FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Evaluace algoritmů lokálně senzitivního hashování (LSH) v doporučovacích systémech
Student:	Ladislav Martínek
Vedoucí:	Ing. Tomáš Řehořek
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

1) Seznamte se s problematikou doporučovacích systémů, konkrétně pak s algoritmy založenými na hledání nejbližších sousedů (nearest neighbors) v kolaborativním filtrování, a rovněž s metodami vyhodnocování úspěšnosti v těchto systémech (např. recall, coverage).
2) Nastudujte algoritmy lokálně senzitivního hashování (angl. Local Sensitive Hashing, LSH).
- navrhněte způsob, jak metody LSH aplikovat za účelem zavedení aproximace do neighborhood-based algoritmů.
- navrhněte framework pro testování a vyhodnocování různě parametrizovaných metod LSH v kolaborativním filtrování.
3) Implementujte několik vhodně zvolených metod LSH a proveďte měření na alespoň 2 dodaných datasetech.
Dosažené výsledky měření vhodným způsobem prezentujte a diskutujte poměr zhoršení přesnosti vs. zrychlení výpočtu.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 22. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Evaluace algoritmů lokálně senzitivního hashování (LSH) v doporučovacích systémech

Ladislav Martínek

Katedra aplikované matematiky
Vedoucí práce: Ing. Tomáš Řehořek

3. května 2018

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Tomáši Řehořkovi za věnovaný čas, cenné rady a vstřícný přístup při vedení této práce. Dále bych chtěl poděkovat firmě Recombee za poskytnutí serveru k testování. V neposlední řadě bych rád poděkoval i rodině za podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 3. května 2018 .....

Odkaz na tuto práci

Martínek, Ladislav. *Evaluace algoritmů lokálně senzitivního hashování (LSH) v doporučovacích systémech*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá aproximací algoritmu k-nejbližších sousedů orientovaného na uživatele pomocí metod lokálně senzitivního hashování a aplikací těchto metod do doporučovacích systémů.

Práce nejprve teoreticky popisuje doporučovací systémy, kolaborativní filtrování, algoritmy hledání k-nejbližších sousedů a metody jejich aproximace pomocí metod lokálně senzitivního hashování.

Na základě analýzy dané problematiky je navrhnut a implementován framework, který umožňuje testovat různé parametrizace metod lokálně senzitivního hashování. Ve frameworku lze testovat přesnost na k-nejbližších sousedů nebo úspěšnost doporučování použitím míry recall v závislosti na catalog coverage.

Popsané metody jsou pomocí frameworku otestovány na dvou odlišných databázích. Z testů jednotlivých metod a parametrizací jsou vyvozeny možnosti jejich kombinace k dosažení optimálních modelů. Při testování optimálních modelů se podařilo dosáhnout velice uspokojivých výsledků. Při čase modelu LSH okolo 3 % času referenčního řešení se podařilo dosáhnout úspěšnosti doporučování mezi 97 a 99 %. Na závěr jsou diskutovány výsledky a různé poznatky z testování jednotlivých metod.

Klíčová slova algoritmus k-NN orientovaný na uživatele, aproximace algoritmu k-NN, lokálně senzitivní hashování, doporučovací systémy, návrh a implementace testovacího frameworku, experimenty a analýza výsledků

Abstract

This thesis deals with the approximation of the user-based k-nearest neighbor algorithm using locality-sensitive hashing methods and the application of these methods in recommender systems.

First the thesis describes the recommendation systems, the collaborative filtering, the k-nearest neighbors algorithms and the method of their approximation using locality-sensitive hashing methods.

Based on the analysis, framework was designed and implemented to test various parameterizations of locality-sensitive hashing methods. The precision of nearest neighbors algorithm or the success rate of recommending (by using the recall rate depending on the catalog coverage) can be tested in the framework.

Described methods are tested on two separed databases using the framework. From the tests of individual methods and parameterizations, the possibilities of their combinations are derived to achieve optimal models. During the testing of the optimal models I was able to achieve very satisfactory results. At around 3 % of the reference solution time, the reached success rate was between 97 % and 99 %. Finally, I discussed the results and different findings from the testing of the individual methods.

Keywords user-based k-NN algorithm, aproximation of k-NN algorithm, locality sensitive hashing, Recommender systems, design and implementation of a test framework, experiments and analysis of results

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza **5**
2.1 Doporučovací systémy 5
2.2 Kolaborativní filtrování 6
2.3 Algoritmy nejbližších sousedů 8
2.4 Metody vyhodnocování úspěšnosti 10
2.5 Algoritmy lokálně senzitivního hashování 12

3 Návrh **19**
3.1 Obecná struktura frameworku 19
3.2 Předzpracování dat 20
3.3 Algoritmus k-NN 21
3.4 Začlenění metod LSH do algoritmu k-NN 22
3.5 Testování a vyhodnocování 23

4 Implementace **27**
4.1 Použité vybrané knihovny 27
4.2 Argumenty frameworku a jejich využití 28
4.3 Optimalizace 30
4.4 Řešené problémy 31
4.5 Modul pro zjednodušenou tvorbu grafů 32

5 Experimenty **35**
5.1 Databáze 35
5.2 Testování modelů s různým počtem projekčních matic 36
5.3 Testování multi-probe LSH modelů 39
5.4 LSH a popularita položek 41

5.5 Využití testovaných metod k vytvoření optimálních modelů . . 45

6 Diskuze **53**
6.1 Rozdělení uživatelů do bucketů 53
6.2 Poměr zhoršení přesnosti a zrychlení výpočtu 54
6.3 Možnosti budoucí práce 54

Závěr **55**

Literatura **57**

A Seznam použitých zkratek **61**

B Obsah přiloženého CD **63**

Seznam obrázků

2.1 Rozdělení prostoru nadrovinami pomocí LSH 13
2.2 Rozdělení 2D prostoru nadrovinami v metrice kosinové podobnosti 15

3.1 Obecná struktura frameworku 20
3.2 Princip vytvoření modelu LSH 22
3.3 Testování úspěšnosti LSH modelů při hledání k-nejbližších sousedů 24
3.4 Leave-one-out křížová validace a testování úspěšnosti doporučovacího systému 25

5.1 Porovnání modelů LSH s 1 až 32 hashovacími funkcemi 37
5.2 Porovnání modelů LSH s 64 až 512 hashovacími funkcemi 38
5.3 Recall a catalog coverage, úspěšnost doporučování více hashovacích funkcí 39
5.4 Graf závislosti přesnosti k-NN a recallu při změně počtu hashovacích funkcí 40
5.5 Porovnání přesnosti k-NN při metodě multi-probe 41
5.6 Rozdělení uživatelů do bucketů, vliv parametru alfa 42
5.7 Ukázka vlivu parametru alfa na přesnost k-NN 43
5.8 Porovnání závislosti přesnosti k-NN a recall při využití parametru alfa 44
5.9 Přesnost k-NN, optimální modely databáze A 47
5.10 Úspěšnost doporučování modelu se 128 hashovacími funkcemi na databázi A 48
5.11 Přesnost k-NN, optimální modely databáze B 49
5.12 Úspěšnost doporučování modelu se 64 hashovacími funkcemi na databázi B 50
5.13 Závislost přesnosti k-NN na hodnotě recall, porovnání databází . . 50
5.14 Závislost času vyhodnocování na hodnotě recall, porovnání databází 51

Seznam tabulek

2.1 Tabulka matice záměn 10

5.1 Tabulka nastavení parametrů a výsledků modelu LSH 46

Úvod

V dnešní moderní společnosti, kde neustále roste objem dat nabízený na internetu, je pro uživatele velmi těžké vyhledávat informace nebo produkty. Proto se společnosti zabývají přizpůsobováním obsahu každému uživateli na internetu např. doporučením, co by se uživateli mohlo líbit. Při doporučovaní se snaží nabídnout např. film, článek, který by mohl uživatele zajímat. Doporučování na internetu je nutné provádět v reálném čase, což se současnými objemy dat je velice problémové. Z tohoto důvodu je výhodné, zároveň i nutné se zabývat zrychlováním doporučovacích systémů.

Existuje mnoho metod, které se zrychlováním zabývají. V práci se věnuji jedné konkrétní metodě a to lokálně senzitivnímu hashování, pomocí kterého lze dosahovat několikanásobného zrychlení výpočtu při malé ztrátě přesnosti. Výstup mé práce by měl pomoci programátorům s testováním a vyhodnocováním různých parametrizací lokálně senzitivního hashování pro zrychlení doporučovacích systémů.

Téma jsem si zvolil, neboť doporučovací systémy jsou využívány mnoha společnostmi na internetových stránkách, kde nám jejich přítomnost usnadňuje hledání informací. Nicméně je velmi výpočetně náročné vyhodnocovat všechna nasbíraná data v reálném čase, proto se v práci zabývám aproximací doporučovacích systémů, konkrétně pak návrhem, implementací a následným vyhodnocením knihovny na aproximaci hledání nejbližších sousedů.

Tato práce dále pokračuje v následující struktuře: Nejdříve se v kapitole 2 věnuji doporučovacím systémům, představím použité algoritmy a metody vyhodnocování jejich úspěšnosti. V následující kapitole 3 navrhnu knihovnu pro testování představených algoritmů a jejich parametrizace, poté budu pokračovat popisem konkrétní implementace v kapitole 4. Následně v kapitole 5 prezentuji výsledky provedených experimentů, na které navážu diskuzí nad výsledky a doporučením pro budoucí práce (kapitola 6).

Kapitola 1

Cíl práce

Cílem rešeršní části práce je získat přehled o doporučovacích systémech, seznámit se s algoritmy nejbližších sousedů a nastudovat algoritmy lokálně senzitivního hashování a možnosti využití těchto metod při aproximaci algoritmů nejbližších sousedů. Dalším navazující cíl rešeršní části je představení metod vyhodnocování úspěšnosti představených algoritmů a doporučovacích systémů založených na těchto algoritmech.

Cílem praktické části práce je navrhnout způsob aplikace algoritmů lokálně senzitivního hashování za účelem aproximace algoritmů hledání nejbližších sousedů, poté návrh a implementace frameworku pro testování úspěšnosti algoritmů lokálně senzitivního hashování v doporučovacích systémech. V neposlední řadě je cílem otestování implementace algoritmů s různou parametrizací na dvou datových sadách a diskuze výsledků, konkrétně poměr zrychlení výpočtu na úkor zhoršení přesnosti.

Kapitola 2

Analýza

V této kapitole představím základní pojmy a principy, které se vztahují k tématu mé práce. Nejprve popíši samotné doporučovací systémy, následně kolaborativní filtrování (angl. Collaborative filtering) a algoritmy nejbližších sousedů. Také v této kapitole popíšu a vysvětlím metody aproximace těchto algoritmů pomocí lokálně senzitivního hashování (angl. Locality-sensitive hashing, LSH) a metody vyhodnocování úspěšností.

2.1 Doporučovací systémy

Doporučovací systémy jsou nástroje a techniky, které generují doporučení položek, které by mohly být zajímavé nebo užitečné pro uživatele [1, s. 1]. Doporučením mohou být například produkty k prodeji, filmy nebo i slevové kupóny od obchodních řetězců.

Systémy začínají být více užitečné, protože narůstá množství obsahu, které je poskytováno. Pro uživatele je tak velice obtížné a zdlouhavé hledání položek, které jsou pro něj zajímavé. Doporučovací systém je pro uživatele velkým přínosem a šetří čas při hledání položek.

Systém však může být přínosem i pro samotnou firmu, pokud doporučení jsou relevantní a zajímavá, uživatelé se na stránku vracejí a zvýší se využití systému [1, s. 5]. S nárůstem počtu uživatelů také vzroste náročnost vyhodnocování a je nutné se zabývat zrychlením těchto systémů.

Pro poskytování personalizovaných doporučení však systém potřebuje mít nějaké informace o každém uživateli pro kterého mají být doporučení generována [2, s. 1]. Z toho plyne, že doporučovací systém potřebuje nějaký model, který uživatelská data vhodně ukládá a zpracovává.

V celé práci budu používat model, který ukládá interakce jednotlivých uživatelů s produkty, kde každé dvojici uživatel–položka je přiřazena váha nebo hodnocení. Například v internetových obchodech jsou ukládány k uživatelům produkty, které si zobrazili, vložili do košíku nebo koupili. Z dat jsou vytvářeny modely, na základě kterých jsou vracena doporučení položek pro uživatele.

2.1.1 Techniky doporučování v doporučovacích systémech

Aby byly schopné doporučovací systémy doporučit některou položku, je nutné, aby systém předvídal užitečnost takové položky. Proto představím různé techniky a modely doporučovacích systémů, které mohou generovat doporučení. Tato doporučení nemusejí být stejná za všech okolností a mohou záviset na různých proměnných např. doba, kdy je potřeba generovat doporučení nebo na geografické poloze. Podle [2] jsou představeny následujících 4 různé základní techniky generovaní doporučení:

- **Doporučování založené na obsahu (angl. Content-based).** Takové doporučování je založeno na podobnosti jednotlivých položek obsahu. Podobné položky jsou řazeny do stejných kategorií, kterými mohou být například videa nebo hudba vybraného žánru. Uživatel dá pozitivní hodnocení písničce a systém mu nabídne podobné ze stejného žánru. Pokud se v databázi objeví nová položka je problém, že není způsob, jak by mohla být doporučována, dokud se neobjeví více informací nebo ji jiný uživatelé neohodnotí, aby byla zařazena do některých kategorií [3].

- **Doporučování na základě znalostí.** Takové doporučování je závislé na specifické znalosti domény podle toho, jaké jsou požadavky a preference uživatelů této domény [1, s. 12]. Tyto systémy jsou většinou v systémech, kde není možné nasbírat velké množství dat pro vyhodnocení.

- **Hybridní systémy.** Tyto systémy jsou založené na kombinaci zde zmíněných případů. Kombinací systémů je možné využívat přednosti různých technik a předcházet jejich nedostatkům. Systémy mohou mezi sebou o výsledku doporučení například hlasovat.

- **Kolaborativní filtrování.** Této technice se podrobněji věnuji v následující podkapitole.

2.2 Kolaborativní filtrování

Základním principem je filtrování obsahu a informací pomocí spolupráce mezi více uživateli, podhledy nebo zdroji dat. V mé práci se budu věnovat filtrování pomocí spolupráce uživatelů. Klíčovou myšlenkou je, že pokud uživatelé sdílejí zájem o určité položky, budou tento zájem sdílet i nadále a budou preferovat podobné položky [4].

Tento přístup filtrování obsahu je velice populární, protože poskytuje velmi dobré výsledky v doporučovacích systémech. Příklad takového filtrování může být například: Mějme uživatele A a B a položky Y a Z. Uživatel A v minulosti kladně hodnotil obě položky Y a Z. Uživatel B hodnotil kladně pouze položku Y. Na základě interakce se stejnou položkou Y jsou uživatelé A a B podobní a je uživateli B doporučena položka Z.

2.2.1 Možné problémy kolaborativního filtrování

V rámci kolaborativního filtrování se mohou vyskytovat potencionální problémy. S těmito problémy se potýkají různé systémy ve větší či menší míře. V doporučovacích systémech založených na kolaborativním filtrování mohou být následující problémy řešeny různě:

- **Škálovatelnost:** S narůstajícím množstvím dat je výpočet časově náročnější. Mohou zde být miliony uživatelů a desetitisíce položek. Je nutné, aby systémy byly dobře škálovatelné, protože je nutné generovat doporučení online v reálném čase.
- **Řídká data (angl. sparse data):** Velké společnosti nabízejí velmi mnoho položek (až v řádech desetitisíců nebo statisíců). Uživatele interagují poté jen s velmi omezenými množinami těchto položek. I populární produkty mohou mít velmi malé množství interakcí.
- **Long-tail:** Tento problém se týká distribucí interakcí nebo hodnocení uživatelů mezi jednotlivé položky. Můžeme předpokládat, že 50% interakcí je koncentrováno na 10 % položek, proto doporučit položku, která se nachází ve zbylých 90 % tedy v tzv. long-tail, je těžší než doporučit populární položku [1, s. 324].
- **Studený start:** Tento problém nastává pokud se registruje nový uživatel nebo je přidána nová položka. Pro nové uživatele je těžké hledat jemu podobné, dokud nejsou uživatelem poskytnuty nějaké informace a nová položka nemůže být doporučována, protože nemá dostatek hodnocení a interakcí s uživateli [5].

2.2.2 Modelově orientované kolaborativní filtrování

Tento přístup nevyužívá celou množinu dat pro výpočet a predikci uživatelského hodnocení nehodnocených položek nebo generování doporučování. Pro modely je využita učící sada, která slouží k naučení modelu pro pozdější vyhodnocování a predikci.

Pro tyto modely je typické, že je velice časově a paměťově náročné je vytvořit, ale následně je velice rychlé a efektivní získání doporučení [5]. Do této kategorie patří například singular value decomposition nebo shluková analýza.

Také lze do této kategorie lze částečně zařadit i LSH. V rámci LSH jsou hashováni uživatelé do bucketů. Metody popisuji dále v práci.

2.2.3 Pamětově orientované kolaborativní filtrování

Tento přístup je založen na vzájemné spolupráci a podobnosti uživatelů nebo položek v databázi. Modely fungují na celé databázi interakcí shromážděné

systémem s možným časovým omezením záznamů [6]. Dělí se na dvě hlavní větve orientované buď na uživatele nebo na položky.

Tyto přístupy se vyznačují velmi rychlým vytvořením modelu, ale při každém vyhodnocení je nutné procházet množinu dat, proto jsou velmi náročné na výpočetní prostředky. Naopak výhodou těchto systémů je jednoduché přidávání nových dat.

Mezi tyto algoritmy patří algoritmy nejbližších sousedů probrané v následující kapitole. Můžeme sem také zařadit metody LSH, pomocí kterého lze aproximovat algoritmy nejbližších sousedů.

2.3 Algoritmy nejbližších sousedů

Algoritmy nejbližších sousedů (angl. k-nearest neighbors, k-NN) vždy fungují v konkrétním měřitelném prostoru, který může být definován různými způsoby. Existence metriky je důležitá pro porovnávání jednotlivých sousedů. Metriky se od sebe liší a lze je například rozdělit na podobnostní nebo vzdálenostní.

Algoritmy k-NN lze rozdělit na dva případy. První z nich je orientovaný na položky (angl. item-based k-NN). Hledá nejpodobnější položky na základě preferencí uživatelů. Naopak druhý přístup je orientovaný na uživatele a hledá nejpodobnější uživatele (angl. user-based k-NN) [7]. V mé práci se zaměřuji pouze na user-based k-NN.

2.3.1 User-based k-NN

Definici User-based k-NN jsem čerpal z [8] a upravil značení, aby korespondovalo v celé mé práci. Zní následovně:

Mějme množinu uživatelů $U = \{u_1, u_2, \ldots, u_n\}$ v metrickém prostoru M, kde každý uživatel $u \in U$ představuje vektor položek. Dále vyhodnocovaný bod $q \in M$. Podobnostní funkcí v prostoru je následující: $s : M \times M \to \mathbb{R}$. Hledání k nejbližších sousedů je definováno jako funkce takto:

$$\mathrm{KNN}\,(q, U, k) = A \tag{2.1}$$

Kde A je množina bodů splňující:

$$\begin{gathered} |A| = k, A \subseteq U \\ \forall a \in A, \forall b \in U - A : s\,(q, a) \geq s\,(q, b) \end{gathered} \tag{2.2}$$

Algoritmus k-NN vrátí vždy k nejpodobnějších uživatelů na množině U, pokud $k < |U|$, jinak jsou vráceni všichni uživatelé.

2.3.2 Metriky podobnosti k-NN

Mezi metriky podobnosti patří mnoho různých metrik. Základní rozdělení je mezi metriku vzdálenosti a podobnosti. Metriky vzdálenosti jsem ve své práci

nepoužíval a jsou využívány spíše pro klasifikaci. Já jsem ve své práci použil metriku podobnosti. Podobnostní metriky lze pak úspěšně využívat i při vážení jednotlivých uživatelů z množiny K. Konkrétně pak v práci a při testování využívám metriku kosinové podobnosti, které při použití v algoritmu k-NN v doporučovacích systémech dává odpovídající dobré výsledky.

Kosinova podobnost je metrika, která je získána výpočtem kosinu úhlu dvou vektorů. Pro vektory $\mathbf{x}, \mathbf{y} \in U$ podle [1, s. 82] nebo [9, s. 3] se kosinova podobnost dvou vektorů spočítá následovně:

$$s(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum_{i=0}^{n-1} x_i y_i}{\sqrt{\sum_{i=0}^{n-1} x_i^2} \sqrt{\sum_{i=0}^{n-1} y_i^2}} \tag{2.3}$$

Pomocí metriky kosinové podobnosti je možné nalézt k-nejpodobnějších uživatelů na množině uživatelů U k dotazovanému uživateli $q \in U$ vyhodnocením podobnosti pro každé dva uživatele. Na základě nejbližších sousedů je možný regresní odhad vah nebo hodnocení položek.

2.3.3 Doporučování na základě algoritmu k-NN

Na základě nejbližších sousedů je pomocí regrese možný odhad hodnocení položek, se kterými uživatel neinteragoval nebo takovou položku nehodnotil.

Hodnocení je počítáno přes všechny nejbližší sousedy z množiny K, kde $K \subseteq U$. Hodnocení $r_{u,i}$ je hodnocení položky $i \in I$ uživatelem $u \in U$. Množina I je množina všech položek v databázi. Tato hodnocení nejbližších sousedů jsou vážena hodnotou kosinové podobnosti a slouží k odhadu uživatelského hodnocení. Predikce pro položku i a uživatele u můžeme spočítat podle [2, s. 20] nebo [9, s. 3] jako:

$$p_{u,i} = \frac{\sum_{j \in K} (r_{j,i} s(u,j))}{\sum_{j \in K} s(u,j)} \tag{2.4}$$

Ve vzorci lze pro účely práce vynechat jmenovatele (nevážit kosinovou podobností). Vzniklým vzorcem lze počítat ohodnocení položek uživatele u, které lze využít pro nalezení těch nejrelevantnějších z nich. Poté tyto položky, se kterými uživatel ještě neinteragoval slouží ke generování doporučení. Ohodnocení pro položku $i \in I$ a uživatele $u \in U$ spočítáme:

$$p_{u,i} = \sum_{j \in K} (r_{j,i} s(u,j)) \tag{2.5}$$

Po seřazení položek podle ohodnocení je vyhodnocovanému uživateli doporučeno n nejrelevantnějších (angl. Top-N recommendations).

Tabulka 2.1: Matice záměn

		Skutečnost	
		ano	ne
Predikce	ano	TP	FP
	ne	FN	TN

2.4 Metody vyhodnocování úspěšnosti

V předchozí kapitole 2.3 jsem popsal fungování algoritmu nejbližších sousedů a generování doporučení pro uživatele. Abychom mohli vytvořené algoritmy a systémy posoudit z hlediska úspěšnosti a výkonu je nutné si představit některé metriky vyhodnocování úspěšnosti.

Pro testování je nutné rozdělit testovací sadu na trénovací a testovací část. Data v trénovací části jsou využívána k učení modelu. Pomocí testovacích dat je testována úspěšnost konkrétního algoritmu.

V následující části nejprve představím techniku křížové validace, která je velice populární a je využívána k testování doporučovacích algoritmů.

2.4.1 Křížová validace (angl. Cross-Validation)

Křížová validace je velmi účinný způsob testování doporučovacího modelu. Data pro křížovou validaci jsou rozdělena na více podmnožin. V každém kole je jiná podmnožina využita jako testovací a zbývající množiny slouží jako trénovací data modelu. Výhodou křížové validace je, že umožňuje využít více různých dat pro testování. Používání odlišných trénovacích sad vede k minimalizaci chyb vzniklých rozdělením dat na trénovací a testovací část. [10]

Konkrétním případem křížové validace je k-fold křížová validace, kde jsou data rozdělena na k podmnožin a jsou k-krát testována. Speciálním případem k-fold křížové validace je leave-one-out křížová validace, kdy je vždy vynechána jen jedna položka.

Tento způsob je nejčastějším způsobem testování doporučovacích systémů. Jedna položka z vektoru uživatele je vynechána. Poté je upravený uživatel testován, zdali mu bude daná položka doporučena. Výsledky je možné zaznamenávat do matice záměn (angl. Confusion matrix), která je popsána v následující části.

2.4.2 Matice záměn

Systémy ne vždy předpovídají přímo hodnocení konkrétní položky, ale mohou se na doporučení dívat jako na binární data, pouze zda položka v doporučených položkách byla (1) nebo nebyla (0) [10]. Takovými daty se zabývám i v mé práci. Do matice záměn jsou pak ukládány čtyři možné případy, jak mohlo testování dopadnout. Matice záměn je ukázána v tabulce 2.1.

Matici záměn obsahuje čtyři buňky. Především první dvě budou mnou v práci sledované. První TP jsou true positives, tedy vynechaná testovaná položka v křížové validaci byla v Ton-N doporučení. Systém doporučil položku správně. FN jsou false negatives, tedy testovaná položka nebyla v Top-N doporučení. Ale ve skutečnosti měla být uživateli doporučena.

Poslední dvě položky matice nebudou mnou sledované při testování doporučování, protože nemám informaci o položkách, které pro uživatele nejsou přínosné, ale pouze interakce s uživatelem vybranými položkami. Aby je bylo možné sledovat je potřeba znát i položky, které se uživateli nelíbí a dán předem daný přesný počet generovaných doporučení. Položky matice TN jsou true negatives, nebo-li položky, které se uživateli nelíbily a nebyly ani v doporučení. FP jsou false positives. To jsou položky, které byly doporučeny, ale jsou negativně hodnoceny.

Z tabulky matice záměn 2.1 lze poté podle [1, s. 275] nebo [10] spočítat počty případů v jednotlivých buňkách a vyhodnocovat především následující populární metriky:

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{2.6}$$

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{2.7}$$

Recall (nebo také senzitivita) je hlavní mírou, kterou využiji pří testování a srovnávání algoritmů a metod LSH. Recall reprezentuje pravděpodobnost, že bude vybrána relevantní položka [11]. Míra je vhodná především, pokud existuje velké množství položek a předem není znám celý průběh testování v Top-N doporučování. Snažím se doporučovat relevantní položky, které by mohly být pro uživatele zajímavé a zaznamenávám počet úspěšných doporučení proti všem pokusům.

Precision je pravděpodobnost, že vybraná položka je relevantní [11]. Pomocí této míry v práci testuji úspěšnost algoritmu k-NN. Z vybraných k-nejbližších sousedů hledám, zdali se nachází v referenčním řešení. Tyto výsledky zaznamenávám do matice záměn a poté počítám precision.

2.4.3 Míra pokrytí (angl. Coverage)

S rostoucím množstvím dat mohou některé doporučovací systémy poskytovat doporučení s velmi vysokou kvalitou, ale jen na omezeném množství nabízených položek, pro které mají obrovské množství dat [1, s. 281]. Systém pak v nejhorším případě doporučuje jen nejpopulárnější produkty tzv. bestsellery. Takové položky mohou být zároveň také lehce dohledatelné uživatelem a systém pak může ztrácet svoji užitečnost.

Na druhou stranu účelný a užitečný bude doporučovací systém, který dokáže doporučovat co nejvíce položek z databáze. Budeme-li nyní uvažovat

pokrytí databáze, je tato míra velice cenná pro doporučovací systémy, které doporučují n nerelevantnějších položek [12].

Míru pokrytí lze rozdělit na dva případy. První je podíl položek, které jsou k dispozici jednomu uživateli. Druhá je míra pokrytí databáze položek (angl. catalog coverage), což je procento všech položek, na které je systém schopen generovat doporučení [12].

Míra pokrytí databáze položek je míra, kterou budu sledovat i při testování doporučovacích systémů založených ma modelech LSH. Tato míra je velice závislá na rozsáhlosti testů a hodnoty se liší podle velikosti testovací množiny, proto je nutné testované algoritmy porovnávat na stejné velikosti testovací množiny. Míru pro počítání catalog coverage je podle [12] zavedena následovně:

$$coverage = \frac{|\bigcup_{u \in U} J_u|}{|I|} \tag{2.8}$$

Kde množina U představuje testované uživatele. J_u je množina doporučených produktů každému konkrétnímu testovanému uživateli $u \in U$ a množina I představuje množinu všech produktů.

2.5 Algoritmy lokálně senzitivního hashování (LSH)

S rostoucím množstvím uživatelů, o kterých jsou ukládána data a množstvím položek roste i výpočetní náročnost algoritmu k-NN, který je třeba na základě dat vyhodnocovat v reálném čase. Z tohoto důvodu jsou zaváděny různé metody aproximace těchto algoritmů. Jejich principem je zrychlení vyhodnocování algoritmů nejbližších sousedů na úkor malého zhoršení přesnosti.

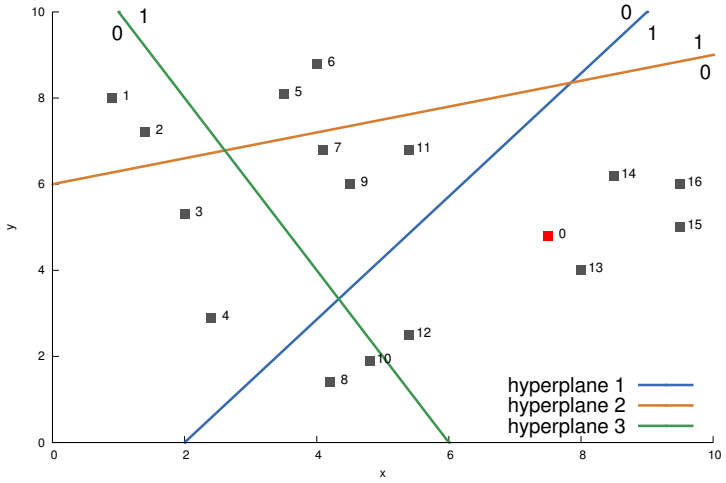
Jednou z takových metod je lokálně senzitivní hashování (dále jen zkráceně LSH). Podobně jako i jiné aproximační metody je LSH randomizované a náhodnost je typicky využita ke konstrukci datové struktury pro vytvoření modelu [13].

2.5.1 Základní princip LSH

Základním principem LSH je rozdělení vícerozměrného prostoru na části nazývané jako kbelíky (angl. buckets). V práci budu pro lepší přehlednost a korespondenci s literaturou používat anglické pojmenování skloňované v češtině např. buckety. Takové obecné rozdělení je zobrazeno na obrázku 2.1.

Rozdělování jednotlivých bodů do bucketů je jedna z odlišností jednotlivých metod. Různé metody LSH jsou zavedeny pro různé metriky podobnosti v prostoru, ale každá metoda je založena na hashovací funkci.

Hashovací funkce LSH transformuje mnoha dimenzionální vektory na sekvence bitů. Odlišností proti klasickým hashovacím funkcím jako jsou SHA



Obrázek 2.1: Rozdělení 2D prostoru náhodnými nadrovinami jako obecná ilustrace indexování prostoru do bucketů pomocí LSH. Obrázek vytvořen na základě popisu v publikaci [14]

a MD5, které se snaží kolizím co nejvíce zabránit, se hashovací přístup v LSH snaží maximalizovat pravděpodobnost kolize blízkých položek [14].

Na obrázku 2.1 popíšu obecné rozdělení prostoru, který je členěn na několik bucketů. Buckety jsou dány jednotlivými nadrovinami dělící prostor na dvě podmnožiny, které určují jeden bit výsledné sekvence. Taková sekvence bitů pak představuje jednoznačně určený bucket a do něj spadající uživatele.

Zahashované vektory uživatelů do jednotlivých bucketů představují model pro aproximaci algoritmu k-NN. Při vyhodnocování dotazu na k-nejbližších sousedů je poté dotazující objekt q zahashován stejnou funkcí do nějakého bucketu (určeného podle stejné sekvence bitů) [15]. Uživatelé ve výsledném bucketu jsou aproximací k-NN a potencionální kandidáti na výsledek.

Začlenění aproximace pomocí LSH do algoritmu k-NN je podle [16] popsáno v následujících dvou pseudokódech. Algoritmus 1 slouží pro vytvoření modelu a předzpracování. Algoritmus 2 slouží k vyhodnocení dotazu na k-nejbližších sousedů.

2.5.2 Hashovací funkce LSH

Dobrá hashovací funkce v algoritmu LSH minimalizuje počet uživatelů v jednom bucketu (se stejným hashem) a maximalizuje počet nejbližších sousedů v tomto bucketu.

Hashovací funkce $h(\mathbf{x}) \rightarrow \langle 0 \dots m)$ převede d dimenzionální vektor $\mathbf{x}$ na číslo bucketu, které je mezi 0 až $m - 1$. Hashovací funkce musí být rychlá na výpočet, jednoduše rozšiřitelná a také pravděpodobnost kolize musí souviset

Algoritmus 1 Předzpracování

1: **procedure** Předzpracování(U ←množina uživatelů, m ←počet hashovacích funkcí)
2: **for** i **from** 0 **to** m **do**
3: H_i hashovací funkci inicializovat jako náhodnou funkci h_i
4: **for each** $u \in U$ **do**
5: **for each** $h_i \in H$ **do**
6: Uživatele u uložit do bucketu $h_i(u)$ příslušné hashovací funkce
return model hashovacích funkcí s rozdělením do bucketů.

Algoritmus 2 Vyhodnocení

1: **procedure** LSH(q ←dotazující uživatel, H ←model hashovacích funkcí, k ←počet nejbližších sousedů)
2: $S \leftarrow [\,]$
3: **for each** $h_i \in H$ **do**
4: $S \leftarrow S \cup \{$uživatele v bucketu $h_i(q)$ hashovací funkce $h_i\}$
return k nejbližších sousedů spočítaných nad seznamem možných uživatelů S.

s odpovídající metrikou podobnosti, tedy podle [17] pro množinu uživatelů U, funkci podobnosti s a hashovací funkci h platí:

$$\forall a, b, c \in U : s\,(a, b) \leq s\,(b, c) \Rightarrow P[h\,(a) = h\,(b)] \leq P[h\,(b) = h\,(c)] \tag{2.9}$$

Výše zmíněné tvrzení není relevantní pro některé hashovací funkce.

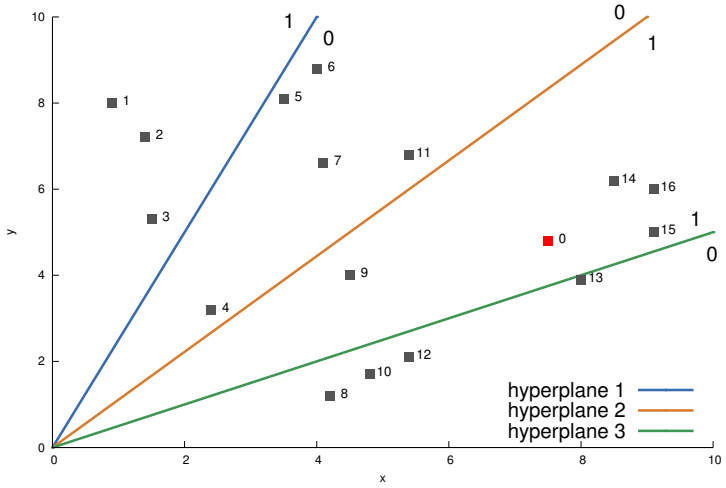
Obecněji bylo LSH poprvé představeno a definováno v publikaci [18]. Definici v práci uvedu upravenou podle [19] a s upraveným značením. Mějme skupinu hashovacích funkcí $H = \{h : U \to T\}$, která je nazývána (r_1, r_2, p_1, p_2)-senzitivní pro metriku podobnosti s, když pro libovolné $\mathbf{x}, \mathbf{y} \in U$ platí:

$$\begin{aligned} &\text{Pokud } s(\mathbf{x}, \mathbf{y}) \geq r_1 \text{ pak } Pr_H[h\,(\mathbf{x}) = h\,(\mathbf{y})] \geq p_1 \\ &\text{Pokud } s(\mathbf{x}, \mathbf{y}) < r_2 \text{ pak } Pr_H[h\,(\mathbf{x}) = h\,(\mathbf{y})] \leq p_2 \end{aligned} \tag{2.10}$$

Aby bylo hashování vektorů lokálně senzitivní je pro podobnostní míru nutné, aby vyhověla nerovnostem $p_1 > p_2$ a $r_1 > r_2$ [18]. Pro různé metriky podobnosti jsou skupiny hashovacích funkcí pro LSH stanoveny odlišně, ale na základě předešlé definice. Dále se v práci věnuji pouze metodám LSH v metrice kosinové podobnosti.

2.5.3 Hyperplanes LSH

Rozdělení nadrovinami je modelem v kosinové podobnosti za předpokladu, že nadroviny dělící prostor procházejí počátkem [14]. Ilustraci rozdělení je vidět na obrázku 2.2. Každá hashovací funkce pak představuje skupinu nadrovin v prostoru, které prostor rozdělují do bucketů.



Obrázek 2.2: Rozdělení 2D prostoru náhodnými nadrovinami jako ilustrace indexování prostoru do bucketů pomocí LSH v metrice kosinové podobnosti. Obrázek vytvořen na základě popisu v publikaci [14] při citlivosti hashovací funkce na kosinovu metriku podobnosti v prostoru.

Na obrázku 2.2 je vidět rozdělení 2D prostoru třemi nadrovinami. Nadrovina je podprostor dimenze n-1 v prostoru dimenze n. V 2D prostoru je nadrovinou přímka. Každá nadrovina rozdělí prostor na dva podprostory. Z toho plyne, že pokud máme d nadrovin, tak celkový počet podprostorů (bucketů), které tyto nadroviny vymezí, bude $\#buckets = 2^d$.

Na obrázku 2.2 jsou tyto podprostory představovány 0 a 1 u jednotlivých přímek. Každá nadrovina může představovat například 1 bit binárního čísla. Buckety jsou pak například binární čísla 000 nebo 010. V případě vyhodnocování bodu 0 z obrázku 2.2 je bod umístěn v bucketu (řazeno podle čísel nadrovin) 011. Vyhodnocení a počítání podobnostní funkce pak proběhne na bodech v tomto bucketu, konkrétně pak pak na bodech 9, 14, 15, 16 a vrátí se k nejbližších.

2.5.4 Náhodná projekce (angl. Random projection)

Algoritmy na principu náhodné projekce vychází z předchozí části 2.5.3 a jsou navrženy pro aproximaci k-NN za použití kosinové podobnosti. Pro uživatelské vektory $\mathbf{x}, \mathbf{y} \in U$, kde θ je úhel, který spolu svírají, a hashovací funkci h podle článku [14] platí:

$$P[h(\mathbf{x}) = h(\mathbf{y})] = 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi} \tag{2.11}$$

Pro redukci dimenze v této metodě se využívá projekční matice M, která může být generována různými pravděpodobnostními rozděleními. Nejčastěji

používaným rozdělením je $\mathcal{N}(0,1)$ nebo případně $\mathcal{U}(-1,1)$, ale také může být M generováno z množiny hodnot $\{-1,0,1\}$.

Označení d je počet hashovacích vektorů v matici nebo též dimenze vektoru výsledného binárního bucketu. Počet hashovacích vektorů v matici také určuje maximální počet bucketů, který je $\#buckets = 2^d$. Označení dimenze uživatelského vektoru je n, pak máme projekční matici jako $M_{n,d}$.

Každý uživatel $u \in U$ je pak zahashován projekční maticí na binární vyjádření bucketu, které je pro každý bit bucketu $i \in \langle 0 \dots d)$ spočítáno následovně:

$$\text{dot product} = \sum_{j=0}^{n} u_j M_{j,i}$$
$$bucket_i = \begin{cases} 0 & \text{if (dot product} < 0) \\ 1 & \text{if (dot product} \geq 0) \end{cases} \tag{2.12}$$

Podobní uživatelé jsou pak s velkou pravděpodobností ve stejném bucketu.

2.5.5 Multi-probe LSH

K dosažení dobrých výsledků je potřeba, aby LSH využívalo více hashovacích fukncí ke generování dobré množiny potencionálních výsledků, ale počet hashovacích funkcí zvyšuje paměťovou a výpočetní složitost. Metoda multi-probe může dosahovat podobných výsledků s menším počtem hashovacích funkcí. Je založena na myšlence a předpokladu, že podobní uživatelé nejsou zahashováni do stejného bucketu, avšak mohou být zahashováni do bucketů, které jsou si navzájem blízké (hash bucketů je velmi podobná a liší se například v jednom bitu). [19]

Cílem této techniky v LSH podle [19] mohou být 2 cíle, které chceme dosáhnout.

- Uložení velkého množství hashovacích funkcí na mnoha dimenzionálních datech je paměťově náročné a využitím blízkých bucketů můžeme snížit počet těchto funkcí, a tím i paměťovou náročnost modelu při dosažení podobných výsledků.
- Dosažení větší přesnosti při aproximaci k-NN v podobném čase při použití stejného množství hashovacích funkcí.

V práci představím techniku pro generování podobných bucketů založenou na hammingovy vdálenosti. Blízké buckety jsou seřazeny v pořadí jejich hammingovy vzdálenosti od bucketu, kam byl zahashován dotazující vektor $\mathbf{q}$.

Blízké buckety lze generovat různými způsoby. Například náhodným otočením vždy jednoho bitu v binární reprezentaci bucketu. Další výhodnější metodou může být prostor a buckety generovat jako okolí, jehož velikost je závislá na konkrétní hodnotě hammingovy vzdálenosti od bucketu, kam by zahashován dotazující vektor $\mathbf{q}$ [20].

Při vyhodnocení algoritmu LSH s metodou multi-probe je jako v základním LSH dotazující uživatelský vektor **q** zahashován do odpovídajícího bucketu. Na základě tohoto bucketu jsou vygenerovány podobné buckety z určité hammingovy vzdálenosti a výslední uživatelé ze všech bucketů jsou vráceni jako potencionální kandidáti na výsledek.

2.5.6 Úprava hashovací funkce na základě popularity položek

Doporučování položek, které se nacházejí v long-tail (tento problém jsem popisoval v kapitole 2.2.1), je pro uživatele velice cenné, ale na druhou stranu je dobře známé, že úspěšnost doporučovacího systému klesá při doporučování položek z long-tail [21].

V každé reálné datové sadě jsou položky, které jsou u uživatelů více oblíbené než jiné. Tuto podobnost lze zjistit z nasbíraných dat. Základní představa je, že je mnohem více pravděpodobnější, že uživatel kladně zareaguje na populární položku než na nepopulární [21].

V [21] je popularita položky brána jako počet relevantních hodnocení uživatelů dané položky. V mé práci pracuji s hodnocením položek jednotlivými uživateli a také s ohodnocením podle významu interakce uživatele s položkou. Popularitu pak spočítám jako váhu w pro každou položku $i \in I$ následovně:

$$w_i = \sum_{j \in U} r_{j,i} \tag{2.13}$$

Takto ohodnocené položky dávají možné pořadí a informaci o popularitě jednotlivých položek v databázi.

Těchto hodnot popularity lze využít k úpravě projekční matice. Každá položka v uživatelském vektoru je násobena s hodnotami na stejných pozicích na matici $M_{n,d}$. Některým hodnotám v projekční matici lze zvětšit hodnotu a tím váhu v rozhodnutí o jednotlivých bitech hashe bucketu. Tím se zvyšuje pravděpodobnost sdílení podobných bucketů pro uživatele, kteří mají oblíbené podobné populární položky.

Projekční matici $M_{n,d}$ lze upravit podle následující formule na matici $M\alpha_{n,d}$:

$$\forall j \in \langle 0 \dots d), \forall i \in I : M\alpha_{i,j} = M_{i,j} w_i^{\alpha} \tag{2.14}$$

Parametr α určuje jaký důraz a vliv má popularita položek na hashovací funkci a rozdělení uživatelů do bucketů. Pro $\alpha = 0$ nemá na projekční matici váha předmětů žádný efekt a jedná se o projekční matici z 2.5.4.

Kapitola 3

Návrh

V této kapitole popíši návrh frameworku, který je určen k testování metod LSH představených v kapitole 2.5 a jejich různých parametrizací. Při návrhu budou využity myšlenky z kapitoly 2. Nejprve popíši základní obecnou strukturu frameworku, z té následně podrobněji popíši jednotlivé moduly, ze kterých se framework skládá. Na závěr této kapitoly představím principy testování algoritmu k-NN a úspěšnosti doporučování.

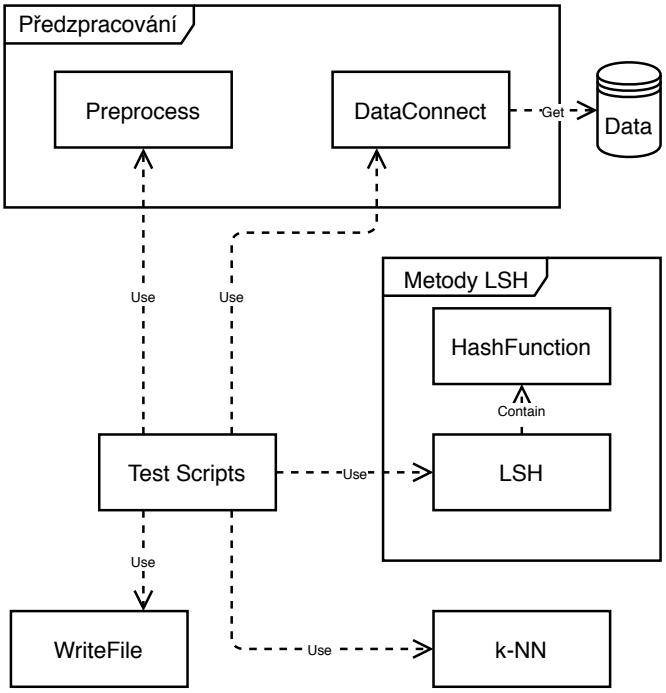
3.1 Obecná struktura frameworku

Na obrázku 3.1 je znázorněna obecná struktura frameworku. Základem jsou testovací scripty, ve kterých jsou systematicky provedeny testy zvolených parametrizací a metod LSH. Ve frameworku se nacházejí dva hlavní testovací skripty, které sdílejí jednotlivé moduly. Různé skripty jsem zvolil z důvodu odlišnosti testování přesnosti algoritmu k-NN a úspěšnosti doporučování na základě algoritmu k-NN.

Dále na obrázku 3.1 je vidět modul předzpracování, který obstarává stažení offline dat z konkrétní datové struktury. Je možné implementovat různé moduly, které mohou stahovat data například z databáze nebo textových souborů. Každý modul musí implementovat příslušnou abstraktní třídu. V části předzpracování probíhá dále zpracování stažených dat pro rychlé vyhodnocování modelů k-NN a LSH.

Dalším důležitým modulem ve frameworku je modul k-NN. Algoritmus k-NN je popsán v kapitole 2.3. Modul slouží pro vytvoření modelu k-NN a vyhodnocování dotazů na k-nejbližších sousedů. Tento modul jsem do systému navrhl zejména z důvodu nutnosti referenčního modelu pro porovnávání a hodnocení metod LSH.

Metody LSH z kapitoly 2.5 představují vlastní modul. Modul slouží k rozdělení prostoru pomocí lokálně senzitivních hashovacích funkcí definovaných v sekci 2.5.2. Modul aproximuje dotaz na k-nejbližších sousedů. Skládá se ze dvou částí, první je vyhodnocení k-nejbližších sousedů na principu k-NN



Obrázek 3.1: Obecná struktura frameworku

a druhý představuje model hashovacích funkcí, který vrací množinu potenciálních uživatelů.

Poslední modul je pomocný a slouží k zápisu dat. Data jsou do třídy posílána jako slovníky klíč-hodnota. V modulu je implementováno řazení dat a zápis do konkrétního formátu výstupu pro tvorbu grafů. Pro práci jsem zvolil formát csv představující zápis tabulky v souboru, kde jsou hodnoty oddělené čárkami.

3.2 Předzpracování dat

Testování úspěšnosti frameworku probíhá na offline datech, proto prvním krokem je získání dat. Modul pro stažení dat implementuje abstraktní třídu. Pomocí konkrétní implementace je možné data získávat z libovolného datového zdroje, například připojením do různých databázích nebo načítáním z textového souboru. Modul musí implementovat metodu, která umožní data načíst jako seznam n-tic ve tvaru (uživatel, položka, hodnocení).

V práci používám modul pro textový soubor, ze kterého načítám data. Modul pouze stahuje data ze specifikovaného souboru. Soubor je možné spe-

cifikovat argumentem na příkazové řádce. Data jsou získávána jako seznam n-tic. Hodnocení může kromě samotného uživatelského hodnocení obsahovat také váhu ohodnocení určité interakce uživatele s položkou.

Taková surová data jsou jako seznam n-tic předávány do modulu předzpracování. Ten zajišťuje dvě hlavní činnosti nutné pro předzpracování dat. Jedna je samotné předzpracování dat a druhá rozdělení dat na trénovací a testovací podmnožiny. Činnosti jsou prováděny v následujících krocích:

1. N-tice jsou postupně zpracovány do slovníku uživatelů, kde klíčem je uživatel a hodnotou je slovník položek. Klíčem slovníku položek jsou jednotlivé položky a hodnotami je hodnocení uživatelem. Dále jsou vytvořeny obousměrné slovníky pro překlad uživatele nebo položky na id a naopak.

   Slovníkovou reprezentaci jsem zvolil, protože matice jsou velice řídké (angl. Sparse matrix). Uživatel přijde do kontaktu pouze s několika položkami. Matice tedy obsahuje většinu nulových pozic. Reprezentace více rozměrným polem by byla paměťově neefektivní.

2. Data jsou rozdělena na trénovací a testovací část. Velikost jednotlivých částí je určena procentuálně vzhledem k počtu uživatelů. Testovací data představují náhodně vybrané procento uživatelů. Zbývající uživatele slouží jako trénovací data pro model.

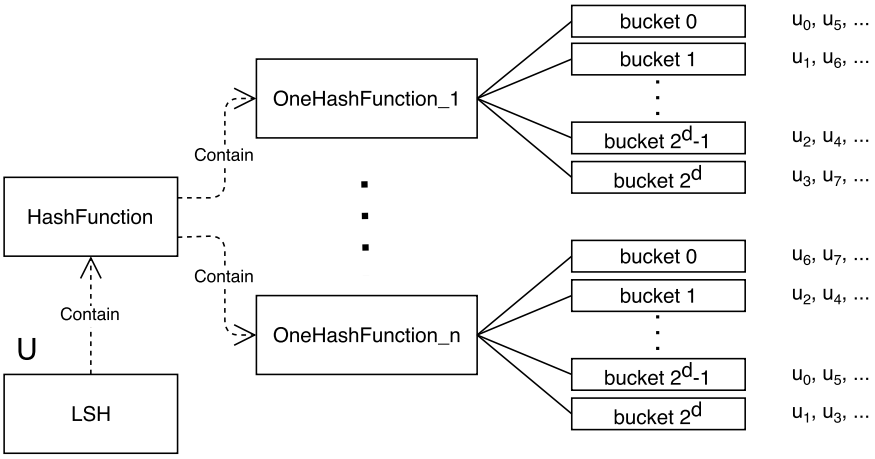
3. Na trénovacích datech dále spočítám popularitu pro každou položku podle kapitoly 2.5.6. Také ze vzorce pro kosinovou podobnost z kapitoly 2.3.2 nejprve vypočítám sumu ve jmenovateli a její odmocninu pro každého uživatele pro urychlení výpočtu kosinové podobnosti při vybavování modelu.

Předzpracovaná trénovací data jsou uložena do datové třídy a slouží k vytvoření konkrétních modelů LSH a k-NN.

3.3 Algoritmus k-NN

Pro testování výkonnosti a úspěšnosti metod LSH jsem v práci vytvořil také modul algoritmu k-NN. Tento modul vždy při hledání nejbližších sousedů prochází celou množinu uživatelů, je to tedy tzv. brute force model hledání podle definice z kapitoly 2.3.

Modul k-NN obsahuje model jako slovník uživatelů s jejich položkami a ohodnocením. Při dotazu na k-nejbližších sousedů je ke každému uživateli spočítána kosinová podobnost s dotazujícím vektorem uživatele. Pole uživatelů s kosinovou podobností je následně seřazeno a algoritmus vrací k sousedů s největší hodnotou podobnosti. Z k-nejpodobnějších uživatelů je při testování doporučování počítáno top-n nejrelevantnějších položek.



Obrázek 3.2: Na obrázku je zobrazen princip vytvoření modelu LSH s buckety z množiny uživatelů U

Výpočetně nejnáročnější část algortimu je počítání kosinové podobnosti přes všechny uživatele podle vzorce z kapitoly 2.3.2. Jednou z možností zrychlení je využití napočítaných sum vektorů jednotlivých uživatelů. Dalších optimalizací může být dosaženo pomocí úprav násobení vektorů.

Hlavní účel začlenění tohoto modulu do frameworku je vytvořit referenční řešení s referenčním časem na dotaz k-nejbližších sousedů nebo top-n doporučení. Vůči tomuto řešení jsou porovnávány modely LSH z následující kapitoly.

3.4 Začlenění metod LSH do algoritmu k-NN

V kapitole 2.5 jsem představil metody LSH pro aproximaci k-NN. Tyto metody jsem začlenil do algoritmu nejbližších sousedů. Na obrázku 3.2 je vidět ilustrace modulů LSH. LSH přijímá množinu trénovacích dat U jako slovník uživatelů. Pomocí HashFunction pro správu hashovacích funkcí z obrázku 3.2 jsou uživatelé rozděleni do bucketů.

3.4.1 Hashovací funkce

V práci jsem se rozhodl pro hashovací funkce využít projekční matice z kapitoly 2.5.4, které jsou dobře citlivé na kosinovou podobnost. Zároveň lze modely z těchto projekčních matic dobře vyhodnocovat a upravovat.

Při vytváření modelu je vytvořena jedna nebo většinou několik hashovacích funkcí určených argumenty z příkazové řádky. Poté vždy celou množinu U za-

hashuji každou hashovací funkcí do jednotlivých bucketů. Počet bucketů se odvíjí od dimenze projekční matice, která je určená parametrem d a popsána v kapitole 3.2.

Na obrázku 3.2 můžeme vidět, že každá hashovací funkce rozdělí uživatele mírně odlišně, proto je nutné, aby framework pracoval s různým množstvím projekčních matic a výsledky z jednotlivých matic efektivně agregoval.

Projekční matice generuji z uniformního pravděpodobnostního rozdělení. U každé matice lze pomocí parametrů specifikovat rozsah generovaných hodnot. Dále je možné hashovací matici upravit podle popularity jednotlivých položek z kapitoly 2.5.6. Framework umožňuje vytvářet různé modely z odlišnými hashovacími funkcemi zadanými v argumentech na příkazové řádce při spuštění testování.

3.4.2 Vybavování modelu LSH při dotazu na *k* nejbližších sousedů

Další částí je samotné vybavování modelu při dotazu. Model umožňuje dva typy dotazů. První dotaz je na k-nejbližších sousedů a druhý pro generování top-n nejrelevantnějších položek pro doporučení.

Dotazující uživatelský vektor je pomocí všech hashovacích funkcí v modulu HashFunction z obrázku 3.2 zahashován do konkrétních bucketů. Uživatelé v těchto bucketech jsou potencionální kandidáti na výsledek. Z každé hashovací funkce je vrácena množina uživatelů. Nad těmito množinami uživatelů je nutné provést sjednocení. Do modulu LSH jsou následně vráceni potencionální uživatelé.

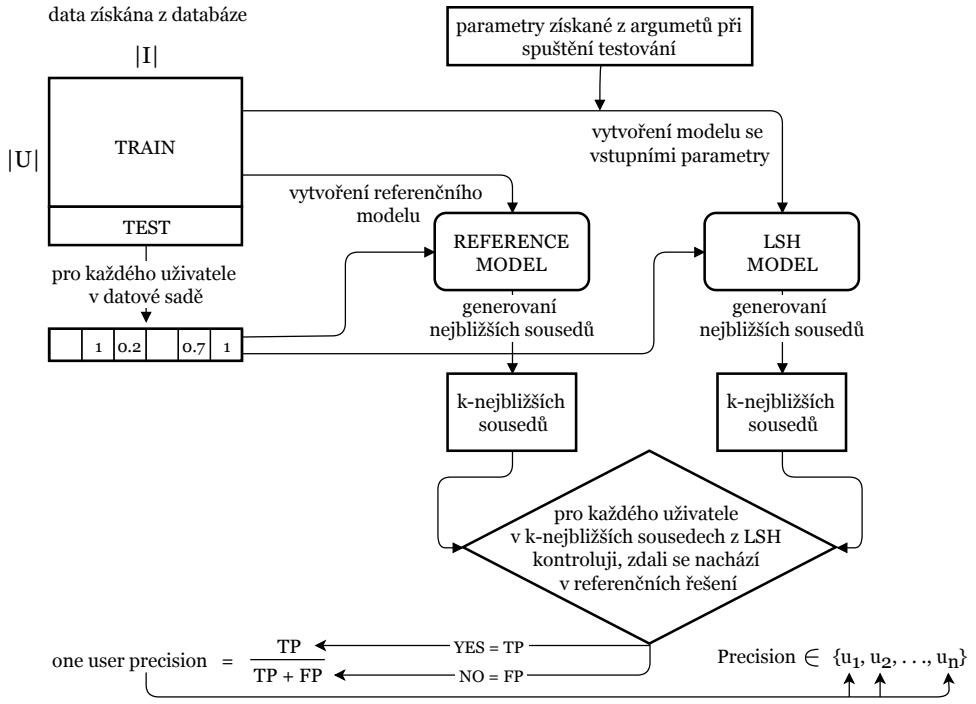
Na množině potencionálních uživatelů z modelu hashovacích funkcí jsou následně spočítány podobnosti s dotazujícím vektorem uživatele. Uživatelé se spočítanou podobností jsou seřazeni a modul vrací k-nejpodobnějších uživatelů nebo top-n nejrelevantnějších položek pro doporučení.

Prohledávání okolních bucketů popsané v kapitole 2.5.5 je možné specifikovat při každém dotazu. Pro stanovení blízkých bucketů existuje v modulu HashFunction metoda, která je k danému číslu bucketu a určené dimenzi schopná vracet blízké buckety podle určené velikosti okolí.

3.5 Testování a vyhodnocování

Součástí celého frameworku jsou skripty, které slouží k testování úspěšnosti. Testování přesnosti k-NN probíhá vždy vůči referenčnímu řešení. Úspěšnost doporučovaní může být měřena i nezávisle na referenčním modelu.

Společnou součástí obou testovacích skriptů je využití modulu předzpracování a modulů k vytvoření modelů k-NN a LSH. Dále se již metody testování liší a popíši je zvlášť v oddělených podkapitolách.



Obrázek 3.3: Na obrázku je zobrazen způsob testování přesnosti metod LSH vůči referečnímu brute force k-NN modelu

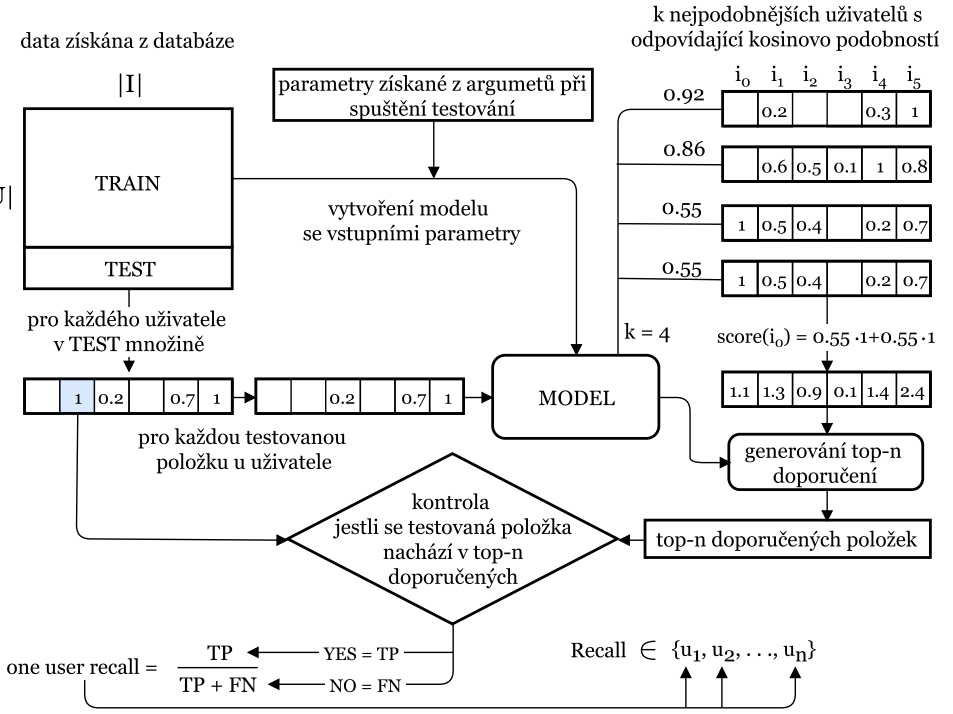
3.5.1 Testování přesnosti k-NN

Na obrázku 3.3 je znázorněno testování úspěšnosti algoritmů k-NN, které popíši v této kapitole. Toto testování je základním testováním modelů LSH ve frameworku. Z trénovacích dat je vždy vytvořen referenční model k-NN. Dále jsou podle parametrů vytvářeny jednotlivé modely LSH, které jsou testovány.

Natrénované modely LSH vrací pro každého uživatele z testovací množiny k-nejpodobnějších uživatelů. Tito nejpodobnější uživatelé jsou kontrolováni vůči referenční množině.

Při hledání uživatelů vrácených modelem LSH v referenční množině není zohledňováno jméno uživatele, ale hodnoty podobnosti uživatelů. Je to z důvodu možnosti shody podobností u více uživatelů (stejní uživatelé) v trénovací množině.

Při testování každého uživatele je vždy určena přesnost podle tabulky matice záměn 2.1 z kapitoly 2.4.2. Dále je pro každého uživatele zaznamenaná rychlost vrácení k-nejbližších sousedů. Výsledná přesnost a rychlost je průměrem hodnot přes všechny uživatele z testovací množiny. Rychlost není ve výsledcích zaznamenaná v jednotkách času, ale procentuálně vzhledem k průměrné rychlosti referenčího modelu.



Obrázek 3.4: Na obrázku je zobrazen způsob testování úspěšnosti doporučování pomocí leave-one-out křížové validace

3.5.2 Testování úspěšnosti doporučování

Na obrázku 3.4 je vidět princip testování úspěšnosti doporučování. Testování doporučování není závislé na porovnávání vůči referenčnímu modelu, ale je výhodné z důvodu porovnání recallu a času u doporučování. Mezi sebou mohou být ale porovnávány i jednotlivé parametrizace a metody LSH.

Podle argumentů z příkazové řádky jsou vytvářeny jednotlivé modely, které jsou následně naučeny trénovacími daty. Při vybavování poté model nalezne k-nejbližších sousedů, ze kterých spočítá příslušné váhy podle kapitoly 2.3.3. Při počítání je použito hodnocení konkrétní položky uživatele, které je váženo hodnotou kosinové podobnosti. Ohodnocení je počítáno pro položky, které uživatel neviděl a je součtem těchto hodnot přes všech k-nejbližších uživatelů. Nakonec je vráceno n položek s nejvyšším hodnocením jako top-n doporučení.

Testování každého modelu následně probíhá přes každého uživatele několikanásobně, protože se testují jednotlivé položky uživatelů. Z vektoru uživatele je vždy vynechána jedna položka a na uživatele bez této položky je generováno top-n doporučení z konkrétního modelu. Každý uživatel je testován na

maximálně m vynechaných položek nebo na všechny, pokud jich je méně než m, kde m je specifikováno konstantou.

Poté se kontroluje, zdali se nachází vynechaná položka v top-n doporučení. Pokud ano – je zaznamenána do matice záměn jako TP. Pokud ne – tak do FN. Podle vzorce z kapitoly 2.4.2 je pak spočítán recall pro jednoho uživatele. Zároveň je také pro každou testovanou položku a testovaného uživatele zaznamenáván čas generování top-n doporučení. Výsledný čas je průměr všech časů při generování doporučení. Výsledný recall je průměrem přes všechny uživatelské recally. Pro testování doporučování se každý model a každá položka testuje pro různé hodnoty k, které jsou mocninami dvojky od $k = 1$ do specifikované hodnoty k.

Kromě času a recallu je dále při doporučování zaznamenáván také catalog coverage z kapitoly 2.4.3. Do mapy všech položek jsou zaznamenány položky, které systém doporučil. Výsledek je pak procento položek, který je schopen systém doporučovat. Hodnota catalog coverage je závislá na velikosti testovací množiny.

Kapitola 4

Implementace

Navržený framework z kapitoly 3 jsem implementoval v programovacím jazyce Python 3.6, který je dobře použitelný pro efektivní implementaci testovacích modulů. Testovací framework není implementován více vláknově z důvodu testování časové náročnosti a porovnání zrychlení jednotlivých metod LSH vůči referenčnímu k-NN.

V této kapitole popíši použité knihovny Pythonu, argumenty použité při implementaci a optimalizace, které byly řešeny. Dále také popíši problémy, na které jsem při implementaci narazil a následně jejich řešení. Nakonec představím modul, který umožňuje rychlou tvorbu grafů z dat generovaných frameworkem.

4.1 Použité vybrané knihovny

Při implementaci frameworku jsem využil některé knihovny, které nejsou dostupné přímo v základním Pythonu. U každé knihovny uvedu k jakému účelu jsem ji ve frameworku využil. Knihovny jsou následující:

numpy Univerzální knihovna, která je využitelná pro velké množství operací, které jsou navíc prováděny časově velice efektivně oproti vestavěným funkcím v Pythonu. Ve své práci ji využívám ke generování hashovacích matic z určeného pravděpodobnostního rozdělení a také při sjednocení množin uživatelů z jednotlivých hashovacích funkcí. Dále také při porovnávání desetinných čísel na danou přesnost a při převodu čísel na textovou binární reprezentaci.

bidict Knihovna, která slouží k vytváření obousměrných slovníků, které využívám pro překlad uživatelů a položek na id a naopak.

bitstring Tuto knihovnu využívám k převodům binární reprezentace bucketů do číselného označení pro získání uživatelů z daného bucketu.

4.2 Argumenty frameworku a jejich využití

Při testování modelů LSH je možné tyto modely a celé testování specifikovat pomocí argumentů z příkazové řádky. Argumenty se dají rozdělit na několik kategorií, které jsou popsány v následujících podkapitolách.

4.2.1 Argumenty pro běh testovacího frameworku

Tyto argumenty nesouvisí přímo s jednotlivými modely LSH, ale slouží k nastavení důležitých vlastností při testování frameworku.

-i, --inputfile Povinný argument, který určuje název datového souboru ve formátu csv, který obsahuje následující tři povinné sloupce v libovolném pořadí. Sloupce jsou „userid“, „itemid“ a „rating“.

-o, --outputfile Povinný argument, který určuje název souboru i s existující cestou, kam jsou při testování zaznamenávány výsledky testování. Pokud soubor neexistuje, bude vytvořen. Pokud existuje, tak jsou data zapisována na konec souboru.

-r, --repeat Nepovinný argument, jehož základní hodnota je nastavena na 1. Určuje počet opakování testování určené specifikace modelu. Jednotlivá opakování jsou zapisována za sebe do souboru konkrétní specifikace.

-t Nepovinný argument, který specifikuje velikost testovací množiny. Velikost je specifikována v procentech vůči celkovému počtu uživatelů v datech. Pokud argument není specifikován, bude použita velikost testovací množiny 10 %.

-k Počet nejbližších sousedů při dotazu. Argument má trochu odlišný význam při testování úspěšnosti doporučování, který popíši zvlášť u speciálních argumentů pří testování úspěšnosti doporučování. Při testování přesnosti na k-nejbližších sousedů specifikuje počet nejbližších sousedů, na kterých jsou modely testovány. Pokud není specifikován, je použita základní hodnota 10.

4.2.2 Argumenty modelu LSH

Argumenty v této části specifikují přímo strukturu jednotlivých modelů, způsob testování a dotazování se do modelů. Argumenty jsou následující.

-d, -di Tyto argumenty specifikují dimenzi projekčních matic konkrétního modelu. Je možné testovat jeden nebo více modelů určených intervalem. Jednotlivé modely LSH s různými hashovacími funkcemi jsou vytvářeny a testovány postupně. Jeden z následujících argumentů je vždy povinný.

-d Určuje dimenzi testovacích matic pro jeden model, který je otestován.

-di Specifikuje interval dimenze matic v hashovacích funkcích. Jednotlivé modely LSH s hashovacími funkcemi s různou dimenzí jsou následně postupně otestovány a výsledky zapisovány do souborů.

-c, -ci Argumenty určující velikost okolí bucketu, kam je zahashován testovaný uživatel, pomocí hammingovy vzdálenosti bucketů (maximální počet změn bitů v binární reprezentaci bucketu). Metoda je popsána v kapitole 2.5.5. Argumenty je možné specifikovat dvojím způsobem a jeden z následujících je vždy povinný.

-c Určuje jednu konkrétní velikost okolí, která bude určena při dotazování a vyhodnocování potencionálních kandidátů na výsledek z hashovacích funkcí.

-ci Specifikuje interval hodnot okolí. Model je vždy pak postupně při dotazování testován na jednotlivé velikosti okolí dané intervalem.

-M Nepovinný argument specifikující počet standardních hashovacích funkcí v modelu. Každá taková hashovací funkce je generována z rozdělení $\mathcal{U}(-1, 1)$ bez použití vah jednotlivých položek.

-m Argument, kterým je možné specifikovat vytvářené hashovací funkce. Přijímá textový řetězec skládající se z několika podřetězců oddělených „+“. Pro n hashovacích funkcí je tvar argumentu následující:

$$\textbf{-m}\ [H_1]\ [+]\ [H_2]\ [+] \ldots [+]\ [H_n]$$

Každé H může být specifikováno třemi různými způsoby, které jsou:

- Když je H prázdný řetězec je použita standardní hashovací funkce.
- H jako jedna hodnota, která určuje velikost alfy pro hashovací funkci (popsaná v kapitole 2.5.6).
- H jako dvojice hodnot specifikuje a, b u rozdělení $\mathcal{U}(a, b)$ při generovaní hashovací funkce.
- H jako trojice hodnot, kde první specifikuje parametry a a b u uniformního rozdělení a třetí určuje alfu.

4.2.3 Speciální argumenty při testování úspěšnosti doporučování

Některé argumenty z této části mají odlišné chování a nebo jsou zadávány pouze při testování úspěšnosti doporučovaní:

-k Tento argument při testování úspěšnosti doporučování určuje horní hranici testovaného k. Při testování jsou testovány hodnoty k, které tvoří řadu $1, 2, 4, 8, \ldots$ a jsou tedy vždy mocninou dvojky. Argument je nepovinný a jeho základní hodnota je nastavena na 8196.

-n Nepovinný argument, kterým se dá specifikovat počet doporučovaných položek v top-n doporučení. Pokud není specifikován, je použita výchozí hodnota 5.

-compare Tento přepínač specifikuje, zdali se má testovat i doporučovací úspěšnost pro referenční brute force model na stejné testovací sadě k porovnání zrychlení a úspěšnosti.

4.2.4 Implementační argumenty frameworku

Tyto argumenty se netýkají přímo testování metod LSH, ale jsou užitečné pro běh frameworku a generování některých výsledků. Argumenty jsou nepovinné a následující:

-buckets Argument specifikuje, zdali budou generovány csv soubory s rozdělením uživatelů mezi jednotlivé buckety v každé hashovací funkci. U každého bucketu je uvedena také průměrná hodnota popularity položky normalizována mezi 0 a 1.

-terminal Argument, který je možno využít při debugování, kontrole dat a nastavení běhu testování. Při zadání argumentu framework vypisuje následující údaje na terminál podle zadané hodnoty.

Existují čtyři úrovně vypisování informací. Pokud není zadána hodnota argumentu, program informuje pouze o úspěšném ukončení běhu. Pokud je zadán parametr „basic“, jsou vypisovány pouze údaje o datové sadě a konci testování. Další úroveň specifikována slovem „info“ vypisuje podrobné informace o datové sadě, o vytváření modelů a o výsledcích jednotlivých modelů LSH. Třetí úroveň je debugovací specifikována slovem „debug“. Při tomto nastavení vypisuje framework maximální množství informací o běhu.

-log Argument, sloužící pro vytváření logu o běhu frameworku. Jako první hodnotu přijímá název souboru s existující cestou, do kterého bude log zaznamenáván. Druhý může specifikovat úroveň výpisu stejně jako u předchozího argumentu (-teminal) pouze s rozdílem, že jsou informace logovány s časem do souboru a je nutné vyplnit jednu ze tří možností.

4.3 Optimalizace

V této části uvedu některé optimalizace, kterými jsem se při implementaci LSH zabýval.

4.3.1 Hledání blízkých bucketů při metodě multi-probe

V kapitole 2.5.5 jsem popsal metodu LSH, která je založena na možnosti, že byli podobní uživatele zahashováni do blízkých bucketů, ale ne do stejného bucketu. Podobnost bucketů je založena na hammingovy vzdálenosti bucketů.

Blízké buckety tedy vždy tvoří okolí, které je specifikováno maximálním počtem změn bitů od binární reprezentace bucketu, kam byl zahashován dotazovaný uživatel. Do velikosti dimenze $d = 10$, kdy jsou uživatelé rozděleni do 1024 bucketů, lze spočítat libovolně velké okolí různými metodami v přiměřeném čase.

Při narůstajícím počtu bucketů je nutné hledání podobných optimalizovat. Při hledání jde o problém generování různých kombinací reprezentace bucketu, kde je změněno maximálně n bitů. Časově nenáročné je hledání bucketů, kde je změněn pouze jeden bit.

Při vytváření modelu tedy předzpracuji mapu, kde klíčem je název bucketu a hodnotou je datová struktura představující bucket se seznamem bucketů, které jsou vzdáleny na hammingovu vzdálenost 1. Konkrétnímu bucketu je při hledání zadána hloubka zanoření, ze které bude buckety vracet. Výsledné buckety jsou vkládány do množiny z důvodu eliminace stejných bucketů. Po provedení potřebného množství zanoření je získána výsledná množina bucketů, ve které se nacházejí možní kandidáti.

4.3.2 Optimalizace sjednocení výsledků z bucketů

Při používání většího množství hashovacích funkcí je velké množství vrácených uživatelů duplikovaných. Duplikované uživatele je nutné odstranit, tedy provést nad vrácenými uživateli sjednocení.

Pro tuto operaci mi časově v Pythonu nejlépe vyšla metoda, při které jsou uživatelé že všech hashovacích funkcí vkládáni do pole. Po vyhodnocení všech hashovací funkcí je použita metoda unique z knihovny numpy, která efektivně eliminuje duplicitní uživatele. Tato metoda měla zanedbatelný čas proti počítání kosinové podobnosti na počtu uživatelů, kteří byli vráceni hashovacími funkcemi.

4.4 Řešené problémy

V sekci řešené problémy popíši problémy, kterými jsem se zabýval při implementaci a testování frameworku.

4.4.1 Mnoho stejných uživatelů s jednou položkou

Prvním řešeným problémem, se kterým jsem se setkal při implementaci, byl problém se stejnými uživateli (většinou uživatele s jednou populární položkou) při vyhodnocování metod LSH na k-nejbližších sousedů.

Při testování algoritmů na hledání k-nejbližších sousedů, kde $k = 10$, se může stát, že k testovanému uživateli existuje v datové sadě např. 50 stejných uživatelů, kteří mají stejnou hodnotu podobnostní funkce. Při vyhodnocení a měření přesnosti určité metody LSH je možné, že byli vráceni jiní uživatelé než z referenčího řešení, ale se stejnou hodnotou podobnosti. Úspěšnost by pak záležela pouze na tom, jak je řazeno pole se spočítanými podobnostmi a podle toho i vráceno k-nejbližších sousedů.

Při vyhodnocování úspěšnosti algoritmu tedy nejsou porovnávána jména uživatelů, ale je porovnávána jednotlivá podobnost vrácených uživatelů. Přesnost poté lépe vypovídá o úspěšnosti metod LSH.

4.4.2 Uživatelé s mnoha položkami

V databázi se nachází uživatelé, kteří viděli velké množství položek. Při testování doporučování popsané v kapitole 3.5.2 by pak tito uživatelé byli testováni tolikrát, že by to mělo příliš velký vliv na celkovou dobu testování. Tento problém je řešen omezením na n testovaných položek u každého uživatele.

4.4.3 Množství načítaných položek

Posledním řešeným problémem bylo velké množství dat, které bylo načítáno z datových struktur. Bylo složité takové množství dat reprezentovat a zpracovávat najednou, proto bylo nutné množství načítaných dat omezit a načíst pouze vybranou podmnožinu dat.

4.5 Modul pro zjednodušenou tvorbu grafů

Pro dobrou interpretaci výsledků testování metod LSH jsem v práci vytvořil skript v Pythonu, pomocí kterého lze jednoduše vytvářet přehledné vektorové 2D grafy ve formátu pdf. Grafy jsou vytvářené pomocí nástroje Gnuplot.

Pomocí argumentů je specifikován název grafu, složka s datovými soubory, oddělovač ve formátu cvs, popis osy X a popis osy Y. Následně existují čtyři argumenty představující možnosti, jak lze vykreslit křivku v tomto grafu.

-plotpoint Pomocí argumentu lze vykreslovat body do grafu. Argument přijímá n-tice hodnot, kterými jsou postupně název csv souboru s daty, název sloupce pro osu X, název sloupce pro osu Y a název série bodů v legendě. Tyto n-tice lze opakovat pro potřebný počet křivek bodů. Opakování funguje také u všech tří následujících argumentů. Pokud je v názvu křivky grafu uveden prázdný řetězec, body křivky nebudou popsány v legendě.

-plotpointlabel Tento argument vykresluje body stejně jako předchozí, ale s tím rozdílem, že při specifikaci série bodů je navíc mezi sloupcem pro

osu Y a názvem specifikován sloupec pro popis daného bodu v grafu, např. velikost k při hledání nejbližších sousedů nebo počet bucketů.

-plot Je specifikován stejně jako argument plotpoint pouze s tím rozdílem, že mezi body vykreslí křivku.

-plotlabel Přijímá stejné hodnoty jako argument plotpointlabel, ale vykresluje mezi body křivku.

Kapitola 5

Experimenty

V této kapitole představím uskutečněné testování a jeho výsledky. V testech porovnám různé metody LSH. Ukážu vliv jednotlivých metod a přístupů na kvalitu modelů a možnosti kombinování těchto přístupů (angl. ensembling) pro dosažení optimálního modelu.

Algoritmy k-NN testuji podle principu navrženém v kapitole 3.5.1 vždy s $k = 10$. Čas a přesnost jsou vždy v procentech vůči referenčního modelu. Pomyslným cílem při testováním modelů LSH je dosažení 100% přesnosti za 0 % času.

Testování úspěšnosti doporučování podle principu z kapitoly 3.5.2 je vždy prováděno s $n = 5$, tedy top-5 doporučování. Dále při tomto testování také porovnám závislost přesnosti k-NN na úspěšnosti doporučování.

Testování bylo prováděno na zapůjčeném serveru z důvodu možnosti testování mnoha modelů současně a také kvůli velké náročnosti na operační paměť, protože jednotlivé modely musí být vytvořené v ní.

Kapitolu jsem rozdělil na několik podkapitol. Na začátku popíši datové sady pro testování, které budu označovat jako databáze. V dalších podkapitolách nejprve představím vliv jednotlivých metod LSH na čas a přesnost vyhodnocování. Tyto experimenty byly provedené na menší databázi. Na závěr této kapitoly ukážu nejlepší modely a jejich kombinace pro dosažení optimálních výsledků. Optimální modely zároveň otestuji i na druhé několikanásobně větší databázi.

5.1 Databáze

V této podkapitole popíši databáze, na kterých jsem prováděl testování. U každé z nich uvedu základní údaje o databázi nebo jejích částech, které byly použity k testování.

5.1.1 Databáze A – obchod s nábytkem

Databáze A je internetový obchod s nábytkem. Tuto databázi jsem použil pro prozkoumání a testování všech představených metod LSH, protože bylo možné spouštět velké množství testů a provést tzv. brut-force hledání optimálních modelů. Databáze A má následující parametry:

Počet unikátních záznamů uživatel-položka: 1 280 189
Počet uživatelů: 290 625
Počet položek: 11 101
Hustota matice: 0,039 %
Průměrný počet položek se kterými uživatel interagoval: 4,40
Velikost testovací množiny: 2538

5.1.2 Databáze B – služba poskytující filmy, videa

Databáze B je internetová služba poskytující filmy, videa a seriály, která je velice rozsáhlá. Její testování je velice náročné především při vytváření referenčního modelu. Proto na této databázi budou otestovány pouze optimální modely, které měly nejlepší úspěšnost na databázi A.

Data z této databáze byla z důvodu velikosti omezována. Omezená data měla následující parametry:

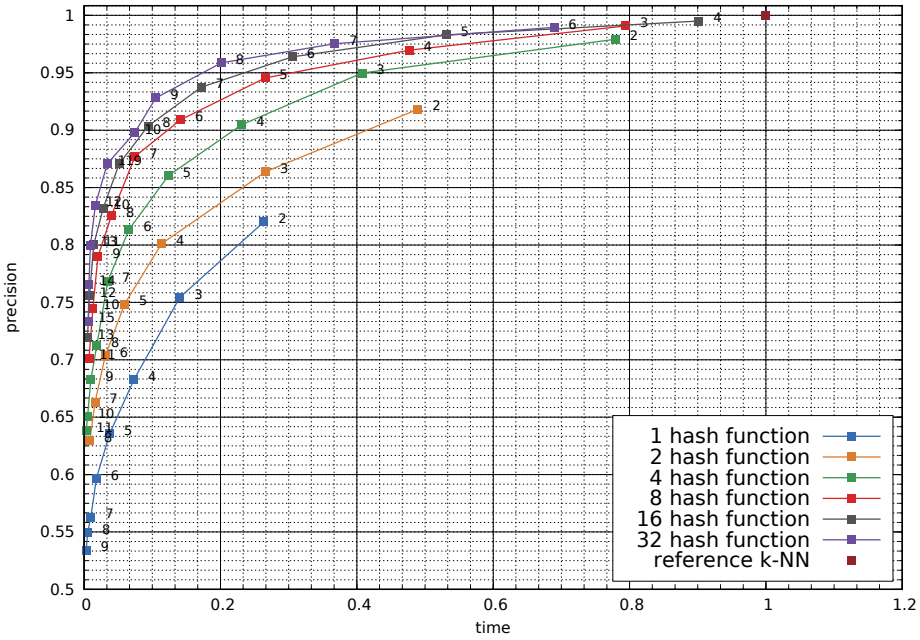
Počet unikátních záznamů uživatel-položka: 54 234 951
Počet uživatelů: 1 028 287
Počet položek: 52 689
Hustota matice: 0,1 %
Průměrný počet položek se kterými uživatel interagoval: 52,74
Velikost testovací množiny: 1066

5.2 Testování modelů s různým počtem projekčních matic

Na začátku jsem otestoval chování modelu při změně počtu hashovacích funkcí. Hashovací funkce byli implementovány jako projekční matice z kapitoly 2.5.4. Byl sledován především vliv na přesnost a čas. Výsledky popíši zvlášť u přesnosti k-NN a u úspěšnosti doporučování.

5.2.1 Přesnost 10-NN

Při testování 10-NN přesnosti je u každého bodu uvedena dimenze hashovací matice, která určuje počet bucketů. Z obrázku 5.1 pro 1 hashovací funkci je patrné, že při zvýšení dimenze o 1, tedy na dvojnásobný počet bucketů, se čas sníží přibližně na polovinu, protože počet uživatelů v bucketu je poloviční,



Obrázek 5.1: Porovnání přesnosti a času modelů LSH s referenčním řešením pro 1 až 32 hashovacích funkcí, u bodů je uvedena dimenze hashovací funkce

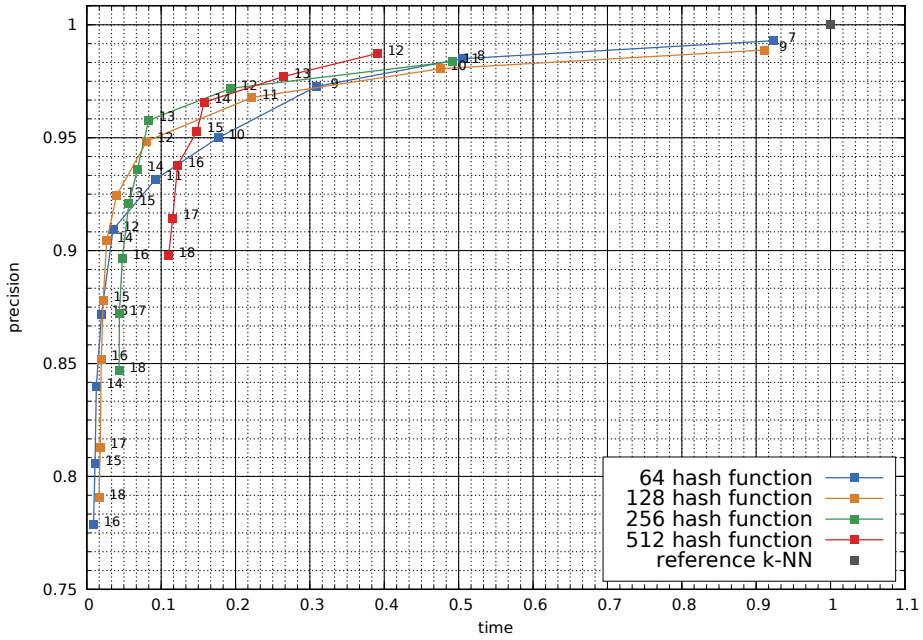
pokud hashovací funkce rozděluje uživatele rovnoměrně. U většího počtu hashovacích funkcí to již není platné, protože každá rozdělí prostor odlišně.

Na obrázku 5.1 je vidět, že s přidáváním hashovacích funkcí roste i přesnost modelů LSH. S přibývajícími hashovacími funkcemi je při zdvojnásobení jejich počtu nárůst úspěšnosti čím dál menší. Počet hashovacích funkcí jsem při testovaní tedy stále zvyšoval pro zjištění hranice, kdy již bude neefektivní přidávat další hashovací funkce. Neefektivní je to především z důvodu, že nad potencionálními uživateli je nutné provádět sjednocení.

Změna je pozorovatelná na obrázku 5.2 pro 256 hashovacích funkcí, kde je vidět trend, který se projevuje při vysokém počtu hashovacích funkcí. Modely s tolika hashovacími funkcemi poté dosahují horších časů při srovnatelné přesnosti. Tento trend platí pro časy pohybující se okolo 10 % a nižší v závislosti na počtu funkcí.

Naplno se projeví na křivce pro 512 hashovacích funkcí, kde je vidět horší čas při stejné přesnosti z důvodu provádění sjednocení. Další hashovací funkce již nepřinášejí možné nové informace s potencionálními uživateli v konkrétní databázi. Optimum pro počet hashovacích funkcí pro databázi A se tedy pohybuje mezi 64 až 128 hashovacími funkcemi.

V některých konkrétních dimenzích bylo dosaženo optimálních modelů i pomocí 256 hashovacích funkcí pro konkrétní časy. Toto lze vidět na obrázku 5.2 pro křivku s dimenzí hashovacích funkcí 13.



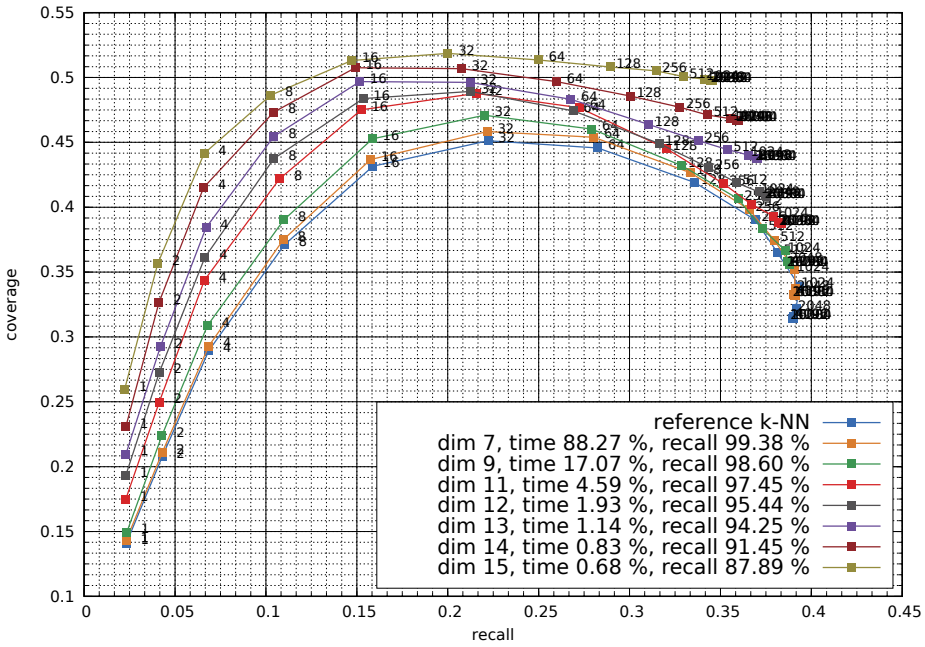
Obrázek 5.2: Porovnání přesnosti a času modelů LSH s referenčním řešením pro 64 až 512 hashovacích funkcí, u bodů je uvedena dimenze hashovací funkce

5.2.2 Úspěšnost doporučování

Na obrázku 5.3 je vidět graf z měření úspěšnosti doporučovaní. Křivka grafu představuje jeden model, neboli například jeden bod z grafu 5.2. V těchto grafech se může lišit i udávaný procentuální čas s časem grafu 5.2 z důvodu testování jednoho uživatele m-krát a rozdílných testovacích sad uživatelů. Testovací sada je vybírána náhodně, ale v jednom grafu je vždy použita stejná sada testovaných uživatelů.

Na grafu 5.3 je patrný trend, který mají křivky při použití metod LSH na této databázi. Při zvyšovaní dimenze hashovací funkce se zhoršuje recall a roste catalog coverage. Tento trend je dán především použitím hashovací funkce. Hashovací funkce má dobrou úspěšnost pro velmi podobné sousedy. Čím jsou si uživatelé méně podobní, tím je možné, že budou zahashováni do jiných bucketů, i kdyby měli být třeba v 1024 nejbližších sousedech.

Při počítání k-nejbližších sousedů se tak n nejpodobnějších sousedů může shodovat, ale čím nižší bude podobnost, tím budou uživatele rozdílní. Tito uživatelé následně zapříčiní zvyšování catalog coverage tím, že se v top-5 doporučení může objevit položka, která by se v doporučení brute-force modelu neobjevila. Stále se však může v doporučení nacházet položka, která byla vynechána při křížové validaci.



Obrázek 5.3: Graf ukazující úspěšnost doporučování modelu s 64 hashovacími funkcemi pomocí závislosti catalog coverage a recallu, procento času recallu je uvedeno v rámci optimállně zvoleného k

5.2.3 Závislost přesnosti k-NN a úspěšnosti doporučování

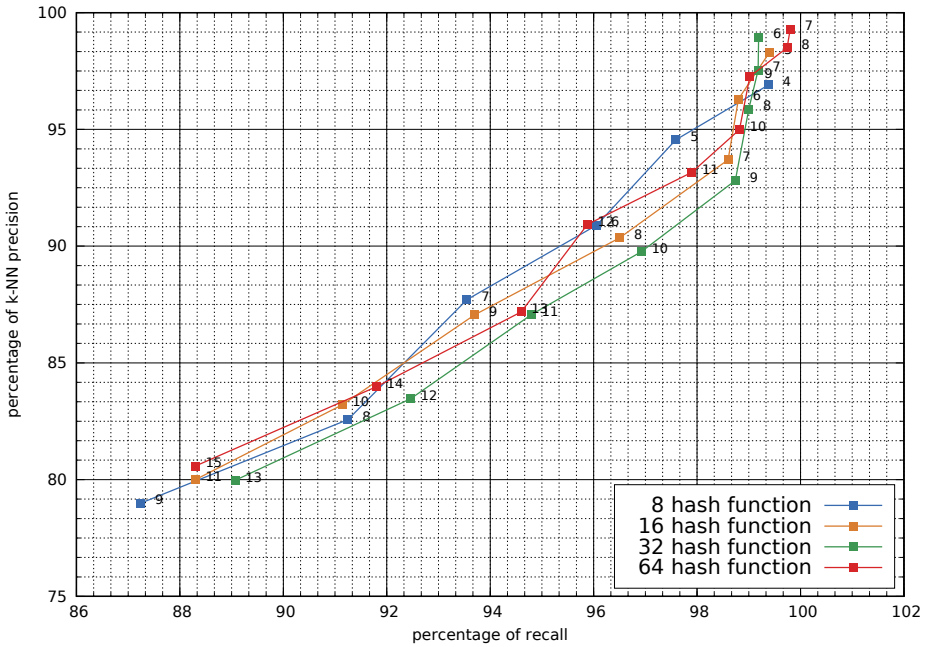
Při testování přesnosti modelů LSH při dotazu na 10 nejbližších sousedů a testování úspěšnosti doporučování (hodnota recall) je velice patrná závislost obou hodnot, která je vidět na grafu 5.4. Hodnota recall je při vyhodnocování použita vždy nejvyšší při optimální hodnotě k.

Je jasně patrné, že úspěšnost doporučování je téměř lineárně závislá na přesnosti dotazu na 10 nejbližších sousedů. Pouze při dosažení hodnoty recallu přibližně 99 % z referenčního modelu je nutné výrazněji zvyšovat přesnost k-NN pro dosažení 100 %.

Díky této závislosti je možné testovat mnoho modelů LSH na přesnost 10 nejbližších sousedů podle schématu 3.3. Testování modelů na přesnost k-NN je mnohem efektivnější a rychlejší než testování úspěšnosti doporučování.

5.3 Testování modelů LSH při aplikaci metody multi-probe

Dalším principem, kterým lze výrazně ovlivnit chování LSH, je metoda procházení blízkých bucketů, kam mohli být zahashováni podobní uživatelé. Vliv této metody představím pouze při přesnosti na 10 nejbližších sousedů díky



Obrázek 5.4: Graf ukazující závislost hodnoty recall na přesnost při dotazu na 10 nejbližších sousedů, přesnost k-NN i recall je uveden v procentech z referenčního modelu, u každého bodu je uvedena dimenze hashovací funkce

závislosti mezi přesností a hodnotou recall z kapitoly 5.2.3. Tato metoda je klíčová pro dosahování optimálních výsledků.

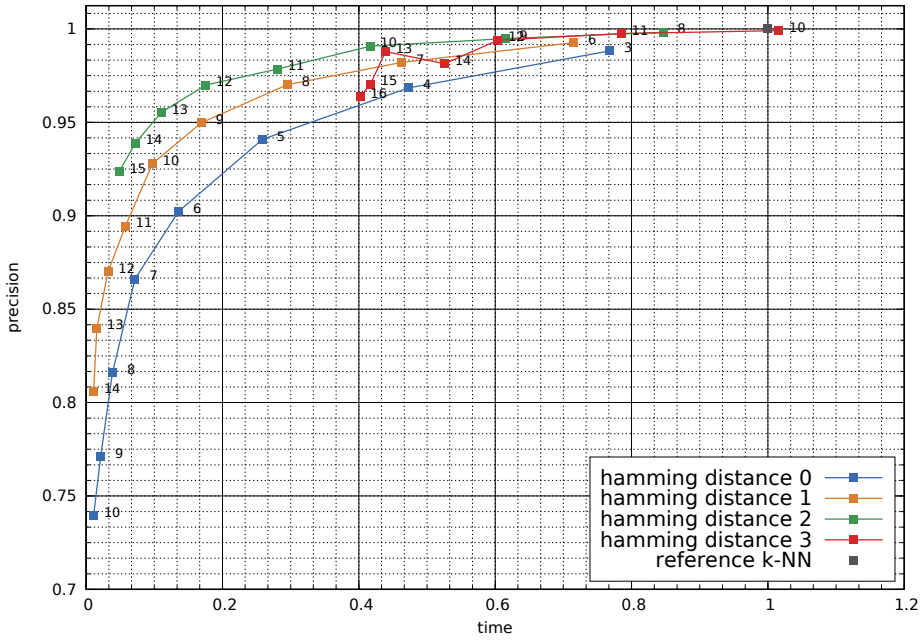
5.3.1 Přesnost 10-NN a vliv metody multi-probe

Na grafu 5.5 je ukázána základní křivka pro model složený z 8 hashovacích funkcí. Na stejný model bylo poté aplikováno procházení podobných bucketů podle hammingovy vzdálenosti binární reprezentace bucketů. Hash podobných uživatelů se může lišit několika bity, protože hodnota podobnosti uživatelů nebyla dostatečně vysoká.

Je patrné, že při použití metody procházení podobných bucketů, lze dosáhnout výrazného zlepšení modelu při stejném počtu hashovacích funkcí. S 8 hashovacími funkcemi použitými v grafu 5.5 lze dosáhnout stejné úspěšnosti jako například se 128 nebo 256 hashovacími funkcemi bez této metody.

Procházení blízkých bucketů má vliv na počet vrácených uživatelů hashovací funkcí, proto je nutné zvýšit dimenzi projekční matice, jak je vidět u popisů bodů na grafu 5.5 tak, aby bylo dosahováno konkrétních časů.

Na grafu 5.5 je na křivce, kde je zvolena maximální hammingova vzdálenost 3, patrná neefektivita modelu. Při velkém okolí je mnoho uživatelů z hashovacích funkcí stejných. Složitost roste se zvyšující se dimenzí hashovacích



Obrázek 5.5: Graf ukazující vliv metody pro procházení blízkých bucketů, pro modely je použito 8 hashovacích funkcí, u každého bodu je uvedena dimenze hashovací funkce

funkcí, velikosti okolí a počtem hashovacích funcí, protože podobné buckety určeného okolí je nutné generovat pro každou hashovací funkci při každém dotazu na k-nejbližších sousedů.

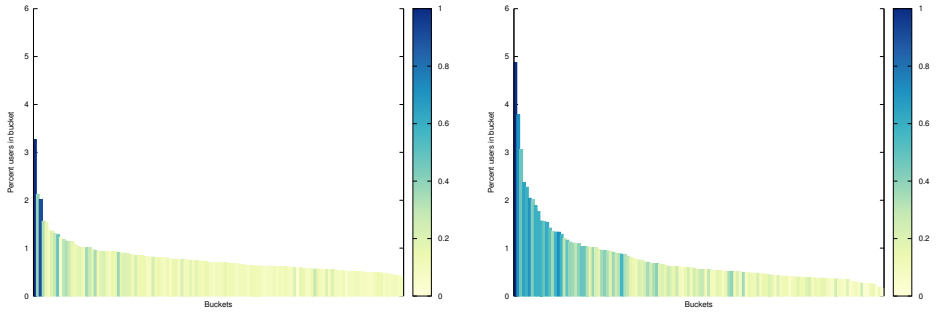
Tuto metodu lze efektivně využít například pro snížení paměťové náročnosti nebo při zlepšování efektivity LSH, pokud je již neefektivní přidávat další hashovací funkce.

5.4 Závislost úpravy hashovací funkcí podle popularity položek na úspěšnosti modelů LSH

V této kapitole ukážu vliv popularity položek popsané v kapitole 2.5.6 na úspěšnost modelů LSH. Úpravou hashovací funkce pomocí popularity položek lze upravit pravděpodobnosti zahashování uživatelů do jednotlivých bucketů. Rozdělení je vidět na obrázku 5.6. Dále dopad úprav také ukážu na grafech úspěšnosti LSH a popíši, kdy je výhodné takto upravovat hashovací matice.

5.4.1 Rozdělení uživatelů do bucketů

Na grafech na obrázku 5.6 je ukázáno rozdělení uživatelů do jednotlivých bucketů. Na levém grafu je vidět, že existují některé buckety, kde se shlukují



Obrázek 5.6: Obrázky ukazují procento uživatelů v jednotlivých bucketech, barva bucketu je průměrná váha položky v bucketu normalizovaná mezi 0 a 1, levý obrázek – rozdělení do bucketů bez použití paramentru alfa, pravý obrázek – rozdělení pomocí hashovací funkce, která byla upravena pomocí popularity položek

uživatelé, kteří viděli populární položky v databázi (mají nejvyšší průměrnou hodnotu vah položek).

Na pravém grafu, kde jsou hodnoty hashovací funkce upraveny podle popularity jednotlivých položek, je vidět tendence, kterou rozdělení má. Při vyšším vlivu populárních položek na výsledný bucket jsou uživatelé s těmito položkami hashováni do stejných bucketů. Z tohoto důvodu roste počet bucketů, ve kterých je vysoká popularita položek oproti rozdělení bez použití popularity položek.

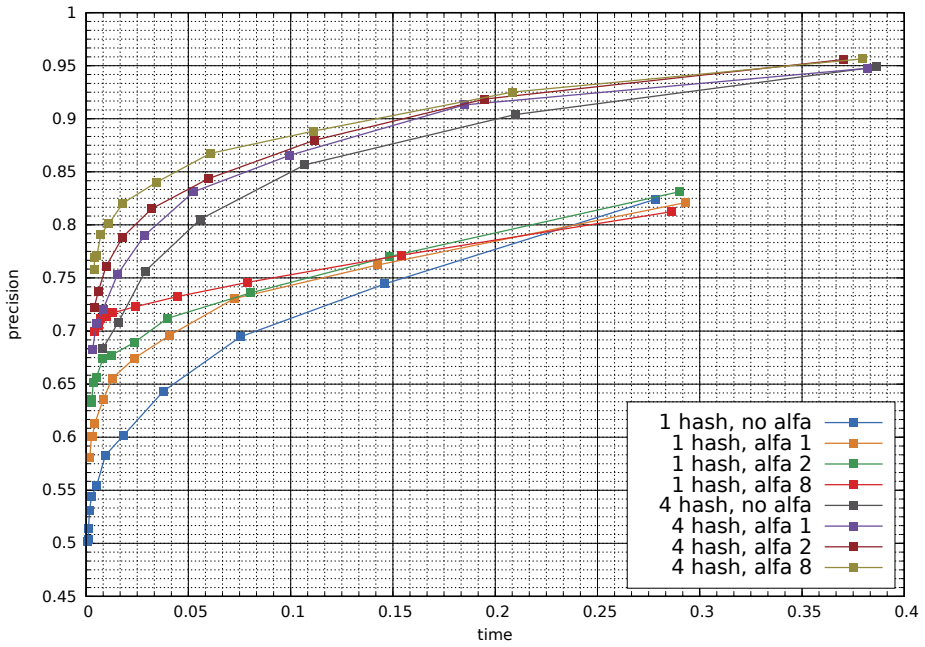
V těchto bucketech s mnoha uživateli se poté nachází především uživatelé se stejnými populárními položkami, a proto roste průměrná popularita a velikost některých bucketů na úkor rovnoměrného rozdělení.

Úpravu hashovacích funkcí jsem neprováděl na všech hashovacích funkcí, ale je provedena vždy na jedné hashovací funkci v modelu. Při úpravě všech hashovacích funkcí tato metoda již nepřináší žádnou výhodu. Proto v legendách grafu používám značení například „alfa 2“, které znamená, že v kombinovaném modelu byla použita jedna hashovací funkce s upravenými pravděpodobnostmi a váha popularity v rozdělení byla 2.

5.4.2 Úspěšnost 10-NN při použití parametru alfa

Na obrázku 5.7 je vidět pozitivní vliv alfy při dotazování na 10 nejbližších sousedů. Tento vliv je výrazný při použití jedné hashovací funkce než při použití více hashovacích funkcí, kde není rozdíl tak patrný.

Je mnohem pravděpodobnější, že se uživatelům budou líbit položky, které jsou oblíbené. Hashovací funkce jsou generovány náhodně z určitého rozdělení. Z tohoto důvodu nejsou vždy uživatelé, kteří by měli být v k-nejbližších sousedech, zahashováni do stejných bucketů. Tuto pravděpodobnost lze výrazně



Obrázek 5.7: Graf ukazující vliv úpravy hashovací funkce pomocí parametru alfa na přesnost při vyhodnocování dotazu na k-nejbližších sousedů

ovlivnit právě použitím vlivu váhy populárních položek nebo také navýšením počtu hashovacích funkcí.

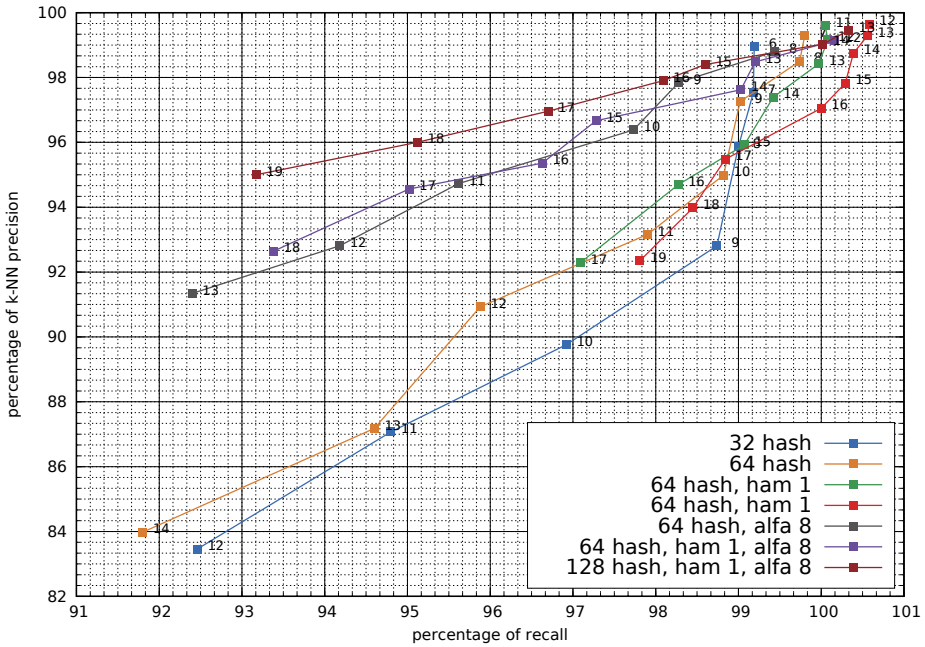
Při využití popularity se zvýší počet potencionálních uživatelů v bucketu, což zhoršuje časovou náročnost, ale výrazně se tím zlepší pravděpodobnosti, že 10 nejbližších uživatelů bude v konkrétním bucketu.

Z grafu 5.7 je jasně patrné, že vliv je výrazný pouze při nízkém čase, tedy velkém množství bucketů, protože zvyšuje pravděpodobnost, že bude 10 nejbližších sousedů sdílet bucket. Při menším množství bucketů již parametr na přesnost 10-NN nemá téměř vliv, protože pravděpodobnost sdílení bucketu 10 nejbližšími sousedy je již dost vysoká i bez využití popularity položek.

Při použití více hashovacích funkcí již není vliv parametru tak výrazný, protože každá hashovací funkce rozděluje prostor jinak, a tím zvyšuje pravděpodobnost, že budou po sjednocení podobní uživatelé mezi potencionálními výsledky.

5.4.3 Úspěšnost doporučování při použití parametru alfa

Na grafu 5.8 jsem do grafu závislosti přesnosti k-NN a recallu zanesl křivku modelu s upravenou hashovací funkcí podle popularity položek. Z grafu 5.8 jde vidět, že parametr alfa má vliv na závislost mezi přesností k-NN a úspěšností doporučování. Parametr „ham“, který budu i nadále používat v legendě grafů,



Obrázek 5.8: Graf ukazující vliv parametru alfa na závislost mezi přesností algoritmu k-NN a recallem při doporučování, u každého bodu je uvedena dimenze hashovací funkce

označuje maximální hodnotu hammingovy vzdálenosti bucketů při hledání blízkých bucketů.

Při zvyšování počtu hashovacích funkcí nebo při využití metody multi-probe je závislost mezi úspěšností na 10 nejbližších sousedech a doporučování stejná. Při využití popularity položek je již závislost odlišná a pro dosažení stejné úspěšnosti doporučování je nutné vyšší úspěšnosti na 10 nejbližších sousedů.

Úprava hashovacích matic podle popularity položek má tedy negativní vliv na úspěšnost doporučování na rozdíl od modelů bez této úpravy. I přes tuto skutečnost dosahují modely dobrých hodnot recallu při doporučování.

Důvod proč parametr alfa má negativní vliv při doporučování je takový, že pro databázi A je maximálního recallu dosaženo pro hodnotu $k = 1024$ podle grafu 5.3. Upravené hashovací funkce tedy fungují dobře při hledání 10 nejbližších sousedů, ale již ne při hledání 1024 nejbližších sousedů. Je totiž pravděpodobné, že 10 zmíněných nejbližších sousedů bude sdílet zájem nad populárními položkami nežli nad těmi nepopulárními. Tento trend ale neplatí pro 1024 nejbližších sousedů.

Parametr alfa zvyšuje pravděpodobnost sdílení bucketu těm uživatelům, kteří sdílejí populární položku, i když třeba jedinou. Nedává takový vliv ostatním položkám, proto ve výsledných k-nejbližších sousedech budou s větší prav-

děpodobností i uživatelé, kteří například nemají vysokou hodnotu podobnosti, ale sdílejí populární položku, která měla na rozdělení vyšší vliv.

Patrné je to právě při dosahování nižšího času, kdy je například procházeno jen 5 % uživatelů, které vrátil model LSH z celkového počtu. V 5 % uživatelů se nachází s větší pravděpodobností 10 nejpodobnějších uživatelů. Pokud hledám 1024 nejbližších uživatelů, tak velká část z nich nebude mít takovou hodnotu podobnosti, kterou by měli při použití referenčního modelu nebo nevyužití popularity v LSH.

V 1024 nejbližších uživatelích je následně mnoho uživatelů s nízkou hodnotou podobnosti. Pro počítání top-n doporučení jsou sice položky váženy podobností. Pokud však rozdíl hodnot není tak výrazný a počet těchto uživatelů je většinový, tak tito uživatelé mají vliv na doporučené položky. Tato většina v 1024 nejbližších sousedech jsou uživatelé sdílející pouze jednu populární položku. Systém pak doporučuje položky, které nejsou relevantní.

5.5 Využití testovaných metod k vytvoření optimálních modelů

V této kapitole využiji poznatků z testovaných metod LSH pro vytvoření optimálních modelů pro dosažení maximální úspěšnosti doporučování, která je pro mě prioritní.

Tyto modely otestuji na obou dodaných databázích. Nejprve popíši volbu modelů. V následující podkapitole naváži nastavením parametrů pro databáze a jejich vliv na model. Dále otestuji přesnost hledání 10 nejbližších sousedů a úspěšnost doporučování vybraných modelů na dodaných databázích.

Jako první optimální model jsem zvolil model se 64 hashovacími funkcemi a procházením blízkých bucketů do maximální hammingovy vzdálenosti 1. Hodnota 2 již pro takto vysoký počet hashovacích funkcí nedávala stabilní výsledky.

Druhý model bude totožný jen s dvojnásobným počtem hashovacích funkcí, tedy 128 a stejně nastaveným procházením blízkého okolí bucketů.

5.5.1 Nastavení parametrů modelů pro různé datové sady

Pro každou databázi je nutné nastavit správné parametry modelu LSH a to především velikost dimenze, množství uživatelů nebo použití metody multi-probe s ohledem na počet hashovacích funkcí a počet uživatelů v databázi.

Konkrétní hodnoty pro vybrané dva optimální modely jsou ukázány v tabulce 5.1. Časy a přesnost doporučování jsou v tabulce uvedeny procentuálně vzhledem k referenčnímu modelu. Z tabulky je patrné, že pro jeden model ale odlišné databáze závisí nastavení především na dimenzi hashovací funkce.

Je vidět, že čím více je v modelu hashovacích funkcí, tím menší je průměrný počet uživatelů na jeden bucket pro dosažení stejného času. Pro jeden zvolený

Tabulka 5.1: Výpis parametrů pro vybrané modely a nastavení parametru pro vybrané databáze, dim – dimenze hashovací funkce, usr – průměrný počet uživatelů na jeden bucket, rec – procentuální recall pro optimální k vůči referenčnímu modelu

Model	Databáze A				Databáze B			
64 hash function multi-probe 1	dim	time	usr	rec	dim	time	usr	rec
	11	70,76	119	100,05	14	13,35	62	99,05
	12	43,68	59	100,08	15	7,89	31	98,82
	13	20,50	30	99,97	16	4,28	16	97,85
	14	9,95	15	99,42	17	2,9	8	97,42
	15	5,26	7	99,07	18	1,89	4	96,76
	16	3,28	4	98,27	19	1,31	2	96,02
	17	2,48	2	97,09	20	1,05	1	95,02
	18	2	1	95,82				
	19	1,8	0,5	94,84				
128 hash function multi-probe 1	dim	time	usr	rec	dim	time	usr	rec
	12	79,80	59	100,58	14	26,95	62	99,31
	13	37,41	30	100,56	15	14,3	31	99,21
	14	18,20	15	100,39	16	8,59	16	98,45
	15	9,76	7	100,29	17	5,41	8	97,77
	16	6,15	4	100	18	3,72	4	97,34
	17	4,37	2	98,84	19	2,64	2	96,73
	18	3,66	1	98,45	20	1,9	1	95,69
	19	3,24	0.46	97,81				

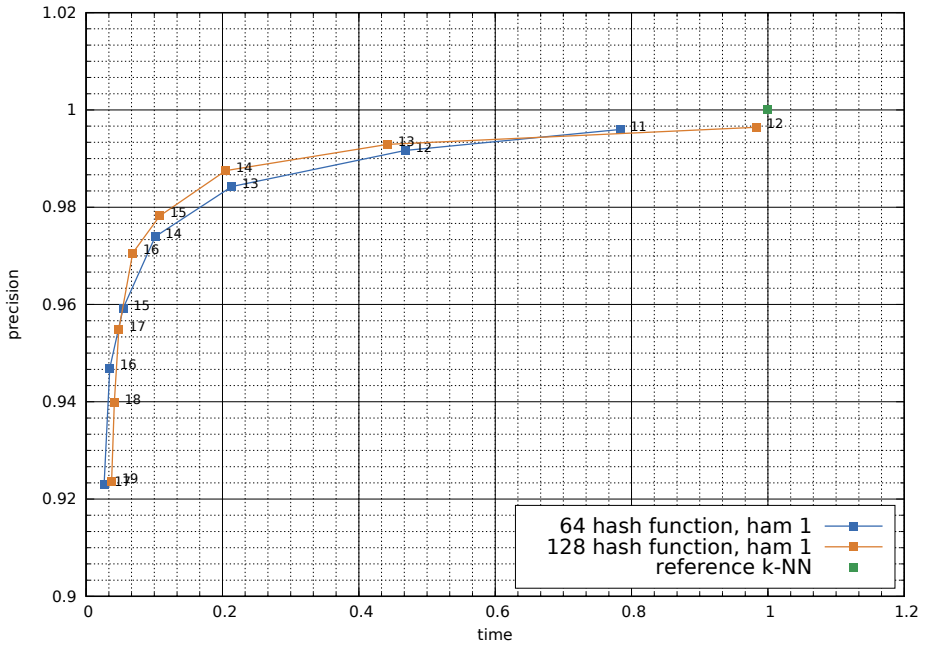
model s konkrétním počtem hashovacích funkcí je pak možné nastavit dimenzi tak, aby se model přiblížil požadovanému času a úspěšnosti, aby se zbytečně netestovaly již předem jasné velmi neefektivní parametrizace modelu.

Poslední možností, jak upravovat jednotlivé metody pro dosažení potřebného času a úspěšnosti, je přidávání a odebírání hashovacích funkcí, které lze pozorovat v tabulce 5.1 mezi modely s 64 a 128 funkcemi.

5.5.2 Testování databáze A

Otestovanou přesnost na 10 nejbližších sousedů pro optimální modely lze vidět na obrázku 5.9. Je zde vidět, že model s 64 hashovacími funkcemi dosahuje při nižších časech lepší úspěšnosti. Na druhou stranu model se 128 hashovacími funkcemi dosahuje lepších výsledku od 5 % času až po 50 %.

Oba modely jsou však velice podobné a na grafu je také patrné, že přidáváním nebo odebíráním hashovacích funkcí od 64 do 128 se lze dobře pohybovat po křivce mezi body modelu se stejnou hodnotou dimenze. To je výhodné zejména pro úpravu modelu v reálném čase, protože změna dimenze hashovacích funkcích znamená nutnost přehashování všech těchto funkcí. Po přidání



Obrázek 5.9: Na grafu jsou zobrazeny oba optimální modely otestované na databázi A

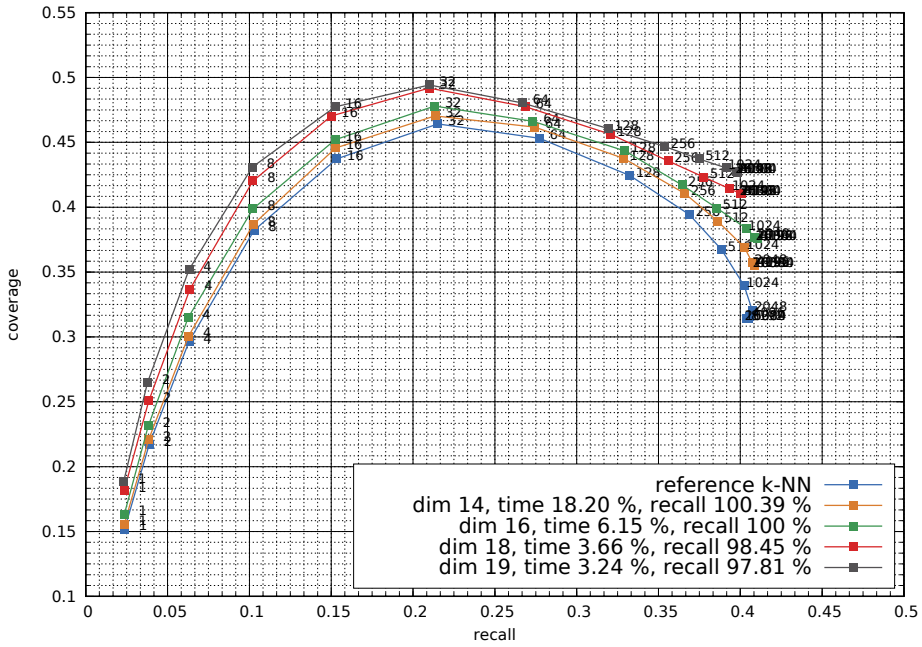
jedné hashovací funkce je potřeba zahashovat uživatele pouze pro jedinou vytvořenou funkci.

Na obrázku 5.10 je vidět úspěšnost doporučování na křivce závislosti recallu a catalog coverage. Je vidět, že lze pomocí LSH dosáhnout 100% recallu při vyšší catalog coverage. Model je tedy schopen doporučovat větší procento položek. Na obrázku 5.10 je zobrazena křivka pro dimenzi modelu 16, která má 100% recall za 6,1 % času. Podrobněji jsou vypsány hodnoty pro optimální k v tabulce 5.1.

Některé křivky databáze A s vyšším časem mají podle tabulky 5.1 vytvořené podle grafu 5.10 procentuální recall vůči referenčnímu modelu vyšší než 100 %, což by se u aproximačních modelů nemělo stát. Tento výsledek nastal pouze u databáze A a je způsoben strukturou dat a způsobem testování.

Důvodem je, že jsem při testování doporučování zvolil zvyšování k exponenciálně, které pro vykreslení vypadá ideálně, protože rozdíly mezi jednotlivými body jsou podobné. Při tomto zvyšování byl optimální recall referenčního modelu na hodnotě $k = 1024$. Při vyšším k klesá recall i catalog coverage. Tento trend se ale neděje tak výrazně u modelů LSH, protože model vrací omezenou množinu uživatelů, která může být dokonce menší než zvolené k.

Z tohoto důvodu model LSH může dosahovat optimálního recallu pro jinou hodnotu k a tento recall může být podobný i pro více vyšších hodnot k. Optimum LSH modelu pro databázi A bylo blíže testovaným hodnotám k než



Obrázek 5.10: Na grafu je vidět úspěšnost doporučování pro model se 128 funkcemi a procházení blízkých bucketů v okolí 1 na databázi A

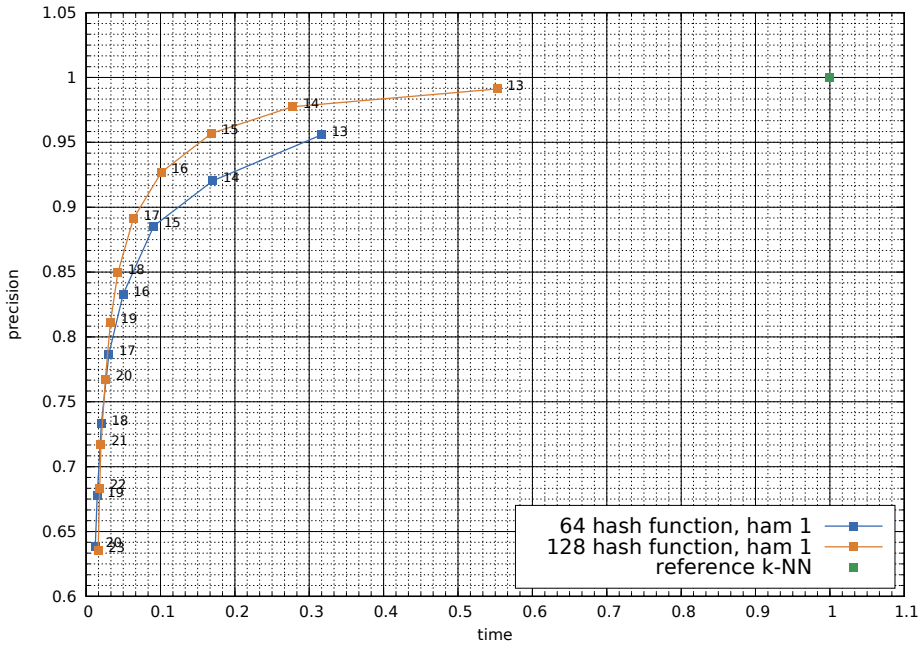
pro referenční model, který by při menším kroku mohl dosáhnout optima při jiném k. Jelikož se procentuální recall blížil ke 100 %, mohlo se stát, že jsem vlivem velkého exponenciálního kroku neporovnával nejoptimálnější hodnoty recallu, a tak procentuální výsledek LSH vyšel až o 0,5 % lépe vůči referenčnímu modelu.

5.5.3 Testování databáze B

Databáze B má odlišné parametry než databáze A. Je to především průměrný počet položek, se kterými uživatel průměrně interagoval, a také celkový počet uživatelů. Jak je patrné v tabulce 5.1, tak model LSH při testování úspěšnosti nedosahuje takových hodnot, jakých se podařilo dosáhnout na databázi A. Rozdíly na jednotlivých databázích jsou však jen několik procent.

Na grafu 5.11 je vidět přesnost při hledání 10 nejbližších sousedů, při kterém je rozdíl mezi testováním databáze A a B vidět nejvíce. I přesto se na databázi B podařilo dosáhnout velmi dobrých výsledků. Při 10% čase je přesnost vyšší jak 90 %.

Při testování úspěšnosti doporučování zobrazené na grafu 5.12 je výsledek výrazně lepší. Pomocí metod LSH se podařilo dosáhnout času, který se pohyboval mezi 2 až 3 % při recallu vyšším jak 96 % vůči referenčnímu řešení. Na grafu si lze všimnout, že pro databázi B je optimální recall při hodnotě



Obrázek 5.11: Na grafu jsou zobrazeny oba optimální modely otestované na databázi B

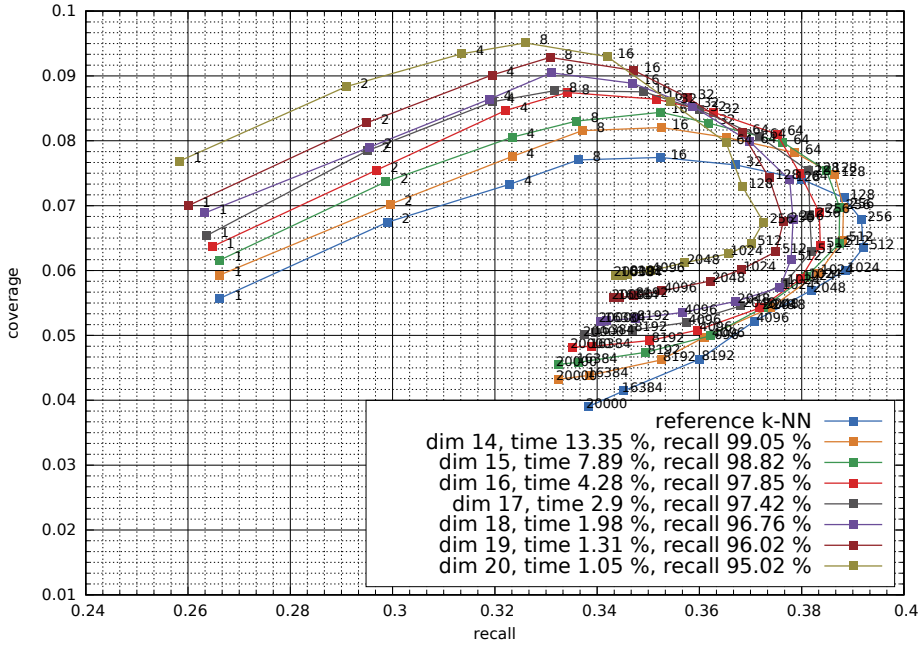
$k = 512$ a pro některé modely LSH i $k = 256$. Pro graf jsem zvolil model se 64 hashovacími funkcemi, protože při nižších časech dosáhl lepších výsledků než model se 128 funkcemi.

Modely na této databázi již nedosahují takového rozdílu v hodnotě catalog coverage, kterého se podařilo dosáhnout na databázi A. Toto může být způsobeno počtem uživatelů v testovací sadě, protože z důvodu velikosti databáze byla testovací sada procentuálně (vzhledem k velikosti databáze) mnohonásobně menší než v případě testování databáze A.

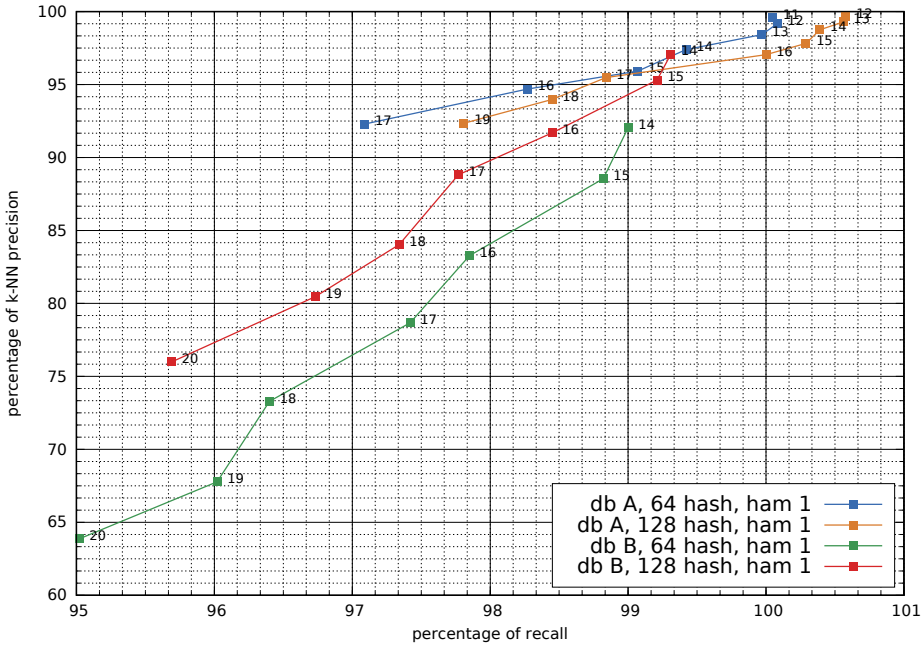
K dosažení výsledků na databázi B jsem použil stejné modely jako na databázi A pouze s tím rozdílem, že jsem podle průměrného počtu uživatelů v bucketu upravil dimenze hashovacích funkcí. Počet hashovacích funkcí mezi 64 až 128 se jeví jako ideální vzhledem k nutnosti provádět sjednocení výsledků z hashovacích funkcí.

5.5.4 Porovnání testování databází A a B

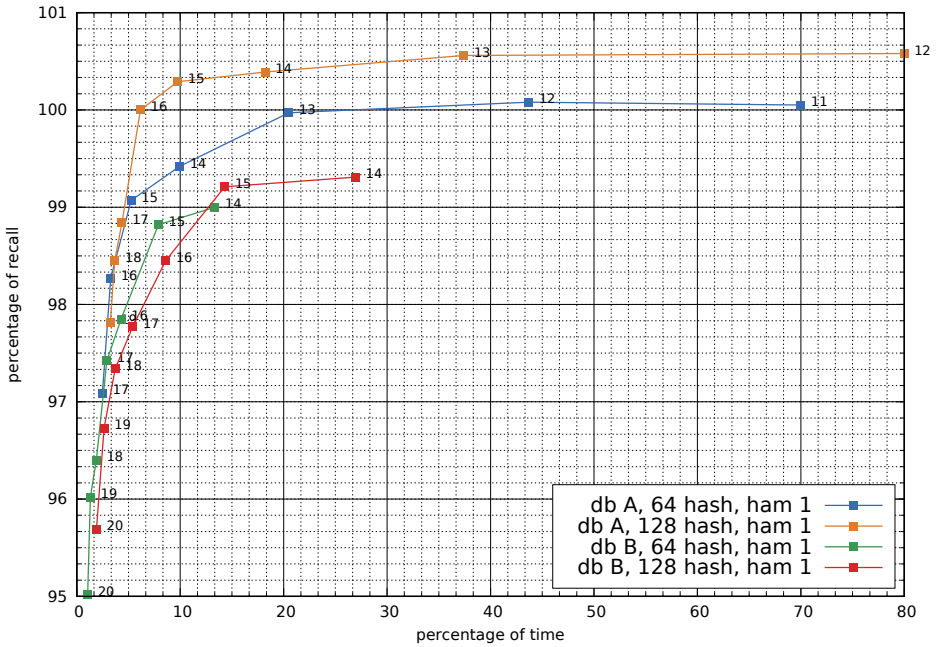
Ze zmíněných rozdílů v testování na databázi A a B vyplývá, že se také změní i závislost přesnosti 10-NN a hodnoty recall. Graf této závislosti je zobrazen na obrázku 5.13. Oproti databázi A na databázi B klesala přesnost 10-NN rychleji při snižujícím se recallu. To způsobuje odlišná struktura dat, jak byla popsána na začátku kapitoly 5.5.3.



Obrázek 5.12: Úspěšnost doporučování pro model s 64 hashovacími funkcemi otestovaný na databázi B



Obrázek 5.13: Závislost přesnosti k-NN na hodnotě recall



Obrázek 5.14: Závislost času vyhodnocování na hodnotě recall pro porovnání stejných modelů na databázi A a B

Na databázi B je tedy větší hodnota recall při stejné přesnosti na 10 nejbližších sousedů. Při testování je tedy nejprve nutné vykreslit křivku závislosti těchto hodnot na dané databázi. Podle této křivky lze testovat modely nejprve na přesnost k-NN pro hledání optimálních parametrizací a modelů, jak bylo zmíněno v kapitole 5.2.3.

Rozdíl mezi dvěma testovanými modely na databázi B je pravděpodobně způsoben volbou příliš malé a rozdílné testovací sady uživatelů, jejíž výběr mohl mít následně vliv na testování.

Na grafu 5.14 jsem vynesl závislost času a hodnoty recall. Je vidět, že mezi modely na databázi A je mnohem větší rozdíl než mezi stejnými modely na databázi B. Tedy čím více je v databázi uživatelů, tím se lze pomocí modelů LSH a přidáváním a odebíráním hashovacích funkcí pohybovat po velice podobné křivce aproximace algoritmu k-NN.

Na grafu 5.14 je vidět, že při dosahování nízkých časů velice prudce klesá recall. Naopak s narůstajícím časem již jen minimálně roste hodnota. Z toho vyplývá, že je optimální volit modely pro databáze s časem okolo 10 % vzhledem k referenčnímu řešení. Při tomto čase je patrný zlom křivky a modely LSH dosahují nejlepších výsledků při poměru času a úspěšnosti.

Kapitola 6

Diskuze

V této kapitole diskutuji některé závěry, které byly zjištěny při testování. Prodiskutuji rozdělování uživatelů do bucketů, které je stěžejní při aplikaci LSH na algoritmy k-NN. Dále prodiskutuji poměr zrychlení na úkor úspěšnosti, kterého se podařilo dosáhnout. Na závěr uvedu možnosti budoucího rozšíření této práce.

6.1 Rozdělení uživatelů do bucketů

V kapitole 5.4.1 je na obrázku 5.6 ukázáno rozdělení uživatelů do bucketů z konkrétního modelu. Zaměřím se hlavně na levý obrázek, kde model není ovlivněn popularitou položek. Na obrázku jde vidět, že v některých bucketech je uživatelů výrazně více než v jiných.

Nerovnoměrné rozdělení má vliv na rychlost vyhodnocování. Z objemných bucketů je vráceno více uživatelů a výpočet podobností je proto náročnější. Model s takovým rozdělením může vyhodnocovat dotazy nerovnoměrně rychle, i když průměrná hodnota dosahuje výborných výsledků, jak bylo prezentováno v kapitole 5.5.

Existuje možnost, jak tomuto problému předcházet. V systému by mohla být nastavena maximální velikost bucketu s ohledem na požadované zrychlení a velikost dat. Pokud by již byl bucket naplněn, uživatel by byl umístěn do jiného bucketu. K nalezení takového bucketu by mohla být využita právě metoda multi-probe. Pomocí podobnosti bucketů by byl uživatel umístěn do některého z podobných bucketů, který není zaplněn.

Touto metodou by mohlo být dosaženo rovnoměrnějšího výkonu s ohledem na čas vyhodnocování. Metoda by nemusela mít tak velký vliv na úspěšnost doporučovacího systému, protože uživatel by byl umístěn do bucketu, který by byl vzdálen od původního tak, aby byl procházen při vyhodnocování. Tedy maximálně do bucketů v okolí nastaveném při dotazování.

6.2 Poměr zhoršení přesnosti a zrychlení výpočtu

Zrychlení na úkor přesnosti je hlavní úkol aproximačních algoritmů. Při přesnosti na k-nejbližších sousedů na databázi A, která má řidší data, se podařilo dosáhnout 2 až 3 % času referenčního modelu a přesnost se pohybovala nad 90 %. Tedy bylo průměrně určeno více jak 9 z 10 nejbližších sousedů správně. Pomocí metod LSH se mi na této databázi A podařilo dosáhnout velice dobrého výsledku. Na databázi B se při stejné úspěšnosti dařilo dosahovat časů mezi 10 až 20 %.

Algoritmus k-NN s více jak 90% úspěšností dává dobré výsledky při generování doporučení. Procentuální hranici pro velmi dobrou úspěšnost doporučovacího systému považuji 99 %, protože takové zhoršení se dá považovat téměř za nulové, jelikož jednotlivé referenční modely se lišili o 1 až 2 % v závislosti na vybrané testovací sadě uživatelů. Lehce nad hranicí 99 % úspěšnosti modely dosahovaly časů, které se pohybují mezi 5 až 6 % času. Zrychlení vyhodnocování na databázi A je tedy velice výrazné.

Při úspěšnosti nad 99 % dosahovaly modely na databázi B časů mezi 10 až 25 %. Na druhou stranu pokud bych snížil tuto hranici, tak při čase mezi 2 až 5 % byla úspěšnost mezi 96 až 98 %, což je velice uspokojivé.

S dalším snižováním času na databázi A se podařilo dosáhnout času na úrovni 1,8 %, ale již s úspěšností 94,8 %. Na druhou stranu na databázi B byla při čase 1 % úspěšnost lehce přes 95 %, tedy modely byly úspěšnější při nižším čase v porovnání s databází A.

Lze pozorovat, že s dalším snižováním času úspěšnost klesá rychleji. Z grafů je tedy patrné, že modely LSH mají velkou variabilitu nastavení tak, aby mohlo být dosaženo požadovaného zrychlení nebo aby úspěšnost nebyla nižší než zvolené požadované procento.

6.3 Možnosti budoucí práce

V budoucnu by mohli být tyto metody implementované jako součást doporučovacího systému, kde by bylo nutné se zabývat jejich paralelizací. Metody LSH lze velice efektivně paralelizovat obdobně jako samotný algoritmus k-NN. V budoucí práci by bylo určitě nutné omezit maximální velikost bucketu, jak bylo zmíněno v části 6.1.

Při využití v online doporučovacím systému by bylo nezbytné se zabývat automatizací nastavení modelu, tedy zvoleným množstvím hashovacích funkcí, množstvím bucketů a velikostí okolí procházených bucketů. Zároveň na online datech není znám předem přesný objem dat ani počet uživatelů, proto by bylo zajímavé navrhnout způsob, kterým by bylo možné přizpůsobovat model při online běhu (např. přehashováním, přidáváním nebo odebíráním hashovacích funkcí).

Závěr

Cílem práce byl návrh a implementace knihovny pro testování a vyhodnocování úspěšnosti metod lokálně senzitivního hashování a jejich různých parametrizací.

Knihovna implementována v rámci práce umožňuje testovat různé parametrizace, které lze zadávat jako argumenty při spuštění testování. Data jsou načítána staticky z textových csv souborů. Z výsledných dat generovaných frameworkem ve formátu csv je možné pomocí nezávislého modulu generovat grafy pro lepší prezentaci výsledků.

Dosažené výsledky při testování různých parametrizací jsou velice uspokojivé. Podařilo se otestovat některé metody LSH s velkým množstvím parametrizací. Některé metody mají velký vliv na úspěšnost a jsou téměř nezbytné. Na druhou stranu existují přístupy, které například zlepšují modely jen v některých ohledech.

Zajímavým poznatkem je zjištěná závislost mezi přesností na k-nejbližších sousedů a úspěšnosti doporučování pro modely LSH. Díky tomuto zjištění bylo možné otestovat velké množství různých parametrizací, protože testování přesnosti na k-nejbližších sousedů je efektivnější. Testováním byly zjištěny některé hranice jednotlivých modelů a také nastavení, pomocí kterého lze dosáhnout optimálních výsledků.

V neposlední řadě je zajímavý poznatek nastavení parametrů testu pro testování metod LSH pomocí frameworku. Pro každou databázi je nutné zvolit požadované parametry při testování s ohledem na velikost databáze. Tyto parametry lze podle počtu uživatelů lehce dopočítávat a tím přizpůsobit testování pro konkrétní databázi.

Literatura

[1] RICCI, F.; ROKACH, L.; SHAPIRA, B.; aj.: *Recommender systems handbook*. Springer New York Dordrecht Heidelberg London, 2011, ISBN 978-0-387-85819-7.

[2] JANNACH, D.; ZANKER, M.; FELFERNIG, A.; aj.: *Recommender systems: an introduction*. 32 Avenue of the Americas, New York, NY 10012-2473, USA: Cambridge University Press, 2010, ISBN 978-0-521-49336-9.

[3] BALABANOVIĆ, M.; SHOHAM, Y.: Fab: content-based, collaborative recommendation. *Communications of the ACM* [online], ročník 40, č. 3, March 1997: s. 66–72, [cit. 2018-03-07]. Dostupné z: https://dl.acm.org/citation.cfm?id=245124

[4] XUE, G.-R.; LIN, C.; YANG, Q.; aj.: Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval* [online], ACM, August 15 - 19, 2005, s. 114–121, [cit. 2018-03-08]. Dostupné z: https://dl.acm.org/citation.cfm?id=1076056

[5] SAVHAN, A.; RICHARIYA, V.: A survey on recommender systems based on collaborative filtering technique. *International journal of Innovations in Engineering and technology (IJIET)* [online], 2013: s. 8–14, [cit. 2018-03-08]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.360.57&rep=rep1&type=pdf

[6] ZHANG, R.; LIU, Q.-d.; GUI, C.; aj.: Collaborative filtering for recommender systems. In *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*, IEEE, 2014, ISBN 978-1-4799-8085-7, s. 301–308.

[7] BELL, R. M.; KOREN, Y.: Improved neighborhood-based collaborative filtering. In *KDD cup and workshop at the 13th ACM SIGKDD international conference on knowledge discovery and data mining* [online], Citeseer, 2007, s. 7–14, [cit. 2018-03-10]. Dostupné z: https://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/Neighbor-Koren.pdf

[8] MIJA, M.; LOWE, D. G.: Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* [online], ročník 36, č. 11, 01 May 2014: s. 2227–2240, ISSN 0162-8828, [cit. 2018-03-10]. Dostupné z: doi:10.1109/TPAMI.2014.2321376

[9] MELVILLE, P.; SINDHWANI, V.: Recommender Systems. In *Encyclopedia of Machine Learning* [online], Boston, MA: Springer US, 2010, ISBN 978-0-387-30164-8, s. 829–838, [cit. 2018-03-10]. Dostupné z: doi:10.1007/978-0-387-30164-8_705

[10] GUNAWARDANA, A.; SHANI, G.: A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research* [online], ročník 10, č. Dec, 2009: s. 2935–2962, [cit. 2018-03-11]. Dostupné z: http://www.jmlr.org/papers/v10/gunawardana09a.html

[11] HERLOCKER, J. L.; KONSTAN, J. A.; TERVEEN, L. G.; aj.: Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Inf. Syst.* [online], ročník 22, č. 1, 1, January 2004: s. 5–53, ISSN 1046-8188, [cit. 2018-03-11]. Dostupné z: doi:10.1145/963770.963772

[12] GE, M.; DELGADO-BATTENFELD, C.; JANNACH, D.: Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems* [online], RecSys '10, New York, NY, USA: ACM, September 26 - 30, 2010, ISBN 978-1-60558-906-0, s. 257–260, [cit. 2018-03-11]. Dostupné z: doi:10.1145/1864708.1864761

[13] SHAKHNAROVICH, G.; DARRELL, T.; INDYK, P.: Introduction. In *Nearest-neighbor methods in learning and vision: theory and practice*, Cambridge, Messachusetts, London, England: The MIT press, 2006, ISBN 0-262-19547-X, s. 1–12.

[14] WANG, J.; SHEN, H. T.; SONG, J.; aj.: Hashing for Similarity Search: A Survey. *CoRR* [online], ročník abs/1408.2927, Wed, 13 Aug 2014, [cit. 2018-03-11], 1408.2927. Dostupné z: http://arxiv.org/abs/1408.2927

[15] ZHANG, Y.-M.; HUANG, K.; GENG, G.; aj.: Fast kNN graph construction with locality sensitive hashing. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ISBN 978-3-642-40991-2, s. 660–674.

[16] GIONIS, A.; INDYK, P.; MOTWANI, R.; aj.: Similarity search in high dimensions via hashing. In *Vldb*, ročník 99 [online], Stanford University, September 07 - 10, 1999, ISBN 1-55860-615-7, s. 518–529, [cit. 2018-03-17]. Dostupné z: http://www.vldb.org/conf/1999/P49.pdf

[17] ARGERICH, L.; GOLMAR, N.: Generic LSH Families for the Angular Distance Based on Johnson-Lindenstrauss Projections and Feature Hashing LSH. *CoRR* [online], ročník abs/1704.04684, Sat, 15 Apr 2017, [cit. 2018-03-17], 1704.04684. Dostupné z: http://arxiv.org/abs/1704.04684

[18] INDYK, P.; MOTWANI, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* [online], STOC '98, New York, NY, USA: ACM, May 24 - 26, 1998, ISBN 0-89791-962-9, s. 604–613, [cit. 2018-03-17]. Dostupné z: doi:10.1145/276698.276876

[19] LV, Q.; JOSEPHSON, W.; WANG, Z.; aj.: Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases* [online], VLDB '07, VLDB Endowment, September 23 - 27, 2007, ISBN 978-1-59593-649-3, s. 950–961, [cit. 2018-03-17]. Dostupné z: http://dl.acm.org/citation.cfm?id=1325851.1325958

[20] CORMODE, G.; DASGUPTA, A.; GOYAL, A.; aj.: An evaluation of multi-probe locality sensitive hashing for computing similarities over web-scale query logs. *PLOS ONE* [online], ročník 13, č. 1, January 18, 2018: s. 1–14, [cit. 2018-03-17]. Dostupné z: doi:10.1371/journal.pone.0191175

[21] STECK, H.: Item Popularity and Recommendation Accuracy. In *Proceedings of the Fifth ACM Conference on Recommender Systems* [online], RecSys '11, New York, NY, USA: ACM, October 23 - 27, 2011, ISBN 978-1-4503-0683-6, s. 125–132, [cit. 2018-03-19]. Dostupné z: doi:10.1145/2043932.2043957

Příloha **A**

Seznam použitých zkratek

LSH	lokálně senzitivní hashování (angl. Locality-sensitive hashing)
k-NN	algoritmus hledání k-nejbližších sousedů (angl. k-nearest neighbors)
TP	správně pozitivní (angl. true positives)
FP	chybně pozitivní (angl. false positives)
FN	chybně negativní (angl. false negatives)
TN	správně negativní (angl. true negatives)
csv	souborový formát určený pro zápis tabulkových dat oddělených čárkami (angl. Comma-separated values)
2D	dvoudimenzionální nebo dvourozměrný prostor
db	značí databázi v legendách u grafů
ham	značí maximální velikost okolí procházených bucketů v legendách grafů (maximální hammingova vzdálenost)
dim	značí dimenzi hashovacích funkcí v legendách u grafů

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `impl` ..... zdrojové kódy implementace
    - `LSH_library` ..... zdrojové kódy testovacího frameworku
    - `GNU_plot` ..... zdrojové kódy modulu pro tvorbu grafů
  - `PlotData.sh` ..... ukázkový script pro volání modulu pro tvorbu grafů
  - `RunTest.sh` ..... ukázkový script pro spuštění testování
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF