Peter Brenkus, ID: 120762

Dokumentácia
Umelá inteligencia
Zadanie č. 3
2023 – 24

Cvičiaci: Ing. Juraj Vincúr, PhD.
Čas cvičení: Štvrtok 18:00

Peter Brenkus, ID: 120762

Table of Contents

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, -1400]

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 (s farbou podľa klasifikácie) a vráti triedu, ktorú pridelila pre tento bod. Na klasifikáciu použite k-NN algoritmus, pričom k môže byť 1, 3, 7 alebo 15.

Cvičiaci: Ing. Juraj Vincúr, PhD.
Čas cvičení: Štvrtok 18:00

Peter Brenkus, ID: 120762

Na demonštráciu Vášho klasifikátora vytvorte testovacie prostredie, v rámci ktorého budete postupne generovať nové body a klasifikovať ich (volaním funkcie classify). Celkovo vygenerujte 40000 nových bodov (10000 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 a 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. Uveďte aj vizualizácie viacerých pokusov. V závere zhodnoťte dosiahnuté výsledky ich porovnaním.

Implementácia

Program som robil v programovacom jazyku Python, verzii 3.11 v programe JetBrains PyCharm 2023.2.5 s využitím knižnice numpy na optimalizáciu kódu, knižnicou matplotlib na vizualizáciu výsledkov a knižnicou random na generovanie náhodných súradníc.

Na začiatku programu si podľa zadania vytvorím pole 40 tisíc bodov. Body sa generujú zakaždým z inej triedy, dokopy je ich 10000 z každej. Na generovanie náhodných súradníc podľa farby slúži funkcia *generate_random_point().* Táto funkcia vygeneruje náhodné súradnice z rozsahu danej farby s 99% šancou a s 1% šancou z celého priestoru. Následne pozrie, či už neboli tieto súradnice vygenerované. Nový bod vracia iba ak je nový – nebol ešte vygenerovaný.

```
points = []
for i in range(10000):
    label = 'R'
    x, y = generate_random_point(label)
    points.append([x, y, label])
    label = 'G'
    x, y = generate_random_point(label)
    points.append([x, y, label])
    label = 'B'
    x, y = generate_random_point(label)
    points.append([x, y, label])
    label = 'P'
    x, y = generate_random_point(label)
    points.append([x, y, label])
```

Cvičiaci: Ing. Juraj Vincúr, PhD.
Čas cvičení: Štvrtok 18:00

Keď je pole bodov vytvorené, spustí sa hlavná slučka, v ktorej sa pre každú hodnotu k, pre ktorú máme klasifikátor testovať, vytvorí objekt klasifikátoru.

Objekt má v sebe pole bodov, hodnotu k – počet susedov, podľa ktorých nové body klasifikuje a všetky potrebné funkcie na klasifikáciu a pridanie bodov do poľa.

Vytvorené body sa postupne klasifikujú a pridajú do jeho zoznamu. V premennej reclassified si podľa zadania ukladám počet bodov, ktorých farba sa klasifikáciou zmenila. Po klasifikovaní všetkých 40000 bodov sa pomocou funkcie *visualize()* body graficky zobrazia a do konzoly sa vypíše štatistika úspešnosti klasifikácie.

Funkcia *classify()* priradí triedu vstupným súradniciam (x, y) na základe vzdialeností od najbližších susedov. Postupne zvyšuje polomer vyhľadávania, až nájde dostatočný počet susedov na klasifikáciu. Na záver vráti triedu, ktorá prevláda medzi najbližšími susedmi.

Funkcia *calculate_distances()* vypočíta vzdialenosti bodov od vstupných súradníc (x, y) v polomere vyhľadávania a zoradí ich od najmenšej po najväčšiu.

```python
k_values = [1, 3, 7, 15]
for k in k_values:
    print(f'Running k-NN with k={k}')
    classifier = KNNClassifier(k)
    classifier.add_initial_points()
    reclassified = 0

    for i in range(40000):
        label = classifier.classify(points[i][0], points[i][1])
        if label != points[i][2]:
            reclassified += 1
        classifier.add_point(points[i][0], points[i][1], label)

    title = f'k-NN Classification (k={k})'
    visualize(classifier, title)
    print(f"Reclassified {reclassified} points")
    print(f"Accuracy: {100 - reclassified / 40000 * 100}%")
```

```python
def classify(self, x, y):
    search_radius = 150.0
    min_points_required = self.k

    while True:
        distances, indexy = self.calculate_distances(x, y, search_radius)
        valid_indexy = [index for index in indexy if index != -1]

        if len(valid_indexy) >= min_points_required:
            k_nearest = distances[:self.k]
            counts = {'R': 0, 'G': 0, 'B': 0, 'P': 0}
            for label, _ in k_nearest:
                counts[label] += 1

            return max(counts, key=counts.get)

        search_radius *= 2.0

# vrati najblizsie body zoradene podla ich vzdialenosti
1 usage
def calculate_distances(self, x, y, search_radius):
    distances, indexy = [], []

    for label, points in self.points.items():
        if len(points) > 0:
            dist = np.linalg.norm(points - np.array([x, y]), axis=1)
            indexy.extend(np.argwhere(dist <= search_radius).flatten())
            distances.extend(list(zip([label] * len(dist), dist)))

    # zoradi body podla vzdialenosti
    distances.sort(key=lambda x: x[1])
    sorted_indexy = [index for _, index in distances]
    return distances, sorted_indexy
```

Peter Brenkus, ID: 120762

Optimalizácia algoritmu

Keďže algoritmus pre klasifikovanie bodov je pri počte dát, na ktorom ho máme testovať časovo náročný, v prvej verzii bežal pre k = 1 vyše pol hodiny, pre vyššie k som to radšej ani netestoval . Kód som išiel rovno optimalizovať.

Prvou optimalizáciou bolo prepísanie matematických operácii a štandardných dátových typov jazyku Python pomocou knižnice numpy. Táto knižnica totiž podporuje operácie nad celými poľami a poskytuje optimalizované algoritmy pre rôzne matematické operácie. Tieto výhody sú v kóde využité vo funkcii *calculate_distances()*:

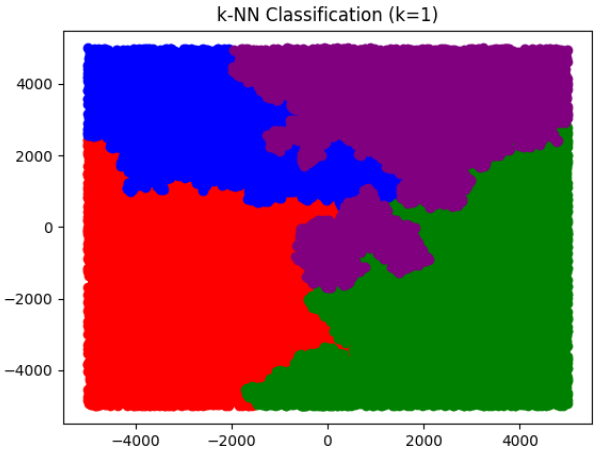
```
for label, points in self.points.items():
    if len(points) > 0:
        dist = np.linalg.norm(points - np.array([x, y]), axis=1)
        indexy.extend(np.argwhere(dist <= search_radius).flatten())
        distances.extend(list(zip([label] * len(dist), dist)))
```

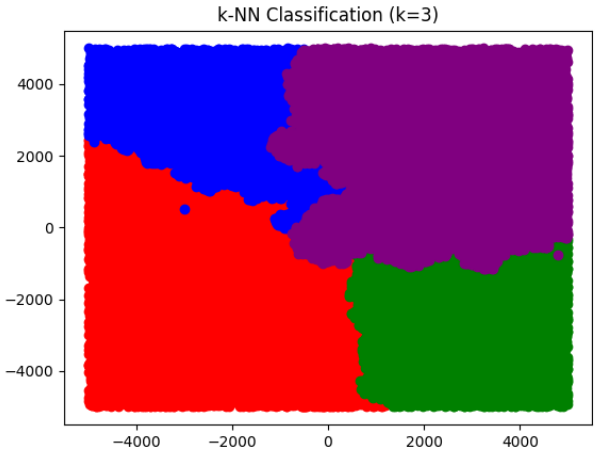
Druhou optimalizáciou bolo zavedenie polomeru vyhľadávania do logiky počítania vzdialeností najbližších bodov. Funkcia síce stále vypočítava vzdialenosti všetkých ostatných bodov, no výsledné pole vzdialeností sa zmenší o všetky také vzdialenosti, ktoré sú mimo polomeru vyhľadávania. To znamená, že program z tohto poľa vymaže body, ktoré sú príliš ďaleko. Toto je dôležité, lebo následne sa všetky body so vzdialenosťou menšou ako je tento polomer zoraďujú podľa vzdialenosti od najmenšej po najväčšiu. Toto zoraďovanie je obrovským faktorom pri časovej zložitosti algoritmu, čím menej prvkov treba zoradiť, tým lepšie.

Testovanie

Optimalizáciami algoritmu sa mi podarilo znížiť čas behu algoritmu na približne 5 minút pre každú hodnotu k, takže som začal testovať. Testovanie prebehlo vždy na tej istej množine bodov.

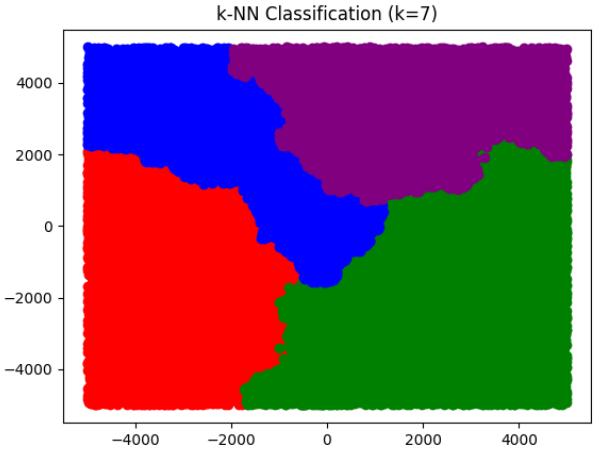
Pokus č. 1

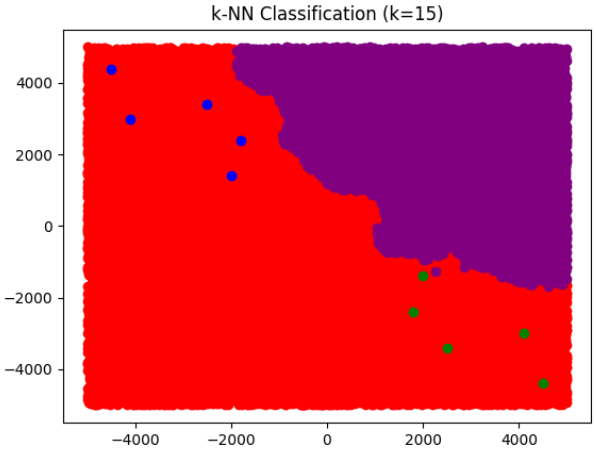




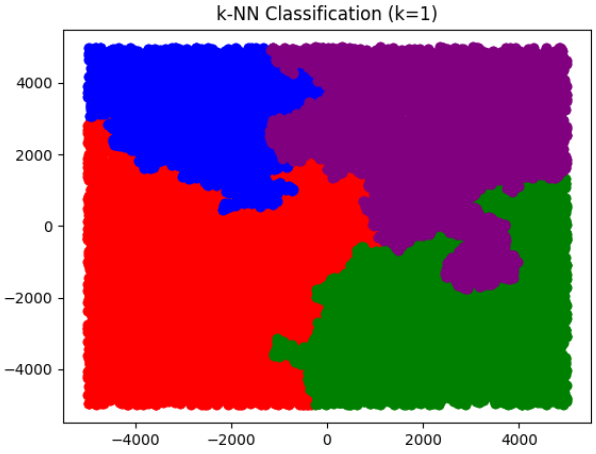
Cvičiaci: Ing. Juraj Vincúr, PhD.
Čas cvičení: Štvrtok 18:00

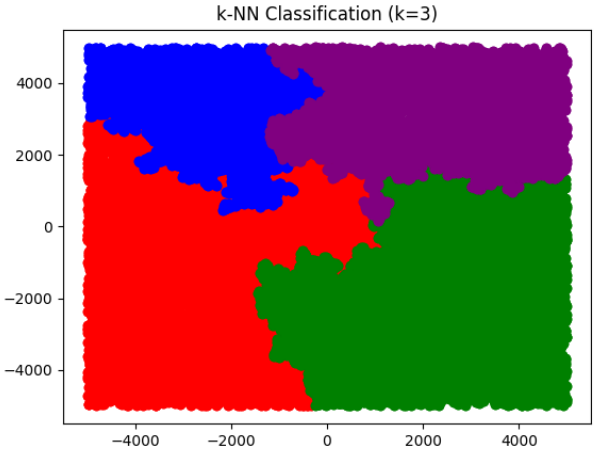
Peter Brenkus, ID: 120762

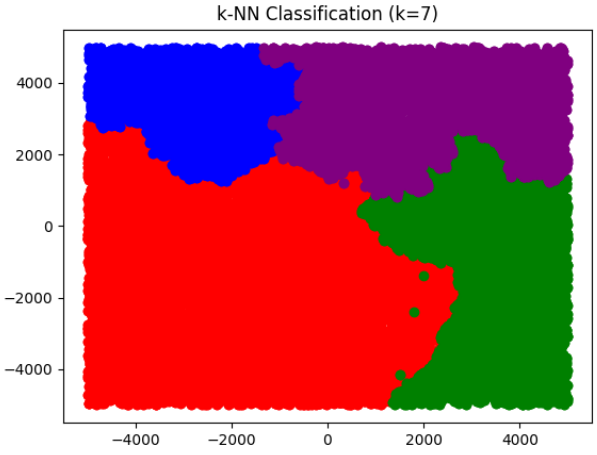


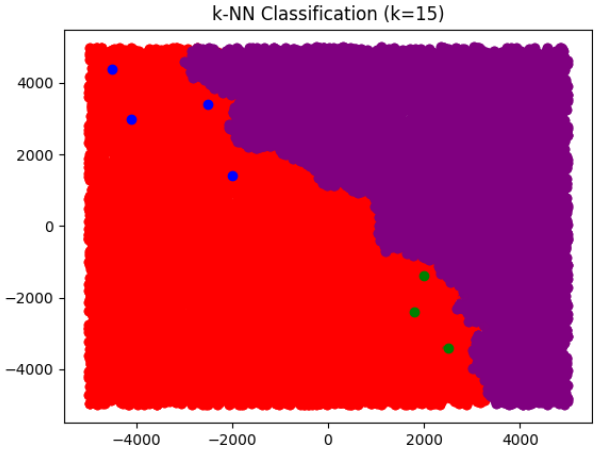


Pokus č. 2









Cvičiaci: Ing. Juraj Vincúr, PhD.
Čas cvičení: Štvrtok 18:00

Peter Brenkus, ID: 120762

Pre každú plochu sa do konzoly vypíšu aj informácie o tom, koľkým bodom sa klasifikovacím algoritmom zmenila farba a percento úspešnosti. Úspešnosť je rátaná ako podiel nereklasifikovaných bodov a celkového počtu bodov.

Výpis v konzole pre oba pokusy:

Pokus 1

```
Running k-NN with k=1
Reclassified 11582 points
Accuracy: 71.045%
Running k-NN with k=3
Reclassified 9656 points
Accuracy: 75.86%
Running k-NN with k=7
Reclassified 11499 points
Accuracy: 71.2525%
Running k-NN with k=15
Reclassified 20858 points
Accuracy: 47.855000000000004%
```

Pokus 2

```
Running k-NN with k=1
Reclassified 10706 points
Accuracy: 73.235%
Running k-NN with k=3
Reclassified 11582 points
Accuracy: 71.045%
Running k-NN with k=7
Reclassified 14276 points
Accuracy: 64.31%
Running k-NN with k=15
Reclassified 20860 points
Accuracy: 47.85%
```

Záver

Z testov je viditeľné, že so zvyšujúcim sa počtom kontrolovaných susedov sa reklasifikuje viac bodov. Prvé tri hodnoty k majú približne rovnakú úspešnosť, no pri k = 15 sa reklasifikuje výrazne viac bodov a úspešnosť výrazne klesá.

Čo sa týka ďalších optimalizácii, program som nekompiloval pomocou PyPy interpretera, lebo pracujem s macOS a na internete som nenašiel zrozumiteľný postup ako to urobiť. Skúšal som to aj zo spolužiakom, ktorému sa to podarilo, no inštalácia PyPy mi stroskotala na tom, že používam staršiu verziu OS, tú však meniť ešte nechcem. PyPy navyše ani nepodporuje matplot knižnicu, ktorú používam na vizualizáciu výsledkov, čiže by som musel okrem vynútenej aktualizácie OS ešte aj meniť proces vizualizácie.

Program aj bez PyPy beží okolo 5 minút, čo je vzhľadom na rozsah počtu bodov rozumné. Z testovania je vidieť aj že program beží správne, moje výsledky sú podobné tým, ktoré boli uvedené v zadaní pre kontrolu.

Cvičiaci: Ing. Juraj Vincúr, PhD.
Čas cvičení: Štvrtok 18:00