VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vyhledávání K nejbližších sousedů na základě filtru

Search K nearest neighbors based on a filter

Bc. Jan Jedlička

Vedoucí práce: Doc. Ing. Radim Bača, Ph.D.

Ostrava, 2022

Abstrakt

Techniky pro efektivní vyhledání K nejbližších sousedů (tzv. KNN problém) jsou základem pro mnoho dnešních aplikací. Velmi často se využívají i techniky pro přibližné KNN vyhledávání. Tyto techniky jsou založeny na grafech. Předmětem této práce rozšíření existující implementace pro přibližné KNN vyhledávání o možnost specifikovat filtr. Filtr bude podmínka, která stanoví, které vektory se při prohledávání vynechají.

Klíčová slova

KNN;HNSW;Filter

Abstract

Techniques for effective nearest neighbor search (so-called KNN problem) are the basis for many of today's applications. Techniques for approximate KNN searches are also very often used. These techniques are based on graphs. The subject of this work is to extend the existing implementation for approximate KNN searches with the ability to specify a filter. The filter will be a condition that determines which vectors are omitted when searching.

Keywords

KNN;HNSW;Filter

Poděkování

Rád bych na tomto místě poděkoval vedoucímu semestrálního projektu, kterým byl pan Doc. Ing. Radim Bača, Ph.D., za pravidelné konzultace a poskytnutí mnoha užitečných rad a nápadů pro řešení samotné práce.

Obsah

Seznam použitých zkratek a symbolů

KNN	–	K-nearest neighbors
HNSW	–	Hierarchical Navigable Small Worlds

Kapitola 1

Úvod

Cílem této semestrální práce bylo pochopit a následně naimplementovat HNSW algoritmus vyhledávající K nejbližších sousedů v prostoru n-diemnzionálních vektorů. Následně bylo zapotřebí tuto mou implementaci rozšířit o vyhledávání prvků na základě zadaných filtrů. Filtry jsou booleovsé podmínky omezující hodnoty jednotlivých atributů u prvků pro K vyhledaných sousedů. Veškerá práce byla napsána v jazyku C++. Tento jazyk je volen vzhledem k jeho nízké úrovni, což umožňuje maximální rychlosti, přímou práci s pamětí, a tato vlastnost je zvlášť u databázových operací a systémů podstatná.

Kapitola 2

KNN

KNN algoritmy slouží pro získání K nejbližších prvků od zvoleného prvku (QueryNode/TargetNode) v n-dimenzionální prostoru. Tyto techniky jsou základem pro mnoho dnešních aplikací a často se i využívají metody pro přibližné KNN vyhledávání. Tato hodnota K bývá relativně malá, nejčastěji získáváme 10 nejbližších prvků, případně hodnoty pod 100, s tím že K může nabývat i mnohonásobně vyšších hodnot což ale není příliš časté.

Pro určení vzdálenosti mezi jednotlivými prvky v n-dimenzionálním prostoru můžeme využít celou řadu metrik. Mezi ty nejznámější patří Euklidova, Hammingova, Minkovského případně Čebyševova. Ve vypracované implementaci se vždy pracuje s Euklidovou vzdáleností, nicméně by nebyl problém zaměnit definici metody pro určení vzdálenosti mezi prvky a tím pádem změnit použitou metriku.

Dimenze prostoru ve kterém vyhledáváme prvky nemusí být vůbec nízká. Dimenze prostoru se může pohybovat v rozmezí od jednotek (2,5,...) do stovek (sift vecdim = 128). A hodnoty jednotlivých atributů se mohou rovněž tak pohybovat v širokém spektru, s tím že v mé práci jsou všechny atributy datového typu float.

Jako datový soubor pro vytvoření bodů v prostoru a možnosti následného testování byl vybrán Sift1M. V souboru se nachází binární data obsahující 1 000 000 rozdílných bodů, v prostoru o dimenzi 128 a floatové hodnotě jendotlivých atributů v rozmezí od 0 do 217, přičemž v atributech jsou pouze celočíselné hodnoty.

Kapitola 3

HNSW

Hierarchical Navigable Small Worlds, dále jen zkráceně HNSW, je jeden ze způsobů jakým řešit KNN problém. Tato metoda je založena na přibližném vyhledávání (ANN) za použití grafů. To znamená že výsledek je poskytován pouze s určitou přesností která je definována pomocí Recall, což je poměr relevantních výsledků které jsme získali.

Přesnost s jakou chceme získat výsledných K sousedů lze měnit. Metoda pro vyhledání KNN má parametr Ef udávající počet prvků které vyhledáváme, z nichž nakonec vracíme K nejbližších. Čím vyšší hodnota Ef bude tím s větší přesností proběhne vyhledání, zároveň tak poroste i čas vyhledávání. Z měření lze vidět že ideální hodnota Ef je okolo 200, získáváme vysoukou přesnost za přijatelný čas vykonání operace.

Důvod přibližného vyhledávání a ne procházení všech prvků je prostý, chceme co nejvyšší propustnost. Určování vzdálenosti mezi prvky nám zabírá mnoho času, okolo 90%, a proto chceme tuto operaci provádět co nejméně. V HNSW se vzhledem k přibližnému vyhledávání zhruba 90% prvků vůbec neprochází což ušetří velkou část času vykonání operace a proto tento způsob vyhledávání KNN dosahuje dobrých výsledků.

Jak již bylo zmíněno HNSW je založen na uložení dat pomocí grafů, konkrétně několikavrstvých grafů. Každý node v prostoru má Mmax sousedů, většinou 16/32, s tím že prvky mohou být a většinou i bývají sousedi navzájem. Vrstvy (layers) grafů fungují tak že v nejvyšší vrstvě, v mé implementaci se jedná o vrstvu 3, se nachází poze zlomek prvků (okolo 500 z 1 000 000) a slouží poze k rychlému průchodu a nálezu nového entryPoint, tedy prvku z kterého se posuneme do nižší vrstvy a kde nakonec začně vyhledávání. Takto se prochází i nižší vrstvy, v mém případě vrstva 2 a hledá se v nich pouze 1 nový entryPoint. Když narazíme na nízkou vrstvu, vrstvu 1 která již obsahuje zhruba 10% všech prvků, a vstoupíme do ní pomocí dříve nalezeného entryPoint a vyhledáváme již EfC prvků. EfC je většinou 200 a EfC prvků se vyhledává ve vrstvě 1 pokud tuto vrstu procházíme z důvodu vložení nového prvku do grafu, pokud bychom do této vrstvy vstupovali z důvodu vyhledání K sousedů tak nám i vrstva vrací pouze 1 entryNode jako vyšší vrstvy. A nakonec vstupujeme do

nejnižší vrstvy, vrstvy 0, buď s W o velikosti EfC v případě vkládání nového prvku nebo W o velikosti 1 v případě vyhledání KNN.

Metoda Search pro vyhledání K nejbližších prvků v grafu od queryNodu s tím že máme zadaný 1 nebo více entryPoint prvků (W) se využívá s drobnými úpravami jak pro operaci vložení prvku do grafu tak i pro vyhledávání KNN prvků. Pokud chceme prvek vložit tak musíme najít pozici kam do grafu ho dát a s kým bude sousedit což nám Search vrátí. Pokud vyhledáváme KNN prvků tak nám Search vrací K nejbližších prvků. Tato metoda funguje na jednoduchém prinicpu. Na začátku máme W již nalezených prvků které následně prohlásíme za ty nejbližší (entryPoint/s), V již navštívených prvku, a C potencionálních kandidátů na nejbližší prvky tedy prvky v W. Před začátkem průchodu jsou v C i V uloženy všechny prvky co jsou v W. Následně začíná průchod, který se opakuje dokud je $C > 0$. Získáme si prvek c, nejbližší prvek z C (vzdálenosti prvků, nejbližší/nejvzdálenější jsou vždy vohodnoceny jako vzdálenost daného prvku ku zadanéme queryNodu) a f, nejvzdálenšjší prvek z W. Pokud je vzdálenost c vetší než vzdálenost f tak průchod končí a našli jsme W s námi hledanými prvky. V opačném případě jdeme dál. Procházíme všechny sousedy e od prvku c. Pokud e není mezi navšívenými prvky V tak jeji přidáme do V, do f opět uložíme nejvzdálenější prvek z W a pokud je vzdálenost mezi e a q menší než vzdálenost mezi f a q a nebo je velikost W menší než zadané Ef tak pokračujeme dál. Do C i W vložíme prvek e a pokud je velikost W větší než Ef tak z W odstraníme nejvzdálenější prvek. Takto procházíme všechny prvky c z C a následně všechny sousedy e z c dokud není C rovno 0 nebo není vzdálenost mezi nejbližšího prvku z C (c) větší jak vzdálenost nejvzdálenějšího prvku z W (f).

Pro lepší pochopení HNSW se již naimplementovaný projekt využíval pouze jako reference pro porovnání podobnosti výsledků a srovnání časů vykonání operací. Implemence HNSW se tedy vytvořila od základu nová dle referenčního projektu a pseudokódu v článku popisujícím principy HNSW. Vlastní implementace celého projektu přinesla výhodu v maximálním porozumění kódu a algoritmům použitým v HNSW. Na druhou stranu je tato implementace přibližně 2.5x pomalejší než referenční kód, výsledky ale vrací rovněž validní jako reference.

Kapitola 4

Filtr

Pod pojmem filtr si můžeme představit booleovskou podmínku omezující hodnoty jednotlivých atributů (dimenzí) prvků v n-dimenzionálním prostoru. Konkrétně tedy filtr říká jaké hodnotě se mají jednotlivé atributy rovnat, případně v jakém intervalu by se měly atributy pohybovat. Toto platí pouze pro atributy jež filtr omezuje, atributy které filtr nezmiňuje vůbec nekontrolujeme a akceptujeme je bez závislosti na jejich hodnotě.

U KNN slouží filtr pro omezení vyhledání výsledných K prvků. Při vyhledávání výsledku se prochází i prvky které filtru nevyhovují, ale do vráceného výsledku jsou vloženy pouze prvky jejichž atributy vyhovují omezení filtru.

Selektivita filtru je číslo v rozsahu od 0 do 1 a udává procentuální počet prvků, které filtr přijme. Čím více je filtr vybíravý tím bližší hodnota k 0 mu bude přiřazena a tím více je filtr selektivnější. Naopak filtry přijímající většinu prvků budou mít přiřazeny číslo blíže k 1 a jejich selektivita bude tedy klesat, s tím že filtry přijímající úplně všechny prvky budou mít hodnotu rovno 1.

Filtr se implementoval jako vector objektů třídy VecDim. Tato třída reprezentuje jeden atribut a obsahuje ID dimenze, 0 - (vecDim - 1), vectory hodnot kterým má atribut (daná dimenze) nabývat nebo intervaly (tuple hodnoty od do) ve kterých se má atribut nacházet. Filtr je tedy nakonec vector který pro všechny atributy které chceme omezovat obsahuje hodnoty a intervaly kterým daná atribut musí vyhovovat, tedy minimálně jedné z těchto hodnot. Atributy které nijak neomezujeme ve vectoru atributů vůbec nejsou, porovnáváme jen ty atributy které nás ve filtru zajímají, ty ostatní přeskakujeme. Pro ověření zda daný prvek vyhovuje filteru jsem vytvořil pomoctou třídu VecDimHelper která pro daný prvek (vector hodnot) vrátí zda vyhovuje filteru nebo ne. Tato pomocná třída také generuje filtry dle určitých kritérií nebo umožnůje parsovat filtry z textové podoby a naopak.

V HNSW se filtr využívá u metody KNNFilter. Této metodě oproti její verzi bez filtru musíme tedy krom queryNodu, K a Ef i samotný filtr. Následně funguje stejně jen s tím rozdílem že používá vlastní SearchLayerFilter pro získání F nejbližších prvků. Tato metoda funguje obdobně jako její verze bez filtru SearchLayerKNN. Rozdíl v implementaci s filtrem je v tom že nevracíme W nejbliž-

ších nalezených prvků ale F nalezených nejbližších prvků které zároveň vyhovují filtru. Na začátku tedy do F uložíme entryPoint (jediný prvek v W) pokud splňuje podmínku filtru v opačném případě začíná průchod jako v popsaném Search v sekci HNSW. Rozdíl je v tom že jedna z ukončovacích podmínek průchodu je původně vzdálenost nejbližšího prvku z C je větší než vzdálenost nejvzdálenějšího prvku z W, v případě SearchLayerFilter je tato ukončovací podmínka rozšířena o to že zároveň musí platit že velikost F je rovna zadanému K. Tím pádem je zaručeno že při průchodu se prvky prochází dokud nenalezneme K hledaných prvků vyhovujících filtru nebo již nemáme co procházet a C je prázdné. Následně je ještě algoritmus rozšířen o část ve které do F vkládáme nově nalezené prvky e (e jsou sousedi nejbližších prvků c z C) pokud tento prvek filtr přijímá a zároveň je vzdálenost e menší než vzdálenost nejvzdálenějšího prvku z F nebo je velikost F menší než K.

4.1 Problém filtru s vysokou selektivitou u HNSW

Při použití implementace HNSW s využitím filtru se může stát že nám metoda KNNFilter nevrátí K hledaných prvků, ale výsledný počet hledaných prvků bude nižší. Pokud tak nastane tak nám ve výsledku chybí nízky počet prvků, nejčastěji 1 nebo 2 a pravděpodobnost že k tomuto jevu dojde je relativně nízká.

Tento jev nastává ve chvíli kdy hodnota K je stejně vysoká nebo jen o něco málo nižší než EF a zároveň je selektivita filtru vyšší, do 25%.

K tomuto problému nastává protože HNSW algoritmus pro procházení grafu a výběr nalezených prvků použitý v metodě SearchKNNFilter lze zakončit dvěma způsoby. První způsob jak zakončit algoritmus průchodu který nás aktuáně nezajímá je ten kdy vzdálenost nejbližšího prvku z C je vyšší než vzdálenost nejvzdálenějšího prvku z W a zároveň je počet prvků v F (původně W) roven zadanému K.

Druhý způsob jak ukončit průchod, ten který způsobí problém, je ten kdy je C prázdný. Stane se tedy že projdeme prvky z C a všechny jejich nenavštívené sousedy kteří jsou blíže než nejvzdálenější prvek z W. Během tohoto průchodu jednoduše nenarazíme na dostatečný počet prvků které by vyhovovaly filtru a F obsahuje tedy méně nejbližších prvků než je počet co hledáme.

Tento problém lze jednoduše vyřešit tím že zvýšíme hodnotu Ef aby byla více rozdílna a vyšší než hodnota K. S roustoucím Ef při zachování K a filtru poroste i čas vykonání operace, ale bude se zvyšovat pravděpodobnost že získáme přesně K prvků ve výsledku a poroste i přesnost operace.

Kapitola 5

Závěr

K závěru bych řekl že část práce určená pochopení a implementaci HNSW byla mnohem pracnější než následné rozšíření o filtry. Je tomu tak protože rozšíření nevyžaduje mnoho nových implementacích stačí jen již naimplementované algoritmy rozšířit o pár podmínek.

Dále bych chtěl zmínil že implementace takovýchto problémů má velice složité debugování a to hned z několika důvodů. Data jsou obrovská a stává se že implementace je obdobná s referencí do doby než se vloží prvek který je v datech daleko. Zabírá to hlavně spoustu času z důvodu dlouhého vytváření gragů a následného hledání kde přesně konkrétně došlo k rozdílu. Ve filnální implementaci se využívá i náhodné vybrání vrstev a pokud dva prky mají stejnou vzdálenost tak může dojít k rozdílným výsledkům.

V práci jsem až moc častou využíval std::vector, použití jednoduchých polí ať už staticky nebo dynamicky alokovaných by přineslo v určitých místech lepší propustnosti při správném využití nebo by graf zabíral méně paměti.

Práce mi určitě přinesla lepší pochopení C++, manipulaci s daty a důležitost jednotlivých rozhodnutí ohledně využití datových struktur nebo algoritmů pro zlepšení výsledné propustnosi a potřebné paměti. Projekt mě bavil a mimo složitý debug jsem si ji užil a jsem rád že si tento projekt zvolil. KNN je zajímavý problém a HNSW je zajímavý, jednoduchý a zároveň efektivní způsob jak jej řešit.

Literatura

1. *ann-benchmarks* [online]. 2022 [cit. 2022-03-06]. Dostupné z: `http://ann-benchmarks.com/index.html`.
2. *git-hnswlib: hnswlib* [online]. 2022 [cit. 2022-03-06]. Dostupné z: `https://github.com/nmslib/hnswlib`.
3. *Nearest neighbor search* [online]. 2022 [cit. 2022-03-06]. Dostupné z: `https://en.wikipedia.org/wiki/Nearest_neighbor_search`.
4. *Optimalizace v INFORMIXU* [online]. 2022 [cit. 2022-04-04]. Dostupné z: `http://www.ms.mff.cuni.cz/~jkoc5219/Optimalizace_v_INFORMIXU.html`.