Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Klasifikácia bodov – KNN algoritmus

2020/21, ZS

Matej Delinčák

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 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

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.

Opis riešenia

Ako je v zadaní napísané, mojou úlohou bolo použiť k nearest neighbours algoritmus na klasifikáciu bodov na 2D ploche. Na začiatku mám dataset s 20 bodmi, ktoré majú štyri rôzne typy – farby. Vytvorím si zoznam bodov, ktoré budem postupne klasifikovať, a týmto bodom určím aj ich typ. Následne pustím môj klasifikátor postupne na každý bod. Funkcia teda zoberie k najbližších susedov a bod ohodnotí na základe najväčšieho počtu rovnakých typov okolitých bodov. Teda ak v k najbližších susedoch budú traja červenej farby a dvaja modrí, tak bod bude ohodnotený ako červený. A ak sa vygenerovaný typ zhoduje s tým, čo mi vráti klasifikátor, tak je to správne. Ak nie, tak to zarátam ako chybu. Mojim cieľom je porovnať, aký parameter k je najlepší (vráti najmenej chýb) pre daný dataset.

Do môjho klasifikátora som implementoval rôzne variácie:

1. **Spôsob vyberanie k najbližších susedov:**
   a. **Halda / Brute force**
      Pre každý bod predtým než ho idem ohodnotiť, vyrátam jeho vzdialenosť ku každému inému bodu. Tieto vzdialenosti vkladám do minimálnej haldy. Potom vyberiem z tejto haldy k najmenších prvok a pokračujem ďalej vo výpočte. Nevýhoda je, že túto haldu musím vždy postaviť pre každý nový bod. Čo je zložitosť O(n log n).
   b. **KD-tree**
      Táto dátová štruktúra funguje na princípe binárneho vyhľadávacieho stromu. Dokáže si udržiavať konkrétne body (nepracuje so vzdialenosťami ako halda) a vie nájsť najbližší bod k hľadanému bodu. Keďže tento strom nemusím vždy vytvárať nový (nové body len vkladám), tak táto štruktúra ma zložitosť O(log n).

2. **K-NN algoritmus**
   Dostanem k-dlhý list najbližších susedov a z nich si zrátam počet jednotlivých typov. Doimplementoval som aj Wk-NN (weighted kNN algoritmus), ktorý neberie len ich typy, ale bod, ktorý je najbližšie, tak jeho typ má najväčšiu váhu, pri určovaní typu pre nový bod.

3. **Výber typu:**
   a. Zo zrátaných typov vyberiem prvý najpočetnejší typ a tento uznám za výsledok klasifikácie.
   b. Ak je viacero typov s rovnakou početnosťou, tak z nich vyberiem, náhodne jeden a ten použijem ako výsledok.

Ďalej som dorobil aj dve rôzne generovania bodov. Jedno také ako je v zadaní a druhé také, že z každého typu bodu si vyrátam medianový stred a tento prehlásim ako stred gaussovho rozdelenia. Čiže čím je bližšie pri tomto strede, tak tým má bod väčšiu šancu na vygenerovanie.

Ako vizualizáciu som zvolil knižnicu tkinter a body vykresľujem do objektu canvas. Užívateľ má možnosť vykresliť len body, alebo použiť vyfarbenie celej plochy na základe vygenerovaných bodov. Na toto by sa dali použiť už vytvorené knižnice, ktoré ste nám odporučili, ale ja som sa rozhodol to spraviť sám. V podstate moje vykresľovanie plôch funguje tak, že plochu rozdeľujem rekurzívne na štvorce. Ak v tomto štvorci sa nachádzajú len body jedného typu, tak vyfarbím danú plochu. Ak nie, tak daný štvorec rozdelím zase na štvorce. Vizualizáciu ukážem neskôr.

Konkretná implementácia

Ako som už opísal vyššie používam rôzne dátové štruktúry ako haldu alebo kd-strom. Kd-strom som si sám naprogramoval, haldu som len použil z knižnice heapq. Zároveň som sa pohral s vizualizáciou v tkinter. Vytvoril som svoju vizualizáciu plôch v 2d ploche. Ako jazyk som si vybral python, či už pre rýchlosť v akej som bol schopný napísať program, ako aj pre množstvo vstavaných funkcií. Pre zrýchlenie behu programu som použil kompilátor PyPy3. Nevýhodou samozrejme bolo, že python je veľmi pomalý pre takýto typ výpočtu. Zadanie som preto skúsil zrobiť aj v jazyku C, ale tento projekt do odovzdania nepriložím.

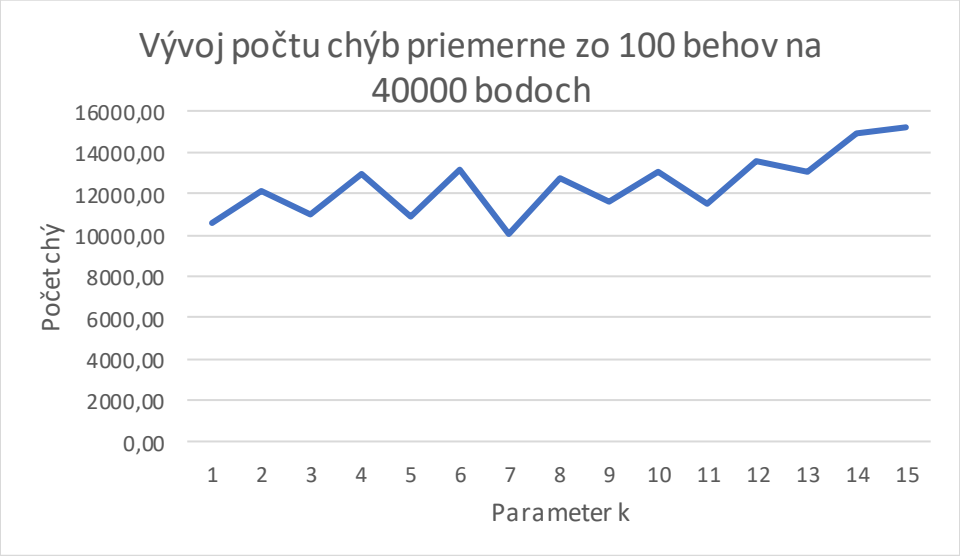
Používateľské prostredie

Užívateľ si môže vybrať s rôznych variácií, ktoré som spomenul vyššie. Navyše som pridal aj možnosť vypnúť pridávanie do pôvodného datasetu. To znamená, že najbližších susedov budem hľadať len medzi pôvodnými bodmi. Užívateľ si môže samozrejme vybrať medzi všetkými variantami, ktoré som implementoval. Vstup, ktorý zadá užívateľ, som pre tento krát neošetroval. Príklad vstupu od užívateľa:

```
Brute force/ kd-tree? 1/2: 2
Knn/ Wknn? 1/2: 1
Pridavat body? a/n: a
Random rozhodnutie o type? a/n: a
Pocet bodov: 40000
Parameter k: 4
```

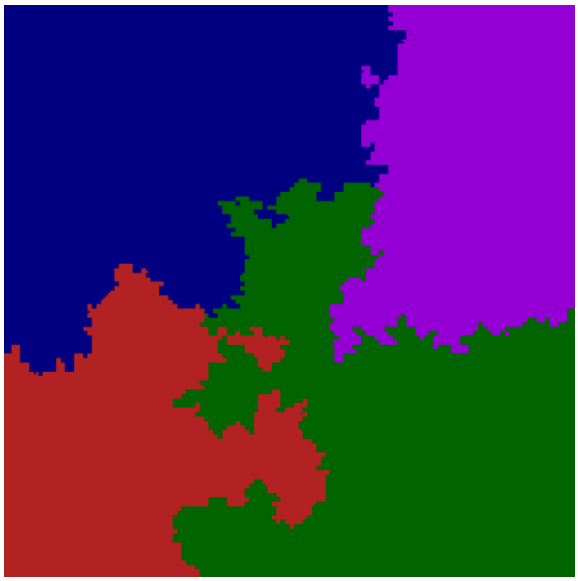
Testovanie

V zadaní máme napísané porovnať 4 hodnoty parametrov, avšak ja som vygeneroval testy pre hodnoty od 1 po 15. Každý test, ktorý je spomenutý v dokumentácií, je vyrátaný na 40 tisíc bodoch. Nasledujúci graf ukazuje vývin počtu chýb k-NN algoritmu s pridávaním bodov do pôvodného datasetu:

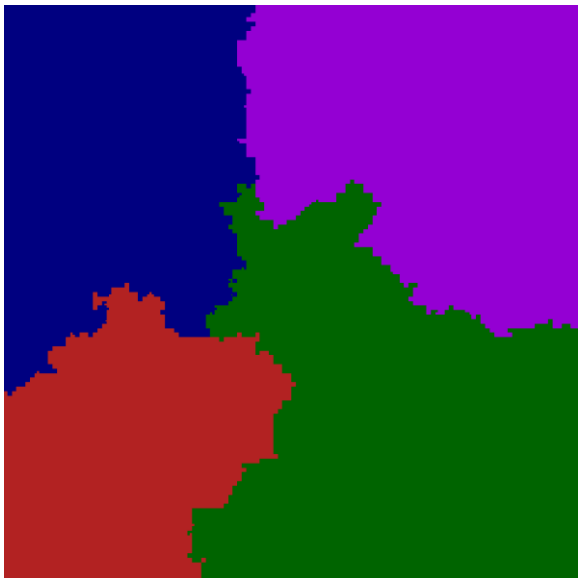


Ako vidno na obrázku tak najmenej chýb mi dal parameter 1, 3,5 a 7. Celkovo do 10 sa to ešte celkom dalo zniesť. Potom to už začalo celkom rásť. Tento graf je priemer zo 100 behov a na 40000 bodoch.

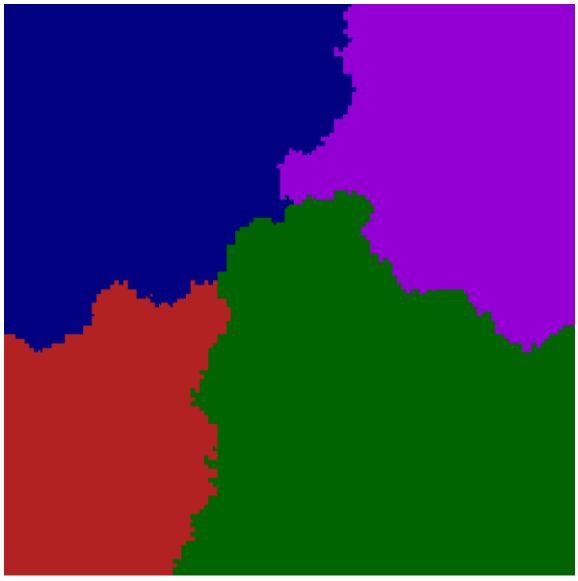
Na obrázkoch teraz ukážem vizualizáciu ako vyzerá plocha bodov pre parametre k = 1, 3, 7, 15.



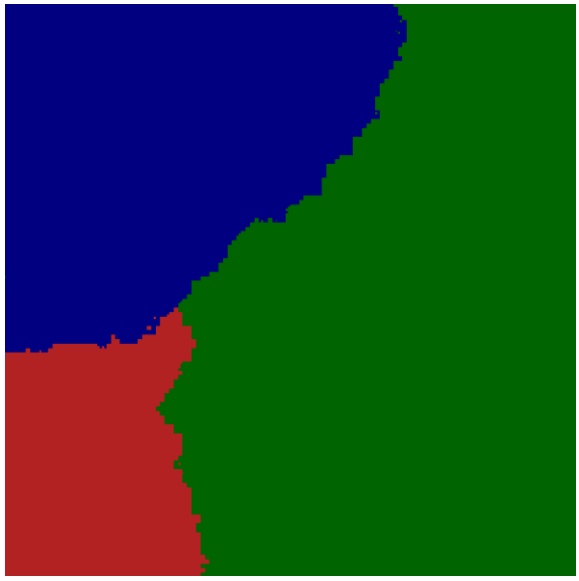
K = 1



K = 3



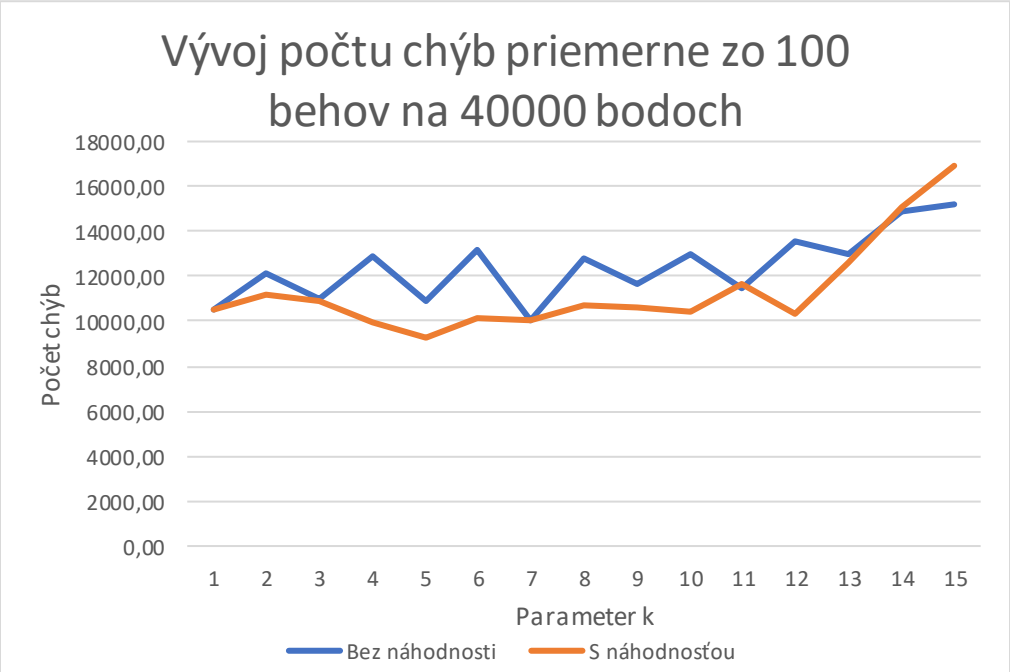
K = 7



K = 15

Ako vidno na obrázkoch najlepší parameter vyšiel 7. Tento najviac dodržuje generovanie bodov zo zadanie a taktiež ma aj najmenej chýb. Najhorší vyšiel 15, kde jedna celá farba bola pohltená inou. Takto veľký parameter je nonsens a teda ho vyhodnocujem ako zlý.

Na grafe vyššie som ukázal ako vyzerá počet chýb, keď zoberiem prvý najväčší počet typov najbližších susedov. Avšak doimplementoval som aj možnosť, kedy sa neberie prvý najväčší počet, ale ak je rovnaký počet najpočetnejších typov viac, tak z nich vyberiem náhodný a ten použijem.

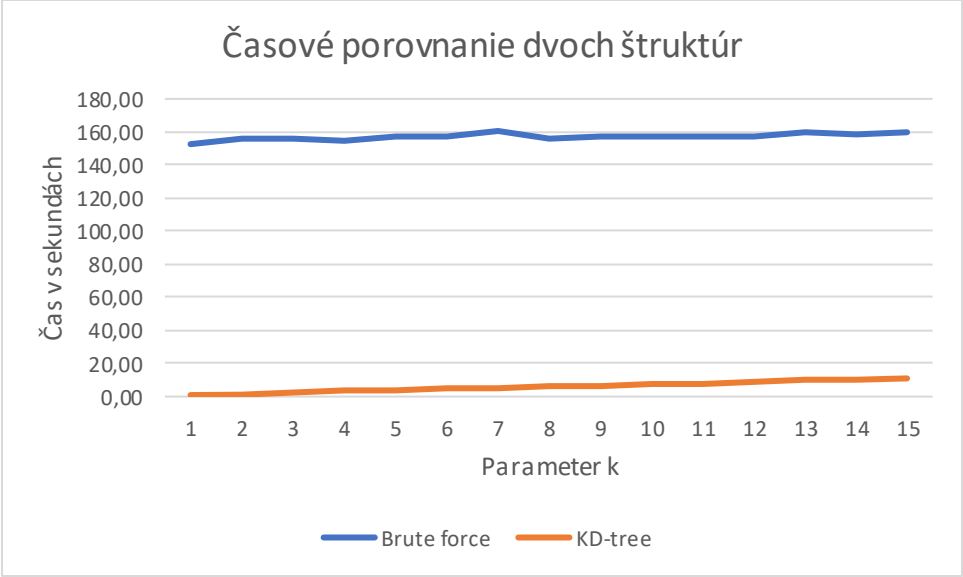


Ako je vidno, tak v priemere zo 100 behov, vychádza náhodne vyberanie farieb lepšie – čiže klasifikácia vytvára menej chýb. Zároveň to krivku aj vyrovnalo. Žiaľ nepochopil som prečo tomu tak je.

Časové porovnanie haldy a KD-stromu

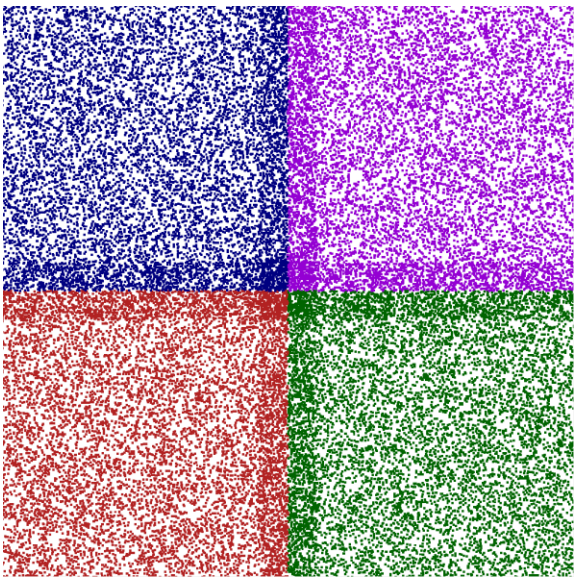
Vytváranie haldy je veľmi pomalé, pretože musím pre každý nový bod vyrátať v podstate kompletne novú haldu. Toto je časovo náročné a neefektívne. A preto som sa rozhodol naštudovať si nejakú dátovú štruktúru, ktorá by mi to urýchlila. Pri hľadaní som našiel práve tento binárny vyhľadávací strom, ktorý dokáže spracovávať aj viac dimenzionálne priestory bodov. Keďže mám pracovať len na 2d ploche, toto nebolo vôbec ťažké implementovať. Táto štruktúra má výhodu, že pre nové body nemusím postaviť nový strom, ale môžem použiť ten čo už mám a tento len rozširujem. Viem doňho vkladať nové body a vyhľadať najbližší bod k danému bodu. Ak chcem nájsť viacero bodov (k najbližších), tak postupne hľadám v strome blízke body a keď ich nájdem, označím tento uzol ako použitý. Pri ďalšom hľadaní tento bod preskakujem.

Vo výsledku je táto štruktúra veľmi rýchla:

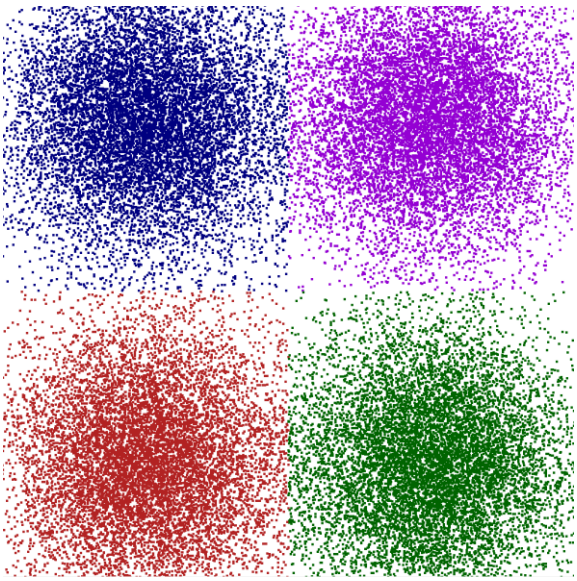


Vytváranie nových bodov viac prirodzene

Ako ste nám odporúčal na cvičení, skúšal som vytvárať nové body nie ako je v zadaní, ale viacej prirodzene ako tomu väčšinou býva. A teda použil som normálne rozdelenie bodov. Pre každý typ bodov z pôvodného datasetu, som si určil ich medoid. Tento som prehlásil ako stred, od ktorého sa budú generovať body. Teda čím ďalej od tohto bodu, tým je menšia šanca pre bod, aby sa vygeneroval.

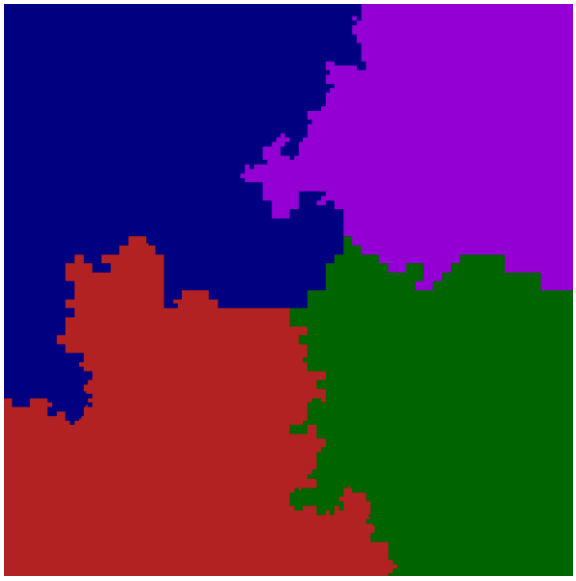


Generovanie bodov ako je v zadaní

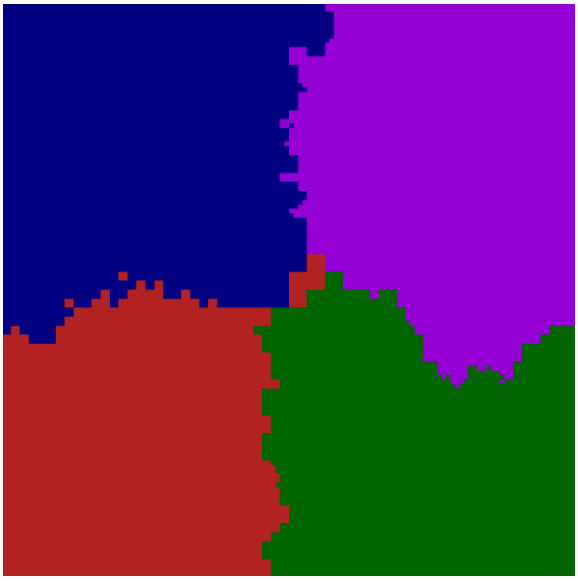


Generovanie bodov pomocou normálneho rozdelenia

Následné obrázky ukazujú ako vyšli výsledné plochy pre rôzne parametre k:



K = 1

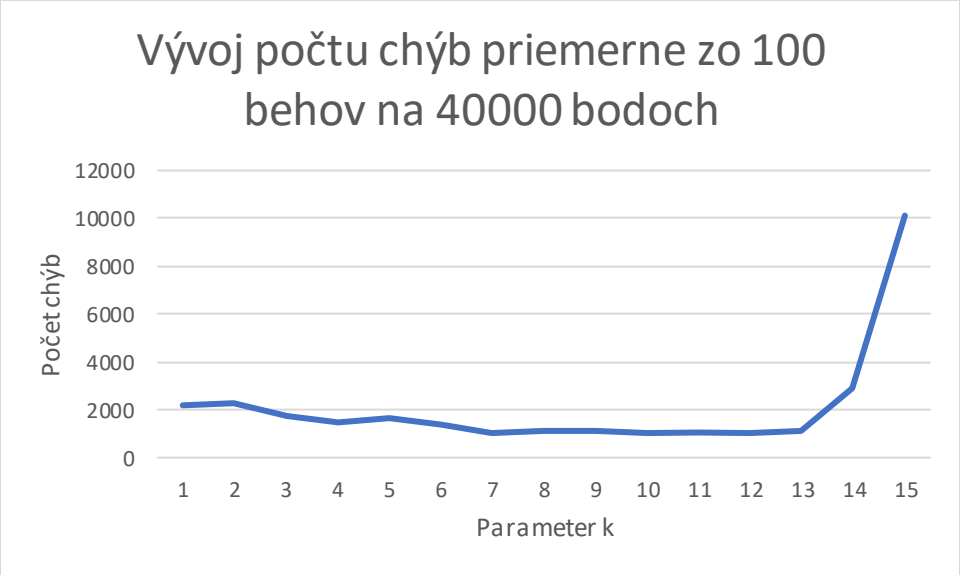


K = 3

K = 7

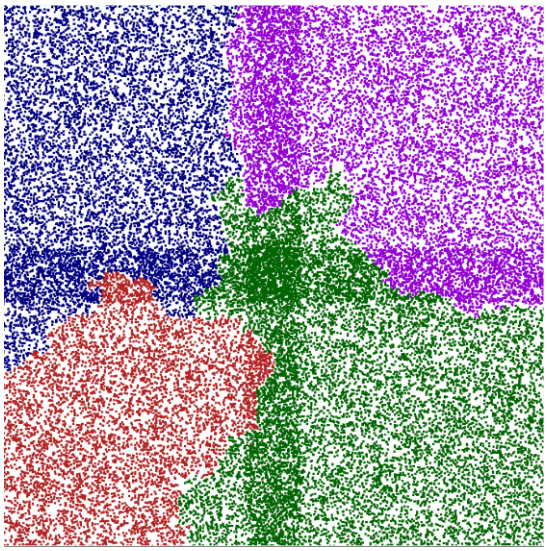
K = 15

Správa sa to veľmi podobne, ako pri generovaní bodov zo zadania, s tým rozdielom, že pre body zo zadania počet chýb postupne rastie. Ale pri normálnom rozdelení to pri parametroch väčších ako 13, začne prudko narastať. Toto je vidno na štvrtom obrázku, kde ostali už len dve farby. Taktiež to je vidno aj na grafe nižšie.

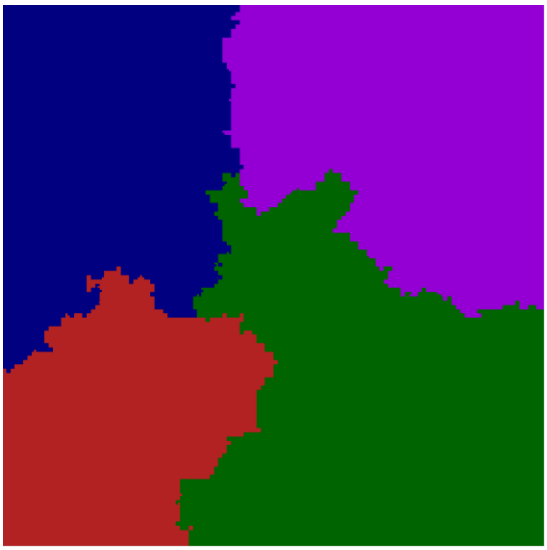


Vizualizácia

Časť zadania je aj vizualizovať výsledky môjho klasifikátora. Rozhodol som sa ručne vykresľovať do canvasu z knižnice tkinter. Do ilustrácií som nevložil súradnicové osi pre ušetrenie miesta a ani nie sú podľa mňa až také dôležité. Jednoducho vľavo hore sú súradnice [-5000, 5000], v strede obrázka je bod [0, 0] a vpravo dole bod [5000, -5000]. Užívateľ má možnosť si vykresli buď samotné body, alebo vyfarbené plochy okolo bodov.

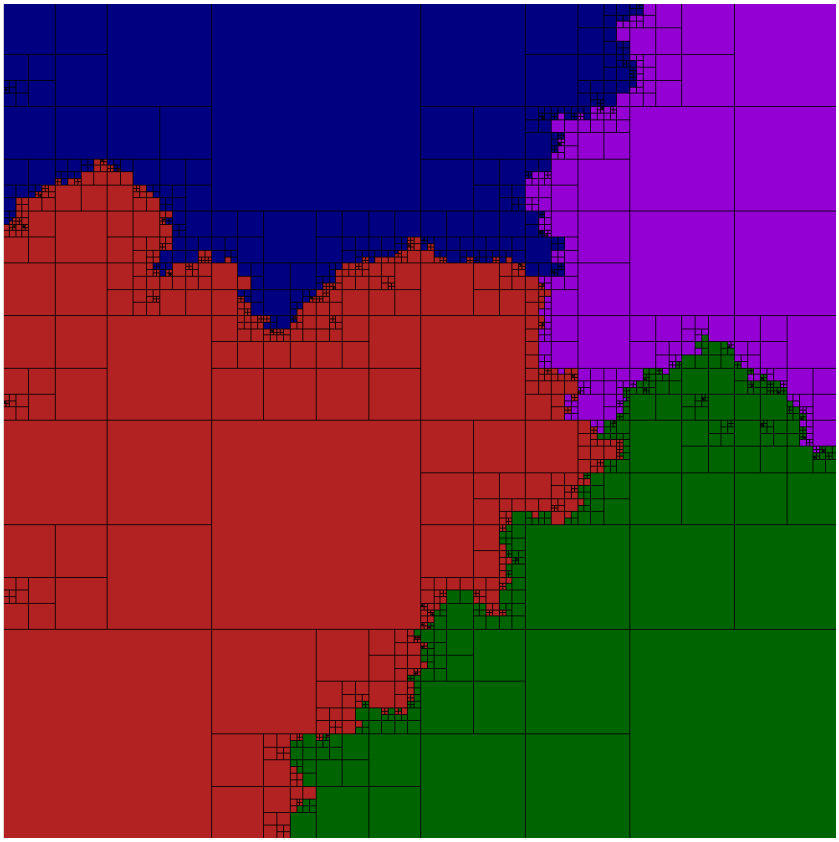


Samotné body



Vyfarbené plochy

Na vyfarbovanie plôch som využil metódu rozdeľuj a panuj. Kde som plochu rozbil na 4 štvorce. Ak v štvorci sa nachádzajú len body rovnakého typu, tak tento štvorec cely vyfarbím. Ak nie, tak tento štvorec zase rozdelím a zavolám rekurzívne túto funkciu. Nižšie uvádzam ukážku rekurzívneho volania, kde som hranice štvorcov obkreslil, aby bolo vidno, ako to funguje:



Záver

Na záver by som zhodnotil, že parameter k veľmi závisí od pôvodného datasetu. Pri skúšaní iného datasetu, vyšlo najlepšie k iné ako pri tom zo zadania. A taktiež aj pri inom generovaní bodov do plochy bolo k odlišné. Tento algoritmus by som ale zhodnotil, že nie je až taký presný ako tie, ktoré sa používajú napríklad v druhom zadaní, ale je veľmi rýchly. A to je jeho obrovská výhoda.

K zadaniu by som povedal len toľko, prosím nenahnevajte sa, ale obtiažnosťou by som ho priradil k strednej škole. Algoritmus sa dá veľmi ľahko pochopiť zo pseudokódu a naprogramovaný som ho mal do pol hodiny. Aby mi toto zadanie dalo aj niečo navyše, tak som sa rozhodol si ho sťažiť už spomínaným kd-stromom a vytvorením vlastnej vizualizácie. Toto navrhujem doplniť aj do zadanie pre budúce ročníky. Ale celkovo by som zhrnul, že zadanie ma bavilo a veľmi rád by som si ho zopakoval.