ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY

GEOINFORMATIKA

Úloha 4 – Clusterizační algoritmy

Rok	Semestr	Skupina	Vypracovali	Datum
2025/2026	ZS	2	František Gurecký Vítek Veselý	5.12.2025

1 Zadání

Vytvořte vlastní implementaci algoritmu shlukování k-means. Funkci volejte na vygenerované alespoň tři shluky ve 2D. Porovnejte vlastní implementaci s využitím již vytvořené funkce *kmeans*. Bonusovou úlohou je poté rozšířit implementaci do obecně N-dimensionálního prostoru.

2 Popis problému

Shlukování jsou metody identifikace skupin o podobných vlastnostech. Při zadaném počtu hledaných shluků by měly tyto metody nalézt takové řešení, kde jsou si body ve shlucích co nejpodobnější a zároveň jsou shluky navzájem co nejvíc odlišené. Obecně se snaží nalézt strukturu v různých datech, která nemusí být na první pohled patrná lidským okem.

3 Popis algoritmů

3.1 Generování dat

Nejprve bylo nutné vytvořit funkci, která bude generovat shluky. Již při první implementaci bylo myšleno na obecnost a funkce tedy přijímá jako parametry dimenzi a počet shluků, které má vytvořit. Samotná implementace v jazyce Python vypadá takto:

```
def gen_pts(dimensions, cluster_count):
points = []
for i in range(cluster_count):
    # Random sized clusters
    cluster_points = random.randint(10, 200)

    # Random offset from center
    offset = [0] * dimensions
    for o in range(dimensions):
        offset[o] = (random.random() - 0.5) * 50
    # Generate cluster points
    for j in range(cluster_points):
        point = []
        for k in range(dimensions):
            point.append((random.random() - 0.5) * 10 + offset[k])

        points.append(point)

return points
```

Tím získáme pole bodů o obecně *dimensions* dimenzích shluknutých do *cluster_count* shluků. Separaci lze ovlivnit upravením koeficientů, kterou násobíme výstup funkce *random*, ta vrací desetinné číslo mezi 0 a 1, které následně přenásobíme a posuneme, aby byl interval symetrický kolem nuly. Tyto parametry budou testovány ve výsledcích.

3.2 Výpočet vzdálenosti

Pro implementaci k-means je nutné také nalézt, kterému centroidu shluků má bod nejblíže, byla tedy implementována jednoduchá funkce pro výpočet vzdálenosti, která očekává souřadnice z námi dané datové struktury a snese obecný počet dimenzí.

Její implementace v jazyce Python pak vypadá takto:

```
def distance(point_a, point_b):
    dim = len(point_a)
    sum_delta_sq = 0

    # Calculate the sum of deltas squared
    for i in range(dim):
        sum_delta_sq += (point_a[i] - point_b[i]) ** 2

    # The distance is the square root
    return sum_delta_sq ** 0.5
```

3.3 K-Means

Tato funkce očekává na vstupu body v námi definovaném formátu a počet shluků, které má vytvořit. Po výpočtu vrací pole náležitostí k centroidům a jejich souřadnice indexované stejně. Inicializace byla zvolena náhodná, v prvních krocích je tedy z vložených bodů určena dimenze na jejímž základě je pak vygenerovaný příslušný počet centroidů shluků náhodně.

Dále byla inicializována iterace tím, že bylo vytvořeno prázdné pole náležení centroidům a koncová podmínka je, že se toto pole po průběhu cyklu nezmění, tedy již se nebude měnit, jaké body centroidům patří a tedy byly vypočteny finální shluky.

V cyklu je poté hledán nejbližší centroid ke každému bodu a jeho index je uložen do pole náležení.

V dalším cyklu jsou přepočítány souřadnice centroidů jako průměrné souřadnice všech bodů, které mu nyní náleží.

Ke konci iterace kontrolujeme podmínku ukončení.

Takto byl algoritmus popsán v poskytnuté prezentaci [1].

V jazyce *Python* je implementace:

```
def k_means(points, cluster_count):
    print("Running k-means")
    dim = len(points[0])
    point_count = len(points)

    centroids = []

    # Generate centroids randomly
    for i in range(cluster_count):
        centroid = []
        for j in range(dim):
            # Random float from -20 to 20 to match the points generation
            centroid.append((random.random() - 0.5) * 40)

        centroids.append(centroid)

    print(centroids)
    # Initialize the iteration
    done = False
    belongs_to = [None] * point_count
    iteration_count = 0

    while not done:
        iteration_count += 1
        print(f"Starting {iteration_count}. iteration")
        done = True

        # Storing the old belongs_to for comparison as the terminal condition
        old_belongs_to = belongs_to.copy()

        # Find closest cluster for all points
        for i in range(point_count):
            nearest = 9999999
            for j, center in enumerate(centroids):
                dist = distance(points[i], center)
                if dist < nearest:
                    belongs_to[i] = j
                    nearest = dist
```

```
    # Recalculate the centroids
    for i, center in enumerate(centroids):
        new_center = [0] * dim
        children = 0

        for j in range(point_count):
            # If this point belongs to this centroid, add it to the sum
            if i == belongs_to[j]:
                children += 1

                for k in range(dim):
                    new_center[k] += points[j][k]

        # We've passed all the points for the centroid,
        # divide the coordinates by the belonging count
        for k in range(dim):
            new_center[k] /= max(children, 1)

        # Update the centroids
        centroids[i] = new_center

    if old_belongs_to != belongs_to:
        done = False

# After the iteration is over,
# return the belonging array and the final centroids
return belongs_to, centroids
```

Tato implementace odpovídá popisu.

3.4 Referenční funkce

Je nutné naši implementaci s již stabilní funkcí porovnat. K tomu byla zvolena třída *KMeans* z balíčku Scikit-learn [2]. Tato třída má metody pro stanovení parametrů výpočtů a také metodu pro samotný výpočet. Pro zjednodušení implementace byla třída s jejími metodami obalena funkcí, která vytvoří její instanci, nastaví parametry a vrátí výsledky stejně, jako námi vytvořená implementace.

V Pythonu pak vypadá takto:

```
def k_means_scikit(cluster_count, init, points):
    # First translate the point data to np.array
    data = np.array(points)
    kmeans_instance = KMeans(
        n_clusters=cluster_count,
        random_state=0,
        n_init="auto",
        init=init
    )
    kmeans = kmeans_instance.fit(data)

    # Prepare the data as simple lists and return them
    # Similar to our belongs_to array of indices
    belongs_to = kmeans.labels_.tolist()
    # The centroids of clusters
    centroids = kmeans.cluster_centers_.tolist()

    return belongs_to, centroids
```

Nastavení třídy odpovídá náhodné inicializaci centroidů.

3.5 Vizualizace

Na závěr je vhodné výsledky také vizualizovat graficky a nejen numericky. K tomu byla vytvořena funkce *vizualize*, která pomocí balíčku *matplotlib* zobrazí grafy z předaných dat. Na vstupu očekává bodová data, pole náležení shlukům a souřadnice centroidů. Tato funkce nic nevrací, pouze data upraví pro své potřeby a vizualizuje, pokud se jedná o 2D data.

V Pythonu:

```
def visualize(points, belongs_to, centroids):
    # Get the same coordinates to list, if the dimension is two
    if len(points[0]) > 2:
        print("Unable to visualize in more than 2D")
        return

    cluster_count = len(centroids)

    # Plot points for the cluster
    for i in range(cluster_count):
        # Separate lists for x and y
        cluster_x = []
        cluster_y = []

        for j in range(len(points)):
            if belongs_to[j] == i:
                cluster_x.append(points[j][0])
                cluster_y.append(points[j][1])

        plt.scatter(cluster_x, cluster_y, s=10)

    # Plot centroids
    cent_x = [c[0] for c in centroids]
    cent_y = [c[1] for c in centroids]
    plt.scatter(cent_x, cent_y, color="black", s=200, marker="X")

    plt.title("K-Means Clustering Visualization")
    plt.show()
```

Jak vypadají grafy je možné vidět v sekci s výsledky 4.

3.6 Vyhodnocení

Pro vyhodnocení bylo provedeno 100 běhů a byly spočteny statistiky, které popisují, jak se referenční a námi určené centroidy podobají. Konkrétně byly pro každý běh spočteny maximální a minimální odchylky centroidů, průměrná vzdálenost centroidů, směrodatná odchylka vzdáleností a rozdíly v souřadnicích. Tyto statistiky byly na závěr shrnuty do průměrného průměru, směrodatné odchylky průměru, průměrné směrodatné odchylky a směrodatné odchylky směrodatných odchylek. Dále také průměrnou maximální vzdálenost, maximální vzdálenost centroidů ze všech běhů a minimální vzdálenost centroidů ze všech běhů.

V Pythonu poté:

```
def calculate_stats(centroids, centroids_ref):
    centroids = np.array(centroids)
    centroids_ref = np.array(centroids_ref)

    # Pair the closest centroids
    distances = []
    diffs = []

    for c in centroids:
        # Find the minimal distance
        d = np.linalg.norm(centroids_ref - c, axis=1)
        i = np.argmin(d)

        distances.append(d[i])
        diffs.append(c - centroids_ref[i])

    distances = np.array(distances)
    diffs = np.array(diffs)

    return {
        "mean_distance": float(np.mean(distances)),
        "std_distance": float(np.std(distances)),
        "max_distance": float(np.max(distances)),
        "min_distance": float(np.min(distances)),
        "diffs": diffs
    }
```

Souhrnné statistiky poté spočteme v Pythonu:

```
def summarize_runs(stats_total):
    # Extract lists
    means = np.array([s["mean_distance"] for s in stats_total])
    stds  = np.array([s["std_distance"] for s in stats_total])
    maxs  = np.array([s["max_distance"] for s in stats_total])
    mins  = np.array([s["min_distance"] for s in stats_total])

    return {
        "mean_of_means": float(np.mean(means)),
        "std_of_means": float(np.std(means)),

        "mean_of_stds": float(np.mean(stds)),
        "std_of_stds": float(np.std(stds)),

        "mean_of_maxs": float(np.mean(maxs)),
        "max_of_maxs": float(np.max(maxs)),
        "min_of_mins": float(np.min(mins))
    }
```

A voláme v cyklu tatko:

```
stats_total = []

    for _ in range(100):
        belongs_to, centroids = k_means(points, cluster_count)
        belongs_to_sc, centroids_sc = k_means_scikit(cluster_count,
                                                     "random",
                                                     points)
        stats_total.append(calculate_stats(centroids, centroids_sc))

    summary_stats = summarize_runs(stats_total)

    print("Summary of 100 runs:")
    for k, v in summary_stats.items():
        print(f"{k}: {v}")
```

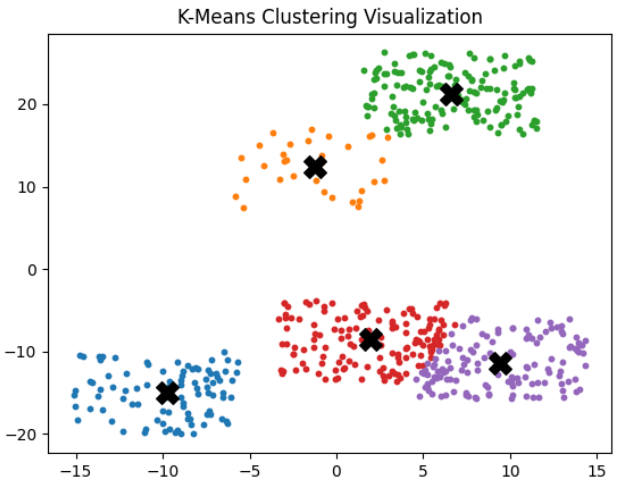
4 Výsledky

Generování shluků probíhá vždy náhodně, otestujme tedy různá nastavení rozlišení shluků. Nejprve budou zobrazeny statistiky ze 100 běhů při základním nastavení a zobrazena vizualizace ukázkového běhu s tímto nastavením.

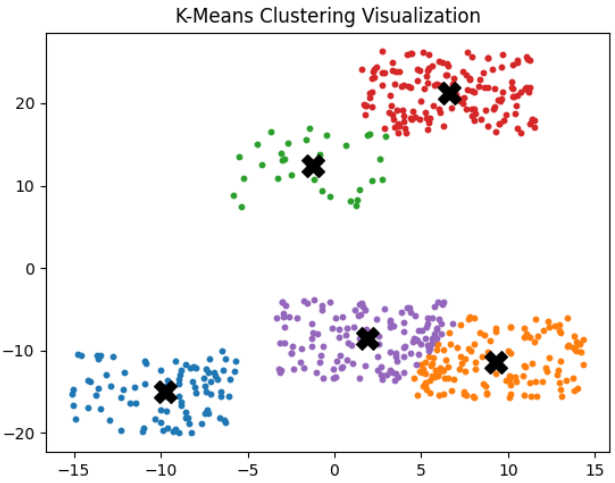
Jak bylo zmíněno, pro všechny nejkratší cesty byla spočtena všechna ohodnocení, výsledky jsou sepsány v tabulkách níže.

Statistika	Hodnota [-]
Průměr průměrných vzdáleností	**9.6451**
Sm. odchylka průměrných vzdáleností	2.3194
Průměr směrodatných odchylek	8.9170
Sm. odchylka směrodatných odchylek	0.5835
Průměr maximálních vzdáleností	19.8272
Největší maximální vzdálenost	**19.8272**
Nejmenší minimální vzdálenost	3.55×10^{-15}

Tabulka 1: Souhrn statistik pro 100 běhů k-means se základním nastavením



Obrázek 1: Naše implementace pro základní nastavení



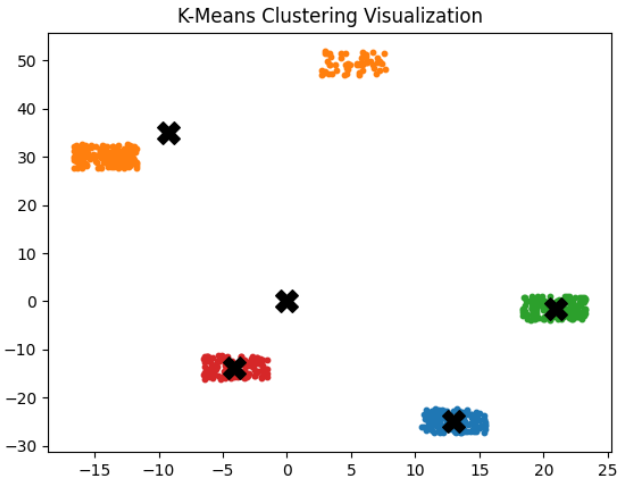
Obrázek 2: Scikit implementace pro základní nastavení

Pro 100 běhů je směrodatná odchylka relativně malá, vizuálne si obě implementace relativně odpovídají. Statistiky se však zdají špatné, průměrná vzdálenost *9,6* je poměrně velká hodnota, vzhledem ke generovanému rozsahu.

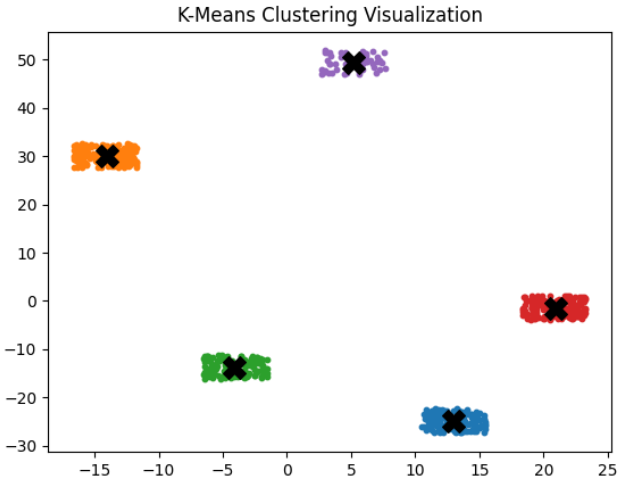
Dále zkusme zvětšit oddělení clusterů, předpoklad je, že se hodnoty zmenší.

Statistika	Hodnota [-]
Průměr průměrných vzdáleností	**13.4765**
Sm. odchylka průměrných vzdáleností	4.4589
Průměr směrodatných odchylek	12.6445
Sm. odchylka směrodatných odchylek	1.8705
Průměr maximálních vzdáleností	28.8474
Největší maximální vzdálenost	**29.4129**
Nejmenší minimální vzdálenost	0.00

Tabulka 2: Souhrn statistik pro 100 běhů k-means s větším odstupem shluků



Obrázek 3: Naše implementace s větším odstupem shluků



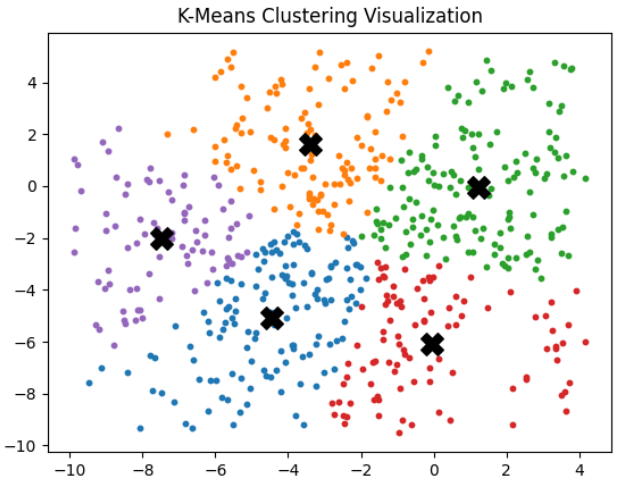
Obrázek 4: Scikit implementace s větším odstupem shluků

Předpoklad se nenaplnil. Zvětšením odstupů se zvýšila také závislost na náhodnosti inicializace, naše implementace poté rychle nalezne lokální extrém a ukončí iteraci před globálním extrémem, ve kterém se nachází řešení, jako to dělá vzorová implementace.

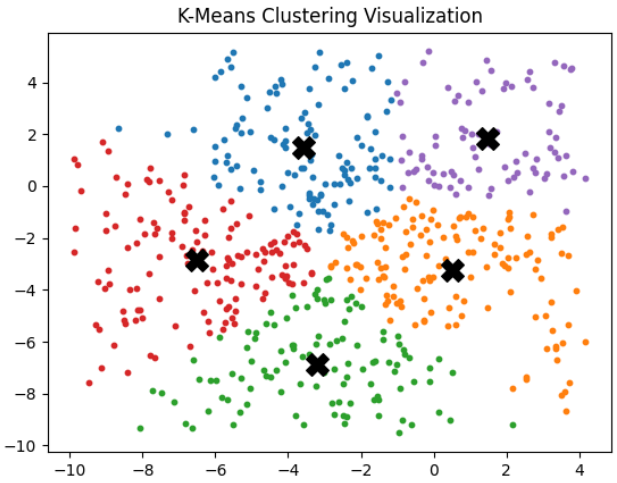
Dále zkusme naopak snížit rozlišení mezi shluky, tedy menší offset a větší rozptyl uvnitř shluků. Předpoklad je nyní, že se zhorší rozlišení mezi shluky a vznikne ještě vyšší závislost na inicializaci středů shluků.

Statistika	Hodnota [-]
Průměr průměrných vzdáleností	**1.1323**
Sm. odchylka průměrných vzdáleností	0.9512
Průměr směrodatných odchylek	0.6954
Sm. odchylka směrodatných odchylek	0.5579
Průměr maximálních vzdáleností	2.2089
Největší maximální vzdálenost	**4.8659**
Nejmenší minimální vzdálenost	2.22×10^{-16}

Tabulka 3: Souhrn statistik pro 100 běhů k-means s větším odstupem shluků



Obrázek 5: Naše implementace s menším odstupem shluků



Obrázek 6: Scikit implementace s menším odstupem shluků

Statistiky se zdají lepší, protože je menší rozsah, ve kterém jsou bodová data generována. Vizuálně si jsou centroidy odlišnější a tvoří shluky rozdílnější než v předchozích nastaveních. Předpoklad se tedy naplnil.

Pro ověření výpočtu ve více dimenzích byl proveden také běh pro 8D data, níže jsou uvedeny souhrnné statistiky pro výchozí nastavení shluků.

Statistika	Hodnota [-]
Průměr průměrných vzdáleností	**4.0177**
Sm. odchylka průměrných vzdáleností	5.1118
Průměr směrodatných odchylek	4.9268
Sm. odchylka směrodatných odchylek	5.8603
Průměr maximálních vzdáleností	12.2710
Největší maximální vzdálenost	**35.3758**
Nejmenší minimální vzdálenost	9.64×10^{-15}

Tabulka 4: Souhrn statistik pro 100 běhů k-means se základním nastavením pro 8D data

Z výsledků se zdá, že pro více dimenzí se obecně odchylky sníží, ale extrém vzdálenosti dvou nejbližších středů shluků je větší. Vícedimenzionální data mohou představovat například multispektrální data z dálkového průzkumu Země, je tedy nutné, aby *K-Means* fungovalo i pro N-dimensionální data.

5 Závěr

Podařilo se pochopit podstatu clusterizačního algoritmu *K-Means*, tento algoritmus byl implementován a testován pro různá nastavení v porovnání s referenční implementací [2].

Z výsledků se dá vyvodit, že velmi záleží na inicializaci shluků, jelikož je prováděno náhodně. Toto by se dalo vylepšit lepším odhadem, například rovnoměrným rozdělením středů mezi minimum a maximum vstupních hodnot, což dělá algoritmus *K-Means++*.

Počet iterací je nepřímo úměrný odlišnosti shluků, pro málo odlišná data je nutno iterovat vícekrát.

Úlohu se podařilo vyřešit také obecně pro N-dimensionální data.

Implementaci je možné otestovat v repozitáři projektu.

Celkově se plně podařilo splnit zadání.

V Praze dne 5.12.2025

František Gurecký
Vítek Veselý

Literatura

[1] Markéta Potůčková. *Shluková analýza*. 2025.

[2] *KMeans*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html` (cit. 06. 12. 2025).