používam ho aj pri divizívnom klastrovaní. Jeho zmysel je rozdeliť zadané pole bodov na K klastrov. Prvý krok algoritmu je nemenný, najprv sa vytvoria klastre s inicializačnými bodmi, ktoré sú náhodne vybrané. Tieto body sú prvé body v klastrov a zároveň sú aj stredom klastru.

```
while len(centroids) != k:
    ch = random.choice(points)
    if ch not in centroids:
        centroids.append(ch)
for c in centroids:
    clusters.append([[c], c])
```

Ďalším krokom je iterovanie cez všetky body, ktoré priradím do ich najbližšieho klastru a po priradení všetkých bodov sa znovu prepočítajú stredy a vymaže sa obsah klastrov, nastáva ďalšia iterácia. Iterovanie končí, keď stredy sa 2 krát po sebe nezmenia a zostanú na tom istom mieste.

```
def get_centroid(cluster):
    x_sum = 0
    y_sum = 0
    for p in cluster:
        x_sum += p[0]
        y_sum += p[1]
    return [x_sum // len(cluster), y_sum // len(cluster)]
```

Stred centroid

Centroid je fiktívny bod, ktorý predstavuje presný priemerný stred klastru.

```
def get_centroid(cluster):
    x_sum = 0
    y_sum = 0
    for p in cluster:
        x_sum += p[0]
        y_sum += p[1]
    return [x_sum // len(cluster), y_sum // len(cluster)]
```

Stred medoid

Medoid je reálny bod, ktorý má najmenšiu celkovú vzdialenosť od všetkých bodov. Je teda nutné manuálne prejsť všetky body v klastri a nájsť minimum, čo zaberá čas.

```
def get_medoid(cluster):
    min = 0
```

```
    min_index = -1
    for p1 in range(len(cluster)):
        dist = 0
        for p2 in range(len(cluster)):
            if p1 == p2:
                continue
            dist += man_dist(cluster[p1], cluster[p2])
        if min_index == -1:
            min = dist
            min_index = 0
        if dist < min:
            min_index = p1
            min = dist
    return cluster[min_index]
```

Divizívne klastrovanie

Tento algoritmus vníma všetky dáta zo začiatku ako v jednom veľkom klastri, z ktorý delí na K klastrov. Vždy rozdelí taký klaster, ktorý má najhoršiu priemernú vzdialenosť medzi stredom a bodom. Algoritmus je veľmi jednoduchý, pretože tu vieme veľmi pekne využiť K-Means-centroid algoritmus na delenie 1 klastru na 2.

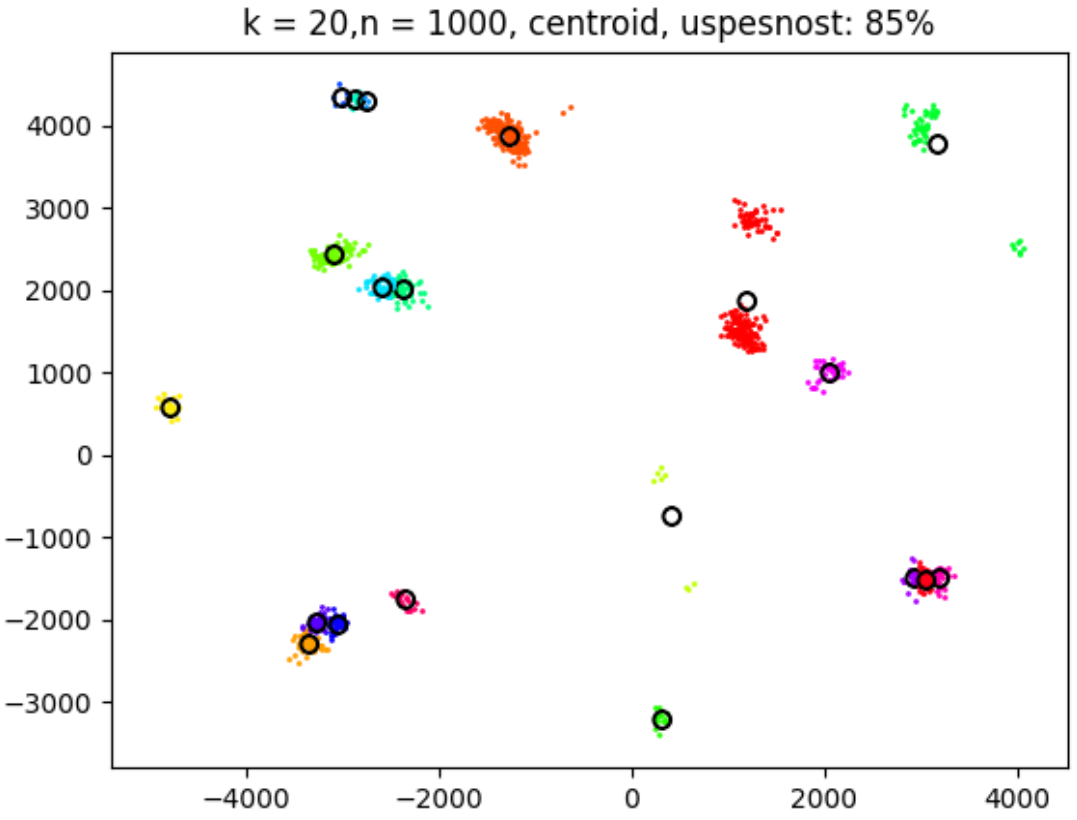
```
def divisive(k, points: list):
    n = 1
    clusters = k_means_centroid(2, points, False)
    n += 1
    while n != k:
        max = clusters[0][2]
        index_max = 0
        for c in clusters:
            next = c[2]
            index = clusters.index(c)
            if next > max:
                max = next
                index_max = index

        new_clusters = k_means_centroid(2,
```

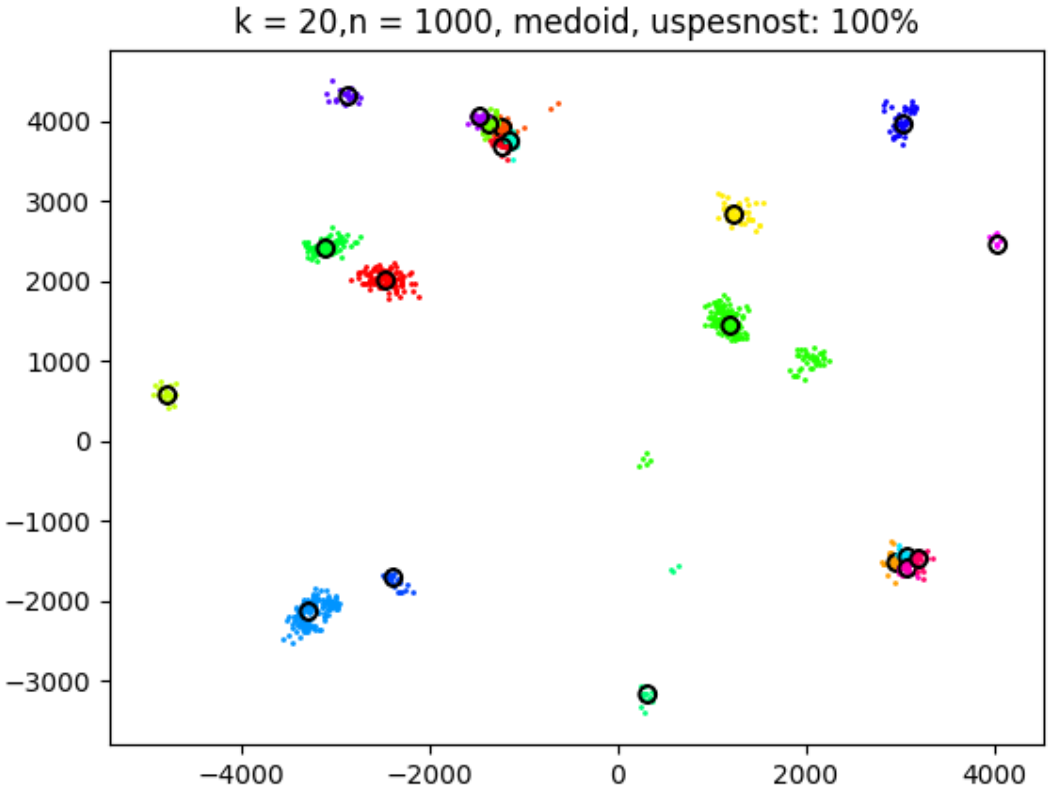
```
clusters[index_max][0], False)
        clusters.pop(index_max)
        clusters.append(new_clusters[0])
        clusters.append(new_clusters[1])
        n += 1
        new_clusters.clear()
    return clusters
```

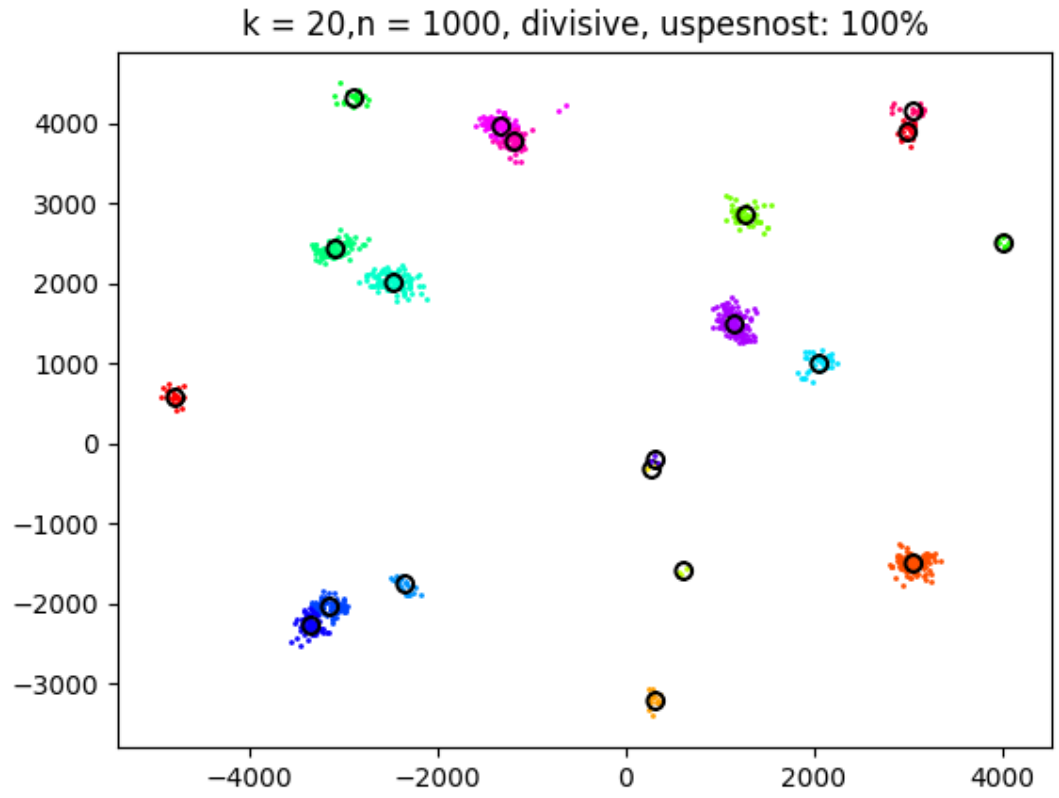
3. Testovanie programu

Celý súbor s vykonanými testami je dostupný v prílohe, sem vložím iba pár testov, ktoré predstavujú ukážku:

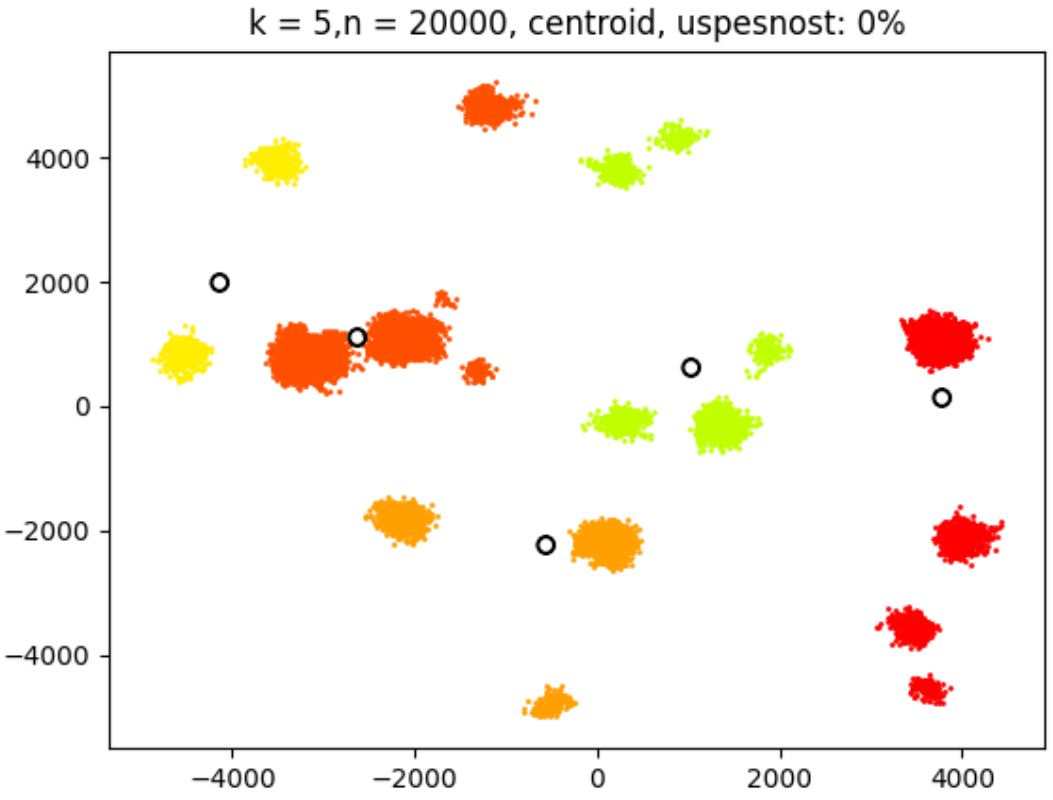
Veľké K, malé N

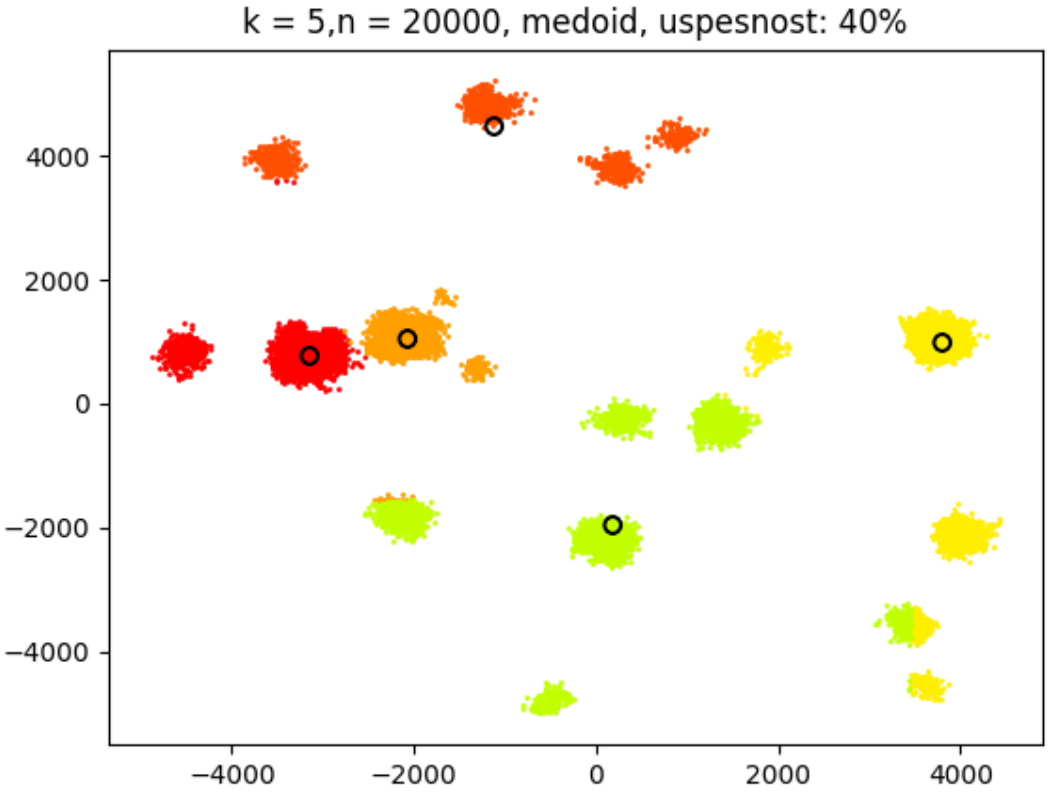


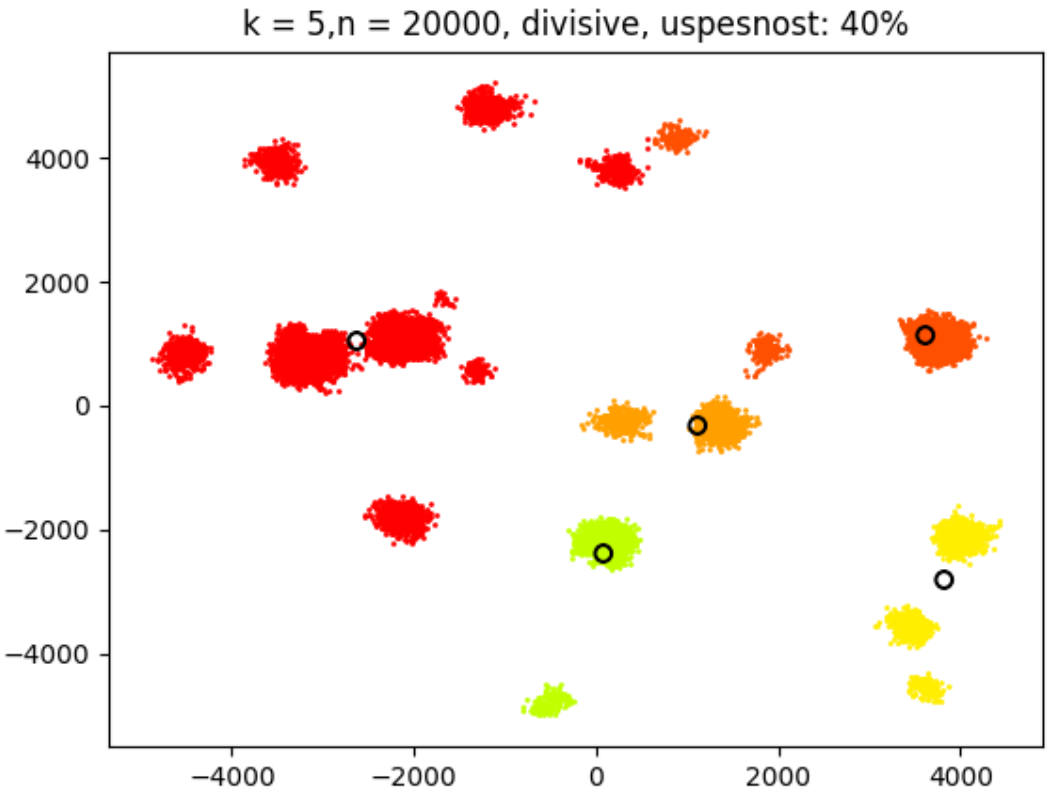

k = 20,n = 1000, medoid, uspesnost: 100%
4000
3000
2000
1000
0
−1000
−2000
−3000
−4000
−2000
0
2000
4000


k = 20,n = 1000, divisive, uspesnost: 100%
4000
3000
2000
1000
0
−1000
−2000
−3000
−4000
−2000
0
2000
4000

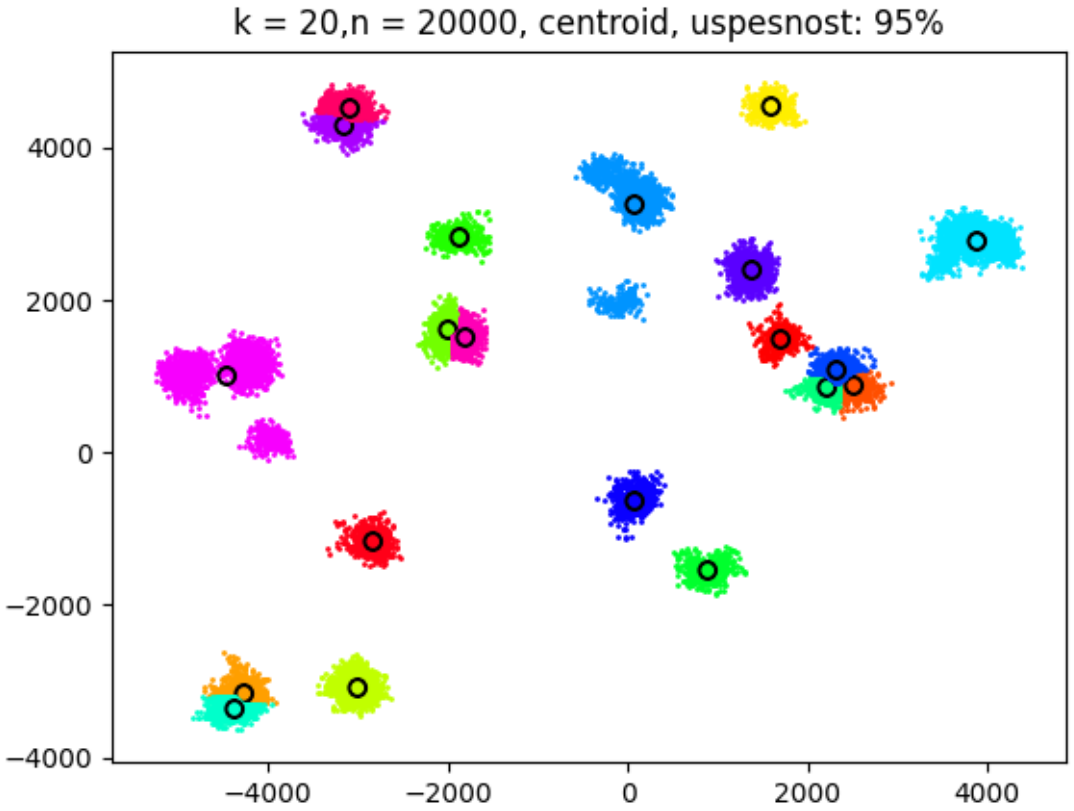
Malé K, veľké N

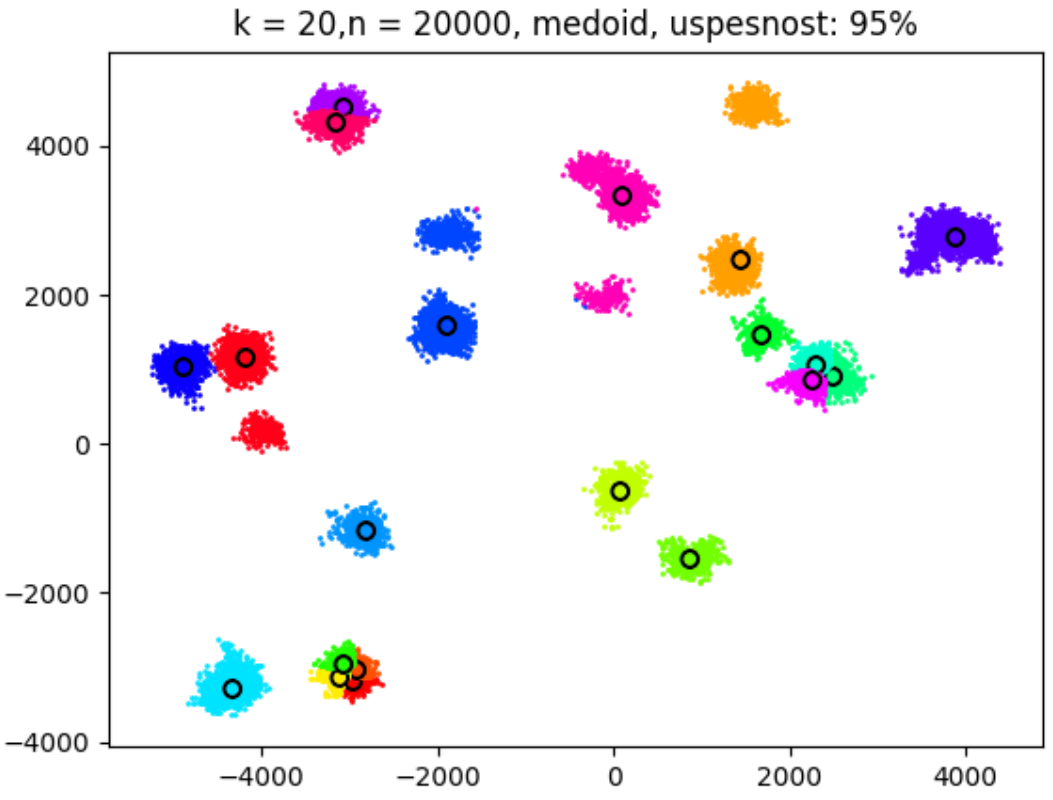


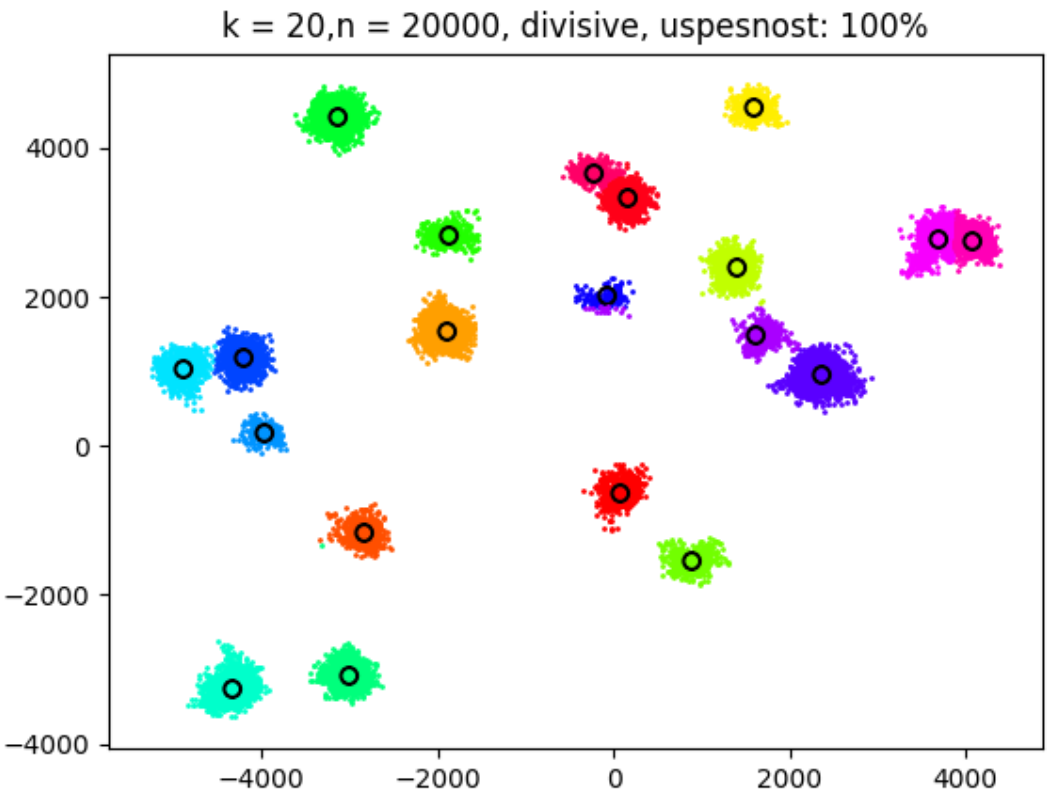




Veľké K, veľké N




k = 20,n = 20000, medoid, uspesnost: 95%
4000
2000
0
−2000
−4000
−4000
−2000
0
2000
4000


k = 20,n = 20000, divisive, uspesnost: 100%
4000
2000
0
−2000
−4000
−4000
−2000
0
2000
4000

4. Záver

Zvyšné testy sa nachádzajú v prílohe.

Centroid - Jeho nájdenie je veľmi rýchle, čo zabezpečuje veľmi veľkú rýchlosť algoritmu v každej situácií. Avšak, pri malom počte klastrov má tento prístup veľmi malú úspešnosť, pravdepodobne najhoršiu zo všetkých algoritmov.

Medoid - Pri malom počte bodov je algoritmus podobne rýchly ako centroid, avšak exponenciálne sa zvyšuje jeho čas s pribúdajúcim počtom bodov. Má ale vysokú úspešnosť, avšak nie najlepšiu.

Divizívne - Algoritmus je pravdepodobne najlepší, čo sa týka úspešnosť/čas. Kvôli použitiu centroidu je extrémne rýchly a keďže redukuje náhodnosť kvôli deleniu zlých klastrov, tak zabezpečuje veľmi vysokú úspešnosť aj pri malom počte bodov/klastrov.

Aglomerativne - Má pravdepodobne najlepšiu úspešnosť, avšak trvá extrémne množstvo času a má extrémne pamäťové nároky. Preto som sa rozhodol tento algoritmus vynechať.