Zadanie 3a
Klasifikácia a zhlukovanie

Laura Fulajtárová

Fakulta informatiky a informačných technológií STU

xfulajtarova@stuba.sk

ID: 120782

UI – Utorok 14:00

1.12.2023

Obsah

1 Parametre môjho počítača

- Processor AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
- Installed RAM 16.0 GB (13.9 GB usable)
- System type 64-bit operating system, x64-based processor

2 Implementačné prostredie

Rozhodla som sa použiť Python ako implementačné prostredie kvôli jeho jednoduchosti, prirodzenému čitateľnému kódu, rozsiahlemu ekosystému knižníc a funkcií, čo umožňuje rýchle a efektívne vývojové práce. Python je ideálny pre implementáciu rôznych algoritmov vrátane evolučných algoritmov, pričom minimalizuje komplikácie spojené s programovaním, takže môžem sústrediť svoju pozornosť na samotný algoritmus a jeho správne fungovanie.

3 Prehľad súborov

3.1 main.py

Môj program kombinuje logickú stránku s estetickým zobrazením výsledkov a je navrhnutý pre jednoduchosť a zrozumiteľnosť. Hlavné funkcie zahŕňajú main(), ktorá riadi celý proces od generovania po vizualizáciu bodov, make_dots() a random_dot() na tvorbu unikátnych bodov, classify() na aplikáciu KNN algoritmu, make_filler_dots() na lepšiu vizualizáciu a visualize_dots() na zobrazenie výsledkov.

3.2 testing.py

V súbore testing.py testujem program, pričom start() umožňuje nastavenie parametrov a make_graphs() slúži na vykreslenie dát do grafov, čo mi poskytuje prehľad o úspešnosti a efektívnosti programu.

4 Riešený problém

Moja úloha je zostaviť klasifikátor bodov na dvojrozmernej ploche o rozmeroch AxB, kde sú rozmery plochy v intervale od -5000 do 5000. Na tejto ploche sa nachádza 20 predgenerovaných bodov v štyroch farbách - červenej, zelenej, modrej a fialovej. Tieto body majú nasledujúce súradnice:

- Červené body: [-4500, -4400], [-4100, -3000], [-1800, -2400], [-2500, -3400], [-2000, -1400]
- Zelené body: [+4500, -4400], [+4100, -3000], [+1800, -2400], [+2500, -3400], [+2000, -1400]
- Modré body: [-4500, +4400], [-4100, +3000], [-1800, +2400], [-2500, +3400], [-2000, +1400]
- Fialové body: [+4500, +4400], [+4100, +3000], [+1800, +2400], [+2500, +3400], [+2000, +1400]

Každý bod má unikátne súradnice. Moja úloha spočíva v náhodnom vygenerovaní ďalších bodov a ich následnom klasifikovaní pomocou KNN algoritmu, kde hodnotu k budem testovať pre 1, 3, 7 a 15. Body budú generované s tým, že nový bod by mal byť v inom farebnom rozsahu. Generovanie bodov prebieha nasledovne:

- Červené body by mali byť generované s 99% pravdepodobnosťou s X < +500 a Y < +500.
- Zelené body by mali byť generované s 99% pravdepodobnosťou s X > -500 a Y < +500.
- Modré body by mali byť generované s 99% pravdepodobnosťou s X < +500 a Y > -500.
- Fialové body by mali byť generované s 99% pravdepodobnosťou s X > -500 a Y > -500.

Zvyšné 1% bodov je generovaných v celom dostupnom priestore. Na určenie správnej triedy každého bodu použijem KNN algoritmus, ktorý porovnám s farebnou triedou vygenerovaného bodu, aby som zhodnotila úspešnosť môjho klasifikátora. Okrem toho, prázdne miesta na ploche musím klasifikovať tiež pomocou môjho klasifikátora. Vytvorené plochy následne vizualizujem, aby som mohla lepšie posúdiť výsledky klasifikácie.

5 KNN algoritmus

Algoritmus K-najbližších susedov (KNN) je základným algoritmom strojového učenia, ktorý je založený na technike učenia pod dohľadom. KNN klasifikuje nové dáta na základe ich podobnosti s existujúcimi dátami. Tento algoritmus je

vhodný hlavne pre klasifikačné úlohy a funguje tak, že uchováva všetky dostupné dáta a pri klasifikácii nových dátových bodov sa spolieha na meranie ich podobnosti s už známymi kategóriami.

Princíp fungovania KNN možno vysvetliť pomocou nasledujúceho postupu:

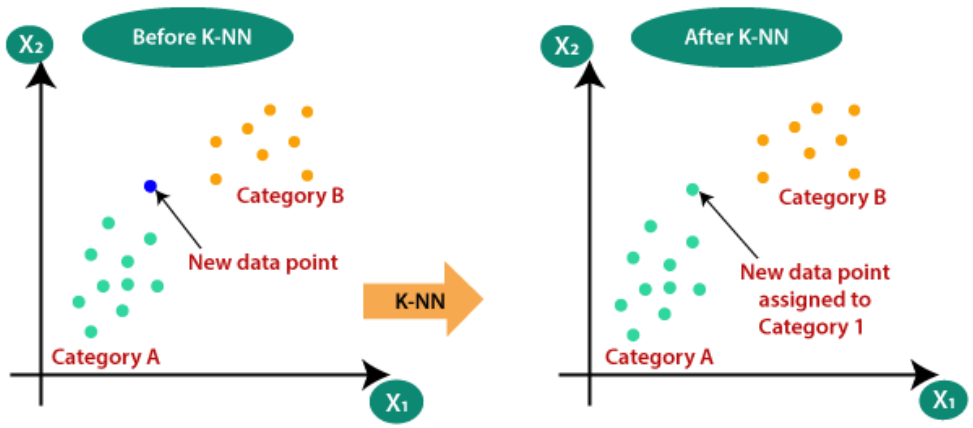
1. Vyber počet susedov K.
2. Vypočítaj euklidovskú vzdialenosť pre K susedov.
3. Vyber K najbližších susedov podľa vypočítanej euklidovskej vzdialenosti.
4. Spomedzi týchto susedov spočítaj počet dátových bodov v každej kategórii.
5. Nový dátový bod priraď do kategórie, v ktorej je najviac susedov.
6. Model je pripravený.

Výhody algoritmu KNN:

- Jednoduchosť implementácie.
- Odolnosť voči šumivým tréningovým dátam.
- Efektívnejší pri veľkom množstve tréningových dát.

Nevýhody algoritmu KNN:

- Potreba určiť hodnotu K, čo môže byť niekedy zložité.
- Vysoké výpočtové náklady kvôli vypočítaniu vzdialenosti medzi dátovými bodmi pre všetky tréningové vzorky.



Obrázok 1 https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

6 Reprezentácia údajov

Vo svojom programe na prácu s bodmi reprezentujem všetky údaje buď vo forme zoznamov (listov) alebo slovníkov, čo umožňuje jednoduchú a efektívnu manipuláciu s dátami. Zoznamy sú ideálne na ukladanie bodov a ich atribútov v poradí, v akom boli generované alebo spracované, zatiaľ čo slovníky poskytujú rýchly prístup k dátam na základe kľúčov, ako sú farba alebo kategória, uľahčujúc tak triedenie a vyhľadávanie špecifických skupín bodov. Navyše, na reprezentáciu súradníc X a Y každého bodu používam štruktúru tuple, ktorá zabezpečuje, že hodnoty súradníc zostanú konzistentné a nezmenené, čím sa znižuje riziko chýb pri spracovaní.

```
dots = {
                "red": [
                    (-4500, -4400),
                    (-4100, -3000),
                    (-1800, -2400),
                    (-2500, -3400),
                    (-2000, -1400),
                ],
                "green": [
                    (4500, -4400),
                    (4100, -3000),
                    (1800, -2400),
                    (2500, -3400),
                    (2000, -1400),
```

```
            ],
            "blue": [
                (-4500, 4400),
                (-4100, 3000),
                (-1800, +2400),
                (-2500, +3400),
                (-2000, +1400),
            ],
            "purple": [
                (4500, 4400),
                (4100, 3000),
                (1800, 2400),
                (2500, 3400),
                (2000, 1400),
            ],
        }
```

7 Moje riešenie

Celý princíp môjho programu je založený na nasledujúcich krokoch:

1. Generovanie bodov: Na generovanie všetkých bodov používam funkcie make_dots() a random_dot(). Funkcia make_dots zabezpečuje vytvorenie všetkých bodov v príslušných farbách tak, aby bol každý bod jedinečný. Samostatný bod je generovaný funkciou random_dot(), ktorá s pravdepodobnosťou 99% vytvorí bod v danom farebnom intervale.

```
def random_dot(color):
    borders = {
        "red": (-5000, 500, -5000, 500),
        "green": (-500, 5000, -5000, 500),
        "blue": (-5000, 500, -500, 5000),
        "purple": (-500, 5000, -500, 5000),
    }

    if random.random() < 0.99:
        x = random.randint(borders[color][0], borders[color][1])
        y = random.randint(borders[color][2], borders[color][3])
    else:
        x = random.randint(-5000, 5000)
        y = random.randint(-5000, 5000)

    return (x, y)
def make_dots(individual_color_num, dots, colors):
    colors = itertools.cycle(colors)
    new_points = []

    for _ in range(4 * individual_color_num):
        color = next(colors)
        point = random_dot(color)

        while point in new_points or point in dots[color]:
            point = random_dot(color)

        new_points.append(point)
    return new_points
```

2. Klasifikácia bodov: V funkcii classify() sa vypočíta euklidovská vzdialenosť všetkých bodov, ktoré sa potom zoradia od najmenšej. Takto získam najbližšie body k nášmu bodu a podľa hodnoty k určím prevládajúcu farbu, ktorú následne vrátim.

```
def classify(point, dots, k):
    distances = [
        (color, ((point[0] - dot[0]) ** 2 + (point[1] - dot[1]) ** 2) ** 0.5)
        for color in dots
        for dot in dots[color]
    ]

    distances.sort(key=lambda x: x[1])

    top_k_colors = [color for color, _ in distances[:k]]
    return Counter(top_k_colors).most_common(1)[0][0]
```

3. Určenie úspešnosti: V hlavnej funkcii main() porovnám vygenerovanú farbu s očakávanou a určím presnosť. Tento proces vykonám pre všetky vygenerované body a vypíšem štatistiku výsledkov.

```
for k in k_values:
        correct = 0
        print(f"\nClassifying with k = {k}...")
```

```
        dots_copy = copy.deepcopy(old_dots)

        start = time.time()
        for i, point in enumerate(new_dots):
            color = classify(point, dots_copy, k)
            expected_color = colors[i % 4]
            dots_copy[color].append(point)
            print(f"Point {i + 1} classified")
            if color == expected_color:
                correct += 1

        end = time.time()
        time_elapsed = end - start
        accuracy = (correct / (total_color_count)) * 100

        print("\nResults for k =", k)
        print(f"Accuracy: {accuracy:.2f}%")
        print(f"Correct classifications: {correct}")
        print(f"Wrong classifications: {total_color_count - correct}")
        print(f"Time elapsed: {time_elapsed:.2f}s")
```

4. Generovanie bodov pre prázdne plochy: Aby bola celá plocha zafarbená, klasifikujem aj ostatné body.

```
def make_filler_dots(dots, k):
    total_range = 5000
    interval = int(total_range / 20)
    filler_dots = {
        "red": [],
        "green": [],
        "blue": [],
        "purple": [],
    }

    for x in range(-total_range + interval // 2, total_range + interval // 2, interval):
        for y in range(
            -total_range + interval // 2, total_range + interval // 2, interval
        ):
            point = (x, y)
            color = classify(point, dots, k)
            filler_dots[color].append(point)

    return filler_dots
```

5. Vykreslenie plochy: Pomocou funkcie visualize_dots a knižnice Matplotlib vykreslím príslušné plochy.

```
def visualize_dots(dots, accuracy, filler_dots):
    fig, ax = plt.subplots()
    for color in filler_dots:
        if filler_dots[color]:
            dot_array = np.array(filler_dots[color])
            ax.scatter(dot_array[:, 0], dot_array[:, 1], c=color, alpha=0.4, s=100)
    for color in dots:
        dot_array = np.array(dots[color])
        ax.scatter(
            dot_array[:, 0],
            dot_array[:, 1],
            c=color,
            label=color,
            marker="o",
            edgecolors="black",
        )
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_title(f"Dots Visualization\nAccuracy: {accuracy:.2f}%")
    ax.set_aspect("equal", adjustable="box")
    plt.show()
```

6. Opakovanie pre rôzne hodnoty k: Môžem pokračovať na ďalšie hodnoty k, pričom proces je rovnaký, ale už nepotrebujem generovať nové náhodné body, namiesto toho použijem už vygenerované. To mi umožňuje efektívne zhodnotiť, ako rôzne hodnoty k ovplyvňujú klasifikáciu príslušných bodov.

8 Používateľské rozhranie

Vytvorila som užívateľské rozhranie v termináli, ktoré je navrhnuté tak, aby užívateľovi intuitívne umožňovalo zadávať potrebné údaje pre spustenie programu. Rozhranie je jednoduché a priamočiare, poskytuje jasné pokyny, čo uľahčuje jeho používanie.

Tu sú kľúčové aspekty môjho užívateľského rozhrania:

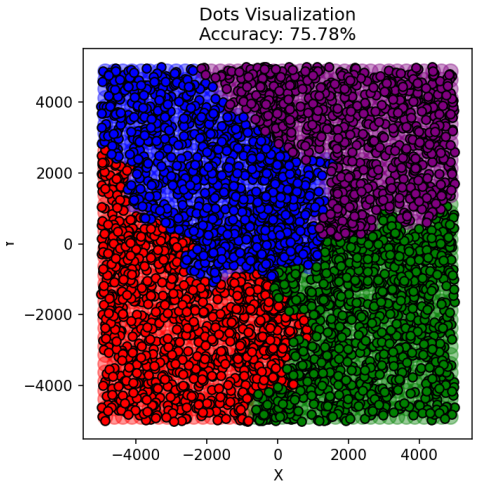
- Počet spustení programu: Užívateľ má možnosť zadať, koľkokrát chce program spustiť. Táto funkcia je obzvlášť dôležitá pri testovaní programu, keďže umožňuje opakované spustenie pre zhromažďovanie dát.
- Počet bodov pre jednotlivé farby: Užívateľ špecifikuje, koľko bodov chce generovať pre každú farbu. Týmto sa určuje množstvo dát, ktoré bude program spracovávať.
- Hodnoty k pre spustenie programu: Užívateľ zadáva hodnoty k, ktoré sa budú používať v algoritme KNN. Tento vstup je kritický pre určenie, ako bude program klasifikovať dáta.

Po úspešnej klasifikácii program vypíše podrobnú štatistiku pre každú testovanú hodnotu k. Táto štatistika zahŕňa:

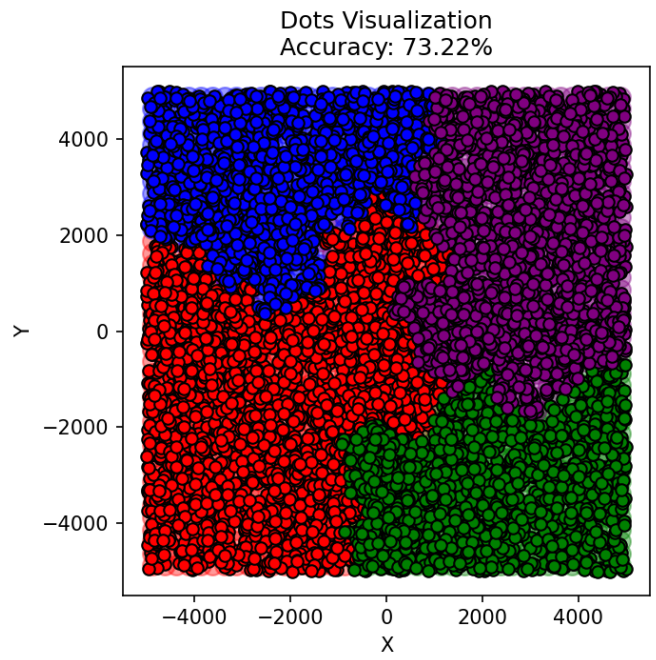
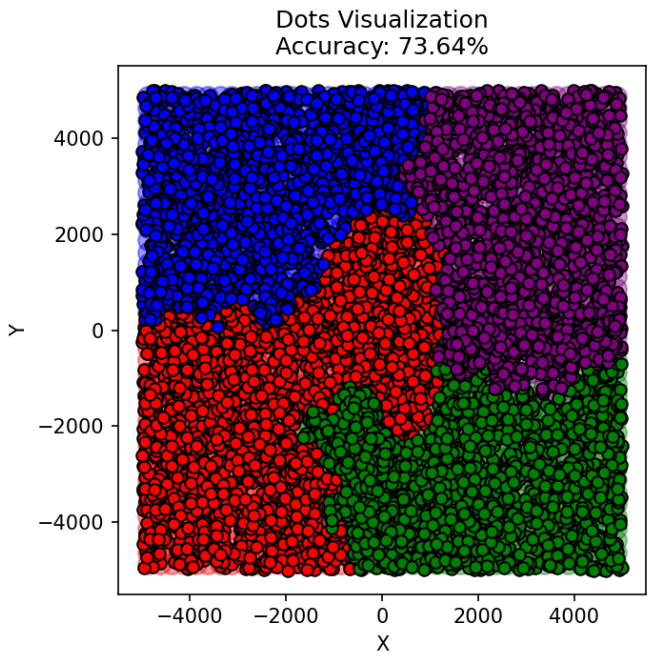
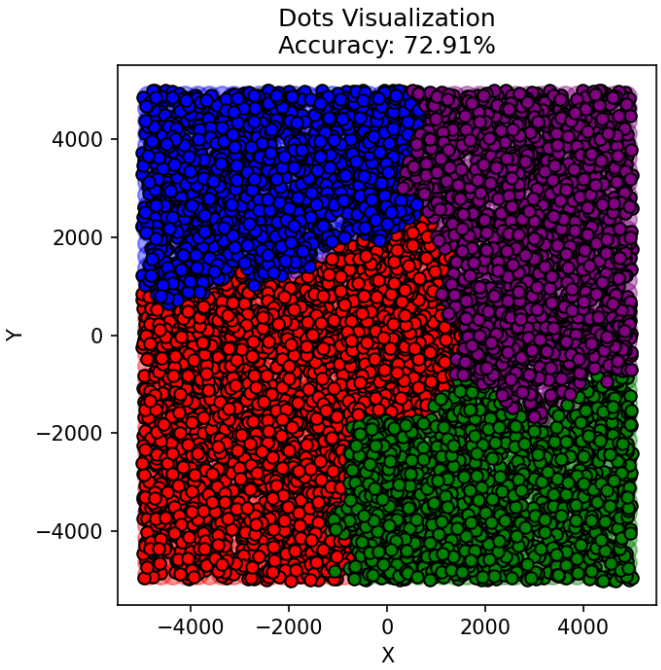
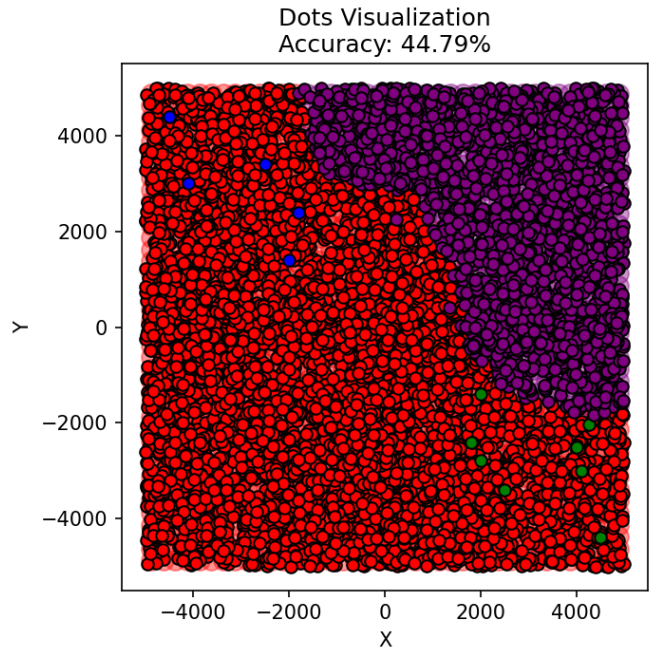
- Percentuálnu úspešnosť: Ukazuje efektivitu klasifikácie pri rôznych hodnotách k.
- Počet správnych a nesprávnych klasifikácií: Poskytuje prehľad o počte úspešne a neúspešne klasifikovaných bodov.
- Výkon a čas: Zobrazuje časovú náročnosť klasifikácie pri jednotlivých hodnotách k.

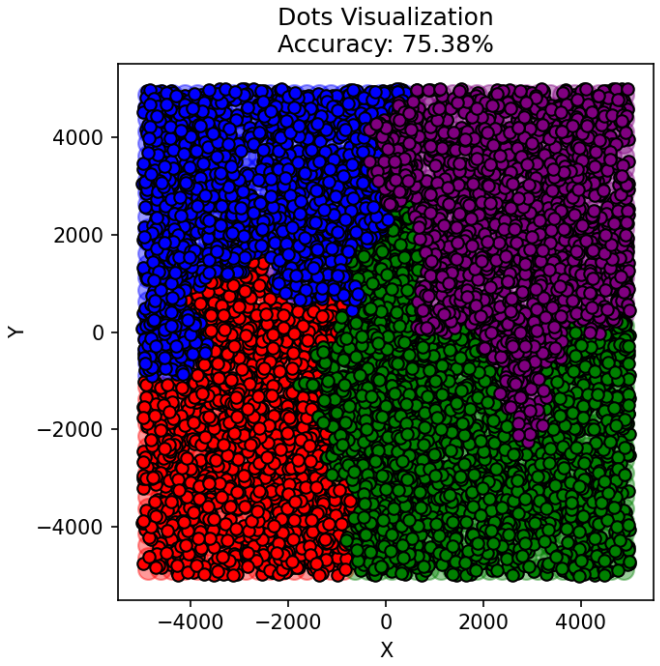
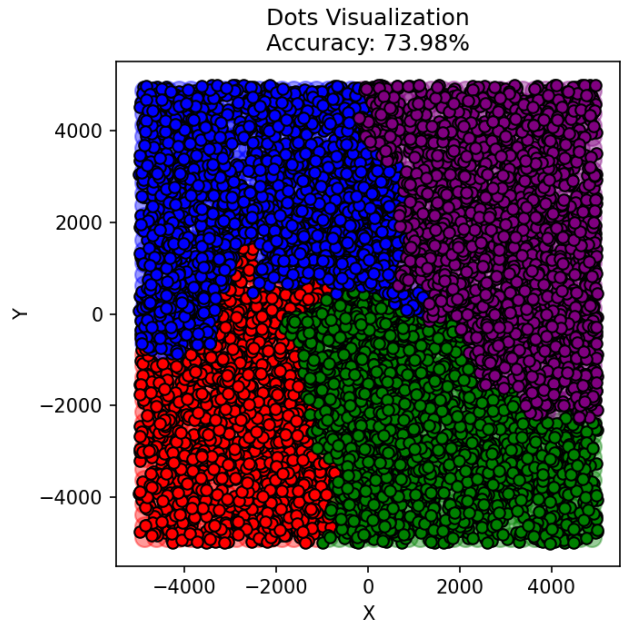
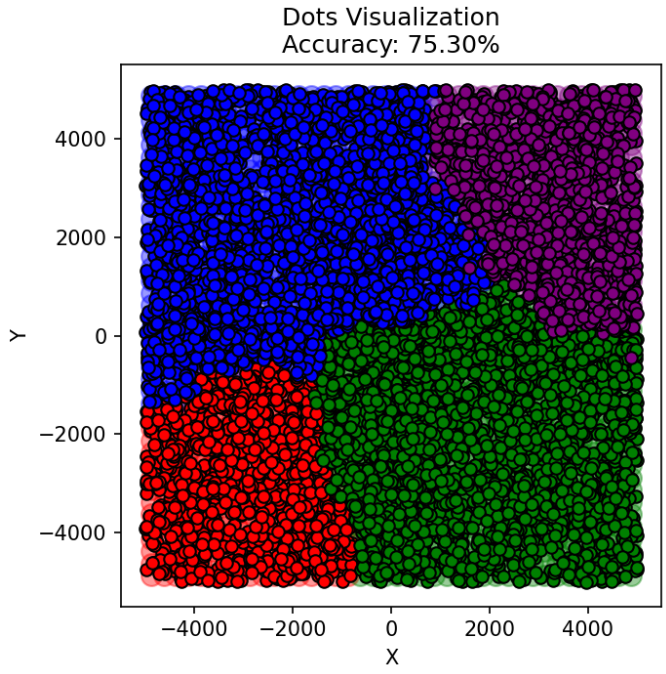
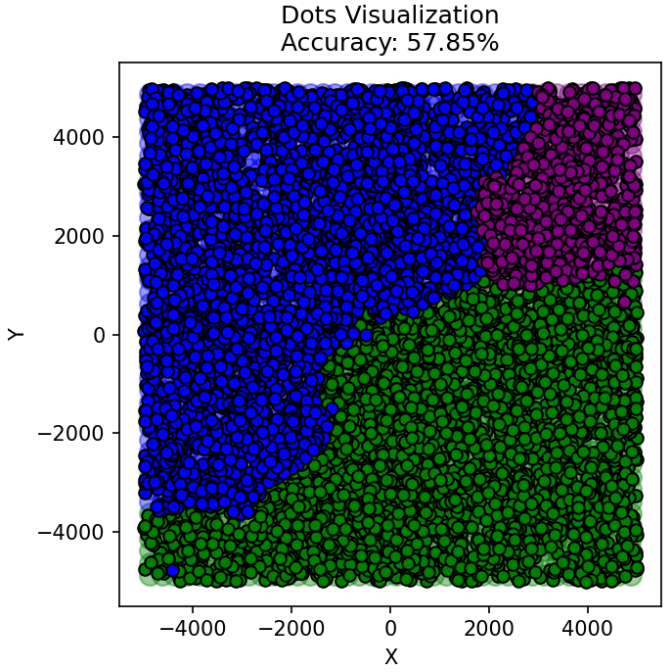
```
Welcome to the KNN Classifier!

Enter how many new dots you want to classify inidvidualy per color: 1000
Enter how many times you want to run the program: 1
Enter the k values you want to use (space-separated): 1 3 7 15
Press enter to start the program...
```

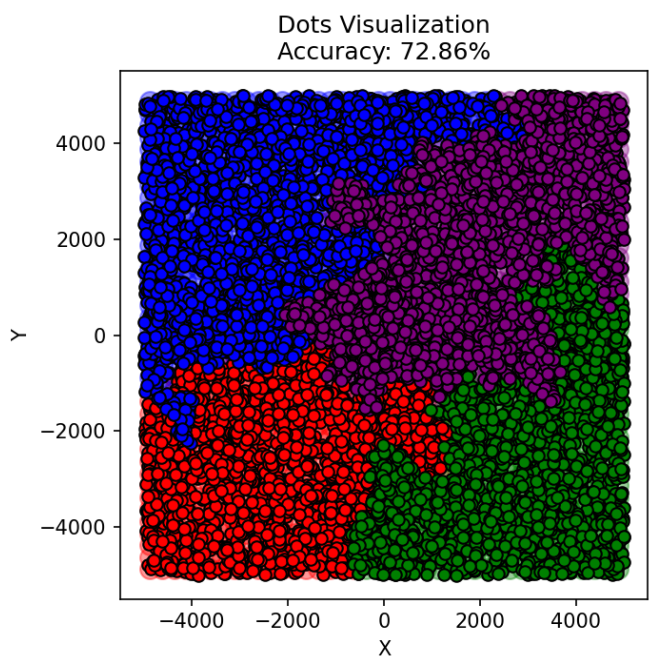
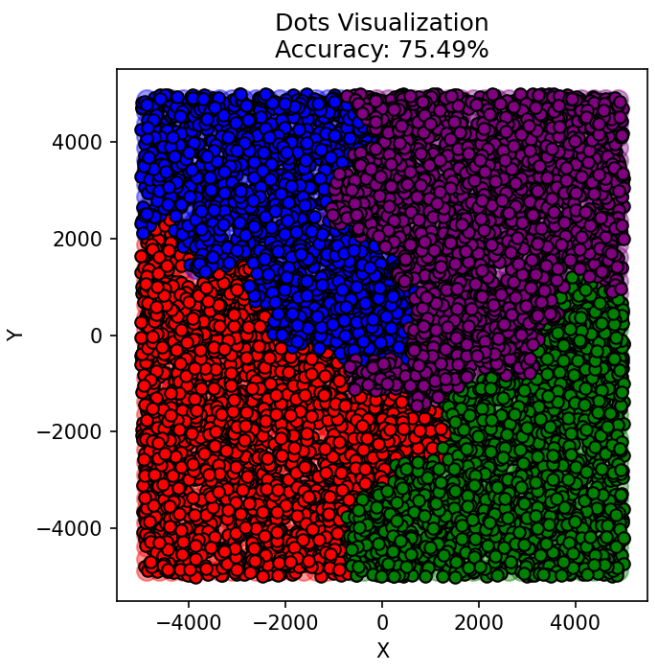
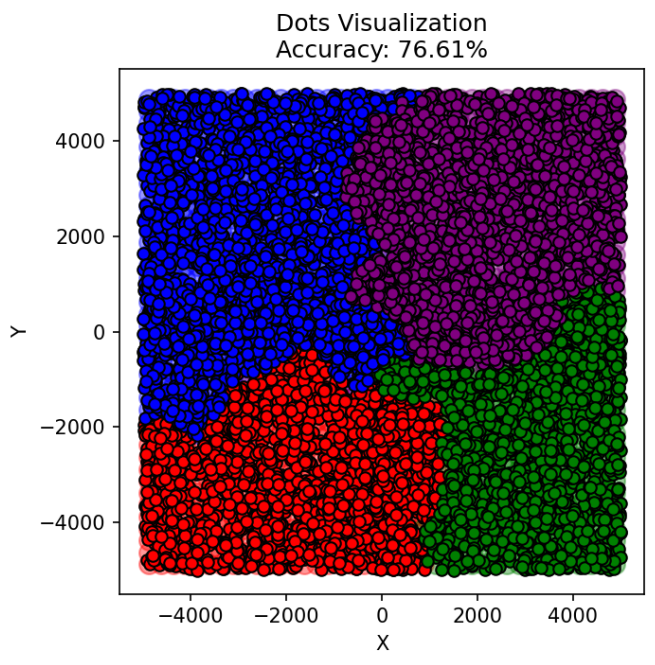
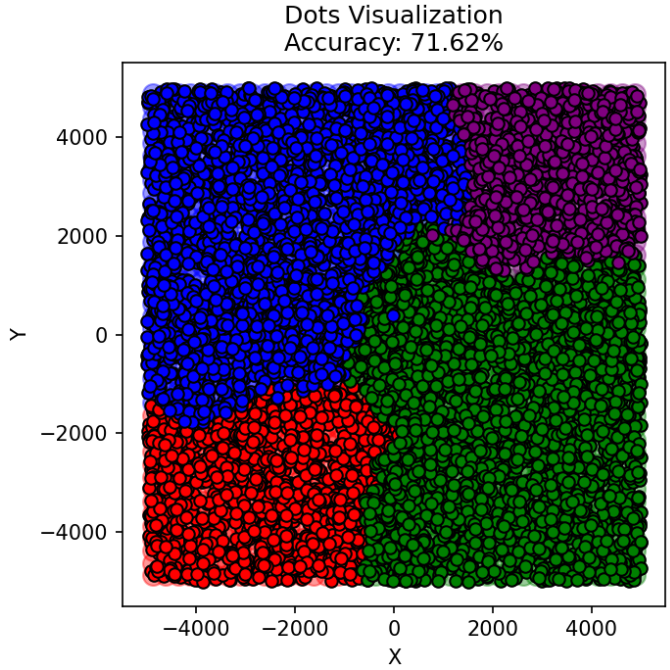
```
Results for k = 1
Accuracy: 75.78%
Correct classifications: 3031
Wrong classifications: 969
Time elapsed: 3.95s
Making filler dots...
Visualizing...
```



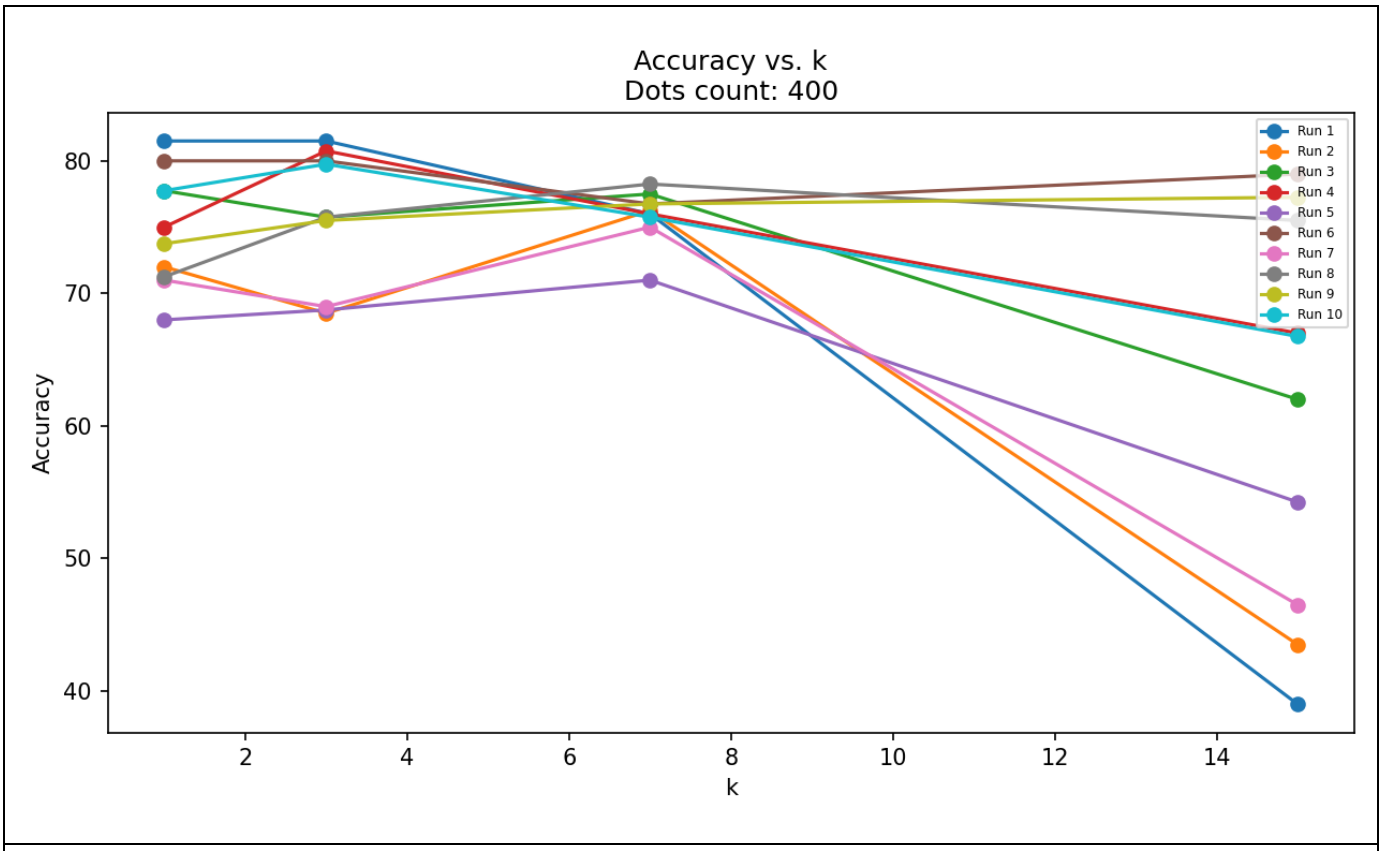
9 Vizuaizácia v main()

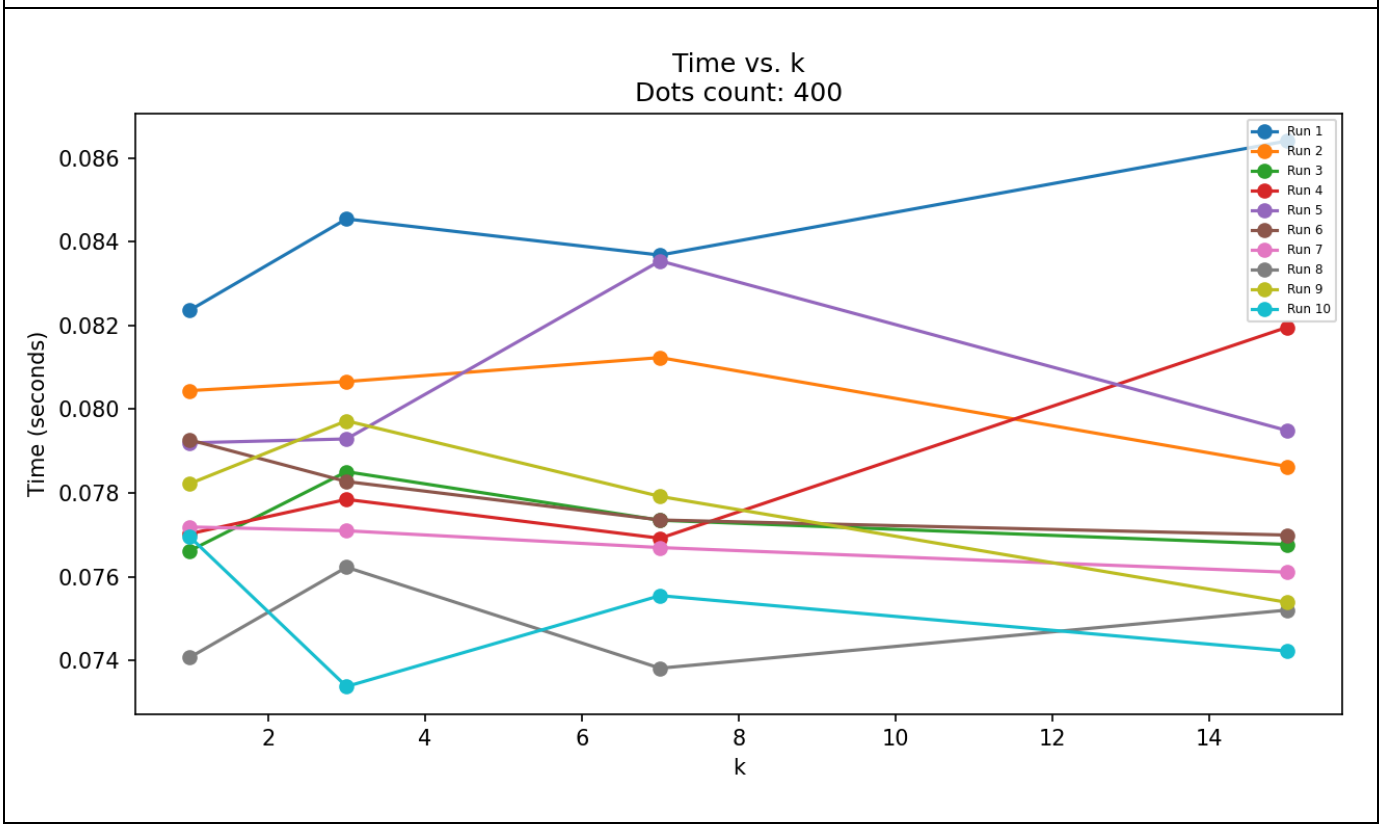
K=1	K=3
	
K=7	**K=15**
	

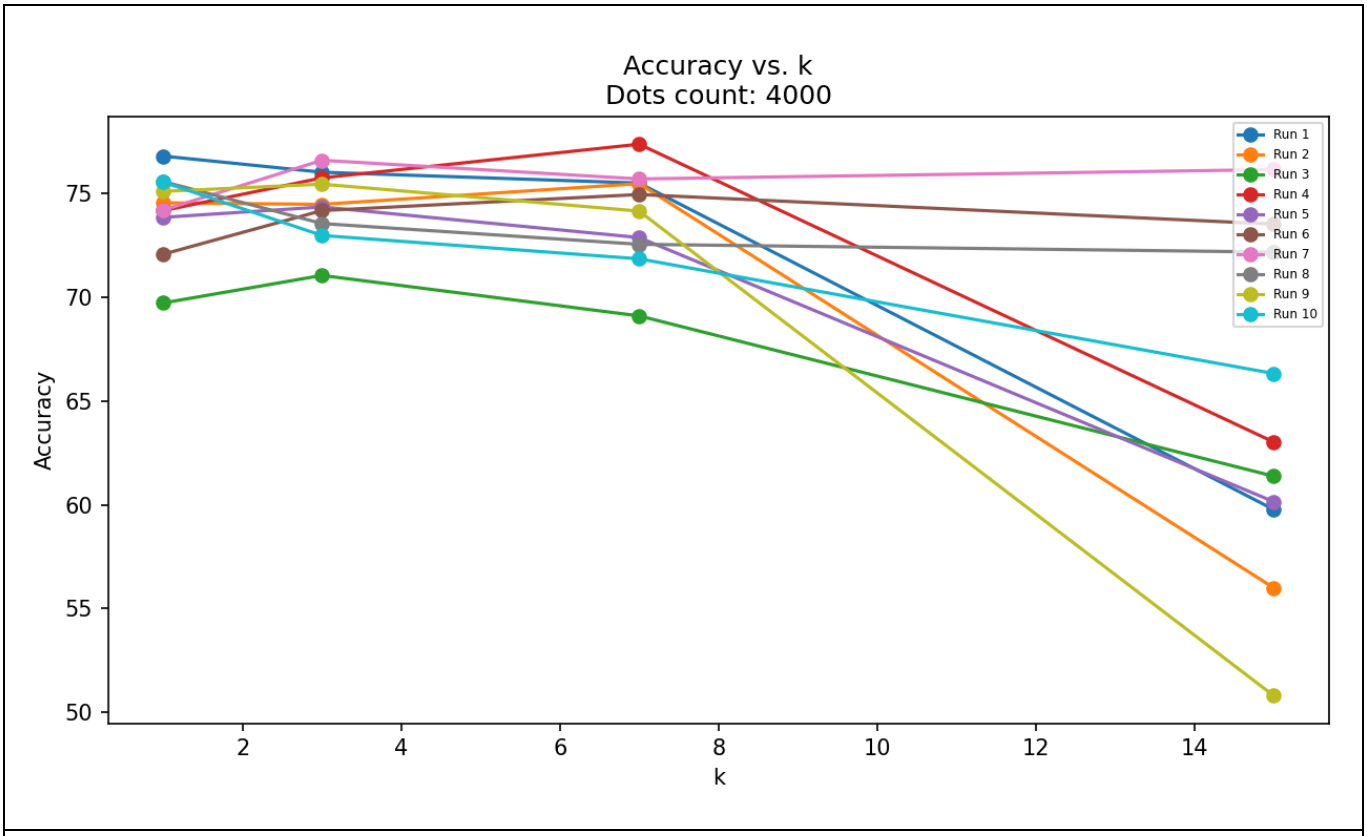
K=1	K=3
 Dots Visualization Accuracy: 75.38% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X	 Dots Visualization Accuracy: 73.98% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X
K=7	K=15
 Dots Visualization Accuracy: 75.30% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X	 Dots Visualization Accuracy: 57.85% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X

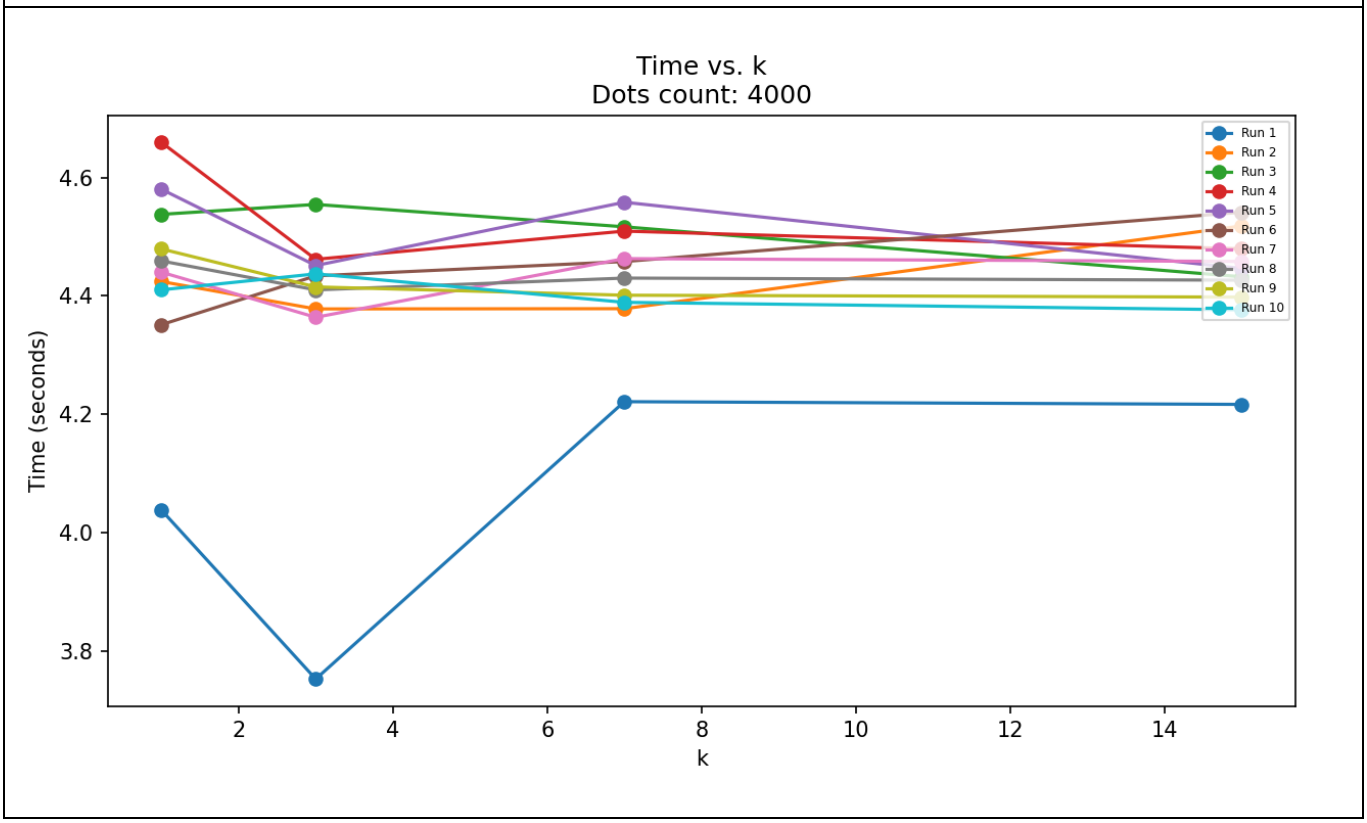
K=1	K=3
 Dots Visualization Accuracy: 72.86% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X	 Dots Visualization Accuracy: 75.49% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X
K=7	K=15
 Dots Visualization Accuracy: 76.61% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X	 Dots Visualization Accuracy: 71.62% 4000 2000 0 −2000 −4000 Y −4000 −2000 0 2000 4000 X

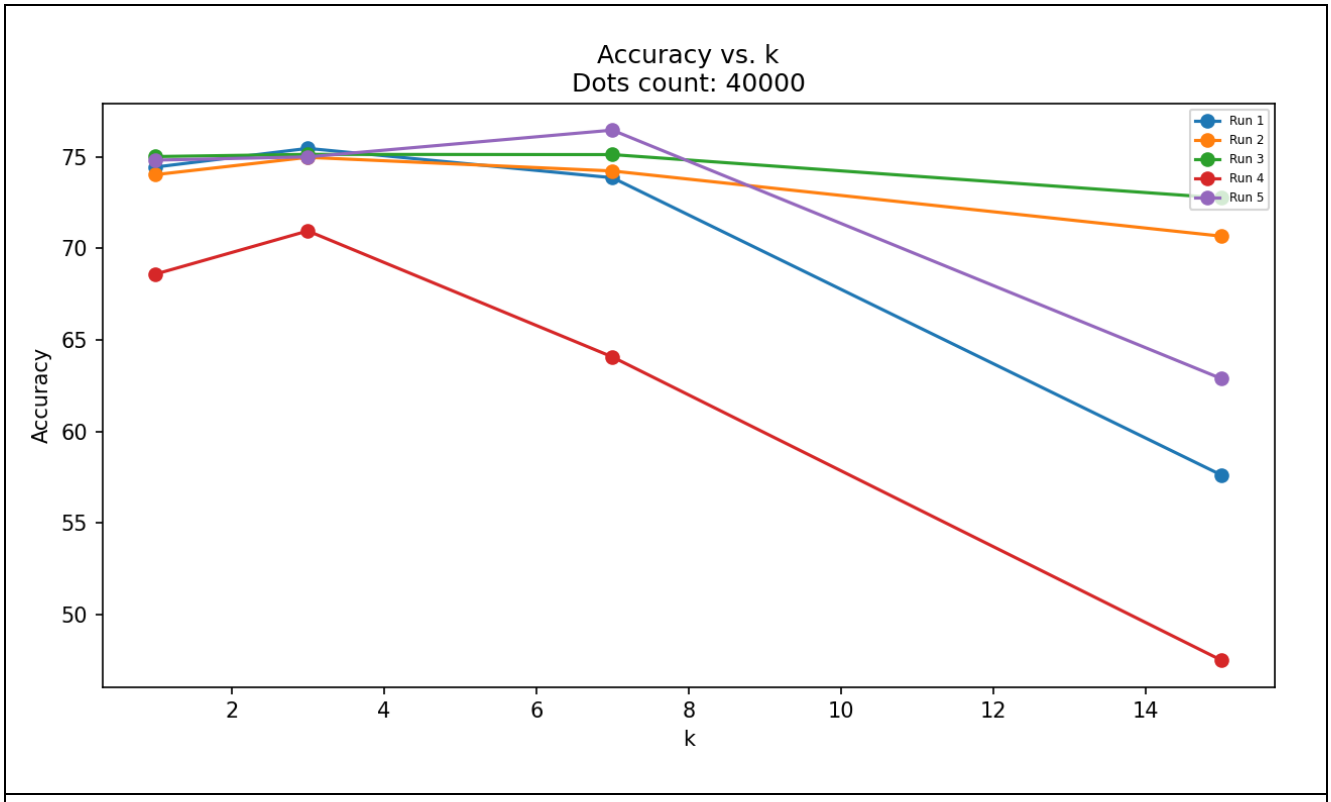
10 Vizualizácia v testing()

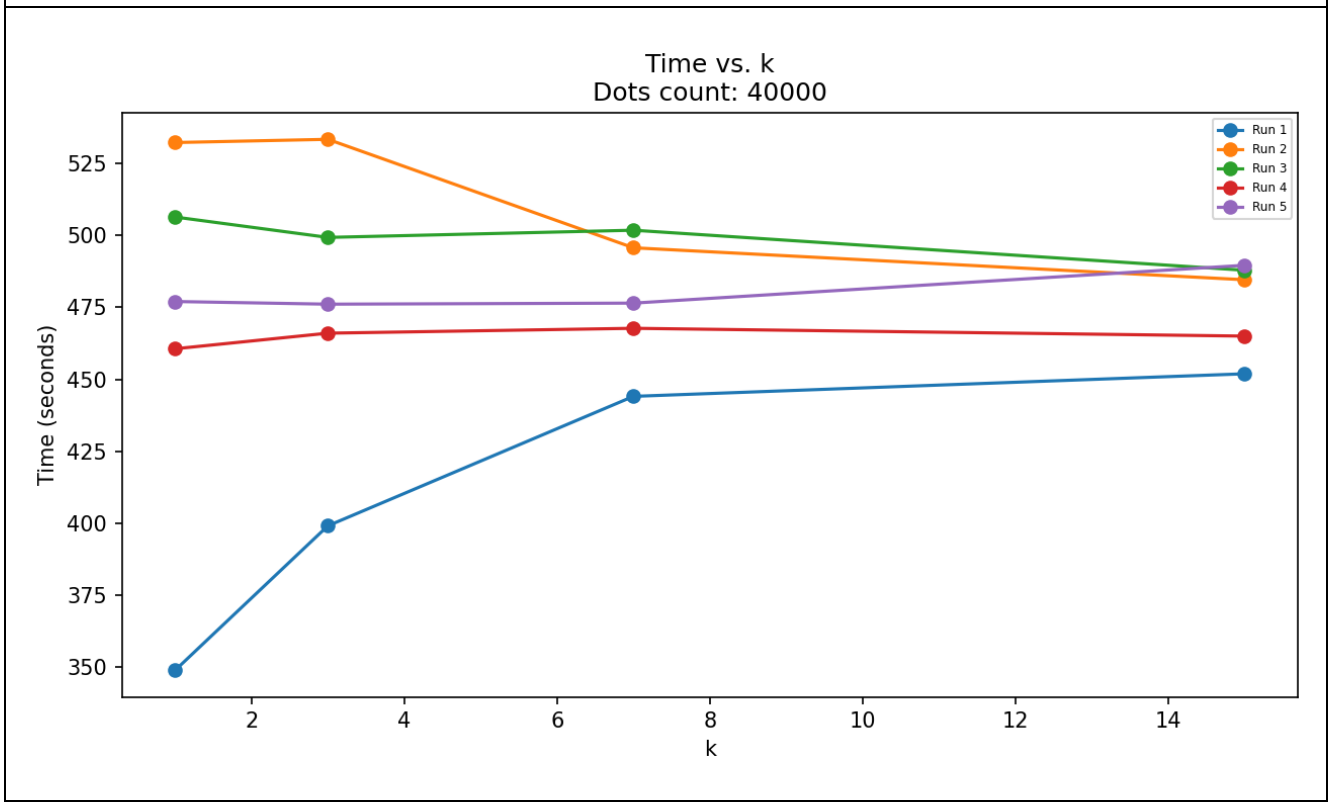





Accuracy vs. k
Dots count: 4000
Accuracy
k
Run 1
Run 2
Run 3
Run 4
Run 5
Run 6
Run 7
Run 8
Run 9
Run 10


Time vs. k
Dots count: 4000
Time (seconds)
k
Run 1
Run 2
Run 3
Run 4
Run 5
Run 6
Run 7
Run 8
Run 9
Run 10


Accuracy vs. k
Dots count: 40000
Run 1
Run 2
Run 3
Run 4
Run 5
Accuracy
75
70
65
60
55
50
2
4
6
8
10
12
14
k


Time vs. k
Dots count: 40000
Run 1
Run 2
Run 3
Run 4
Run 5
Time (seconds)
525
500
475
450
425
400
375
350
2
4
6
8
10
12
14
k

11 Zhodnotenie výsledkov

Pri analýze úspešnosti môjho KNN algoritmu s rôznymi hodnotami k som zistila, že výber správnej hodnoty má zásadný vplyv na výsledky klasifikácie. Keď som použila k=1, úspešnosť klasifikácie sa pohybovala medzi 65-75%, ale vizualizácie odhalili nesúvislé plochy s nepravidelnými okrajmi, čo naznačuje určitú nestabilitu v rozdelení. Zvýšenie k na 3 prinieslo lepšiu úspešnosť, okolo 75%, a zároveň vizuálne koherentnejšie a stabilnejšie hranice. Pri k=7 som zaznamenala ešte lepšie výsledky s úspešnosťou presahujúcou 75%, kde plochy boli suvislejšie a celkovo väčšie. Avšak, zaujímavým obratom bolo, keď som zvýšila k až na 15. V tomto prípade som zaznamenala pokles úspešnosti na priemerne 40-50%, čo ukázalo, že príliš vysoké hodnoty k môžu negatívne ovplyvniť kvalitu klasifikácie. Z mojich testov a analýz vyplýva, že optimálna hodnota k pre algoritmus KNN by nemala byť ani príliš vysoká, ani príliš nízka. Táto rovnováha je kľúčová pre dosiahnutie najlepšej kombinácie presnosti a vizuálnej súdržnosti výsledkov. Príliš malé hodnoty k môžu viesť k nesúvislým a nepravidelným oblastiam, zatiaľ čo príliš vysoké hodnoty môžu spôsobiť stratu dôležitých detailov a znížiť celkovú presnosť.

12 Možné vylepšenia

Môj súčasný program na klasifikáciu bodov pomocou algoritmu KNN je stabilný a spoľahlivý. Výpočet euklidovskej vzdialenosti pre každý bod so všetkými ostatnými bodmi zaručuje presnosť klasifikácie, čo je kľúčové pre korektné výsledky. Tento prístup je efektívny pri menších dátových sadách, kde je dôležitá presnosť nad rýchlosťou.

Na zvýšenie výkonnosti, najmä pri väčších dátových sadách, by bolo možné zaviesť metódy ako KD-stromy pre rýchlejšie vyhľadávanie najbližších susedov, alebo rozdeliť plochu na menšie sekcie na zníženie počtu bodov potrebných k porovnaniu. Tieto vylepšenia by mohli výrazne skrátiť čas potrebný na spracovanie bez významnej straty presnosti.

Použitá literatúra

1. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning