Autor: Martin Pažický
ID: 103086

Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Umelá inteligencia

Zadanie 4a

Klasifikácia

Autor: Martin Pažický

Vedúci cvičení: Ing. Ivan Kapustík

Akademický rok: 2020/21

Autor: Martin Pažický
ID: 103086

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 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 volaním ich (funkcie classify). Celkovo vygenerujte 40000 nových bodov (10000 z každej triedy). Súradnice nových bodov generujte náhodne, pričom nový bod by mal mať za každým 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 $Y > -500$

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.

V závere zhodnoťte dosiahnuté výsledky ich porovnaním.

Autor: Martin Pažický
ID: 103086

Riešenie

1) Hlavná myšlienka

Hlavný cyklus programu je jednoduchý – vygeneruje sa bod podľa zadania, nájde sa k najbližších susedov a podľa toho, ktorá farba u nich bola najviac krát zastúpená sa určí farba nového bodu. Ohodnotený bod sa potom pridá do pôvodného datasetu a klasifikácia ďalších bodov už bude ovplyvnená aj novo pridanými bodmi.

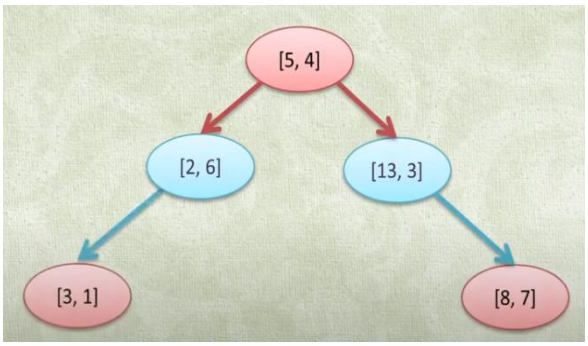
Nosná časť zadania je hľadanie najbližších susedov bodu. V prípade, že dáta nie sú vhodne reprezentované, je pre nájdenie najbližších susedov nutné prejsť všetky body z datasetu. Týmto postupom dramaticky narastá časová náročnosť a pre dataset o veľkosti 40000 by program bežal veľmi dlho.
Rozhodol som sa problém hľadania najbližších susedov preto riešiť pomocou dátovej štruktúry k-d tree. Táto štruktúra rozdelí priestor na oblasti a vyhľadávanie najbližších susedov značne urýchli.

2) Reprezentácia údajov

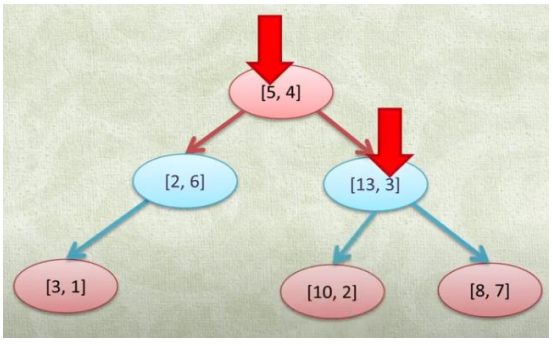
2D body reprezentujem pomocou triedy Point, ktorá pozostáva zo súradníc x,y a farby bodu. Súradnice sú vyjadrené celými číslami a farba môže nadobúdať hodnoty od 1 do 4, pri čom každé číslo zodpovedá jednej farbe.
Dataset, do ktorého sa body pridávajú je k-d strom. Na úvod sa do neho pridajú počiatočné body a potom sa postupne pridávajú aj novo vygenerované.

3) K-d Strom

K-d strom je dátová štruktúra, ktorá sa využíva na rozdelenie k-rozmerného priestoru a usporiadanie bodov v ňom.
K-d strom je veľmi podobný obyčajnému BST, až na to, že namiesto 1 hodnoty každý vrchol uchováva k hodnôt (v prípade tohto zadania je k rovné 2 a hodnoty sú súradnice x a y).
Keď sa do stromu pridáva nový vrchol, rovnako ako pri BST sa porovnajú hodnoty a pokiaľ je hodnota menšia, ide sa doľava, pokiaľ väčšia doprava.
To, aká hodnota sa porovnáva je určené aktuálnou hĺbkou v strome. Pri dvoch hodnotách sa len jednoducho strieda x a y.



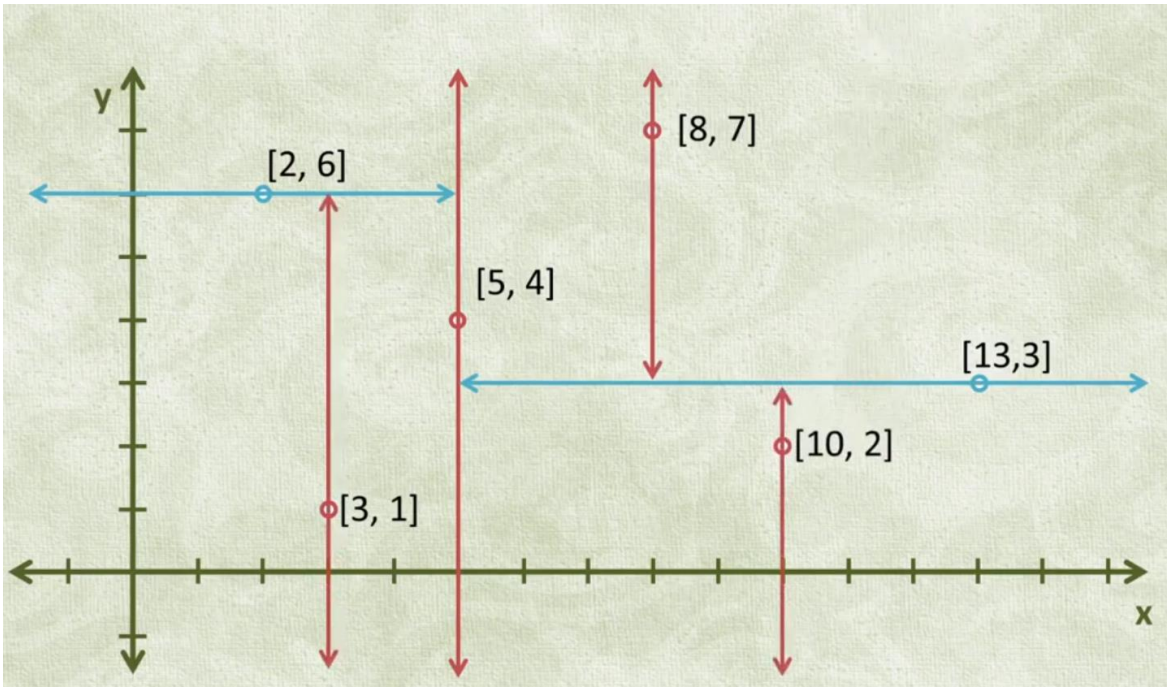
Obrázok 1: k-d strom pred pridaním bodu [10,2]



Obrázok 2: k-d strom po pridaní bodu [10,2]

Na obrázku sú 2 typy vrcholov – červené a modré. V červených sa porovnáva súradnica x, v modrých y.

Vizualizácia toho, ako strom z obrázku 2 rozdelí priestor vyzerá nasledovne:



Obrázok 3: rozdelenie priestoru k-d stromom

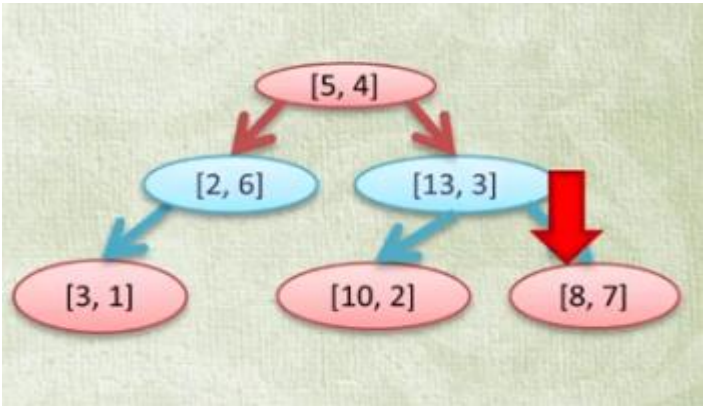
Keď si zoberieme napríklad koreň stromu (bod [5,4]), je vidieť ako delí priestor na 2 časti podľa osi x. Pri ďalších vrcholoch stromu platí to isté.

Implementácia stromu je v mojom zadaní pomerne jednoduchá. Vrchol pozostáva z 2D bodu (trieda Point) a ukazovateľa na pravého a ľavého potomka. Na skonštruovanie stromu stačí operácia insert, ktorú mám v rekurzívnej forme a opäť je takmer totožná ako pri BST.

4) Hľadanie najbližšieho suseda

Keď je k-d strom skonštruovaný, dá sa využiť na efektívne hľadanie najbližšieho suseda. Najprv popíšem hľadanie 1 suseda, potom hľadanie k susedov.

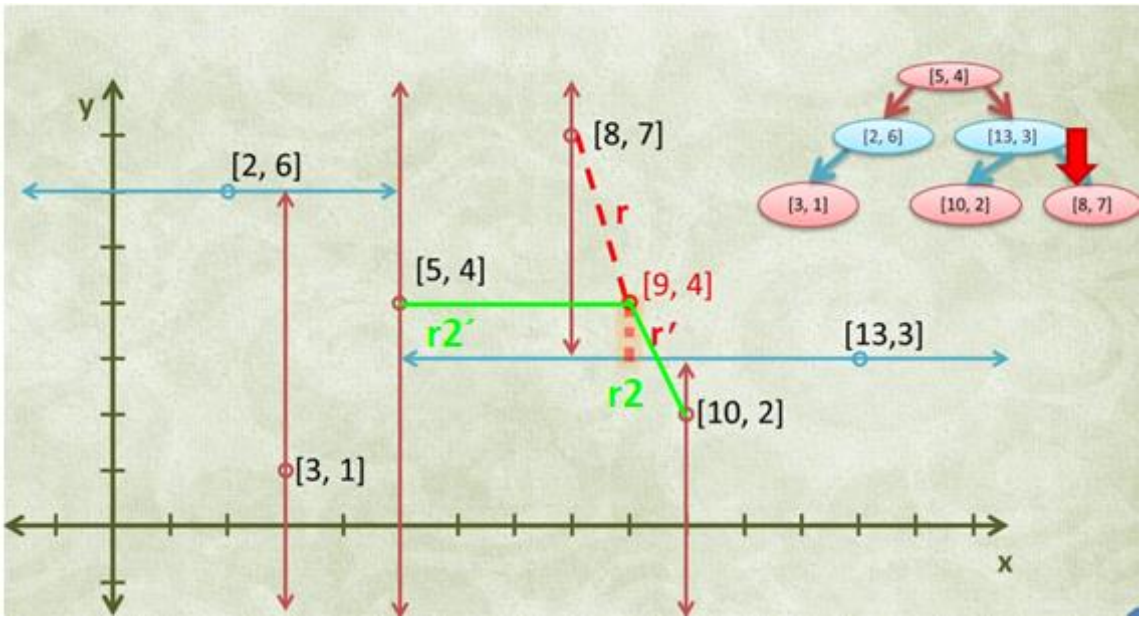
Pri hľadaní najbližšieho suseda sa najprv postupuje ako pri operácií insert (alebo search) – bod, pre ktorý sa hľadajú susedia sa vkladá (skutočne sa však neuloží) a za najbližší bod sa najprv považuje rodič vloženého bodu.



Obrázok 4: hľadanie najbližšieho suseda – bod [9,4]

Na obrázku sa hľadá najbližší sused pre bod [9,4], rodič po vložení a teda najbližší sused by

bol bod [8,7].
Tento výsledok však ešte nie je správny, je potrebné pozrieť aj ostatné vetvy stromu. To sa zabezpečí pomocou techniky backtrackingu.
V strome sa postupuje spätne a kontrolujú sa vetvy, ktoré predtým kontrolované neboli. Netreba však kontrolovať všetky vetvy stromu .
To, ktoré vetvy sa ďalej kontrolujú sa rozhodne porovnaním vzdialenosti doposiaľ najbližšieho suseda a vzdialenosti k priamke rovnobežnej s osou, na ktorej leží bod, ktorý predstavuje ďalšiu oblasť. To, s ktorou osou je priamka rovnobežná sa opäť rozhodne podľa hĺbky v strome.



Obrázok 5: hľadanie najbližšieho suseda (bod [9,4])

Postup popisovaný vyššie je lepšie pochopiteľný z obrázku 5. Za najbližšieho suseda sa najprv považuje bod [8,7]. Bod [13,3] rozdelil plochu a predtým sa skontrolovala len horná časť. Zistí sa však, že v dolnej časti plochy sa môžu nachádzať bližšie body ako je ten, ktorý bol nájdený (r´< r), preto sa ide kontrolovať aj táto vetva.
Príde sa na to, že bod [10,2] je bližšie ako bod [8,7] a uloží sa ako nový najbližší sused.
Potom sa opäť postupuje spätne, avšak pri koreni stromu sa už spravilo rozhodnutie, že sa jeho ľavá vetva kontrolovať nebude, keďže žiadne bližšie body obsahovať nemôže (r2´ > r2).
Najbližším bodom ostane bod [10,2], čo je správny výsledok.
Zanedbanie niektorých vetiev dokáže ušetriť množstvo času a práve to robí hľadanie najbližších susedov pomocou k-d stromu efektívnym. Ľavá vetva koreňu mohla pokojne obsahovať obrovský podstrom, ktorý by sa nemusel vôbec prechádzať.

5) Hľadanie k najbližších susedov

Pri hľadaní k najbližších susedov sa postupuje podobne ako pri hľadaní jedného, avšak namiesto uchovávania jedného najbližšieho suseda sa použije priority queue, kde je uchovaných k susedov.
Priority queue sa usporadúva podľa vzdialenosti k bodu, ku ktorému sa hľadajú najbližší susedia.

Konkrétne v mojej implementácií sa radí najvzdialenejší bod na vrchol priority queue.
Pri rozhodovaní, či prehľadať nejakú oblasť sa potom porovnáva vzdialenosť oblasti a najvrchnejšieho bodu v priority queue.
Tiež som priority queue obmedzil maximálnu veľkosť na hodnotu k. Je to zabezpečené tak, že ak sa veľkosť prekročí, z priority queue sa rovno aj vytiahne najvrchnejšia hodnota.

6) Klasifikácia

Samotná klasifikácia bodu prebieha po nájdení najbližších susedov. Pole najbližších susedov sa prechádza a rátajú sa výskyty jednotlivých farieb. Na počítanie používam hashmapu (kľúč je farba a hodnota je počet).
V počítadle potom nájdem maximum a ešte raz prechádzam počítadlo pre prípad, že sa tam nachádza viacej maxím.
Tento proces by šlo zefektívniť, avšak vzhľadom na to, že počítadlo ma len 4 prvky, som proces nezefektívňoval, keďže vplyv na výkon by bol minimálny.
Pokiaľ farieb s maximálnym počtom zastúpenia bolo viac, vyberie sa z nich výsledná farba náhodne.

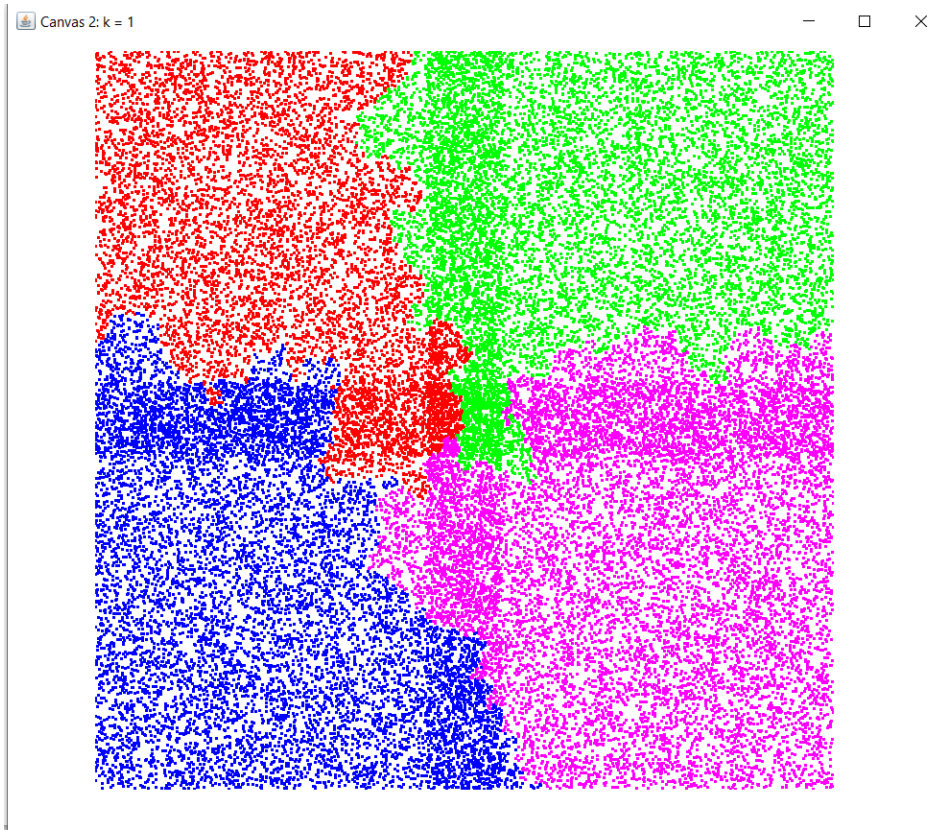
7) Časová zložitosť

Časová zložitosť je ovplyvnená najmä hľadaním k najbližších susedov. S využitím k-d stromu je časová zložitosť pre klasifikáciu n bodov: O(n * log(n) + k*log(k)); prvé n vyjadruje počet bodov, log(n) vyhľadanie najbližšieho suseda v strome (v skutočnosti je táto časť viac ako log(n), keďže strom nemusí byť vyvážený a prechádza sa viac vetiev ako len pri vkladaní) a k*log(k) vyjadruje vkladanie do priority queue obsahujúcej k najbližších bodov).
Do časovej zložitosti som ešte nepripočítal samotnú tvorbu k-d stromu – O(n*log(n)).
Pri brute force algoritme by s podobným využitím priority queue časová zložitosť bola O(n * n + k*log(k)) + O(n).
K-d strom je teda o mnoho efektívnejší, keďže sa tam nevyskytuje kvadratická zložitosť.

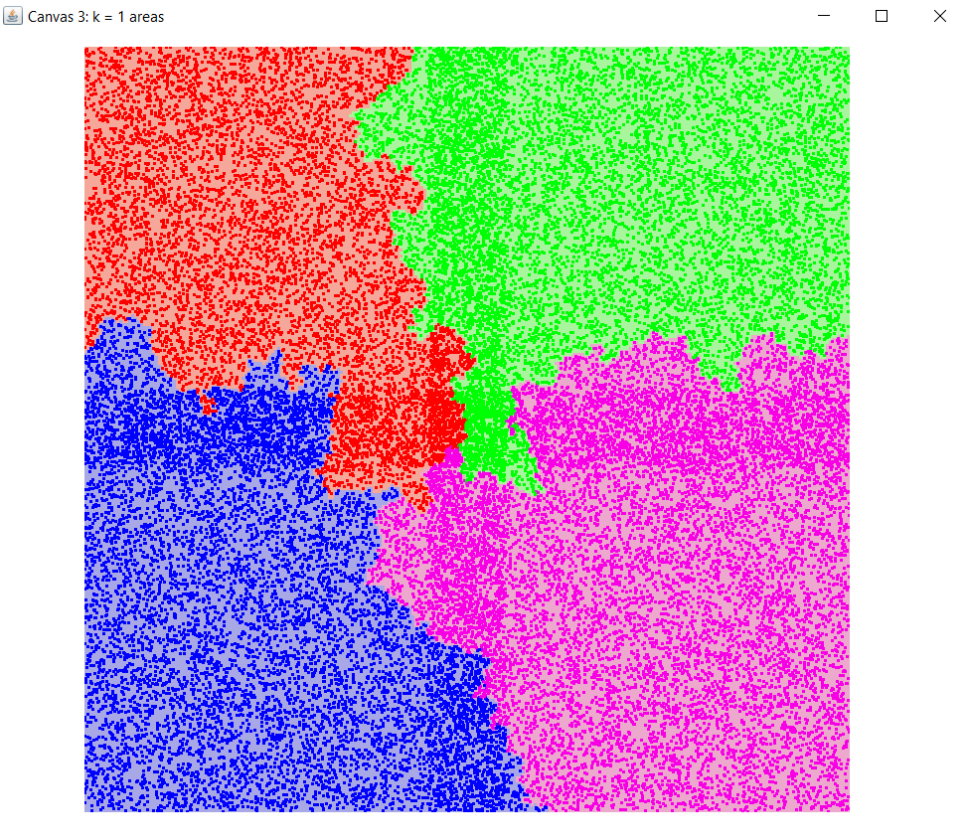
Vizualizácia

Na vizualizáciu vygenerovaných dát som využil knižnicu Java AWT (Abstract Window Toolkit).
Po dokončení generovania bodov sa k-d strom prevedie na pole a body sa z poľa vykresľujú. K súradniciam každého z bodov sa pripočíta konštanta 5000 (keďže plocha začína bodom [0,0]) a vydelia sa takou konštantou aby sa zmestili na plochu (zvolil som rozmery 770x770). Takto sa môžu niektoré body sčasti prekrývať, delenie je však nutné keďže by inak bola potrebná plocha až 10000x10000.
Po vykreslení bodov ešte vyfarbujem aj celú plochu. Postupne generujem body z celej plochy a posielam ich klasifikátoru, ktorý na základe k-d stromu vytvorenom pri prvom generovaní určuje farbu všetkých bodov. Tento postup je výpočtovo pomerne náročný keďže sa musí klasifikovať ďalších cca 200000 bodov (770 * 770 / 3, 3 je šírka bodu) avšak tieto body sa už do pôvodného datasetu nepridávajú a s využitím k-d stromu je potrebný čas minimálny.
Pri vyfarbovaní celej plochy používam priehľadne farby, aby bolo stále vidno aj body, ktoré boli reálne vygenerované. Vykresľovanie transparentných farieb je náročnejšie pre grafiku počítača, preto môže na niektorých zariadeniach táto časť trvať dlhšie.
Na obrázku 7 je vidieť plochu po vykonaní druhej fázy vizualizácie. Časti zafarbené priehľadnou farbou hovoria o tom akú farbu by dostal bod, keby bol vygenerovaný na dané

miesto.



Obrázok 6: vizualizácia – prvá časť



Obrázok 7: vizualizácia – druhá časť

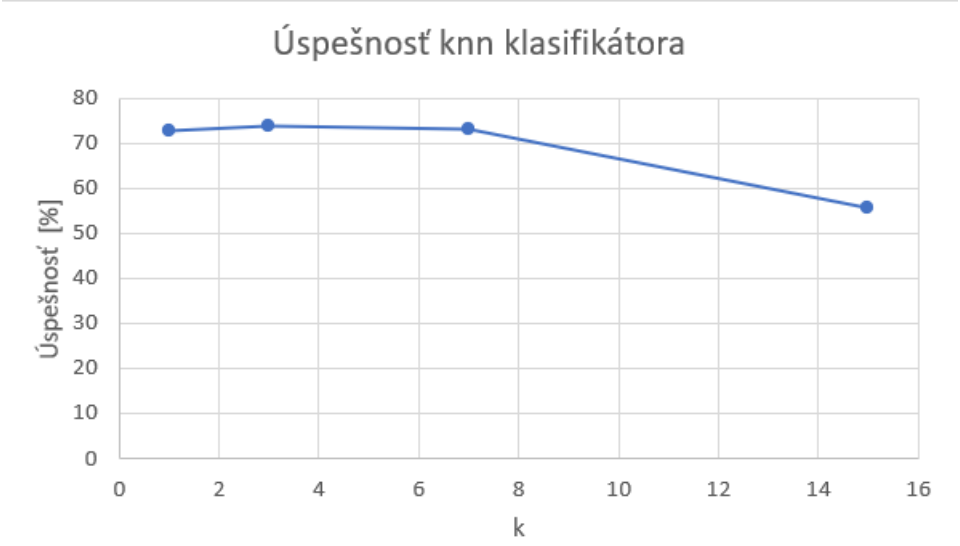
Autor: Martin Pažický
ID: 103086

Testovanie

Klasifikátor sme mali otestovať s rôznymi hodnotami k – 1, 3, 7, 15.

Budem vyhodnocovať úspešnosť klasifikátora a čas trvania pri jednotlivých hodnotách k. Za úspešne ohodnotenie považujem také, pri ktorom klasifikátor ohodnotí bod rovnakou farbou, s akou bol vygenerovaný.

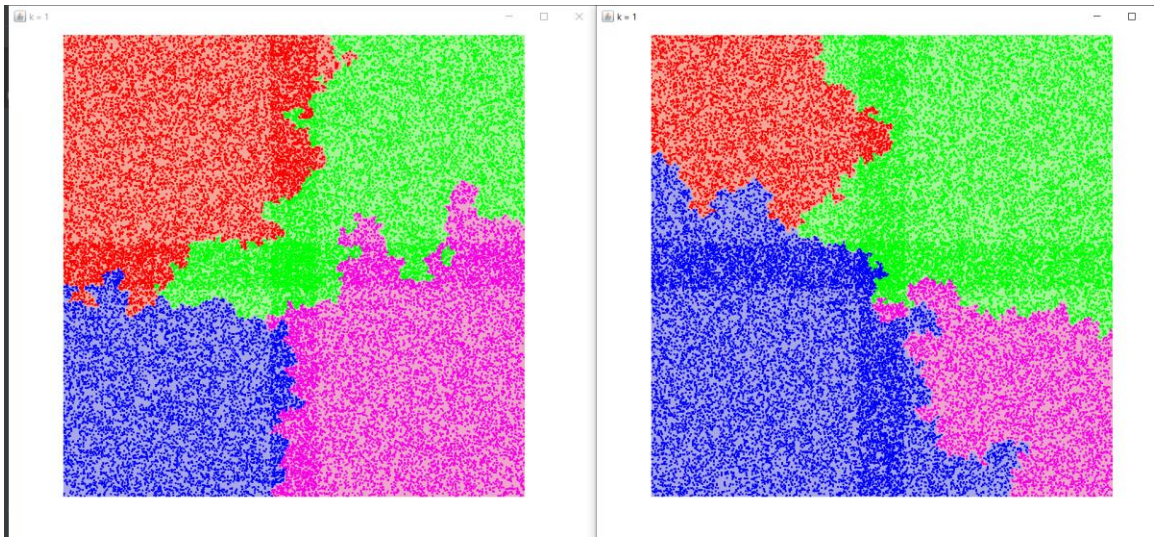
Všetky testy púšťam na datasete o veľkosti 40000 (podľa zadania) a každý test púšťam 1000 krát. Generátor náhodných čísel je spúšťaný so seedmi od 1 do 1000 (prvý test sa spustí so seedom 0, druhý 1 atď.). Výsledky testov je teda možné ľahko zreprodukovať.



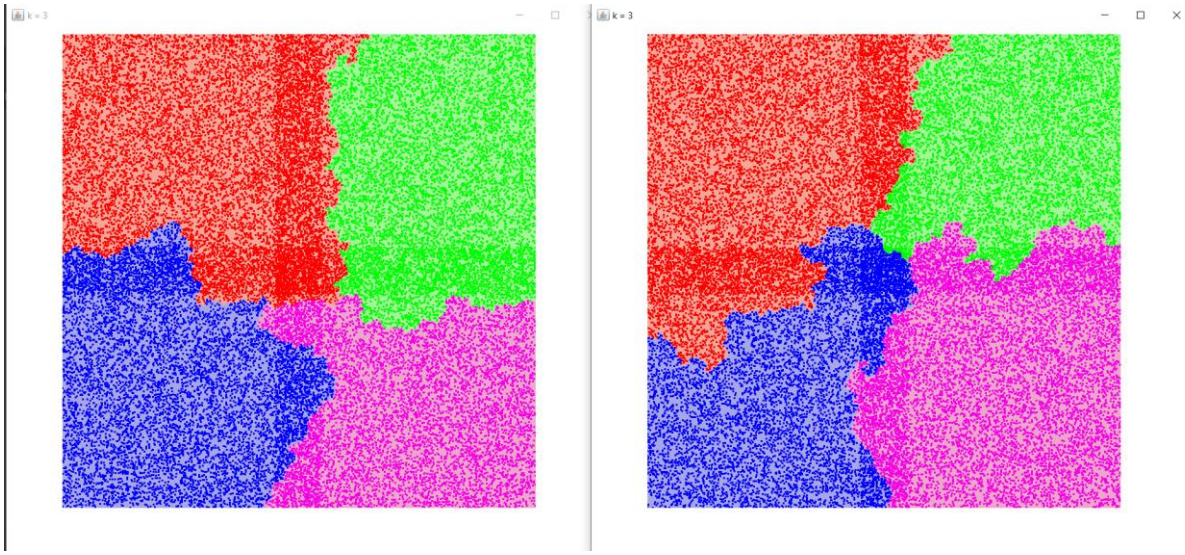
Graf 1: úspešnosť knn klasifikátora

Graf 2: trvanie knn klasifikácie

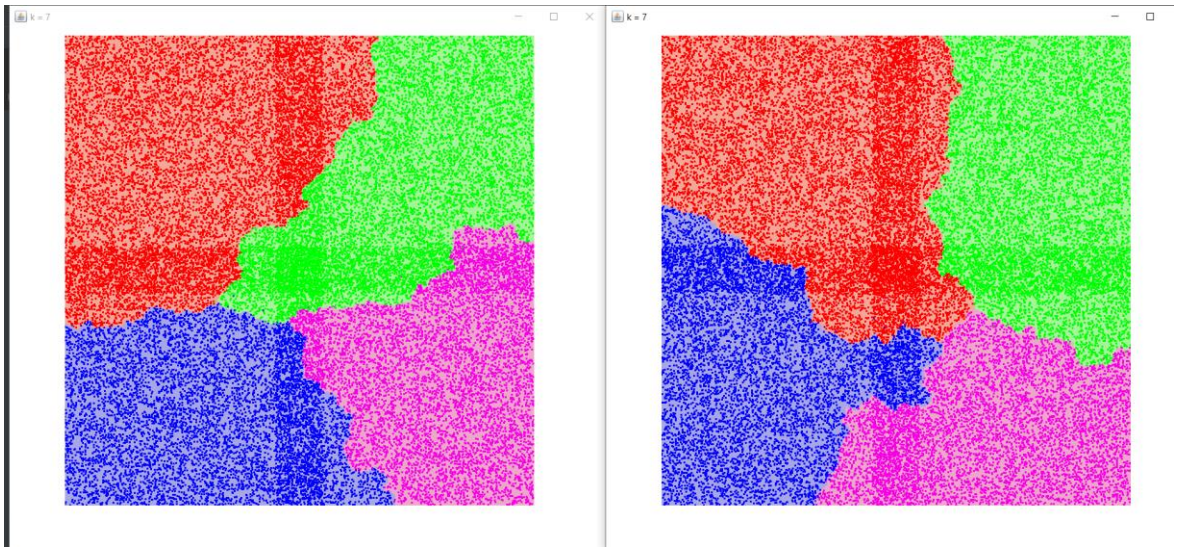
Autor: Martin Pažický
ID: 103086



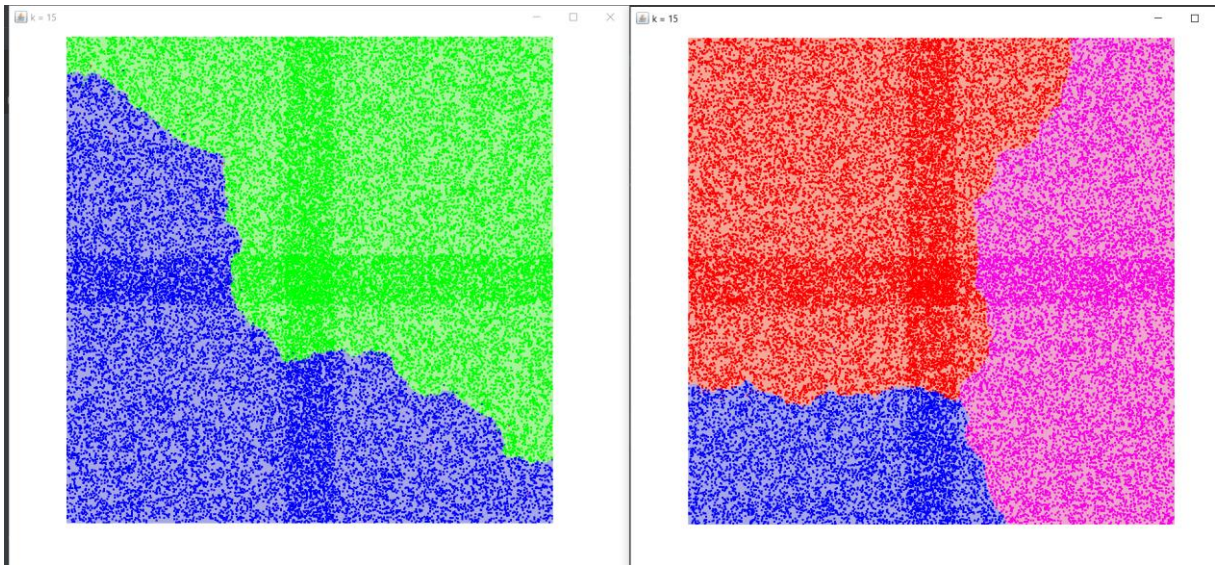
Obrázok 8: vizualizácia, k = 1



Obrázok 9: vizualizácia, k = 3



Obrázok 10: vizualizácia, k = 7



Obrázok 11: vizualizácia, k = 15

Okrem grafov som priložil aj vizualizácie na rôznych hodnotách k (vždy 2 pri rozdielnych random seedoch).

Pri hodnotách k rovných 1, 3 alebo 7 je vidieť len minimálne rozdiely v úspešnosti. Keď je však k rovné 15, úspešnosť klasifikátora prudko klesne.
Na začiatku je v datasete len 20 bodov, preto je klasifikátor pri väčších hodnotách k (napr. 15) chybovejší. Pri klasifikovaní prvých bodov majú vplyv na výslednú farbu aj body zo susedných oblastí, keďže bodov je málo. Bodu sa teda často priradí farba susednej oblasti a tento problém sa môže ďalej nabaľovať. Vysoké hodnoty k preto nie sú vhodné práve z tohto dôvodu, keby bolo na počiatku bodov v datasete viac, vyššie hodnoty k by mohli byť úspešnejšie.
Príliš nízka hodnota k (napr. 1) tiež nemusí byť ideálna, keďže je veľmi citlivá a výsledná farba klasifikátoru môže byť ľahko ovplyvnená zatúlanými bodmi.
Treba teda zvoliť hodnotu k, ktorá nie je ani príliš vysoká ani nízka, v tomto experimente boli najlepšími hodnotami 3 a 7.
Trvanie klasifikácie sa dá popísať už jednoduchšie, čím väčšie k, tým dlhšie klasifikácia trvá, keďže sa musí nájsť viac najbližších susedov.
Testy som spúšťal metódou test v triede Main. Menil som len parameter k.

Užívateľské rozhranie

K programu som pripravil aj jednoduché konzolové užívateľské rozhranie, po spustení užívateľ môže špecifikovať parametre klasifikátoru, následne sa vypíšu informácie o behu programu (čas, úspešnosť). Tiež je možné zvoliť si spôsob vizualizácie, pokiaľ by vizualizácia

s vyfarbením celých plôch trvala príliš dlho, môže sa zvoliť vizualizácia len vygenerovaných bodov.

```
1 - spustit program
2 - koniec
1
zadaj hodnotu k (1 az 20)
7
zadaj mnozstvo bodov ktore chces generovat (MUSI BYT DELITELNE 4!)
odporucane mnozstvo bodov je 40000 (zadanie)
1000000
zadaj seed generatora nahodnych cisel (hociake cislo)
6
-- vygenerovanie bodov trvalo 3,497 sek
-- 255620 bodom bola chybne urcena farba (uspesnost 74,44 %)
zvol sposob vizualizacie
1 - vykreslit len vygenerovane body
2 - zafarbit celu plochu (moze trvat dlhsie)
1
-- body sa teraz vykresluju v druhom okne...
1 - spustit program
2 - koniec
1
zadaj hodnotu k (1 az 20)
1
zadaj mnozstvo bodov ktore chces generovat (MUSI BYT DELITELNE 4!)
odporucane mnozstvo bodov je 40000 (zadanie)
40000
zadaj seed generatora nahodnych cisel (hociake cislo)
99
-- vygenerovanie bodov trvalo 0,154 sek
-- 10421 bodom bola chybne urcena farba (uspesnost 73,95 %)
zvol sposob vizualizacie
1 - vykreslit len vygenerovane body
2 - zafarbit celu plochu (moze trvat dlhsie)
2
```

Obrázok 12: užívateľské rozhranie

Zhodnotenie

Cieľom zadania bolo vytvoriť klasifikátor s knn algoritmom. Pôvodne som na hľadanie najbližších susedov používal brute force algoritmus (prechádzal sa každý bod v datasete a hľadalo sa k extrémov), čo sa však ukázalo ako značne neefektívna stratégia.
Využitím k-d stromu sa proces hľadania najbližších susedov dramaticky zlepšil (časy pri 40000 bodoch padli z 30 – 90s na 0.08 – 0.6s) a tým pádom klasifikátoru nerobili problém ani veľké množstvá bodov.
Verziu s brute force algoritmom som využíval na kontrolu funkčnosti verzie s k-d stromom, výsledky sa zhodovali.

Klasifikátor som otestoval pri rôznych hodnotách k a rozdiely v experimentoch sa dali pozorovať či už na čase, úspešnosti alebo aj pri samotnej vizualizácií.

Zdroje

Informácie o fungovaní k-d stromu ako aj príkladne vizualizácie som čerpal zväčša z videa https://www.youtube.com/watch?v=Glp7THUpGow&ab_channel=StableSort.