Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológii

15. November 2021

Umelá inteligencia

Zadanie č. 3a- Zenová záhrada

Evolučný algoritmus

Roland Vdovják
id: 110912

Roland Vdovják
ID: 110912

Obsah

Zadanie....2
Riešenie....4
Princíp programu....4
Make()....4
Set_point()....5
Classify()....5
Testovanie....7
Výsledky....8
Behy pre k = 1, 3, 5, 15....8
k = 1, 3, 5, 7, 9, 11, 13, 15, 17....11
Zhodnotenie....14

Roland Vdovják
ID: 110912

Zadanie

Máme 2D priestor, ktorý má rozmery X a Y, v intervaloch od -5000 do +5000. V tomto priestore sa môžu nachádzať body, pričom každý bod má určenú polohu pomocou súradníc X a Y. Každý bod má unikátne súradnice (t.j. nemalo by byť viacej bodov na presne tom istom mieste). Každý bod patrí do jednej zo 4 tried, pričom tieto triedy sú: red (R), green (G), blue (B) a purple (P). Na začiatku sa v priestore nachádza 5 bodov pre každú triedu (dokopy teda 20 bodov). Súradnice počiatočných bodov sú:

```
R: [-4500,-4400], [-4100,-3000], [-1800,-2400], [-2500,-3400] a [-2000,-1400]

G: [+4500,-4400], [+4100,-3000], [+1800,-2400], [+2500,-3400] a [+2000, -400]

B: [-4500,+4400], [-4100,+3000], [-1800,+2400], [-2500,+3400] a [-2000,+1400]

P: [+4500,+4400], [+4100,+3000], [+1800,+2400], [+2500,+3400] a [+2000,+1400]
```

Vašou úlohou je naprogramovať klasifikátor pre nové body – v podobe funkcie classify(int X, int Y, int k), ktorá klasifikuje nový bod so súradnicami X a Y, pridá tento bod do nášho 2D priestoru a vráti triedu, ktorú pridelila pre tento bod. Na klasifikáciu použite k-NN algoritmus, pričom k môže byť 1, 3, 7 alebo 15.

Na demonštráciu Vášho klasifikátora vytvorte testovacie prostredie, v rámci ktorého budete postupne generovať nové body a klasifikovať ich (volaním funkcie classify). Celkovo vygenerujte 20000 nových bodov (5000 z každej triedy). Súradnice nových bodov generujte náhodne, pričom nový bod by mal mať zakaždým inú triedu (dva body vygenerované po sebe by nemali byť rovnakej triedy):

- R body by mali byť generované s 99% pravdepodobnosťou s X < +500 a Y < +500
- G body by mali byť generované s 99% pravdepodobnosťou s X > -500 a Y < +500
- B body by mali byť generované s 99% pravdepodobnosťou s X < +500 a Y > -500
- P body by mali byť generované s 99% pravdepodobnosťou s X > -500 a Y > -500

  (Zvyšné jedno percento bodov je generované v celom priestore.)

Návratovú hodnotu funkcie classify porovnávajte s triedou vygenerovaného bodu. **Na základe týchto porovnaní vyhodnoťte úspešnosť** Vášho klasifikátora pre daný experiment.

Experiment vykonajte 4-krát, pričom zakaždým Váš klasifikátor použije iný parameter k (pre k = 1, 3, 7 alebo 15) a vygenerované body budú pre každý experiment rovnaké.

Vizualizácia: pre každý z týchto experimentov vykreslite výslednú 2D plochu tak, že vyfarbíte túto plochu celú. Prázdne miesta v 2D ploche vyfarbite podľa Vášho klasifikátora.

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

Poznámka 1: Je vhodné využiť nejaké optimalizácie na zredukovanie zložitosti, napríklad pre hľadanie k najbližších bodov si rozdelíme plochu na viaceré menšie štvorce, do ktorých umiestňujeme body s príslušnými súradnicami, aby sme nemuseli vždy porovnávať všetky body, ale len body vo štvorci, kde sa nachádza aktuálny bod a susedných štvorcoch. Je tiež možné využiť algoritmus na hľadanie k najmenších hodnôt.

Poznámka 2: Úlohu je možné riešiť aj pomocou neurónovej siete, pričom je vhodné použiť nejaký framework (napríklad PyTorch).

Riešenie

Program sme riešili v pythone. Aj napriek nevýhode ohľadom rýchlosti sme uprednostnili python.

Princíp programu

Na začiatku program definuje prvých 20 bodov zo zadanie. Volá funkciu kde podľa vstupného parametra určujeme okrem iného aj hodnotu k.

Make()

Funkcia *make()* na začiatku vyčistí hashtabuľku *data_set*, kde držíme všetky body, vloží do *data_set* len počiatočné body. Ďalej vytvára nové body (*set_point()*). Keď je hashtabuľka naplnená efektívne vkladá body do podgrafu.

Ukážka kódu:

```
# Nastavenie premennych pred spustenim tvorenia bodov
k = k1
random.seed(5000)
start = time.time()
data_set = {}
counter = 0
for i in range(5):      # Pociatocne body
        ...

# Vytvaranie bodov
for i in range(size-20):
        ...
        set_point(color)

# Efektivne vlozenie do grafu
data = list(data_set.values())
x_arr = []
y_arr = []
color_arr = []
for val in data:        # Naplnenie suradnic
        ...

end = time.time()

p = (size-counter)/size*100

plt.title("K= {}".format(k))
plt.xlabel("T = {:.2f}s P = {:.2f}%".format(end-start, p))

plt.scatter(x=x_arr, y=y_arr, color=color_arr, s=10)
```

Set_point()

Hlavnými úlohami tejto funkcie je určiť hranice pre vytvorenie bodu a vytvoriť bod, po úspešnom vytvorení sa bod klasifikuje. Hranice sa na 99% určujú podľa farby bodu pred klasifikáciou a s pravdepodobnosťou 1% na celej ploche.

Ukážka kódu:

```
# Urcenie hranic pre priradovanie random cisel
if random.randint(1, 100) < 100:
    if color == 'red':
        boundries_x = [-5000, 500]
        boundries_y = [-5000, 500]
    elif color == 'green':
        boundries_x = [-500, 5000]
        boundries_y = [-5000, 500]
    elif color == 'blue':
        boundries_x = [-5000, 500]
        boundries_y = [-500, 5000]
    elif color == 'purple':
        boundries_x = [-500, 5000]
        boundries_y = [-500, 5000]
else:
    boundries_x = [-5000, 5000]
    boundries_y = [-5000, 5000]

# Najdenie random cisla
while True:
    x = random.randint(boundries_x[0], boundries_x[1])
    y = random.randint(boundries_y[0], boundries_y[1])
    key = (x, y)

    # Ak naslo random cislo
    if key not in data_set:

        point = POINT(x, y, color)  # Vytvori bod
        classify(point)          # Klasifikuje
        data_set[key] = point    # Priradi do dir
        return data_set[key]
```

Classify()

Klasifikácia bodu je určená na určenie farby bodu podľa jeho k najbližších susedov. Na určenie susedov potrebujeme vedieť vzdialenosť bodov od toho, ktorý klasifikujeme. Euklidovskú vzdialenosť počítame pre každý bod z hashtabuľky(*distances*). Do listu priraďujeme vzdialenosť a farbu daného bodu. Keď máme list plný, zoradíme ho podľa vzdialenosti od najmenšej a priradíme do iného listu(*k_id*).

Pre k prvých prvkov, t.j. k najbližších susedov zisťujeme, akej sú farby a podľa toho plníme polia *closest*. Je to pole zložené z troch polí. Okrem počítanie počtu farieb riešime aj vzdialenosť danej farby. Ak sa totiž vyskytne viac ako jedna farba s rovnakým počtom, vyberieme tú, ktorej priemerná vzdialenosť je menšia.

Ukážka kódu:

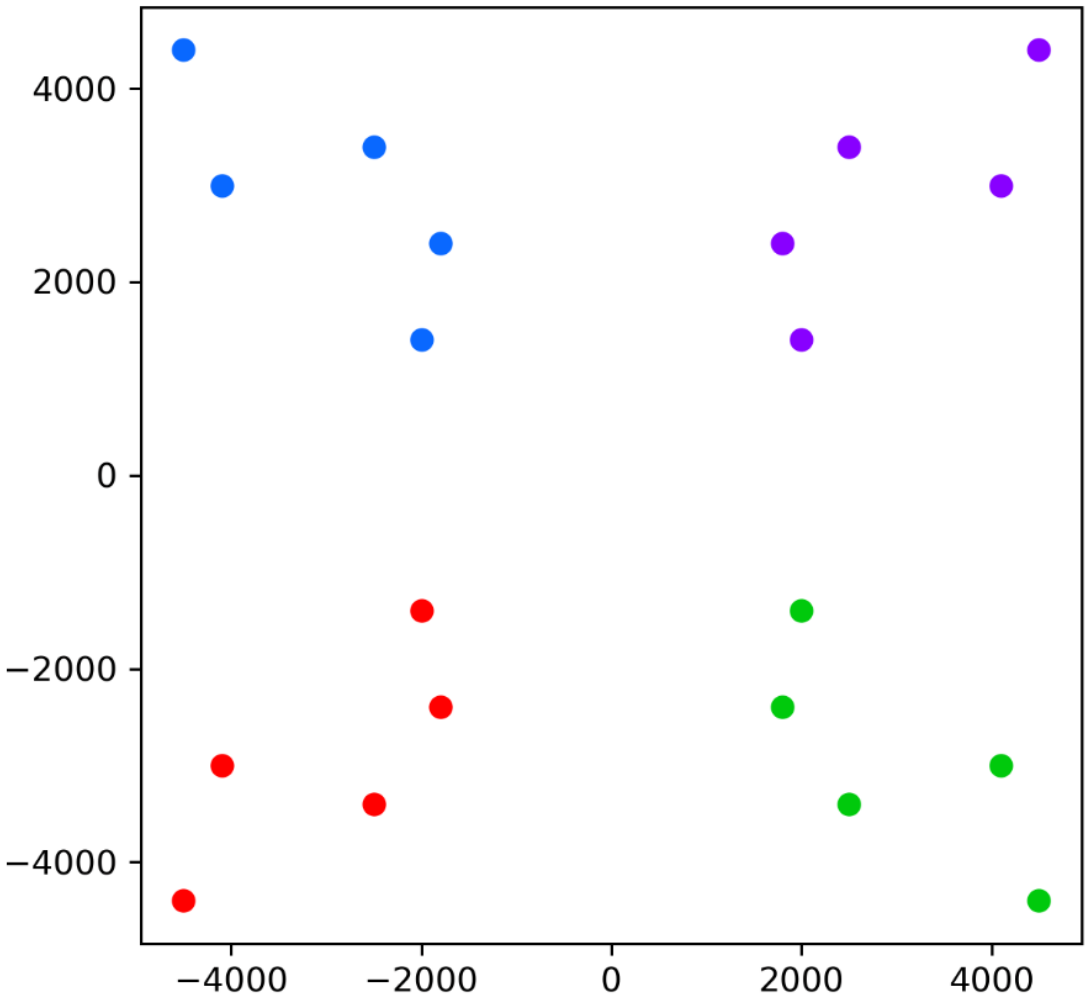
```
disatnces = [[np.sqrt((point.x-p.x)**2 + (point.y-p.y)**2), p.color]for p
             in data_set.values()]

# Zoradenie vzdialenosti
k_id = sorted(disatnces, key=lambda x:x[0])

# Priradovanie poctu farbam ktore su blizke
for i in range(k):
    if k_id[i][1] == 'red':
        id = 0
    elif k_id[i][1] == 'green':
        id = 1
    elif k_id[i][1] == 'blue':
        id = 2
    elif k_id[i][1] == 'purple':
        id = 3

    closest[0][id] += float(k_id[i][0])
    closest[1][id] += 1

# Spocitanie priemernej vzdialenosti farieb
for i in range(4):
    if closest[1][i] > 0:
        closest[2][i] = closest[0][i] / closest[1][i]

# Rovnaky pocet blizkych bodov
col = np.where(closest[1] == np.max(closest[1]))

# Ak je viac rovnakych farieb v rovnakom pocte, vyberie tesn s priemerne
blizsou vzdialenostou
if len(col[0])>1:
    point.color = c[np.argmin(closest[2])]
else:
    point.color = c[np.argmax(closest[1])]
```

Testovanie

Program sme testovali pre rôzne hodnoty k. Od hodnoty *k = 1* až po *k = 17*. Tiež bolo dôležité aby sa pre rôzne hodnoty tvorili rovnaké náhodné body. To sme zabezpečili priradením random seedu na určitú hodnotu. Testovali sme len na veľkosti 20 020 bodov.

Na začiatok program umiestni 20 bodov zo zadania, potom vyberá náhodne body a klasifikuje ich.

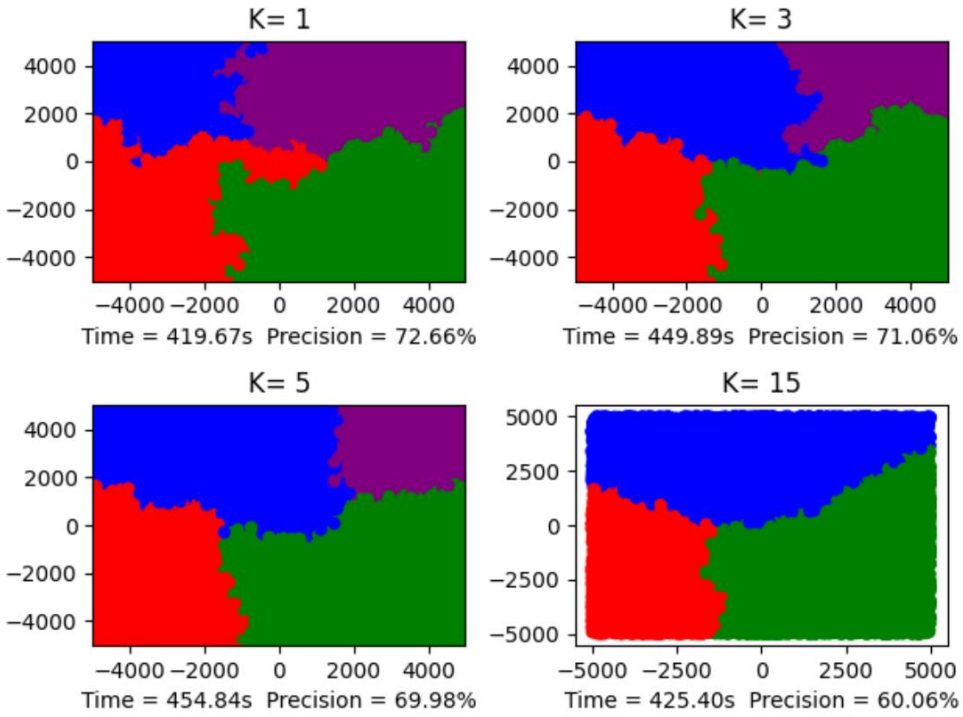
Začiatok po naplnení bodov vyzerá takto:



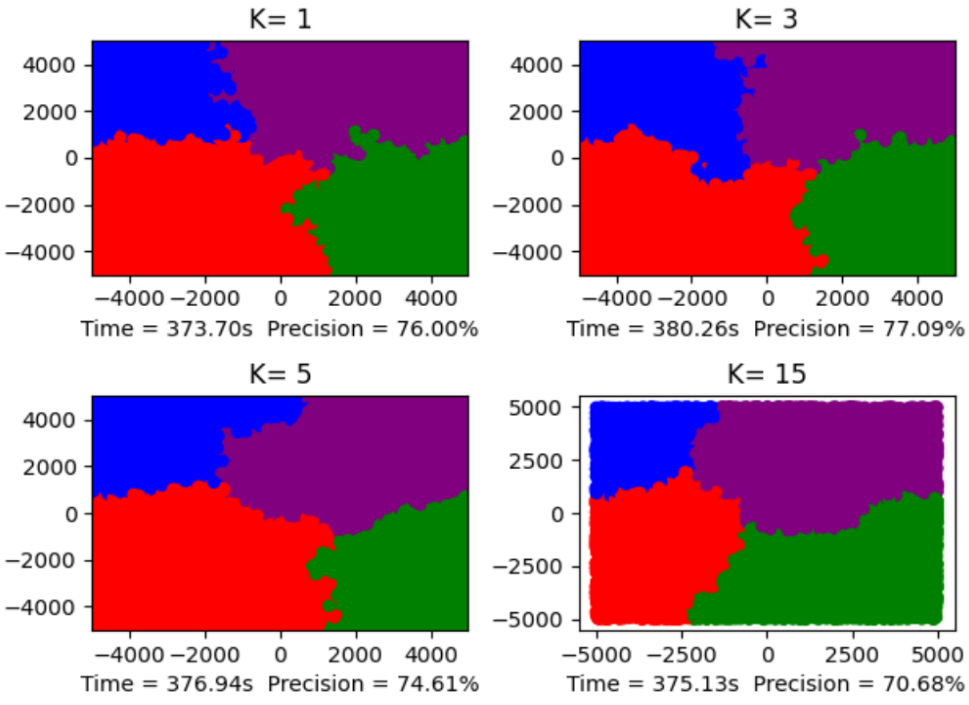
Výsledky

Program sme zo začiatku testovali pre k = 1, 3, 5, 15, až neskôr sme zistili, že je vyžadované k = 7. Pre tieto štyri hodnoty sme testovali päťkrát, aby sme mali lepšiu predstavu o trvaní programu a presnosti klasifikácie pri rôznych k hodnotách.

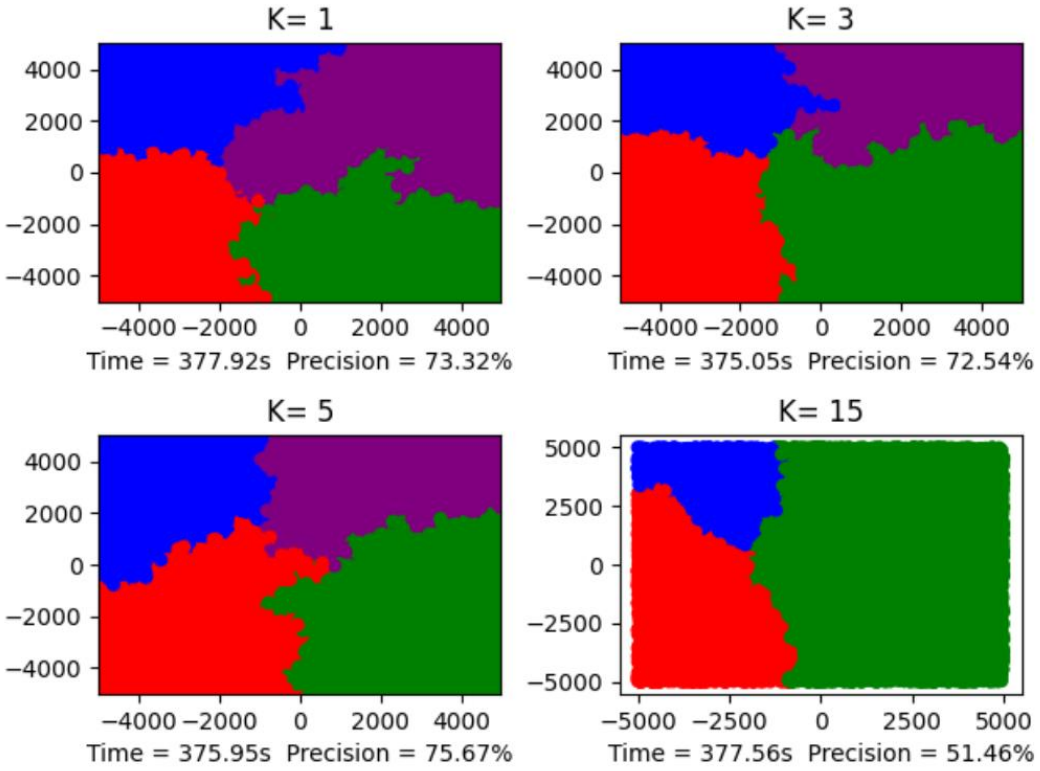
Behy pre k = 1, 3, 5, 15



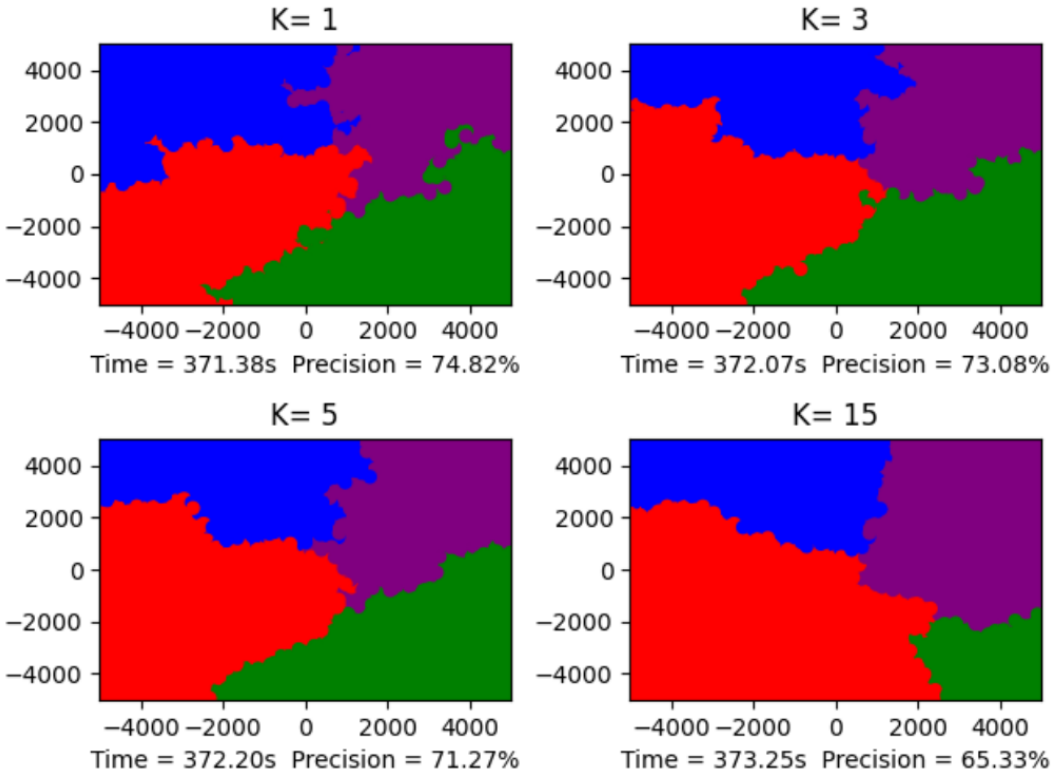
Obrázok 1: Beh 1



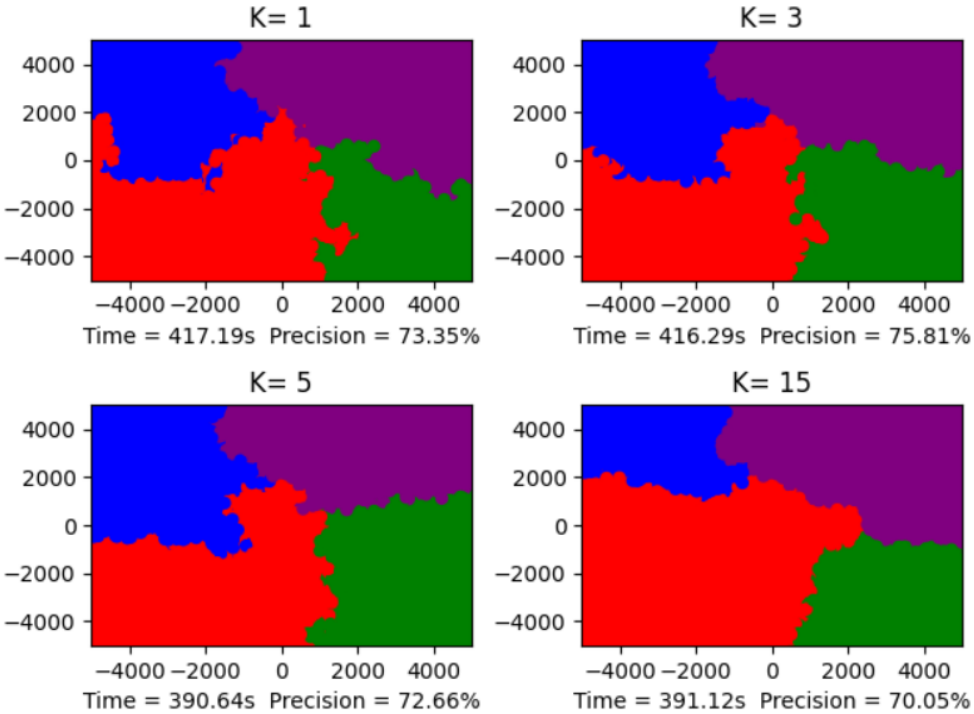
Obrázok 2: Beh 2



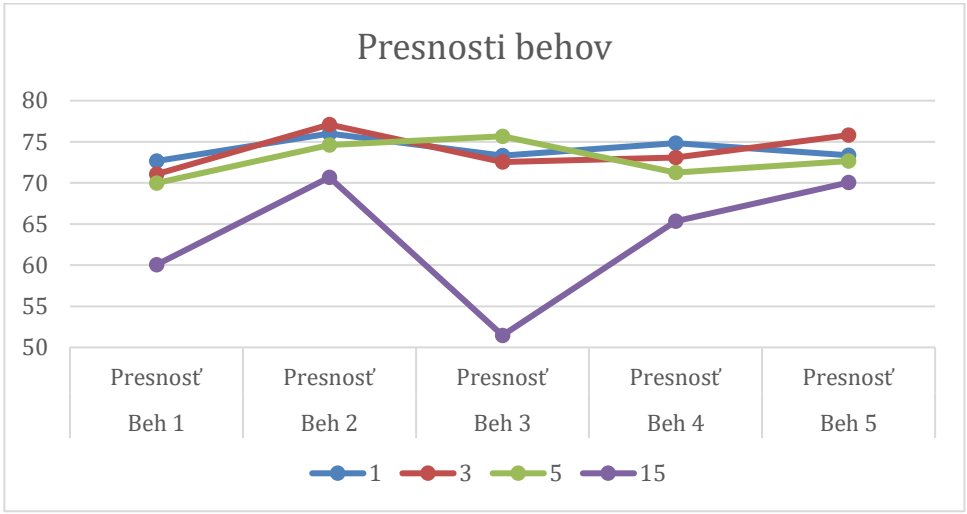
Obrázok 3: Beh 3



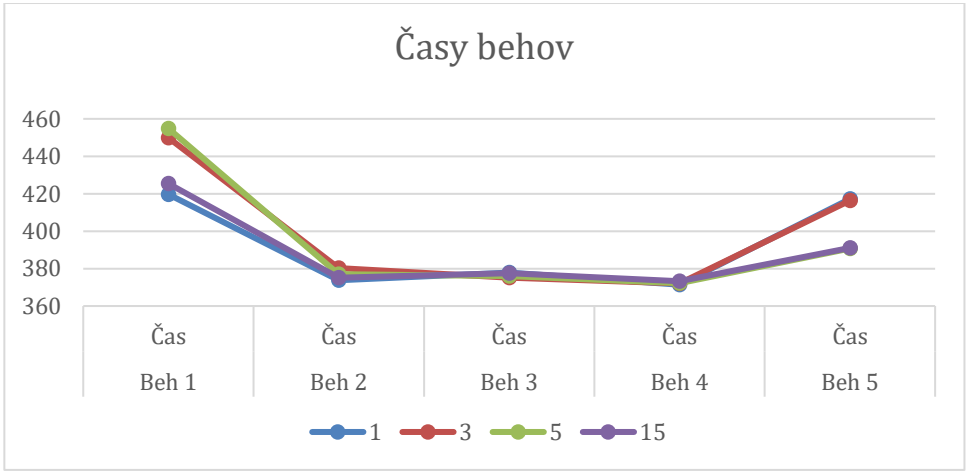
Obrázok 4: Beh 4



Obrázok 5: Beh 5

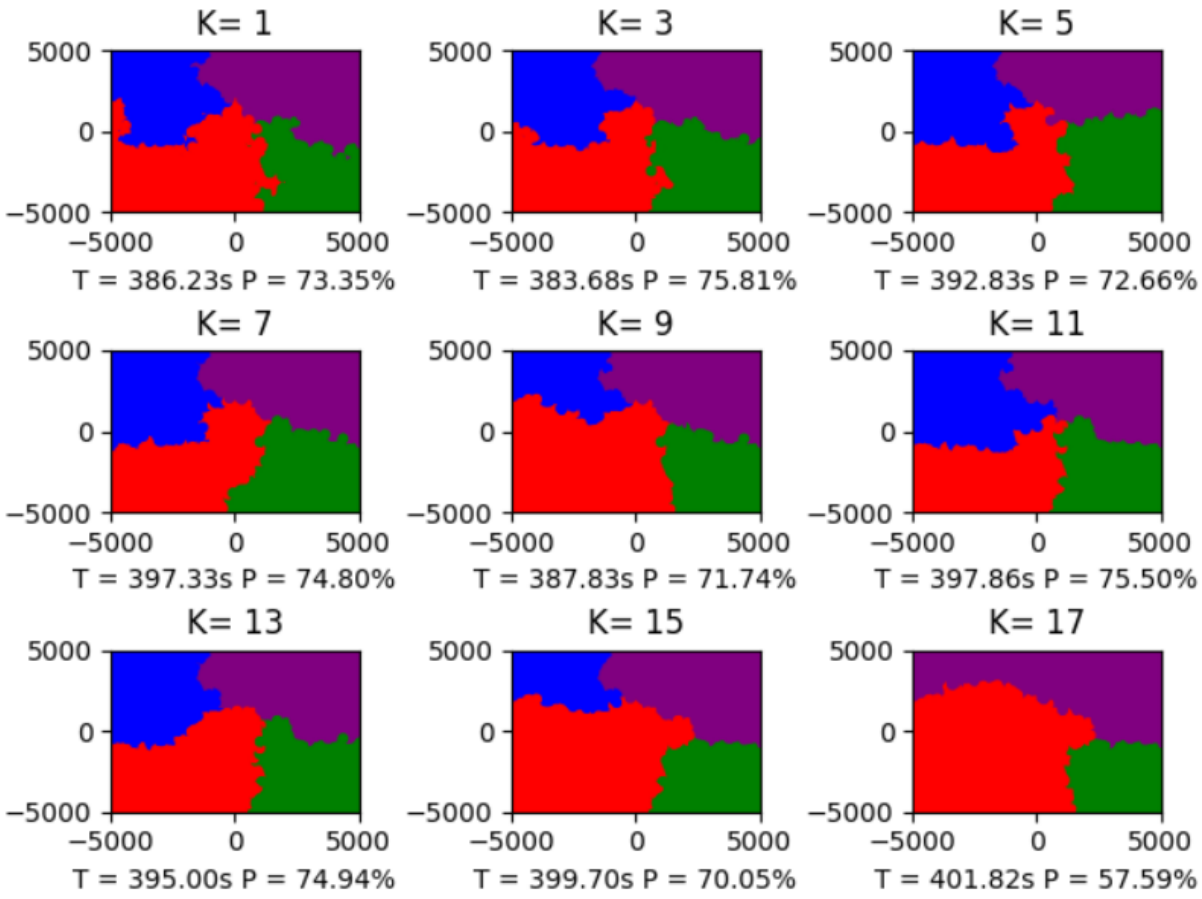


Graf 1: Presnosti behov

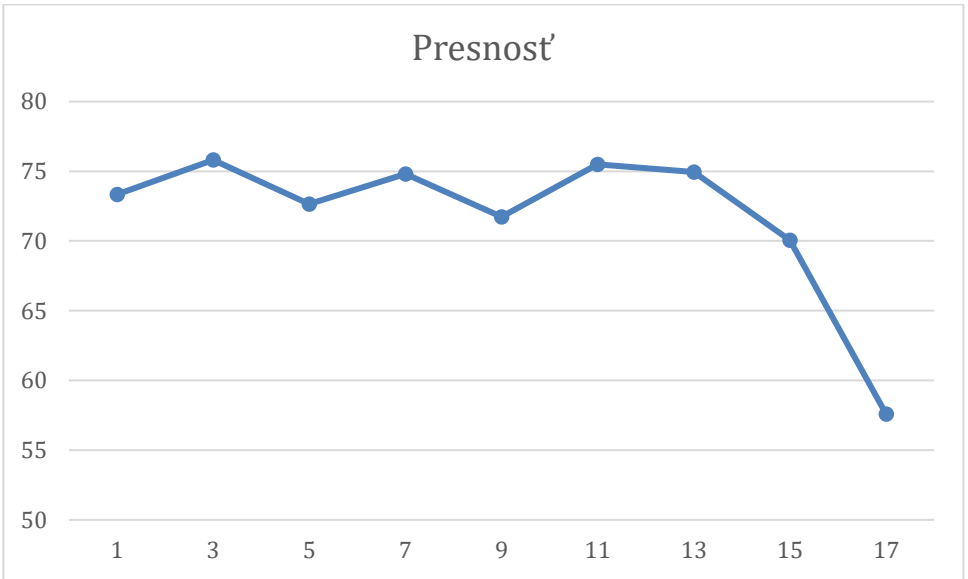


Graf 2: Časy behov

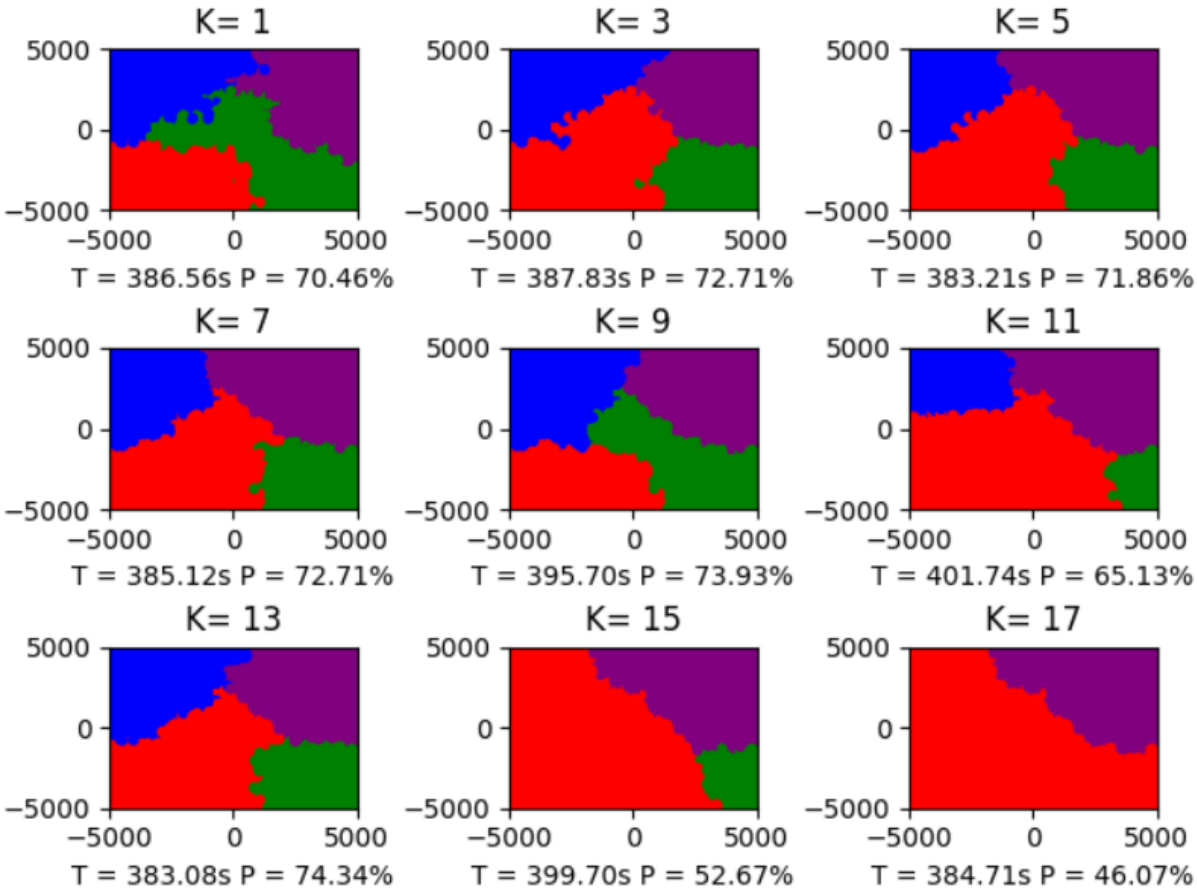
k = 1, 3, 5, 7, 9, 11, 13, 15, 17



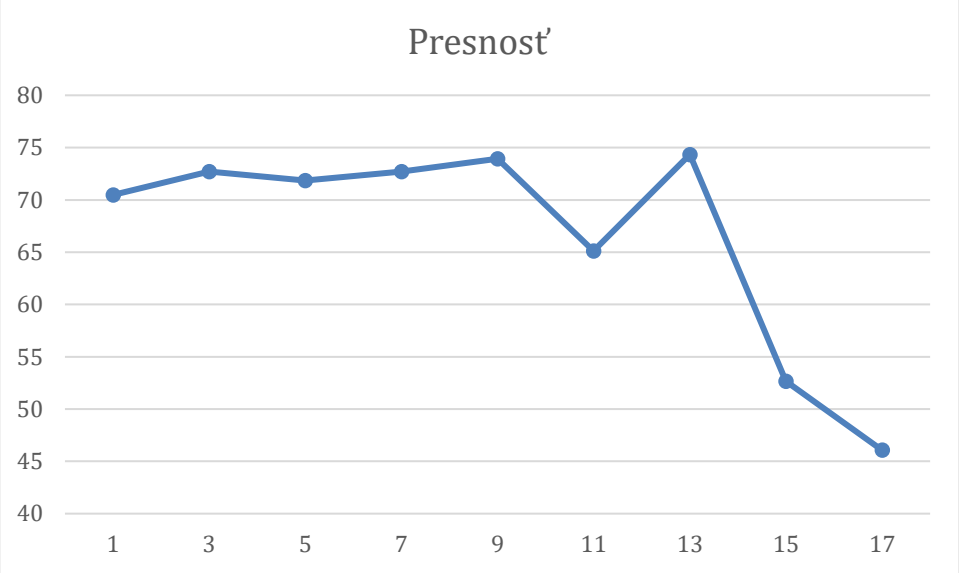
Obrázok 6: Beh 1 k = 1 až k = 17



Graf 3: Porovnanie presnosti pre rôzne hodnoty k, beh 1



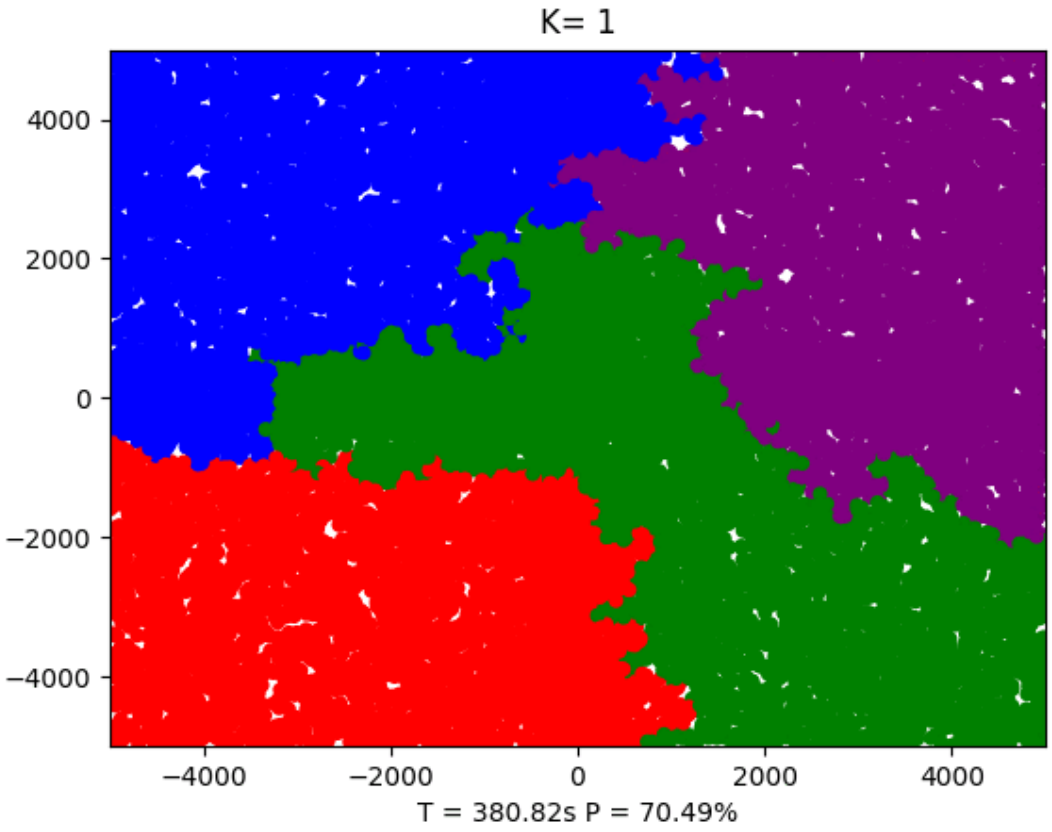
Obrázok 7: Beh 2 k = 1 až k = 17



Graf 4: Porovanie presnosti pre rôzne k, beh 2

k = 1 až 20

Vďaka implementácií rozhodovania v prípade rovnakého počtu susedov s rovnakými farbami sme mohli testovať aj pre párne hodnoty k.



Zhodnotenie

Výsledky nie sú úplne optimálne, dôvodom je hlavne absencia optimalizácie kNN algoritmu. Časy sú závislé aj od vyťaženia pc, na ktorom som pri testovaní robil aj iné projekty, ktoré vyťažovali výkon počítača. Aj tak sa však dalo testovať a doba nebola taká dlhá, v priemere do 400 sekúnd (Graf 2).

Presnosti sa líšili podľa použitej hodnoty *k*, ako sme to predpokladali. Najlepšia presnosť hodnoty k sa líši podľa seedu, podľa ktorého sme tvorili náhodné body. Záleží veľmi na poradí vkladaných bodov a aj na ich pozícií.

Vo všeobecní platilo, že moc malé alebo moc veľké hodnoty mali menšiu presnosť, ako tie medzi nimi.

Vylepšením môže byť implementácia grid systému alebo interpretácia kódu cez pypy.