VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE DOPORUČOVACÍHO SYSTÉMU

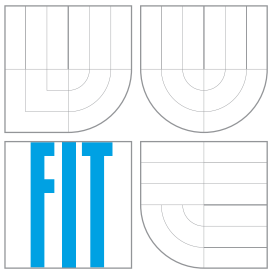
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IGOR KONÍČEK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE DOPORUČOVACÍHO SYSTÉMU

WEB APPLICATION OF RECOMMENDER SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IGOR KONÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. JAROSLAV ZENDULKA, CSc.

BRNO 2015

Abstrakt

Tato práce řeší tvorbu doporučovacího systému, který je využit v reálné aplikaci serveru cbdb.cz. S využitím přístupů kolaborativního filtrování a filtrování založeného na obsahu se podařilo vyvinout funkční doporučovací systém. Díky zpětné vazbě uživatelů bylo zjištěno, že většina doporučených knih je pro ně relevantní. Hlavním přínosem této práce je rozšíření stávající funkčnosti serveru cbdb.cz o doporučovacím systémem, který využívá jeho rozsáhlé databáze hodnocení, uživatelů a knih.

Abstract

This master's thesis describes creation of recommender system that is used in real server cbdb.cz. A fully operational recommender system was developed using collaborative and content-based filtering techniques. Thanks to many user feedback, we were able to evaluate their opinion. Many recommended books were tagged as desirable. This thesis is extending current functionality of cbdb.cz with recommender system. This system uses its extensive database of ratings, users and books.

Klíčová slova

Doporučovací systémy, dolování z dat, kolaborativní filtrování, filtrování založené na obsahu, doporučování knih, kosinová podobnost

Keywords

Recommender systems, data mining, collaborative filtering, content-based filtering, books recommendation, cosine similarity

Citace

Igor Koníček: Webová aplikace doporučovacího systému, diplomová práce, Brno, FIT VUT v Brně, 2015

Webová aplikace doporučovacího systému

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Doc. Ing. Jaroslava Zendulky, CSc. Další informace mi poskytl Ing. Martin Hlosta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Igor Koníček
21. května 2015

Poděkování

Rád bych poděkoval mému vedoucímu práce Doc. Ing. Jaroslavu Zenulkovi, CSc za výborné myšlenky na vylepšení algoritmů. Dále děkuji Ing. Martinu Hlostovi za odborné konzultace. A také děkuji Ing. Jiřímu Jandovi ze serveru cbdb.cz za vynikající spolupráci.

Obsah

Kapitola 1

Úvod

21. století je bez pochyby dobou internetu. S klesající cenou paměťových zařízení rapidně stoupá velikost databázových serverů, kde jsou uložené různorodé informace. Data každý den přibývají enormní rychlostí a už se stává nemožné, aby se v nich člověk vyznal bez patřičné interpretace. Proto do popředí vstupují moderní techniky, jako například dolování z dat (Data Mining). Tyto metody nám umožňují hledat vzory v rozsáhlých datových strukturách a napomáhají nám k jejich analýze a vyhodnocování významu těchto dat.

V dnešní době vzniká spousta nových internetových portálů, jejichž cílem je nabízet návštěvníkovi nějaké produkty, položky, či služby. Provozovatel je často tlačen k tomu, aby jeho sortiment byl co nejširší a pokryl tak vetší škálu návštěvníků. Tento fakt ovšem může hrát negativní roli, protože návštěvník samotný může snadno ztratit přehled. Tento problém můžeme vyřešit použitím doporučovacího systému.

Cílem doporučovacích systémů je poznat své uživatele, naučit se jejich preference a chování. Na základě takto zjištěných informací jim pak nabízí nové, dosud neviděné položky, které by mohly mít pro návštěvníka pontencionální přínos.

Čtenář je v kapitole 2 seznámen s doporučovacímy systémy. Nalezne zde obecné informace o těchto systémech, dále pak jejich přínos a význam. Následně je obeznámen s různými druhy dat, které doporučovací systémy využívají. Poté čtenář nalezne informace o jednotlivých doporučovacích technikách, které se v dnešní době používají a nakonec je seznámen s problémy, se kterými se tyto systémy potýkají.

Ve 3. kapitole jsou popsány základní informace o dolování z dat (Data mining). Čtenář zde nalezne informace ohledně předzpracování dat pro následné doporučování. Poté je seznámen s pojmem klasifikace.

Kapitola 4 seznamuje čtenáře se serverem cbdb.cz, je zde popsána datová sada, která je použita pro doporučení. Následně jsou zde sepsány služby, které tento knížní server poskytuje svým uživatelům. Dále je zde popsána stěžejní část této práce, a to rozdělení vývoje do čtyř iterací. Pro každou iteraci je zde popsáno, co se v ní událo, jaké kroky byly uskutečněny. Životní cyklus programu je popsán od samotného soupisu požadavků, až po závěřečnou iteraci zobrazující kompletní doporučovací systém.

V 5. kapitole popisuji dosažené výsledky s doporučovacím systémem. Doporučovací systém byl implementován do reálné aplikace s tisíci uživateli, proto byla zvolena metoda zpětné vazby. Je zde popsáno, jaká data uživatelé poskytli a jak by se dala tato data následně využít.

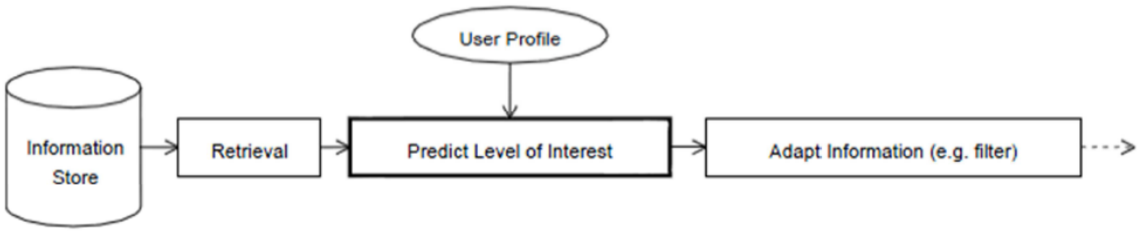
V závěrečné kapitole jsou shrnuty výsledky celé diplomové práce. Jsou zde také uvedeny další myšlenky a nápady, které by mohly systém obohatit.

Kapitola 2

Doporučovací systémy

Doporučovací systémy jsou softwarové aplikace určené k doporučování a navrhování položek, či uživatelů. Jako položku si můžeme představit například zboží v internetovém obchodě, písničku v přehrávači, film v půjčovně atd. Doporučováním uživatelů se může rozumět například návrhy přátel na sociálních sítí.

Doporučovací systémy jsou cíleny na konkrétní jedince, kteří nemají dostatek zkušeností s výběrem konkrétních položek, či produktů, a nebo nejsou schopni se rozhodnout mezi desítkami různých variant, které jim systém bez doporučení nabízí. Například webová stránka Amazon.com má online doporučovací systém, který navrhuje produkty návštěvníkům. Díky tomu, že doporučovací systémy berou v potaz preference každého zákazníka zvlášť, tak může každému zákazníkovi doporučit produkt, který by mohl co nejvíce vyhovovat jeho preferencím. V kontrastu s takovými systémy jsou nepersonalizované systémy. Takovéto programy jsou mnohem jednodušší a běžně jsou k vidění na novinových serverech - například novému návštěvníkovi zobrazí 10 nejčtenějších článků posledního týdne. Zdrojem informací této kapitoly je [16].



Obrázek 2.1: Schéma komponent v doporučovacím systému. Zdroj [13]

Stěžejním úkolem doporučovacích systémů je předpověd pro konkrétního uživatele, jaký produkt by mohl vyhovovat jeho zájmům, či preferencím. Aby systém mohl takto předpovídat vhodné položky, musí od uživatelů získávat jejich hodnocení. Hodnocení můžeme rozdělit na explicitní - uživatel ručně zadá hodnocení produktu například ve formě hvězdiček; a implicitní - pouhá interakce uživatele s produktem se bere jako pozitivní hodnoceni, například zobrazení produktu nebo komentování.

Obrázek 2.1 zachycuje základní stavební prvky doporučovacího systému. Na nejnižší úrovni je zdroj informací (information store), z něj se extrahují data (retrieval). Následně jsou na tato data použity algoritmy počítající výši přinosu (predict level of interest) pro uživatele (user profile). Nakonec proběhne filtrování takovýchto dat (adapt information).

Doporučovací systémy mají své kořeny v polovině 90. let [16]. Tedy ve srovnání s ostatními disciplínami (databázové systémy, vyhledávací systémy,..) je to nové odvětví. V posledních letech jsou tyto systémy na vzestupu - velké internetové servery implementovaly doporučovací systémy do svých služeb, například Amazon, Netflix, GoodReads, Yahoo, Last.fm.

2.1 Význam doporučovacích systémů

V úvodu této kapitoly bylo řečeno o použití doporučovacích systémů. Nyní se seznámíme s výhodami a důvody proč se zajímat o doporučovací systémy:

- **Navýšení prodeje produktů**. Toto je jedna ze stěžejních a nejdůležitějších vlastností doporučovacích systémů. Příkladem může být prodej souvisejících produktů k právě kupovanému produktu, či prodej podobných produktů.
- **Prodej více různorodých produktů**. Některé produkty mohou být ve velké nabídce těžko k nalezení, pokud na ně neupozorní doporučovací systém.
- **Zákazníkova spokojenost**. Dobře navržený doporučovací systém zvyšuje návštěvníkův prožitek. S dobrým grafickým designem, s vhodným přístupem počitač-člověk může být návštěva hodnocena jako velmi zajímavá a přínosná. S těmito aspekty také stoupá konverze na zákazníky.
- **Lepší porozumění uživatelům**. Doporučovací systém má schopnost se učit zvyky uživatele, a tím zpřesňovat a vylepšovat své předpovědi. S těmito znalostmi se poté dá dále pracovat. Například v cestovním průmyslu může systém novému zákazníkovi doporučit destinaci v závislosti na jeho poloze. Nebo můžeme hledat vztahy mezi jednotlivými produkty a nabízet je pospolu.

Robin Burke ve své práci [6] definoval jedenáct úkolů, kterým může doporučovací systém vypomoct. Zde uvádím některé z nich:

- **Nalezení některých doporučení**. Tento úkol řeší mnoho doporučovacích systémů. Jádrem je zobrazení uživateli pouze několika doporučení (běžně 10 nejlepších). Takovéto systémy často nezobrazují bodové ohodnocení jednotlivých předpovědi.
- **Nalezení všech doporučení**. Doporučovací systémy mohou také zobrazovat všechna relativní doporučení. Tyto systémy nacházejí uplatnění v situacích, kde je malý počet produktů na výběr, či je vhodné uživateli zobrazit všechny možné návrhy (například v oboru finančnictví či zdravotnictví).
- **Doporučení posloupnosti**. Systém nezobrazí pouze jednotlivé produkty, ale i produkty návazné. Například doporučí knihy z trilogie, či televizní seriály nebo hudební nahrávky.
- **Obyčejné prohlížení**. V tomto úkolu si zákazník pouze prohlíží katalog produktů bez úmyslu zakoupení. Cílem systému je na základě prohlédnutých položek nabídnout zákazníkovi nové položky, o které by mohl jevit zájem.
- **Nalezení věrohodného návštěvníka**. Někteří uživatelé mohou zkoušet ovlivňování doporučovacího systému v negativním směru a sledovat, jak se systém zachová. Proto systém může obsahovat ochranné funkce, kterými prověří zda je návštěvníkovo chování akceptovatelné.

- **Vyjádření názoru**. Některým uživatelům nejde o samotné doporučování tak jako o vyjádření vlastního názoru a tím přispět komunitě.

2.2 Data a zdroje znalostí

Pro správně fungující doporučování musí systémy shromažďovat data. Ovšem zdroje dat a znalostí mohou být velmi rozličné. Můžeme mít doporučovací techniky, které mají malou bázi znalostí - například používají velmi jednoduchá data, jako jsou uživatelská hodnocení. Jiné techniky jsou provázané se znalostní bází - například doporučování v závislosti na sociálních interakcích jednotlivých uživatelů. Data používaná v doporučovacích systémech můžeme rozdělit na položky, uživatele a transakce (např. vztah mezi uživatelem a položkou).

Položky

Položky jsou samotným předmětem doporučování. Ohodnocení položky pro konkrétního uživatele může být pozitivní, pokud pro něj má kladný přínos, nebo negativní, pokud je položka irelevantní pro uživatele. V závislosti na použité technologii doporučování, může systém využívat rozličné atributy popisující jednotlivé položky. Například v doporučování filmů může systém využívat žánr (komedie, horor, ...), ale i také režisér a herci mohou být použiti pro detailnější popis struktury jednotlivých filmů a systém se může také naučit sílu vlivu jednotlivých atributů na výsledné doporučování. Položky mohou být reprezentovány použitím různorodých přístupů: od minimalistické reprezentace použitím jediného identifikačního kódu až po složitou strukturu atributů.

Uživatelé

Uživatelé doporučovacích systémů mají rozdílné preference a unikátní charakteristiku popisující, co by se danému uživateli mohlo líbit. Aby se doporučovací systém mohl přizpůsobit preferencím uživatele, musí znát jeho profil - být schopný extrahovat různorodé atributy popisující konkrétního uživatele a tím nalézt vzorec pro doporučení konkrétních položek systému. V závislosti na použité technologii doporučování se extrahují rozdílné informace. Například v kolaborativním filtrování (collaborative filtering) jsou uživatelé modelování jako seznam, kde každý uživatel je spárován s hodnocením jednotlivých produktů, které sám uživatel ohodnotil. V demografických doporučovacích systémech je uživatel svázán s informacemi o věku, pohlaví, povolání, vzdělání, atd.

Tato uživatelská data tvoří uživatelský model - ten zakódovává uživatelské preference a potřeby. Různorodé modely byly studovány a využívány v průběhu vývoje doporučovacích systémů [2, 3]. Jelikož žádné doporučování nelze zprovoznit bez vhodného uživatelského modelu, uživatelé budou vždy hrát centrální roli v systému. Výjimku mohou tvořit systémy, které takovýto model nevyužívají, například systémy typu '10 nejkupovanějších produktů'. Uživatelé mohou být také popsání vzorcem chování. Například jakými cestami procházejí konkrétní webové stránky [22], či jaké destinace vyhledávají pro dovolené. Uživatelé také mohou mít různé stupně věrohodnosti pro doporučovací systém a tím ovlivnit sílu vlivu na ostatní uživatele.

Transakce

Obecně pojmem transakce je myšlena interakce mezi uživatelem a daným doporučovacím systémem, která se uloží do interní databáze systému. Informace v transakcích si lze před-

stavit jako zaznamenávání uživatelského chování během používání doporučovacího systému. Tyto transakce poté napomáhají k vylepšení doporučovacích algoritmů, které daný systém využívá. Například transakce může uchovávat informace o zvolené položce uživatelem a jakým způsobem se k položce dopátral (vyhledávání, přímý odkaz,...). Transakce také může uchovávat uživatelskou zpětnou vazbu (textový komentář, či hodnocení položky,...)

Nejběžnějším použitím transakcí, které doporučovací systémy globálně využívají, je uživatelské hodnocení produktu. Toto hodnocení může být získáno explicitně či implicitně. Explicitním hodnocením může být uživatelské hodnocení, či jiná zpětná vazba, kterou uživatel musí vyplnit. Dle [19] hodnocení mohou mít různé podoby:

1. Nejčastějším hodnocením, se kterým je možné se setkat je 1-5 hvězdičkové ohodnocení, kde větší počet znamená lepší ohodnocení (například hodnocení knih na Amazon.com), či varianta 1-10 stupnice na IMDb.com. Také se můžeme setkat s inverzní variantou, která je podobná školnímu hodnocení.

2. Ordinální hodnocení. Uživatel je vyzván o vybrání varianty slovního vyjádření, která nejblíže vyjadřuje jeho názor: „Silný souhlas, souhlas, neutrální, nesouhlas, silný nesouhlas“.

3. Binární hodnocení. K dispozici jsou pouze 2 hodnoty, kdy uživatel volí pouze mezi hodnotami „dobrý“ a „špatný“. Například server YouTube.com umožňuje hodnocení „Palec nahoru“ a „Palec dolů“.

4. Unárním hodnocení může vyjadřovat, zda uživatel navštívil danou položku zobrazil, či ji vložil například do košíku. V takovém případě je položka ohodnocena kladně. Pokud takovéto hodnocení chybí, znamená to, že o položku není zájem.

Doporučovací systémy mohou z transakcí odvozovat uživatelovy preference v závislosti na jeho akcích. Například pokud uživatel vyhledává klíčové slovo „zdraví“ na serveru Amazon.com, dostane na výběr širokou škálu položek. Uživatel poté rozklikne detail konkrétní položky. V tomto okamžiku systém odvozuje doporučení následných položek v závislosti jakou položku si uživatel zobrazil [16].

2.3 Doporučovací techniky

Jádrem doporučovacího systému je algoritmus, který identifikuje užitečné položky vhodné pro konkrétního uživatele. Algoritmus musí předpovědět, která položka je vhodná pro dané doporučení. Aby toto systém zvládl, musí znát nebo být schopen předpovědět užitečnost jednotlivých položek a poté z této znalosti musí být schopen extrahovat informaci o míře vhodnosti doporučení pro konkrétního uživatele. Algoritmus předpovídání nemusí být přímo implementován uvnitř algoritmu doporučování, ale může být jako samostatný modul.

Nejjednodušší doporučovací technikou je obyčejné, nepersonalizované doporučování. Mějme například systém na doporučování nejoblíbenějších cestovatelských destinací. Důvodem použití takovéhoto systému může být nedostatek uživatelských preferencí a populární destinace je taková položka, která má vysoké hodnocení, či popularitu velkým množstvím uživatelů. Pro nového uživatele je tedy vyšší pravděpodobnost, že si takovouto destinaci také oblíbí, než jinou náhodně zvolenou destinaci [13].

Podle [6] můžeme rozlišovat následujících šest rozdílných tříd doporučovacích systémů:

1. Filtrování založené na obsahu (Content-based filtering)

2. Kolaborativní filtrování (Collaborative filtering)
3. Filtrování založené na znalostech (Knowledge-based filtering)
4. Komunitní filtrování (Comunity-based filtering)
5. Filtrování založené na demografických informacích (Demographic filtering)
6. Hybridní doporučovací systémy (Hybrid recommender systems)

2.3.1 Filtrování založené na obsahu

Systém založený na této technice doporučuje položky, které jsou podobné těm, které daný uživatel v minulosti ohodnotil. Pokud uživatel například v minulosti ohodnotil knihu, která patří do žánru sci-fi, tak se systém tuto skutečnost naučí a v budoucnu bude doporučovat knihy z tohoto žánru. V tomto přístupu je vybudována struktura klíčových slov popisujících jednotlivé položky a každý uživatel je definován svým profilem preferencí (sestáván také z jednotlivých klíčových slov). Klíčová slova, též nazývána termy, mohou být přiřazeny automaticky či manuálně. Jednotlivé položky jsou poté reprezentovány vektory a uživatelské profily váhovanými vektory položek, kde jednotlivé hodnoty vah vyjadřují sílu významu jednotlivých popisujících vlastností.

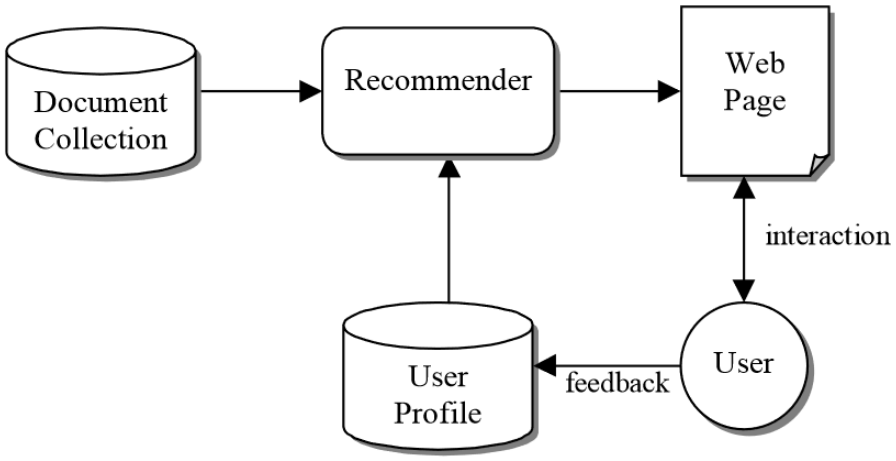
Pro výpočet hodnot vah lze užít různě složité a výpočetně náročné algoritmy. Jednoduchými principy výpočtu může být například obyčejné průměrování hodnot. Použitím složitějších a pokročilejších technik lze dosáhnout vhodnějšího doporučení a tím zvýšit pravděpodobnost, že doporučená položka bude více vyhovovat uživatelovým preferencím. K tomu lze využít metody strojového učení - shluková analýza, rozhodovací stromy, neuronové sítě, apod.

Při výběru metod hraje hlavní roli jejich účinnost a v druhé řadě také jejich prostorová náročnost, ta roste s počtem uživatelů systému a velikostí jejich profilu. Genetické algoritmy a neuronové sítě jsou zpravidla účinnější, ale pomalejší v porovnání s ostatními metodami, jelikož potřebují více iterací k zjištění relevantnosti položky k danému uživateli. Rychlost jednodušších algoritmů se snižuje s rostoucím počtem nových položek, protože každá musí být porovnána se všemi ostatními položkami nacházející se v databázi.

Obrázek 2.2 zachycuje pět základních elementů, které se vyskytují v těchto systémech. V systému máme nějakou databázi informaci (document collection), která vstupuje do doporučování (recommend). V tomto momentu nastává v systému smyčka, jak se jednotlivé komponenty navzájem ovlivňují. Samotné doporučení je zobrazené na výstup, např. webová stránka (web page). Z tohoto výstupu může uživatel (user) pomocí zpětné vazby (feedback) změnit svůj uživatelský profil (user profile), který je brán v potaz při doporučování pro tohoto uživatele.

Doporučování probíhá v následujících třech krocích:

1. **Analýza obsahu** - hlavním cílem tohoto kroku je analyzovat obsah jednotlivých položek v systému. Probíhá zde extrakce příznaků (nejčastější položky bývají textové soubory a extrahované příznaky může být analýza textu, jeho význam, apod.)
2. **Učení uživatelských preferencí** - v tomto kroku algoritmus sbírá a vyhodnocuje data o uživateli a tím získává jeho preference a na jejich základě vytvoří jeho profil, který využije k doporučování



Obrázek 2.2: Content-based filtering. Zdroj [12]

3. **Filtrování položek** - systém páruje položky a uživatele a vyhodnocuje míru vhodnosti tohoto párování (na základě uživatelského profilu a vektoru příznaků položky). Následně systém uživateli nabídne výsledné, nejvhodnější položky.

Nespornou výhodou filtrování založeného na obsahu je nezávislost na jiných uživatelích systému, aby bylo nalezeno doporučení pro konkrétního uživatele. Další výhodou je transparentnost této metody - jsme schopni určit na základě jakých vlastností (či příznaků) položky jsme provedli doporučení danému uživateli. A také systémy založené na této metodě nemají problém s doporučením nových položek, které ještě nebyly viděny / hodnoceny.

Hlavní nevýhodou tohoto přístupu je fakt, že vždy bude doporučena pouze položka, která má podobné rysy odpovídající uživatelskému profilu. Další nevýhodou je omezená analýza příznaků položek - pokud se pracuje s nízkým vektorem příznaků, systém nemůže nikdy kvalitně popsat položky a tím bude docházet k nepřesnému doporučování. Doporučování novým uživatelům je také problematické, jelikož nemají vybudovaný dostatečně kvalitní profil - musí tedy při vstupu do systému vyplnit určitý počet hodnocení, nebo případně dotazník a teprve následně může být spuštěno doporučování.

2.3.2 Kolaborativní filtrování

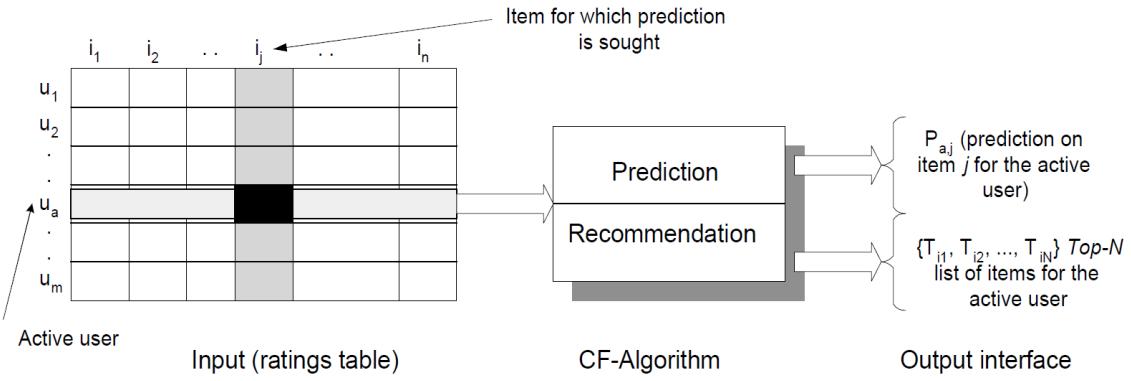
Kolaborativní filtrování je oblíbená technika doporučovacích systémů, která zakládá své předpovědi a doporučování na hodnocení položek, či chování uživatelů. Základním principem je myšlenka, že uživatelské hodnocení položky může být agregováno a využito k doporučení pro konkrétního uživatele. Pokud například dva uživatelé mají podobný vkus (hodnotí stejné položky) a někdy v budoucnu jeden uživatel ohodnotí novou položku, je tato doporučena druhému uživateli. Tento přístup je vhodný k použití v doménách s velkým počtem položek a uživatelů (například databáze filmů, knih, písniček, novinových článků)

Kolaborativní filtrování můžeme rozdělit na dvě kategorie [11] - **filtrování založené na podobnosti uživatelů** (user-based filtering) a **filtrování založené na podobnosti položek** (item-based filtering). První zmíněná kategorie je založena na podobnosti hodnocení jednotlivých uživatelů (výsledkem může být například seznam Top-10 položek, které by se konkrétnímu uživateli mohly líbit). Druhá kategorie hledá doporučení založené na podobnosti hodnocení položek v systému (například při zobrazení konkrétní položky, může systém vypsat Top-10 souvisejících položek s danou zobrazenou položkou).

Výše zmíněné kategorie můžeme také rozdělit na **paměťové** (memory-based) a **modelové** (model-based). Paměťový přístup vypočítá z uživatelského hodnocení podobnost mezi jednotlivými uživateli / položkami a poté provede doporučení. Tento přístup je jednoduchý na implementaci. Tento přístup podává přesnější výsledky než modelově založený přístup, protože využívá všechny položky dostupné v systému k vytvoření výsledného doporučení.

Nevýhodou tohoto přístupu je složitá práce s velkými databázemi, které mohou mít velmi nízké procento zaplnění. V druhém přístupu je vytvořen model, za využití technik data miningu a algoritmů strojového učení, které jsou aplikovány na trénovací sadu. Výsledný model je poté aplikován v doporučovacím algoritmu. Tento přístup zvládá problém s velkou databází a škálovatelností lépe, než předešlý přístup. Nevýhodou je složité budování modelů a také problém vhodného zvolení testovací sady.

Obrázek 2.3 zobrazuje princip kolaborativního filtrování. Na vstupu je matice hodnocení. V řádcích je veškeré hodnocení uživatele u_a a ve sloupcích je veškeré hodnocení položek i_b. Tedy na pozici u_x, i_y se nachází konkrétní hodnota hodnocení uživatele u_x pro položku i_y. Tato matice je poté vstupem predikce nebo doporučení. Výstupek predikce je položka j pro konkrétního uživatele a. Výstupem doporučení je seznam N položek pro konkrétního uživatele a.



Obrázek 2.3: Základní schéma procesu kolaborativního filtrování. Zdroj [18]

Filtrování založené na podobnosti uživatelů (user-based filtering)

Tato metoda je přímou implementací hlavní myšlenky kolaborativního filtrování: 1) najdi uživatele (tzv. nejbližší sousedy), jejichž hodnocení je podobné s hodnocením právě zkoumaného uživatele; 2) využij hodnocení těchto sousedů k doporučení položky, kterou zkoumaný uživatel ještě nehodnotil. Tento princip byl poprvé publikován v článku GroupLens o doporučování [15].

Abychom mohli určit doporučení pro uživatele Alice, musíme zjistit seznam uživatelů, které mají podobné hodnocení jako Alice. K tomu slouží níže popsané přístupy. Poté se vezmou hodnocení položek jednotlivých uživatelů, váhovaných jejich podobností s Alicí a následně je predikována předpověď pro Alici pro konkrétní položky, které sama ještě nehodnotila. Výpočet podobnosti je funkce $s : U \times U \rightarrow \mathbb{R}$, výsledkem pro dva zadané uživatele je reálné číslo vyjadřující míru podobnosti. Základní typy podobnostních funkcí:

- **Pearsonova korelace** - nejčastější funkce pro výpočet podobnosti, výsledná podobnost je vyjádřena číslem v intervalu $< -1, 1 >$, kde záporné číslo značí negativní

korelaci. Nevýhodou této funkce je vysoká korelace mezi uživateli s nízkým počtem hodnocení - to lze vyřešit nastavením prahu minimálního počtu hodnocení.

- **Spearmanova korelace** - jedná se o modifikaci pearsonova přístupu. Vzorec k výpočtu zůstává stejný, jenom se mění hodnoty atributů. Mění se význam hodnocení - položka s nejvyšším ohodnocením je na pozici důležitosti 1. Nejhůře hodnocená položka je na poslední pozici
- **Kosinová podobnost** - tento přístup je odlišný od předešlých dvou. Ke svému výpočtu využívá vektory, popisující uživatele. Uživatelé jsou reprezentování $|I|$-dimenzionálním vektorem a podobnost dvou uživatelů je vypočítána jako kosinová vzdálenost těchto dvou vektorů ohodnocení.

Po výpočtu nejpodobnějších uživatelů vezmeme K-nejpodobnějších. Od těchto K sousedů vezmeme hodnocení položek, které uživatelka Alice nehodnotila. Vhodnou agregační metodou se poté vytvoří seznam položek, které by se Alici mohly líbit.

V průběhu používání systému uživatel hodnotí (či mění hodnocení) položky, tím pádem se mění i vektor podobnosti s ostatními uživateli. To komplikuje předvýpočet matice podobnosti. Ta je závislá jak na hodnocení uživatele, tak i na hodnocení uživatelů jemu podobných. Proto systémy tohoto typu provádí výpočet až v případě nutnosti.

Tento přístup se nedá použít na systémy, kde se nachází tisíce uživatelů a miliony položek, protože výpočet by probíhal nad enormně velikou maticí, což není v reálném čase možné, při běžné výpočetní kapacitě [11].

Filtrování založené na podobnosti položek (item-based filtering)

Narozdíl od filtrování založeného na podobnosti uživatelů, kde byla vypočítávána podobnost mezi uživateli, aby následně byly nalezeny položky pro doporučení, tento přístup využívá podobnost mezi položkami systému. Základní myšlenkou je, že pokud mají dvě položky podobné hodnocení od mnoha uživatelů, jsou si tyto položky podobné. Například pokud si uživatel zobrazí nějakou položku, může mu systém nabídnout seznam dalších, podobných položek (právě v závislosti na podobnosti hodnocení od ostatních uživatelů).

Pro výpočet podobnosti lze použít metody zmíněné ve filtrování založeném na podobnosti uživatelů. Tím ovšem na první pohled dostáváme stejný problém se škálovatelností tohoto přístupu. Hlavním rozdílem je fakt, že můžeme provést výpočet matice podobnosti v offline režimu. V předešlém přístupu změna hodnocení položky mohla silně ovlivnit matici podobnosti uživatelů. Za předpokladu, že máme v systému počet hodnocení produktu větší než počet hodnocení uživatelů, můžeme počítat s vlastností, že nové hodnocení položky uživatelem neovlivní výslednou podobnost. Díky této vlastnosti můžeme snížit rozměry výsledné matice podobnosti položek a přepočítávat její hodnotu v momentě, kdy není systém tolik vytížený.

2.3.3 Filtrování založené na znalostech

Předešlé dvě techniky mají své výhody v doporučovacích doménách, kde můžeme jednotlivé položky kvalitativně ohodnotit - například filmy, knihy, písničky nebo novinové články. Ovšem pro oblast prodeje počítačů, aut, bytů nebo finančních produktů jsou tyto přístupy nevhodné - například nemovitost není položka, která by byla denně kupována, a tudíž získat relevantní počet hodnocení je téměř nemožný (jelikož hodnocení je základem předchozích dvou přístupů, nejsou proto vhodné k použití).

Filtrování založené na znalostech překonává tyto problémy podrobnou analýzou uživatelských preferencí a jednotlivých položek v systému. Na základě uživatelské preference probíhá filtrování individuálně pro každého uživatele zvlášť. Výhodou tohoto přístupu je zjišťování uživatelových preferencí při vstupu do systému a to odstraňuje problém studeného startu. Velkou nevýhodou je důraz kladený na návrháře tohoto systému, aby jednotlivé položky byly popsány co nejdetailněji.

Existují dva základní přístupy filtrování na základě znalostí. Prvním je **doporučení na základě požadavků** [4] (case-based recommending) a druhým je **doporučení na základě omezení** [7] (constraint-based recommending). Oba přístupy si jsou velmi podobné - uživatel musí detailně specifikovat své požadavky. Hlavní rozdíl je v přístupu k doporučování [7]. Doporučení na základě požadavků počítá doporučení na základě podobnosti položek vzhledem k uživatelovým požadavkům. Doporučení na základě omezení využívá předdefinovanou znalostní bázi, která obsahuje pravidla. Ty určují, jakým způsobem uživateli doporučit položky.

- **Doporučení na základě omezení** - systém tohoto typu bývá běžně určen pomocí množiny dvou proměnných (V_c a V_{prod}) a množiny tří omezení (C_r, C_f, C_{prod}). Význam proměnných a omezení je následující:
  - V_c - uživatelské vlastnosti - zde jsou popsány požadavky zadané uživatelem (například požadované rozměry bytu, lokace, maximální cena,...)
  - V_{prod} - produktové vlastnosti - popisuje vlastnosti konkrétní položky systému (napřklad velikost bytu, jeho lokace a cena, počet místností,...)
  - C_r - omezení - popisuje jednotlivá omezení pro uživatelské vlastnosti (například pokud je byt v centru a je nový, cena bude převyšovat 3 miliony Kč)
  - C_f - podmínky filtrování - definuje podmínky, za jakých by měly být položky vybrány. Je určen vztah mezi uživatelskými a produktovými vlastnostmi (například uživatelům s nízkým kapitálem nebudou doporučovány byty s vysokou cenou)
  - C_{prod} - definuje dostupné položky. Je to podmnožina množiny V_{prod}, užitím omezujících podmínek.
- **Doporučení na základě požadavků** - uživatel systému definuje atributy položek, které chce maximalizovat (například výměra nového bytu) a zároveň chce některé atributy minimalizovat (například vzdálenost bytu od centra a/nebo jeho cena). Tyto atributy se poté berou v potaz při výpočtu podobnosti položek v systému. Výpočet najde podobnost položek s uživatelovým atributem, který chce maximalizovat a poté následně provede výpočet pro atribut, který chce minimalizovat a zobrazí uživateli jednotlivé výsledky.

2.3.4 Komunitní filtrování

Tento typ doporučovacích systémů je založený na preferencích přátel daného uživatele. Systém dodržuje následující motto: *„řekni mi kdo jsou tví přátele, já ti povím kdo jsi ty“* [1]. Bylo zjištěno, že uživatelé raději inklinují k doporučení odvozené od jejich přátel, než od totožného doporučení od anonymního uživatele [21]. V posledních letech jsou sociální sítě na vzestupu, čímž mohou využívat tuto doporučovací techniku čím dál snadněji - vzniká nová oblast sociálních doporučovacích systémů [9].

Systém založený na komunitní filtrování získává hlavní znalosti z hodnocení, která byla poskytnuta lidmi, které si uživatel označil jako své přátele. [9] uvádí, že výhoda tohoto přístupu nemusí být až tak znatelná v porovnání s kolaborativním filtrováním, zdroj také zmiňuje možnou výhodu přístupu v překonání problému studeného startu, jelikož položky jsou odvozeny od skupiny přátel.

2.3.5 Filtrování založené na demografických informacích

Systémy doporučování založené na tomto principu doporučují položky v závislosti na demografických vlastnostech uživatele. Mezi takové vlastnosti můžeme řadit například věk, pohlaví, bydliště, vzdělání, náboženství, příjem, aj. Základní myšlenkou těchto systémů je fakt, že rozdílné demografické skupiny mají rozdílné preference.

Například věková skupina 10-18 let bude preferovat články o pop-kulturních celebritách, kdežto skupina 35-40 bude preferovat zprávy o aktuálním dění ve světě. Velké internetové stránky mohou uživatelům nabízet obsah v závislosti na území, kde se uživatel nachází. Například server YouTube.com na hlavní stránce bude zobrazovat jiná videa pro obyvatele Ameriky a jiná pro obyvatele Česka.

2.3.6 Hybrid doporučovací systémy

V posledních letech probíhá výzkum v oblasti hybridních doporučovacích systémech. Jádrem této techniky je kombinace výše zmíněných přístupů do jednoho komplexního. Kombinovat může dvě nebo více doporučovacích technik. Nejčastějším příkladem je kombinace kolaborativního filtrování a nějaké další techniky. Tím lze dosáhnout vyšší výkonosti a/nebo přesnosti jednotlivého doporučování pro uživatele.

Hybridní přístup může být implementován několika přístupy, nejběžněji: použití jednotlivých technik odděleně a poté vytvořit jednu sadu výsledků (například filtrování založené na obsahu aplikované na kolaborativní filtrování). Dalším přístupem je kombinace jednotlivých technik do jednoho kompaktního modelu. Studie ukázaly, že hybridní přístup může zvýšit kvalitu doporučování v porovnání s „čistými“ technikami. Výhodou hybridního přístupu může být překonání nejběžnějšího problému doporučovacích systému - tzv. problém studeného startu a problému řídkých dat (viz 2.5).

V [5] je definováno sedm typů hybridních přístupů:

- **Váhování** - skóre jednotlivých doporučovacích modulů je kombinováno do jediného výsledku použitím váhovaného vektoru. Výhodou tohoto přístupu je jednoduchá implementace. Nevýhodou může být fakt, že v některých případech doporučení je potřeba váhy dynamicky měnit (například snížit váhu kolaborativního filtrování, pokud doporučujeme novou položku, která nemá dostatek uživatelských hodnocení)

- **Přepínání** - systém se rozhoduje, kterou doporučovací techniku využije pro danou situaci doporučení a vybere pouze jednu konkrétní. Výhodou je, že pokud selže jedna doporučovací technika, stačí se přepnout na další techniku. Nevýhoda tohoto přístupu tkví na nutnosti definice (často složitých) kritérií, při kterém dojde k přepnutí na další techniku.

- **Míchání** - výsledky od více doporučovacích technik jsou prezentovány dohromady.

- **Kombinování příznaků** - výsledkem doporučovacích technik je sada příznaků, které jsou následně zkombinovány a předloženy do jediného doporučovacího algoritmu.

- **Úprava příznaků** - velmi podobný předešlému přístupu. Doporučovací systém vytvoří sadu příznaků, které jsou vstupem dalšího doporučovacího systému.
- **Kaskáda** - jednotlivé doporučovací techniky jsou seřazeny v tzv. kaskádě - výstup jedné techniky je vstupem druhé techniky. Kaskáda je určena prioritami jednotlivých technik
- **Meta-úroveň** - výstupem jedné doporučovací techniky je model, který je potom vstupem druhé doporučovací techniky

2.4 Měření přesnosti doporučení

Cílem doporučovacích systémů je najít vhodnou podmnožinu položek, které by mohly mít přínos pro uživatele. Abychom mohli vyhodnotit míru přínosu, musíme použít měřící techniku. Měření přesnosti je typicky nezávislé na uživatelském rozhraní a tudíž měření může probíhat v offline režimu. Následující rozdělení vychází z [20]:

2.4.1 Měření přesnosti předpovědi hodnocení

Pro výpočet lze použít následující techniky:

Střední kvadratická odchylka (Root Mean Squared Deviation - RMSD)

Systém generuje množinu předpovědí $\hat{r}_{ui}$ hodnocení položky i uživatelem u pro testovací sadu N, tvořenou páry uživatel-položka (u, i), pro které je známo hodnocení r_{ui}. Výpočet přesnosti předpovědi a aktualního hodnocení je dán vzorcem:

$$RMSD = \sqrt{\frac{1}{|N|} \sum_{(u,i) \in N} (\hat{r}_{ui} - r_{ui})^2} \tag{2.1}$$

kde $|N|$ je celkový počet hodnocení v sadě N. Vzorec počítá průměr druhých mocnin rozdílu mezi očekávaným hodnocením a aktuálním hodnocením

Střední absolutní chyba (Mean Absolute Error - MAE)

Jedná se o alternativu k RMSD. Vzorec místo druhé mocniny rozdílu počítá absolutní hodnotu rozdílu předpovězeného hodnocení se skutečným hodnocením

$$MAE = \sqrt{\frac{1}{|N|} \sum_{(u,i) \in N} |\hat{r}_{ui} - r_{ui}|} \tag{2.2}$$

2.4.2 Měření užitečnosti předpovědi

V některých případech nechceme vyhodnotit míru přesnosti předpovědi hodnocení, ale chceme určit, zda předpověď má přínos, či nikoliv. Výsledkem je binární hodnota: 1 - položka má přínos; 0 - položka nemá přínos. V offline režimu vyhodnocení máme běžně datovou sadu sestávající z položek, které uživatelé ohodnotili. Poté zvolíme testovacího uživatele, některé jeho položky skryjeme a spustíme doporučovací algoritmus. Následně máme čtyři různé možnosti, které mohou nastat. To ilustruje tabulka 2.1

	Doporučeno	Nedoporučeno
Užitečné	True-Positive	False-Negative
Neužitečné	False-Positive	True-Negative

Tabulka 2.1: Klasifikace možných výsledků doporučení položky uživatelovi

- **True-Positive** (TP) - počet položek, které byly užitečné a systém je doporučil
- **False-Positive** (FP) - počet položek, které nebyly užitečné, ale systém je doporučil
- **False-Negative** (FN) - počet položek, které byly užitečné, ale systém je nedoporučil
- **True-Negative** (TN) - počet položek, které nebyly užitečné a systém je nedoporučil

Z hodnot v této tabulce můžeme odvodit metriky vypovídající o přesnosti předpovědi. Falešně pozitivní míra (False Positive Rate), též označována jako chyba prvního druhu se počítá pomocí vzorce $FPR = \frac{FP}{TN+FP}$. Toto číslo vyjadřuje míru výskytu položek, které byly doporučeny a označeny jako neužitečné. Další metrikou je falešně negativní chyba (False Negative Rate), též značena jako chyba druhého druhu. Výpočet probíhá vzorcem $FNR = \frac{FN}{TN+FP}$. Číslo vyjadřuje míru výskytu položek, které nebyly doporučeny, ale přesto byly užitečné.

Z údajů v tabulce můžeme taky vypočítat celkovou přesnost doporučení - F_1 score (jinak též F-score, nebo F-measure). Pro výpočet používá přesnost a úplnost doporučení.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

$$F_1 score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.5}$$

2.5 Problémy doporučovacích systémů

Doporučovací techniky ze své podstaty definice zahrnují různé typy problémů, které mohou v menší, či větší míře ovlivnit míru úspěšnosti samotného doporučení.

Studený start (cold start)

Jedná se o nejčastější problémem npvých doporučovacích systémů [11]. Sem spadají problémy typu „co doporučit novému uživateli, o kterém nemáme informace?“ a „co dělat s položkami, které nemají žádné hodnocení (či jiné relevantní informace)“. Problém studeného startu má nejhorší dopad na kolaborativní filtrování. Existuje řešení, které za pomocí hybridních technik umožní tento problém minimalizovat.

Problém řídkých dat (data sparsity)

V reálných systémech se vyskytují velké databáze obsahující tisíce (miliony) položek a uživatelů. Pokud bychom vytvořili matici hodnocení, kde řádky budou uživatelé a sloupce položky a hodnocení uložili do buněk, dostali bychom hustotu zaplnění v řádech desetin až setin procent. Řešení tohoto problému mohou poskytnout hybridní techniky.

Šedá ovce (gray sheep)

Uživatel je považován za tzv. šedou ovci, pokud se jeho hodnocení, či názor neshoduje s žádnou existující skupinou uživatelů v systému. Doporučení pro takové uživatele tedy nemá velký přínos. Speciálním případem takových uživatelů jsou ti, jejichž hodnocení je natolik radikální, že doporučení pro ně je téměř nemožné.

Šilinkový útok (shilling attack)

Hlavní motivací pro útoky tohoto typu je osobní, či finanční zisk útočníka. Ten svým hodnocením zvyšuje popularitu svých položek a/nebo snižuje hodnocením popularitu položek svých konkurentů [10]. Pokud je cílem doporučovacího systému prodej položek, může se tímto útokem dostat do popředí podvržené položky. Zdroj [10] dále uvádí možné způsoby řešení tohoto problému.

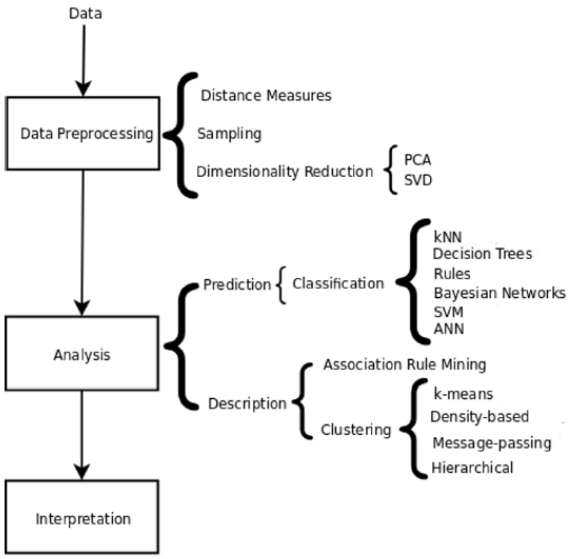
Kapitola 3

Dolování z dat

Doporučovací systémy typicky využívají technik a metod z dalších počítačových odvětví. Nejdůležitějším odvětvím je dolování z dat (Data mining). Podle [23] můžeme tvrdit, že dolování z dat je extrakce zajímavých (netriviálních, skrytých, dříve neznámých a potenciálně užitečných) modelů dat a vzorů z velkých objemů dat.

Obrázek 3.1 ilustruje tři fáze, které v dolování z dat probíhají:

- Předzpracování dat (data preprocessing)
- Analýza dat (data analysis)
- Interpretace výsledků (result interpretation)



Obrázek 3.1: Obrázek ukazuje kroky a metody dolování z dat. Zdroj [16]

3.1 Předzpracování dat

Data z reálných systémů bývají často zašuměná, a proto je potřeba předzpracování (např. vyčištěním, filtrací, transformací,...), aby je bylo možné použít pro strojové učení. Data

jsou definována jako seznam objektů, které jsou popsány atributy.

3.1.1 Vzorkování

Vzorkování (sampling) je základní technikou využívanou v dolování. Metoda vybírá podmnožinu relevantních dat z celé datové sady. Použitím vzorkování můžeme redukovat počet prvků datové sady, která může být velmi rozsáhlá, a tím snížit výpočetní náročnost. Jádrem této techniky je nalezení reprezentativního vzorku - ten má přibližně stejné hodnoty atributů jako zbylé položky ve vzorku.

Nejjednodušší vzorkovací technikou je náhodné vzorkování, kde reprezentant je vybrán zcela náhodně z datové sady (s rovnoměrným rozdělením pravděpodobnosti). Další technikou je stratifikované vzorkování. Data rozdělíme do několika skupin podle jejich atributů (skupiny nemusí nutně obsahovat stejný počet položek). Nyní na každou skupinu zvlášť použijeme náhodné vzorkování pro výběr reprezentanta. Vzorkování můžeme provádět bez navrácení - vybraná data vyjmeme z datové sady, tím zaručíme, že budou vybrány pouze jedenkrát; nebo s navrácením - data mohou být opakovaně vybrána ze sady. Běžnou praktikou pro náhodné vzorkování bez navrácení je rozdělit datovou sadu v poměru 80:20 na trénovací a testovací sadu.

Doporučovací systémy často využívají vzorkování, když chtějí rozdělit hodnocení položek od uživatelů na trénovací a testovací sadu. V tomto kroku je často využívána křížová validace. Náhodné vzorkování není často vhodnou metodou a využívají se různé modifikace - například zvýšíme pravděpodobnost pro nejnovější hodnocené položky. Dále můžeme také požadovat, aby velikost množiny vybraných vzorků byla v poměru s počtem hodnocení jednotlivých uživatelů. Zmíněné problémy jsou předmětem vyhodnocování kvality doporučovacích systémů a stále v této oblasti probíhá vývoj [16].

3.1.2 Redukce dimenzionality

Rozsáhlé datové sady pro doporučovací systémy často trpí problémem řídkosti (viz 2.5). Tento problém například zhoršuje výsledky shlukovacích metod, či hledání odlehlých hodnot. Techniky pro redukci dimenzionality pomáhají řešit tyto problémy tím, že transformují datovou sadu z vysoce dimenzionálního prostoru do prostoru s nižší dimenzionalitou. I v nejprimitivnějším doporučovacím systému můžeme mít rozsáhlou, řídkou matici s tisíci řádky a sloupci (např. uživatelé a položky), kde většina hodnot je rovna nule. Použitím technik pro redukci dimenzionality na takovou matici můžeme zrychlit výpočet predikce a snížit paměťové nároky algoritmů.

Základními technikami pro redukci dimenzionality je metoda PCA (Analýza hlavních komponent - Principal Component Analysis) a SVD (singulární rozklad - Singular Value Decomposition)

Analýza hlavních komponent

PCA je statistickou metodou, která se snaží nalézt vzory ve vysoce-dimenzionálních prostorech. Metoda pomocí lineární kombinace rysů, které nesou nejvíce informací, umožňuje snížit dimenzionalitu. Je založena na statistických charakteristikách dat reprezentovaných kovariační maticí, jejích vlastních číslech a odpovídajících vlastních vektorech [23].

3.1.3 Čištění zašuměných dat

Některé hodnoty atributů dat v sadě mohou chybět, či nějakým způsobem mohou být nekonzistentní. Data také mohou obsahovat odlehlé hodnoty, které pro dolování nemají přínos. Čištění dat je jednou z nejdůležitějších částí předzpracování dat. Mezi metody čištění zahrnujeme: ošetření chybějících hodnot, odstranění šumu, úpravu nekonzistentních dat, či identifikaci odlehlých hodnot.

V kontextu s doporučovacími systémy rozlišujeme přirozený a úmyslný šum [14]. Přirozený šum vzniká neúmyslným hodnocením uživatele. Úmyslný šum vytváří uživatelé cíleně za účelem poškození kvality doporučování. Odstranění přirozeného šumu můžeme dosáhnout algoritmicky, nebo můžeme požádat uživatele o znovu ohodnocení položky.

3.2 Klasifikace

Úlohou klasifikace je zařazení daného objektu do jisté třídy na základě jeho atributů. Klasifikace probíhá ve dvou fázích - trénování a testování. V trénovací fáze je klasifikátoru předložena trénovací datová sada. Výstupem je poté model, který popisuje do jaké kategorie budou zařazena nová, dosud neviděná data. Existují dvě základní kategorie klasifikace - s učitelem a bez učitele.

Učení s učitelem

Trénovací sada obsahuje informace o jednotlivých datech a jejich příslušnost k dané klasifikační třídě. Klasifikátor poté tuto informaci využívá k tvorbě modelu. Ten se poté využívá při samotné klasifikaci nad daty, u nichž chceme nalézt jejich zařazení do třídy. Mezi metody řadíme K-nejbližších sousedů, rozhodovací stromy, klasifikace založená na pravidlech, Bayesovský klasifikátor, neuronové sítě, SVM [17].

Učení bez učitele

Vstupní trénovací data nemají informaci o příslušnosti k dané třídě, tedy cílem těchto metod je v těchto datech nalézt strukturu. Mezi hlavní metody patří shluková analýza.

Shluková analýza

Shlukování je proces rozdělování objektů do shluků na základě podobnosti objektů [23]. Výsledné shluky nazýváme třídy. Algoritmy kladou důraz na to, aby data uvnitř shluků si byla co nejvíce podobná a data mezi různými shluky byla co nejvíce rozdílná. Podobnost dat můžeme měřit například vzdálenostní funkcí (Euklidovská vzdálenost, manhattanská metrika,...). Shluková analýza nachází vztahy mezi vstupními daty, aniž by podrobně znala jejich strukturu.

Na jednotlivé metody shlukové analýzy klademe následující požadavky [23] : škálovatelnost, schopnost zpracovávat různé typy atributů, vytvářet shluky různého tvaru, schopnost vyrovnat se zašuměným obsahem, necitlivost na pořadí vstupních záznamů, schopnost zpracovávat vysokodimenzionální data, aj.

Shlukovací metody můžeme rozdělit následovně [23]: Metody založené na rozdělování, hiearchické metody, metody založené na hustotě, metody založené na mřížce, metody založené na modelech, metody pro shlukování vysoce-dimenzionálních dat.

Kapitola 4

Návrh doporučovacího systému

Základním problémem nových systému pro doporučování je nedostatek dat, nad kterými by mohly algoritmy provést výpočty a následně pro uživatele zobrazit výsledné doporučení - tzv Problém studeného startu (viz 2.5). Po dohodě s vedoucím práce byla zvolena aplikační oblast zaměřena na doporučování knih. Na českém trhu dominují dva významné weby, které sdružují aktivní čtenáře, databázi knih a zároveň hodnocení jednotlivých uživatelů ke konkrétním knihám. Po jednání se zmíněnými weby se úspešně podařilo domluvit spolupráci se serverem ČBDB[1] - zastupovaným panem Jandou.

Abych dostal přístup k datové knižní sadě serveru, byla součástí dohody s panem Jandou podmínka, že zdrojové kódy doporučování budou implementovány do jeho stávajícího systému. Zároveň také bylo podmínkou, že přístup k reálné datové sadě dostanu až při hotovém programu - tedy při vývoji jsem musel použít náhodně generovaná data pro knihy a jejich hodnocení.

Tato kapitola popisuje návrh jednotlivých částí, které dohromady tvoří doporučovací systém. Pro jednoduchost škálovatelnost a udržovatelnost kódu je využit princip objektového programování.

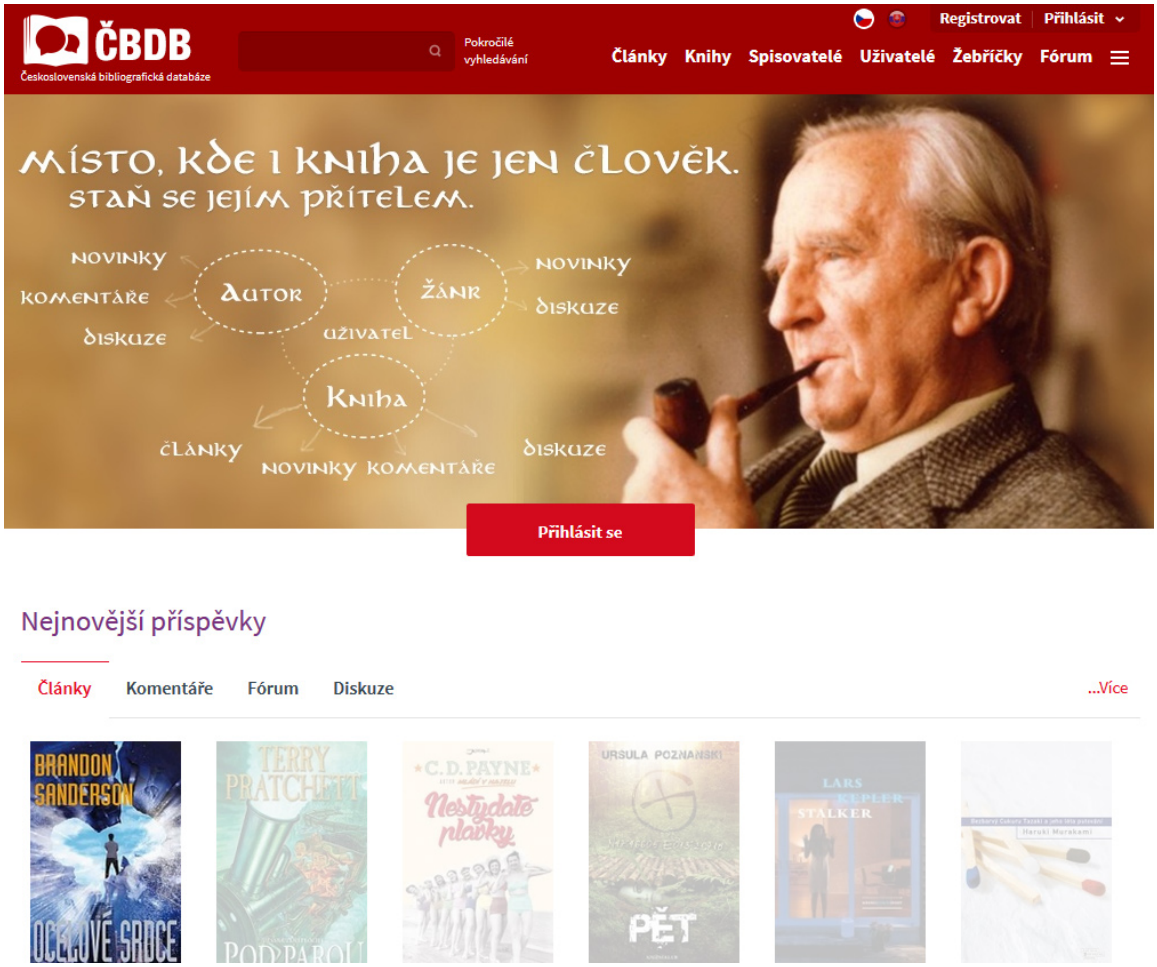
4.1 Popis stávajícího systému

Československá bibliografická databáze vznikla v roce 2009. Za tuto dobu nashromáždila značně rozsáhlou datovou sadu. Obrázek 4.1 zachycuje úvodní stránku serveru. Návštěvník má možnost číst články, otevřít katalog knih a následně detail vybraných knih. Má také možnost zobrazit seznam autorů a jejich detailní kartu. Dále má možnost vypsání všech registrovaných uživatelů a také různé žebříčky popularity. V neposlední řadě má uživatel možnost se přihlásit do svého profilu a nebo si takovýto profil vytvořit.

Služby serveru cbdb.cz

V úvodu byly zmíněny možnosti neregistrovaného návštěvníka. Ty jsou ovšem velmi limitované v porovnáním s možnostmi registrovaného uživatele. Registrace probíhá zadáním jedinečného uživatelského jména, heslem, kontaktním emailem a preferovaným jazykem. Zobrazení možností neregistrovaného návštěvníka a přihlášeného uživatele zobrazuje diagram použití na obrazku 4.2

[1] http://www.cbdb.cz - Československá bibliografická databáze



Obrázek 4.1: Úvodní stránka serveru cbdb.cz

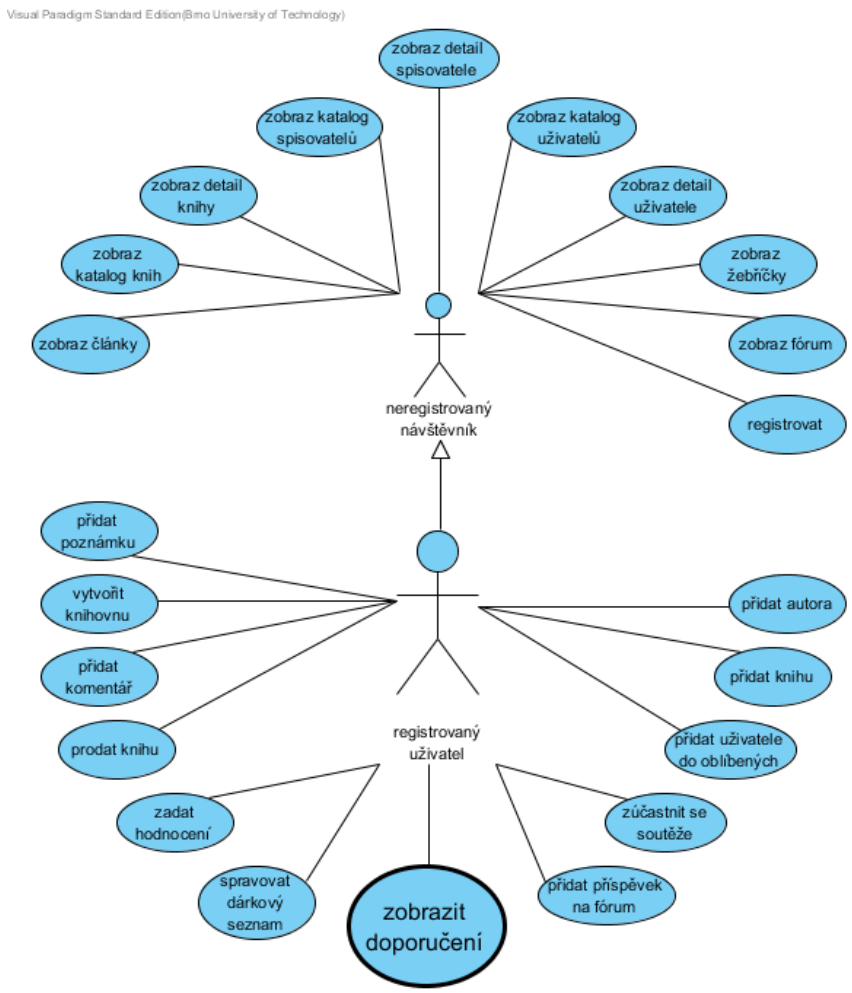
Datová sada

Každá jednotlivá kniha nacházející se v databázi byla ručně vložená uživateli serveru a následně schválena tamními moderátory. Orientační velikost databáze je následující:

- 280 žánrů
- 40 000 uživatelů
- 130 000 knih
- 1 000 000 hodnocení knih

Obrázek 4.3 znázorňuje diagram schématu relevantního úseku databáze, nad kterou bude doporučovací systém vystaven. V jádru se vyskytuje tabulka *hodnocení*, která udržuje informace o konkrétním hodnocení daného uživatele pro zvolenou knihu. Na tuto tabulku jsou dále napojeny tabulky *kniha*, *uživatel* a *autor*. Samotné hodnocení je celočíselná hodnota v rozmezí 1-5 (vyjadřující stupnici velmi nelíbilo → velmi líbilo).

Pro každou knihu je také uložen její žánr. Vazba je m:n, tedy jedna kniha může mít více žánrů a jeden žánr může být přiřazen více knihám. Dále je ke knize také napojen její autor

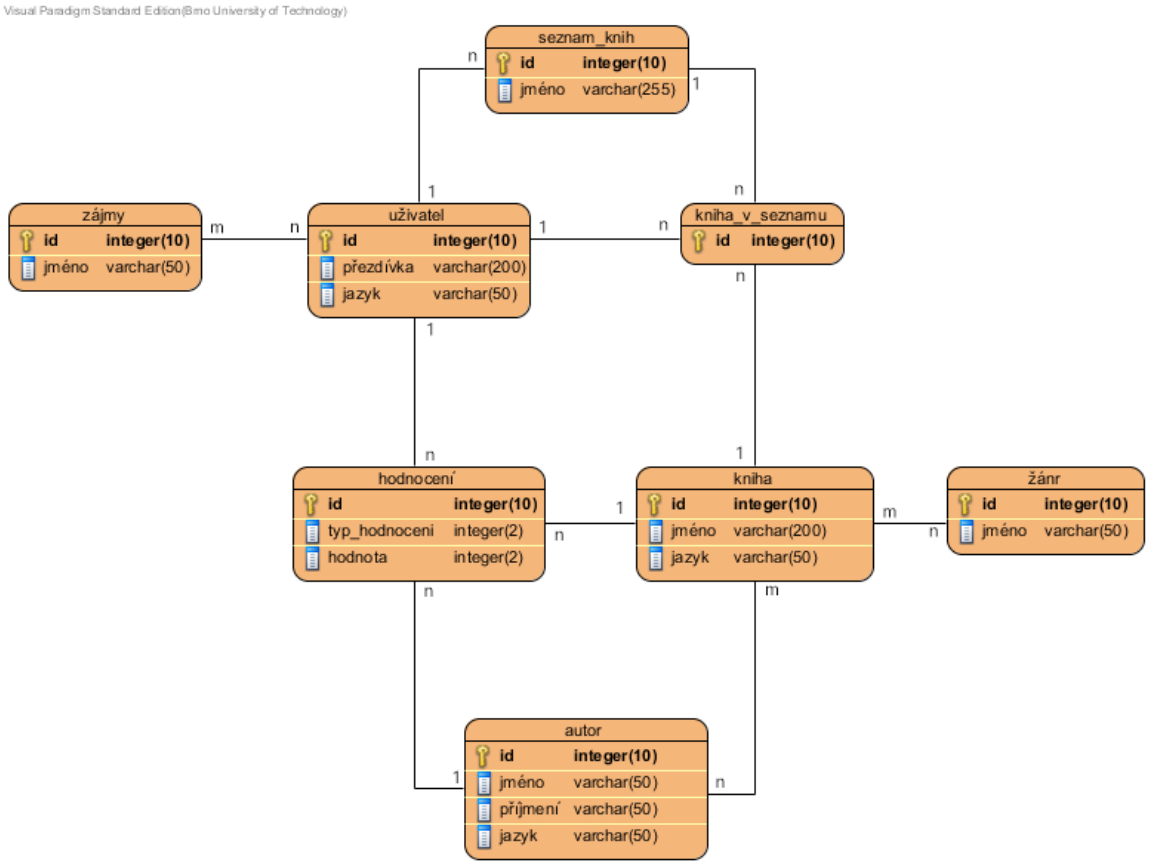


Obrázek 4.2: Diagram použití pro dva různé uživatele serveru cbdb.cz

(resp. autoři), vazba opět m:n. Každý takovýto autor v systému může mít od uživatelů přiřazené hodnocení, tedy hodnocení může být směrované jak na knihu, tak i na autora.

Uživatelé mají možnost si vytvářet něco jako virtuální knihovny - pojmenované seznamy, do kterých si uživatel přiřadí knihy. K tomuto slouží tabulka *seznam_knih*, kde každá virtuální knihovna má sve jméno a do ní přiřazené knihy. Jedna konkrétní kniha může být přiřazena do vícero knihoven.

V rámci této práce bude algoritmus pro doporučování primárně využívat tabulku *hodnocení* a následně *žánry*. Žánry budou hrát velkou roli pro počítání podobnosti jednotlivých uživatelů a také pro doporučení samotné. Užitím hodnocení pro doporučení dostáváme doporučovací systém typu kolaborativního filtrování (viz 2.3.2). Užitím historie hodnocených knih a extrahováním informace o oblíbenosti žánrů a následném doporučení dostaneme systém filtrování založený na obsahu (viz 2.3.1). Nevýhodou systémů kolaborativního filtrování je škálovatelnost. Pro malou sadu uživatelů fungují algoritmy relativně rychle. Ovšem s rostoucí uživatelskou základnou roste i velikost matice hodnocení (Obrázek 2.3). Pro naši datovou sadu dostáváme $|uzivatele \times knihy| = 5.2 * 10^9$ všech možných kombinací hodnocení $(uzivatel_i, kniha_j)$. V současném okamžiku se v databázi nachází téměř 10^6 uživatelských hodnocení. Podílem těchto čísel zjistíme, že zaplnění matice hodnocení je $1.92 * 10^{-4}$, neboli



Obrázek 4.3: Diagram schématu databáze zachycuje relevantní část databáze, která bude dostupná pro doporučovací modul

0.019%. Toto číslo upozorňuje na problém řídkých dat (viz 2.5).

V přístupech kolaborativního filtrování se podobnost uživatelů obecně počítá na základě počtu společně ohodnocených knih. V databázi se ovšem nachází 40 000 uživatelů (a předpokládejme, že toto číslo do budoucna poroste). Abychom spočetli podobnost uživatelů, museli bychom porovnat každého uživatele s každým. V takovém případě se dostáváme na $40\ 000^2$ vzájemných porovnání. V rámci každé dvojice uživatelů se poté porovnají jejich knihy, které ohodnotili oba současně. Dostáváme složitost $O(n^2m)$, kde n značí počet uživatelů a m počet ohodnocených knih.

Užitím vlastností systémů založených na obsahu můžeme toto číslo razantně zmenšit. Tyto systémy se zakládají na užití informací, které poskytne pouze uživatel sám. Pro každého uživatele si tedy uložím jeho preference - žánry, které čte nejčastěji. Každé hodnocení je svázané s knihou a každá kniha je svázaná se svým žánrem. Díky této vazbě lze extrahovat hodnotu četnosti výskytu žánru. Pro každého uživatele zvlášť uložím tři nejčtenější žánry (vycházím z předpokladu, že uživatel nebude záporně hodnotit spoustu spolu-nesouvisejících knih). Následně budu porovnávat pouze uživatele, jejichž preference se shodují. Díky tomuto omezení lze významně snížit počet porovnání.

Díky kombinaci přístupů systémů založených na obsahu a kolaborativních systémů se vytvářený systém řadí do kategorie hybridních doporučovacích systémů. Tímto můžeme docílit přesnějších a sofistikovanějších doporučení. Snížíme tím výpočetní nároky na server.

Dosavadní doporučovací systém na cbdb.cz

Před samotným začátkem vývoje mého doporučovacího systému se již na serveru cbdb.cz nacházel základní doporučovací systém. Každý uživatel má možnost označit libovolné knihy do seznamu svých oblíbených knih. Tímto faktem dává uživatel najevo, že takto vybrané knihy má opravdu rád. Následně algoritmus vybere náhodně dvě knihy z tohoto seznamu a snaží se najít jiné uživatele, kteří mají tyto dvě knihy taky označené jako oblíbené. Následně se vyberou knihy takto nalezených uživatelů a zobrazí se všechny, které daný uživatel ještě nehodnotil. Výpočet takovéhoto seznamu doporučených knih probíhá pokaždé, když uživatel zažádá o tento seznam (otevření stránky 'Doporučené knihy'). Tento přístup nepředpočítává žádná data a všechno počítá až v momentě žádosti uživatele. Jelikož výběr dvou knih probíhá náhodně, může nastat situace, že uživatel si zobrazí seznam doporučených knih a při každém zobrazení může systém zobrazit jiné doporučené knihy (jelikož náhodně vybral dvě jiné, než v předchozím požadavku). V extrémním případě může být vybrána kombinace dvou knih taková, že žádný jiný uživatel tuto kombinaci nemá. Tudíž v tomto případě nebudou uživateli doporučeny žádné knihy.

4.2 Životní cyklus vývoje doporučovacího modulu

Životní cyklus celé aplikace probíhal v iteritvním vývoji. V každé iteraci probíhal návrh a následně byla provedena implementace dle tohoto návrhu. V první iteraci byla provedena analýza současného stavu serveru cbdb.cz a proběhl návrh rozšíření stávající databáze. Dále byly zvoleny programovací jazyky pro implementaci. V druhé iteraci byla vytvořena základní struktura objektů databázové vrstvy a generátoru, který bude simulovat reálná data pro doporučování (seznam uživatelů, seznam knih, seznam žánrů a hodnocení). Ve třetí iteraci vývoje se na tuto vrstvu napojila vsrtva doporučování, která již z předešlého kola měla naimplementované metody pro práci s databází. Ve čtvrté, závěrečné, iteraci proběhla refaktorizace kódu, zavedení dependency injection (viz kapitola 4.2.4).

4.2.1 1. iterace - požadavky

Cílem je vytvořit doporučovací zásuvný modul, který bude snadno implementovatelný do stávajícíh kódu serveru cbdb.cz. Z tohoto důvodu je nejdůležitějším požadavkem na implementaci jeji **jednoduché použití** pro získání doporučení. Důraz je kladen na použití co nejméně operací tohoto modulu, aby byl navrácen seznam doporučených knih. Dalším požadavky se týkají výpočtů, které budou probíhat s týdenní periodou. Každá část těchto výpočtů bude možna **volat odděleně** (uživatelské preference, podobnost žánrů, aktivní uživatelé - výpočty jsou detailně popsány v druhé iteraci). Pro toto řešení jsem se rozhodl z důvodu flexibility. Až bude systém nasazen, můžeme zjistit, že některé z těchto výpočtů mohou být pouštěny s menší či větší periodou.

Použité technologie

Pro doporučovací systémy existují rozsáhle knihovny psané převážně v jazyce Java - např. Apache Mahout[2], která se využívá tam, kde je potřeba strojového učení. Webová aplikace serveru cbdb.cz je napsána v jazyce PHP a tudíž jsem jako vývojový jazyk použil také PHP. Po průzkumu existujících řešení doporučovacích systémů založených na jazyce PHP jsem

[2] http://mahout.apache.org/

došel k závěru, že neexistuje žádná vhodná knihovna, která by urychlila vývoj aplikace. Proto jsem se rozhodl celou aplikaci napsat vlastní silou.

Návrh databáze

Současnou databázi serveru cbdb.cz bylo potřeba rozšířit o šest tabulek pro uchování potřebných informací:

aktivni_uzivatele	zde bude seznam uživatelů, kteří mají více než 20 ohodnocených knih
feedback	zde bude zpětná vazba od uživatelů systému, kteří pro dané doporučení dají +1 nebo -1, vyjadřující jejich pocit k doporučení
uzivatele_doporuceni	pokud si uživatel požádá o seznam doporučených knih, po prvním výpočtu se zde uloží seznam N doporučených knih s týdenní platností
uzivatele_preference	zde jsou uloženy pro každého aktivního uživatele tři žánry, které hodnotil nejčastěji, včetně míry podílu tohoto žánru vzhledem k počtu všech ostatních žánrů
uziv_pref_tmp	pomocná tabulka sloužící pro uložení mezivýsledků výpočtu uživatelských preferencí
zanr_podoba	tabulka sloužící k uložení informace o podobnosti dvou žánrů $(zanr_A, zanr_B, podoba_{AB})$

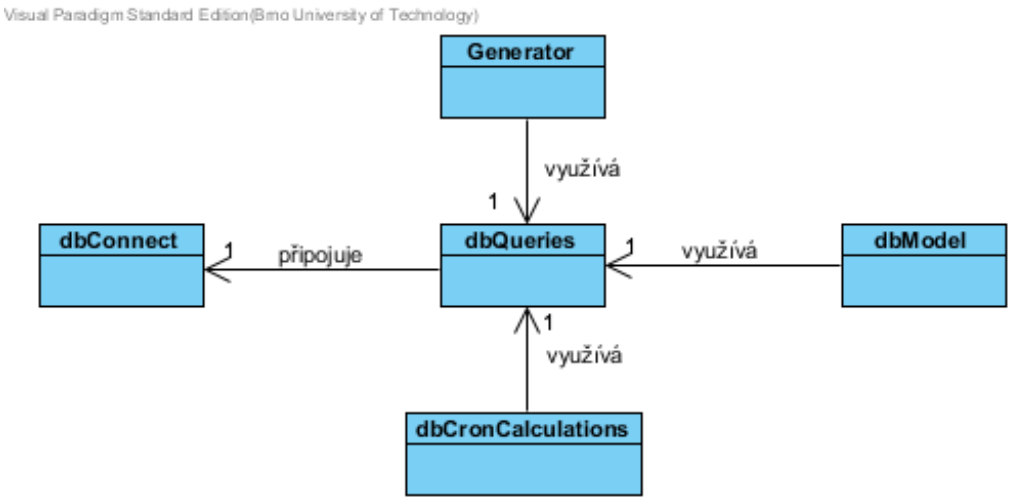
4.2.2 2. iterace - databázová vrstva

Přístup do databáze s reálnými daty mi byla umožněna až v pozdních obdobích vývoje. Ze schématu databáze, který odpovídá fragmentu celé databáze jsem vytvořil jeho obraz na lokální databázi. Pro samotné doporučení budu využívat tabulky *hodnocení*, *uživatel*, *kniha*, *žánr* (plus jejich vazební tabulky mezi vztahy m:n).

Primárním účelem této vrstvy bude obstarávat vnější požadavky na data z databáze, resp. o uložení dat do databáze. Jelikož se tato vrstva bude připojovat do databáze, která v ostré verzi bude na serveru cbdb.cz, tak musím rozčlenit tuto vrstvu tak, aby neovlivnila chod části databáze, kterou výsledný program nebude využívat. Na nejnižší úrovni vrstvy se bude nacházet inicializační soubor obsahující přihlašovací údaje do databáze. Tímto principem se můj doporučovací program odstiňuje od nutnosti mít přihlašovací údaje uvnitř zdrojových kódů.

Obrázek 4.4 zobrazuje existenci pěti objektů:

- dbConnect - singleton uchovávající si připojení do databáze
- dbQueries - zde budou přichystány všechny potřebné SQL dotazy
- dbModel - třída pro komunikaci ostatních tříd systému s databází
- dbCronCalculations - tato třída bude obstarávat offline předvýpočty
- Generator - třída užitá v prvotní fázi vývoje pro generování dat



Obrázek 4.4: Konceptuální diagram tříd databázové vrstvy

Třída **dbConnect** bude obstarávat propojení mého systému s databázovým serverem. Tato třída je vhodným kandidátem na využití návrhového vzoru *jedináček* (angl. singleton), který zajišťuje, že v systému se bude nacházet pouze jediná instance této třídy. Tímto přístupem snížíme počet vytvořených spojení do databáze na jedna. Pokud bude chtít nějaká třída použít služby databáze, dotáže se na tuto třídu a mohou nastat dvě situace: databázové spojení ještě nebylo vytvořené a tudíž neexistuje instance této třídy. Spojení se naváže s využitím přihlašovacích údajů vyskytujících se v inicializačním souboru. Následně se uloží do třídní proměnné odkaz na toto spojení. Druhou situací je skutečnost, že spojení již bylo v minulosti vytvořené a tedy stačí využít již existující spojení. Tedy jsem ukázal, že doopravdy využijeme návrhový vzor jedináčka.

Třída **dbQueries** bude obsluhovat veškeré SQL dotazy do databáze. Budou zde umístěny všechny potřebné SQL příkazy pro manipulaci s daty: Insert, Update, Delete, Truncate, Create a Drop table. Žádná jiná třída nebude mít možnost volat instanci dbConnect, tím zajistím, že veškeré operace nad databází budou prováděny metodami této třídy.

Třída **dbModel** bude mít na starosti komunikaci s doporučovací vrstvou a také bude provádět výpočty nad daty z databáze. Metody této třídy budou obstarávat plnění dvourozměrného pole ve tvaru

$$(uziv_i, kniha_j) = hodnoceni_{ij}$$

které se následně využije v doporučovací vrstvě pro výpočet doporučených knih. Dále bude třída obsluhovat transformaci databázových výsledků do polí, které budou opět využity v doporučovací vrstvě. Dále bude obsluhovat uložení doporučených knih do databáze. Před samotným uložením vymaže zastaralé doporučení pro konkrétního uživatele a uloží nový, aktuální seznam knih.

Lokální databázi poté bude potřeba naplnit daty. Pro tyto účely vytvořím třídu **Generator**, která bude po nasazení na databázi s ostrými daty smazána. Dočasný přístup je použitím náhodného generátoru, který bude vhodným způsobem simulovat data na serveru. Toto má ovšem svá úskalí, protože tímto postupem nelze vygenerovat žádná relevantní data, na kterých by se dala ověřit míra správnosti doporučení. Ovšem pro účely vývoje modulu toto ničemu nevadí. Nejprve je potřeba správně implementovat algoritmy, jejich syntax. Po přístupu do reálné databáze se poté algoritmy poupraví, aby lépe doporučovaly nad

reálnými daty.

Velikost ostré databáze vím, díky tomu si mohu na lokální databázi vytvořit její kopii. Pro účely vývoje a testování tuto databázi zvětším a to následovně:

- počet uživatelů 100 000 (nárůst 2.5x)
- počet knih 250 000 (nárůst 1.9x)
- počet hodnocení 2 500 000 (nárůst 2.5x)
- počet žánrů 20 (snížení 14x)

Na takto vygenerovaných datech poté bude probíhat samotná implementace mého doporučovacího systému. Velikost byla navýšena zhruba 2.5x, abych simuloval postupný nárůst, který nastává na reálné databázi.

Třída **dbCronCalculations** bude sloužit jako komunikační rozhraní pro CRON[3], který bude spouštěn v periodickém časovém intervalu. Výpočet doporučení seznamu knih pro konkrétního uživatele je komplexní a časově náročná operace, proto můj systém rozdělím tak, aby co nějvětší část složitých výpočtů byla prováděna s časovou periodou. Tato třída bude obstarávat výpočty dat, které mají vysokou časovou a paměťovou náročnost.

Periodu opakování těchto výpočtů zvolím na jeden týden. Beru v potaz fakt, že průměrně aktivní čtenář je schopen přečíst přibližně jednu až dvě knihy během jednoho týdne. Proto předvýpočty nemusí být každý den, či častěji, jak je tomu na různých internetových obchodech, kde stovky až tisíce uživatelů denodenně zanechají velké množství relevantních dat - umístění zboží do košíku, návštěva stránky, napsání recenze, ohodnocení produktu.

Doporučování budu provádět pouze pro tzv. *aktivní uživatele*. Jako takovéto označím všechny uživatele, kteří mají více než dvacet ohodnocených knih. Výběr tohoto čísla je inspirován z největšího doporučovacího knižního portálu GoodReads[4]. Díky tomuto omezení lze snížit počet jednotlivých porovnávání. V prvotní fázi vývoje, kdy jsem vyvíjel systém na náhodně vygenerovaných datech se z celkového počtu 100 000 uživatelů snížil počet na 65 207. Tedy téměř 35% snížení počtu uživatelů. V případě reálné databáze serveru cbdb z celkového počtu 38 027 uživatelů se počet snížil na 7 960. Tedy snížení o 79% původního počtu. Takto velké snížení může být způsobeno například tím, že spousta uživatelů se zaregistrovala, ale svůj účet nijak aktivně nevyužívají. Nelze tedy takovéto uživatele brát v potaz pro doporučování ostatním uživatelům.

$$genreSim_{A \to B} = \frac{|A|}{|A + B|} \tag{4.1a}$$

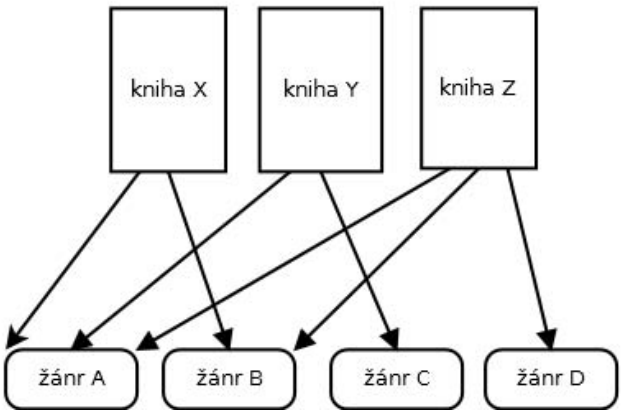
$$genreSim_{A \to B} = \frac{|A \wedge B|}{|A + B|} \tag{4.1b}$$

Další výpočet, který bude probíhat v časových intervalech je spočtení *podobnosti jednotlivých žánrů*. Vím, že každá kniha má jeden a více přiřazených žánrů. V situaci, kdy kniha má více než dva žánry, můžeme začít počítat podobnost jednotlivých žánrů. Pro všechny druhy žánrů a pro všechny knihy v databázi spočteme, kolikrát se konkrétní žánr A vyskytuje s ostatními žánry současně u stejné knihy. Mějme například knihy X, Y a Z.

[3]Cron je softwarový démon, který v operačních systémech automatizovaně spouští v určitý čas nějaký příkaz resp. proces (skript, program apod.), zdroj: http://cs.wikipedia.org/wiki/Cron

[4]http://www.goodreads.com/

První kniha X patří do žánru A, B. Druhá kniha Y patří do žánru A, C a třetí kniha Z patří do žánru A, B, D - grafickou ilustraci znázorňuje obrázek 4.5. Použitím jednoduchého vzorce 4.1a pro výpočet podobnosti žánru $A \rightarrow B$ dostaneme podobnost 66%. Použitím stejného vzorce na opačnou podobnost $B \rightarrow A$ ovšem dostaneme 100%. Jemnou úpravou tohoto vzorce dostaneme vzorec 4.1b, kde pro oba směry porovnání dostaneme hodnotu 40%. Z podstaty vzorce plyne, že maximální hodnota podobnosti je 50%. Jelikož pořadí prvků uspořádané množiny je neměnné vzhledem ke skalárnímu násobení všech prvků, tak nemusím provádět transformaci na interval 0-100%. Pro každý žánr si uložím tři žánry, se kterým má největší míru podoby.



Obrázek 4.5: Grafická ilustrace přiřazení knih X, Y a Z k žánrům A, B, C a D

Po dokončení výpočtu podobnosti žánru proběhne výpočet *uživatelských preferencí*. Jako preference označuji množinu:

$$(pref_1, pref_2, pref_3, user) \in Z \times Z \times Z \times U \tag{4.2}$$

Kde Z je množina žánrů a U je množina uživatelů v systému. Pro každého aktivního uživatele vypočítám tři žánry, které hodnotil nejčastěji. Pro všechna hodnocení, která uživatel zanesl do systému si zjistím, o kterou knihu se jednalo a pro každou takovouto knihu si poznačím s jakými žánry je spojena. Výsledný počet agreguji a seřadím. Z tohoto seznamu vezmu první tři žánry a uložím je. Tento přístup v určitých okolnostech může zvyšovat entropii doporučování. Mějme situaci, kdy pro uživatele F máme seřazenou posloupnost četnosti žánrů ve tvaru (žánr,počet výskytů): (C,15), (E,10), (A,6), (B,6), (D,3). Vidíme, že 2 žánry se ucházejí o třetí místo. Abych nezvýhodňoval pouze jeden žánr (např. ten, který abecedně předchází jiným) zavedu v tomto okamžiku náhodný výběr. Tuto vlastnost aplikuji i v případě, že je více uchazečů zároveň o druhé místo, respektive první místo. Důsledkem této vlastnosti je fakt, že může docházet k doporučení větší variace knih pro konkrétního uživatele, při zachování stejného počtu hodnocení.

Připomeňme, že základem doporučovacího systému je výpočet podobnosti všech dvojic uživatelů. Abychom tohoto dosáhli, museli bychom na ostré databázi provést téměř $40\,000^2$ porovnání. Pokud bychom provedli výpočet pouze nad všemi aktivními uživateli, dostali bychom se přibližně na počet $8\,000^2$ porovnání. Touto vlastností jsme ušetřili 64% porovnání. Ovšem $8\,000^2$ je stále vysoké číslo a s rostoucí databází počet porovnání kvadraticky poroste. Proto pro výběr dvojic uživatelů, které budu porovnávat s uživatelem F za účelem zjištění jejich podobnosti využiji jejich preference a to následovně: porovnám všechny

aktivní uživatele, kteří se shodují v první preferenci (tedy nejčetěnší uživatelův žánr) a zároveň se musí shodovat alespoň v jedné ze zbylých preferencí. Toto se dá formálně zapsat následovně:

$$sim_F = \{b \mid b \in U \wedge F.p_1 = b.p_1 \ \wedge \ (F.p_2 \in (b.p_2, b.p_3) \ \vee \ F.p_3 \in (b.p_2, b.p_3))\} \tag{4.3}$$

Kde p_1, p_2, p_3 jsou uživatelské preference popsané v 4.2. Využitím této vlastnosti dostávám v průměru pro každého uživatele 120 jiných uživatelů. V tomto okamžiku se tedy dostáváme na úroveň $8\ 000 * 120 = 960\ 000$ párů porovnání. Ve srovnání s prvotním naivním porovnáváním všemi se všemi dostáváme pouze 0.06% porovnání. S tímto přístupem se nyní dostáváme na lineární složitost $O(nm)$, namísto předchozí kvadratické složitosti.

Pokud víme, které uživatele budeme vzájemně porovnávat, jakým způsobem uložit tuto informaci? V tomto okamžiku se nabízí dvě možnosti přístupu k následujícímu problému. Prvním řešením je v rámci předvýpočtů ukládat pro všechny páry porovnání do databáze trojici:

$$(uzivatel_1, uzivatel_2, podobnost) \tag{4.4}$$

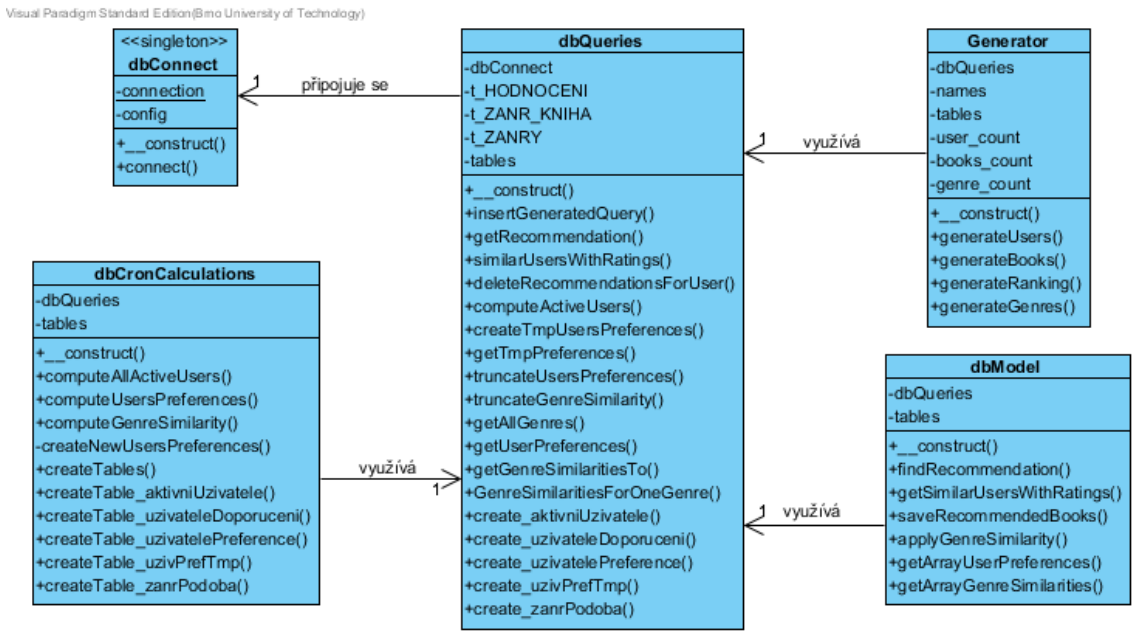
Toto řešení má výhodu v tom, že pokud budem dělat samotné doporučení pro $uzivatel_1$, dotážeme se do databáze na seřazený seznam N-nejpodobnějších uživatelů a tyto následně použijeme pro výpočet doporučených knih. Nevýhodou tohoto přístupu je velikost dat v databázi, která časem bude narůstat. Druhým řešením, pro které jsem se rozhodl já, je výpočet podobnosti uživatelů až v momentě, kdy $uzivatel_1$ žádá o zobrazení samotného doporučení. Výhododou tohoto přístupu je snížení nároků na úložný prostor. Na druhou stranu se zvýší výroky na početní výkon. V naivním přístupu by pro každý dotaz uživatele pro zobrazení doporučení proběhl výběr N uživatelů a následně by byl vypočítán seznam doporučených knih. Toto je ovšem velmi nepraktické a pokud by v jednom okamžiku žádalo větší množství různých uživatelů současně o zobrazení doporučení, zvýšil bych tím zátěž na výpočetní kapacitu serveru. Proto jsem se rozhodl o poupravení tohoto přístupu následovně:

Uvažujme situaci, kdy $uzivatel_1$ ještě nikdy nežádal o zobrazení doporučení. Zašle dotaz pro výpis doporučených knih. Výpočet doporučení knih popisuje odstavec Online výpočet (viz 4.2.3). Po výpočtu doporučených knih pro $uzivatel_1$ se seznamu uloží do databáze a zároveň se tato data vypíšou na výstup, kde si je už může $uzivatel_1$ přečíst a zvolit nějakou knihu, kterou si přečte. Pokud $uzivatel_1$ zažádá opět o zobrazení doporučených knih, již nedochází k žádnému výpočtu, ale pouze se program dotáže do databáze, kde je uložený seznam doporučení a pouze dojde k výpisu. Tímto přístupem minimalizuji počet výpočtů na opravdové minimum. Nyní je potřeba ještě brát v úvahu fakt, že každý týden bude probíhat předvýpočet dat. Tudíž je potřeba uložená doporučená data každý týden zneplatnit. Když uložím vypočtené doporučené knihy, připojím k těmto datům i informaci s datem vytvoření. Pokud se $uzivatel_1$ dotáže na doporučené knihy a data v databázi budou starší než sedm dní, proběhne výše popsaný výpočet znovu.

Implementace

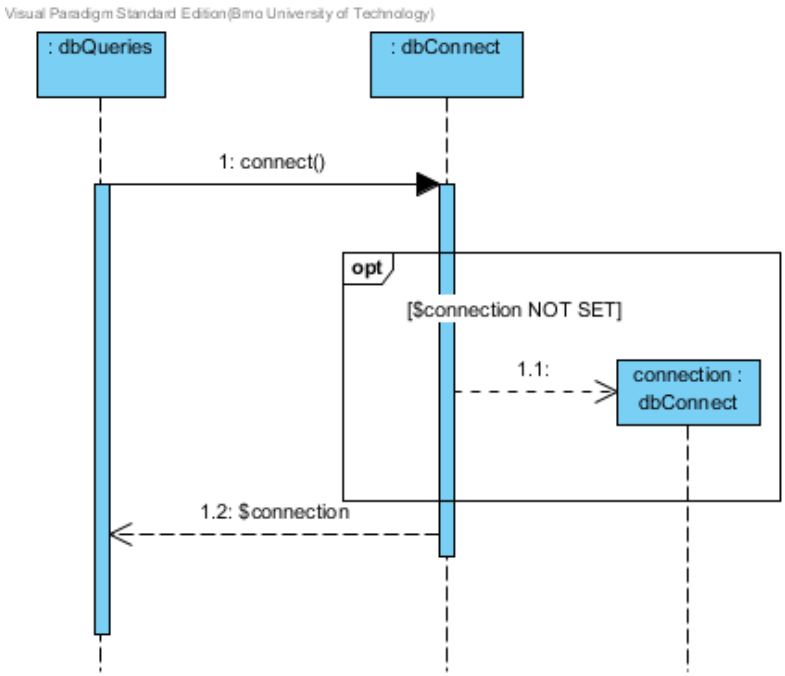
Během této iterace byla implementována databázová vrstva. Schéma této vrstvy zachycuje obrázek 4.6.

Na nejnižší úrovni se nachází třída **dbConnection**. Tato má na starosti veškerou komunikaci mezi doporučovacím systémem a databázovým serverem. Podle návrhu je implementována jako jedináček (návrhový vzor *singleton*). Díky této vlastnosti existuje během života programu pouze jediná instance této třídy a tím zajistíme, že existuje pouze jediné



Obrázek 4.6: Diagram tříd první iterace

spojení do databáze. Pokud by nebyl využit jedináček a instance třídy by se vytvářela při každém volání SQL dotazu. To znamená, že při každém volání by se vytvářelo nové spojení do databáze, čímž by se navýšila režie. Vznik objektu zachycuje diagram sekvence na obrázku 4.7.



Obrázek 4.7: Sekvenční diagram reprezentující práci s jedináčkem

Další implementovanou třídou v této fázi byla **dbQueries**. Cílem bylo, aby se všechny SQL dotazy, které bude systém využívat, nacházely na jediném místě kódu. K tomuto právě slouží tato třída. Jelikož celkový doporučovací systém bude sloužit jako zásuvný modul, je potřeba, aby systém byl schopný jednoduše provést instalaci = vytvoření nových tabulek, které se na serveru ještě nenacházejí. Seznam nových tabulek zanesených do systému popisuje kapitola 4.2.1. Aby byl systém pružný, všechny názvy tabulek jsou uchovávány v třídních proměných. Konstruktor třídy se postará o naplnění správných jmen těchto tabulek. Tato třída je implementována tak, aby nedošlo k zápisu do existující tabulky serveru cbdb, ale jenom pouze ke čtení z těchto tabulek. Tím odpadají starosti s nechtěným, či neoprávněným zápisem (resp. smazáním dat). Tato třída může zapisovat pouze do tabulek, které jsou nově vytvořeny speciálně pro doporučovací systém. Tato třída slouží jako prostředník, kdy je v systému požadavek na databázová data, třída požadavek zpracuje, vytvoří SQL dotaz, s využitím databázového spojení v třídě **dbConnection** provede dotaz a výsledná data vrací zpět do systému bez jakékoliv úpravy. O tuto úpravu se již postará funkce, která požadovala data.

Nejdůležitější třídou této iterace je **dbCronCalculations**. Tato třída implementuje offline zpracování dat doporučovacího systému tak, jak jej popisuji v kapitole 4.2.2. Cílem této třídy bylo, aby co největší počet výpočetně náročných operací proběhl v jednom okamžiku a vypočítaná data se zaznamenala do databáze. Tato data pak následně využije doporučovací vrstva. Časově a prostorově nejnáročnějším výpočtem je tvorba uživatelských preferencí, o toto se stará metoda `computeUsersPreferences()`. Nejdříve je zavolána metoda `createTmpUsersPreferences()` třídy **dbQueries**, která obstará naplnění dočasné tabulky preferencí a to následovně: tabulku uživatelů spojí s tabulkou hodnocení a následně na každé toto hodnocení připojí tabulku knih. V tomto okamžiku připojí tabulku žánrů na každou knihu. Nyní jednoduchou agregační funkcí nad uživateli a žánry zjistím počet, kolikrát se pro daného uživatele vyskytuje daný žánr. Tuto posloupnost pro každého uživatele seřadím a uložím ji do tabulky *uziv_pref_tmp*. Nyní metoda vyprázdní tabulku *uzivatele_preference* a následně zavolá metodu `createNewUsersPreferences()`. Tato metoda pro každého uživatele vybere tři nejčtenější žánry a pro každý z nich vypočte jejich podíl v celku. Tyto informace uloží do tabulky *uzivatele_preference* ve tvaru:

$$(pref_1, share_1, pref_2, share_2, pref_3, share_3, user) \in Z \times \mathbb{R}^+ \times Z \times \mathbb{R}^+ \times Z \times \mathbb{R}^+ \times U$$

Tyto předvypočtené hodnoty jsou následovně využívany při doporučování. Jednotlivé preference využívá doporučovací systém pro párování uživatelů za účelem zjištění jejich podobnosti. A jednotlivé podíly žánrů jsou poté přičteny ke skóre doporučených knih. Tento princip je podrobně popsán v 4.2.3.

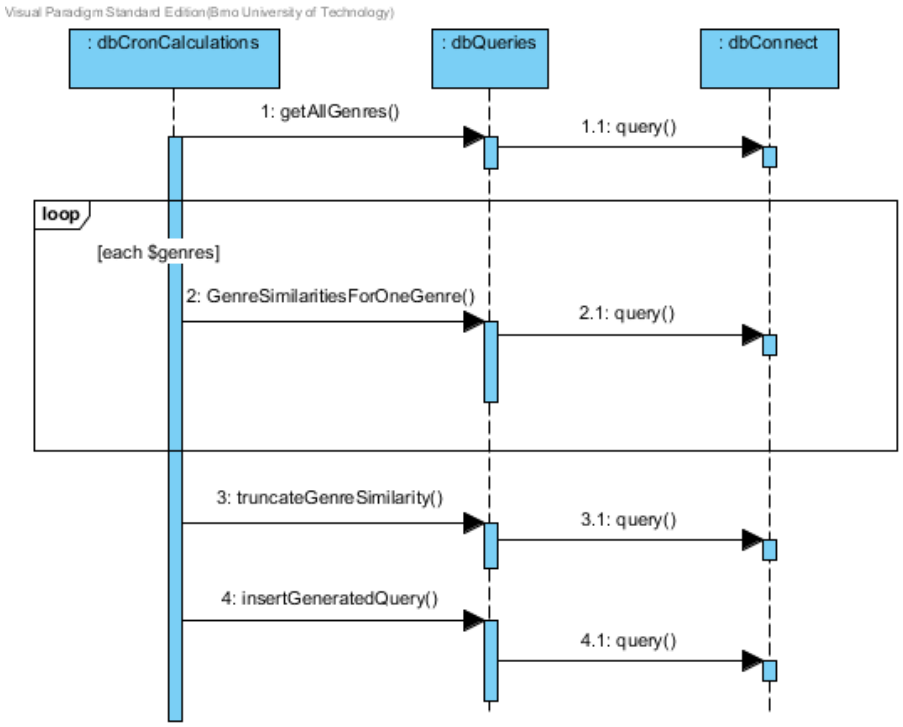
Druhou důležitou metodou této třídy je výpočet podobnosti žánru. Tato metoda vezme seznam všech žánrů z databáze a postupně pak hledá všechny dvojice žánrů, kdy se oba tyto žánry vyskytují u jedné knihy současně. Následně jě vypočtena podobnost žánru přesně tak, jak je navržen vzore v kapitole 4.2.2. Pro každý žánr uložím tři žánry, se kterými je nejpodobnější. Toto skóre je uloženo do databáze a princip využití je popsán v 4.2.3.

Tělo metody je zachyceno na obrázku 4.8. Aby byl minimalizován počet INSERT dotazů do databáze, je do proměnné `$query` kumulován výsledek a po dokončení veškerého výpočtu je proveden INSERT. Tímto způsobem pošleme do databáze pouze jeden dotaz, který provede vložení více řádků najednou. Díky tomuto přístupu snížíme časovou náročnost režie. Sekvenční diagram této metody je zachycen na obrázku 4.9.

Další implementovanou třídou v této iteraci je **dbModel**. Primárním účelem této třídy je transformovat data z databáze do použitelného formátu pro výpočty - často se jedná

```php
public function computeGenreSimilarity() {
        $allGenres = $this->dbQueries->getAllGenres();
        $query = "";
        while ($genres = $allGenres->fetch_assoc()) {
            $genre_id = $genres["id"];
            $oneGenre = $this->dbQueries->GenreSimilaritiesForOneGenre($genre_id);
            while ($genre = $oneGenre->fetch_assoc()) {
                $query.="($genre_id, $genre[kid], $genre[podil]),";
            }
        }
        $this->dbQueries->truncateGenreSimilarity();
        $this->dbQueries->insertGeneratedQuery($this->tables->t_zanr_podoba, $query);
    }
```

Obrázek 4.8: kód metody pro výpočet podobnosti žánrů



Obrázek 4.9: Sekvenční diagram zachycující získání doporučení pro uživatele

o dvourozměrná pole. Třída obsahuje dvě důležité metody `findRecommendation($id)` a `getSimilarUsersWithRatings($id)`. První zmíněná metoda má na starost zjištění, zda se v databázi vyskytuje již spočtené doporučení pro konkrétního uživatele se zadaným identifikačním číslem. Pokud ano, toto doporučení nesmí být starší než sedm dní (jelikož perioda offline výpočtů je právě sedm dní). Pokud vše proběhne v pořádku, je naplněno pole s informacemi o knize (id, jméno, url, autor, atd.) a tyto informace jsou poté předány na výstup. Takto vytvořené pole je následně v cyklu zpracované a jednotlivé knihy jsou zobrazeny uži-

vateli. Pokud ovšem doporučení neexistuje, či je doporučení zastaralé, je vráceno prázdné pole. To signalizuje doporučovací vrstvě, že je potřeba provést výpočet doporučených knih. Druhou zmíněnou metodou je funkce, která pro daného uživatele z databáze vybere kandidátní uživatele, dle vzorce uvedeného v rovnici 4.3. Následně pro všechny takovéto uživatele vytvoří trojrozměrné pole, kde první dimenzí je identifikační číslo uživatele, druhou dimenzi tvoří identifikační číslo knihy a třetí dimenzi tvoří informace o hodnocení dané knihy daným uživatelem a také jsou zde uloženy identifikátory žánrů dané knihy. Takto komplexní informace je poté předána na výstup, kde jej zpracuje doporučovací vrstva.

Pro každého aktivního uživatele se v systému udržují informace o jeho preferencích = tři žánry, které uživatel hodnotil nejčastěji (resp. knihy spadajících do těchto žánrů). Ke každému z těchto tří žánrů je také uložená informace, jaký poměrový podíl tento žánr zastupuje v celku všech ostatních hodnocených žánrů. Tato hodnota je poté použita k vylepšení skóre doporučených knih. Do vylepšení skóre se také započítává i skóre podobnosti žánrů, které jsou v páru s uživatelovými preferencemi. Jelikož tyto dvě hodnoty (poměrový podíl preferencí a skóre podobnosti dvou žánrů) nejsou normalizovány, nelze potom tyto hodnoty bez úpravy přičíst ke skóre doporučené knihy. Uživatelské preference musí mít větší váhu ve vylepšení skóre než žánry, které jsou podobné. K tomuto výpočtu slouží metoda `getArrayGenreSimilarities()` třídy **dbModel**. Podrobnější popis principu vylepšení skóre je popsán v 4.2.3.

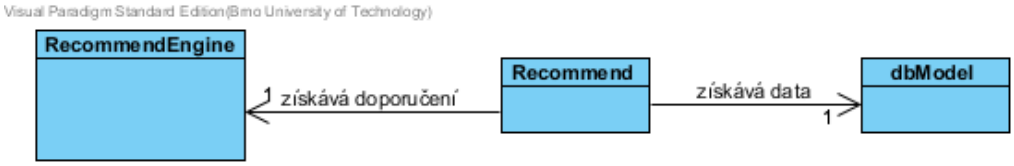
Čtvrtou implementovanou třídou v této iteraci je **Generator**. Jak již názevy vypovídá, jde o třídu, která má na starosti vytvoření dat. Tato data byla použita pouze v prvotní fázi vývoje. Tvorba této třídy byla nutná, jelikož přístup k ostrým datům serveru cbdb.cz mi byl umožněn až v pozdější době vývoje. Tento fakt měl jedno pozitivum a to, že jsem si mohl naddimenzovat databázi dat, abych tím nasimuloval nárust, ke kterému časem dojde. Velikost původní databáze mi byla známa již v začátcích vývoje, proto jsem mohl na základě této informace navýšit počet - navýšení dat jsem provedl přibližně 2.5x. Generování uživatelů a jejich jmen obstarala metoda `generateUsers()`, generování knih obstarala metoda `generateBooks()`. V ostré databázi se nachází 280 různých žánrů. Pro generování náhodných čísel používám vestavěnou PHP funkci `rand()`, která má rovnoměrné rozložení vygenerovaných čísel. Tento fakt má úskalí v tom, že kdybych ke knihám tímto způsobem generoval náhodně žánry, dostal bych také rovnoměrné pokrytí všech žánrů, což ovšem ve skutečnosti nenastává. Některé okrajové žánry mají mnohem menší četnost výskytu, než jiné. Proto jsem se rozhodl pro testovací databázi vygenerovat pouze dvacet žánrů. Toto generování obstará metoda `generateGenres()`. Posledním úkolem této třídy je generování hodnocení knih - to provede metoda `generateRanking()`. Pro všechny vygenerované uživatele náhodně vyberu číslo v intervalu $< 5; 50 >$. Toto bude značit, kolik daný uživatel bude mít hodnocení. Pro všechna tato hodnocení náhodně vyberu knihu, a poté pro tuto knihu vygeneruji hodnocení v intervalu $< 0; 5 >$. Následně tuto trojici uložím do databáze. Tato třída byla smazána před implementací na server cbdb.cz.

4.2.3 2. iterace - doporučovací vrstva

Celé jádro doporučování se nachází v této vrstvě. Základním požadavkem této práce je jednoduché rozhraní mezi vnějším systémem - webovým serverem cbdb - a mým systémem pro doporučování. K tomuto bude sloužit jednoduché volání třídy **Recommend** s parametrem identifikátoru uživatele a toto volání se již postará o veškerou režii, volání a výpočty. Doporučovací systém bude mít dvě fáze: **offline výpočet** a **online výpočet**. První fáze bude provádět časově náročnější výpočty, které se budou provádět v pravidelných časových

intervalech (například jednou týdně ve 2 ráno, kdy na serveru není vysoká návštěvnost). Popis výpočtů probíhajících v offline fázi zajišťuje třída **dbCronCalculations** - výpočet aktivních uživatelů, uživatelské preference a podobnost žánrů. Výpočty jsou popsané v kapitole 4.2.2.

Obrázek 4.10 zobrazuje závislosti mezi třídami **Recommend**, **RecommendEngine** a **dbModel**



Obrázek 4.10: Diagram tříd druhé iterace, s vazbou na třídu dbModel z první iterace

Online výpočet: Výpočet doporučení bude probíhat po volání funkce s identifikačním číslem konkrétního uživatele. Doporučovací systém v tomto okamžiku vybere X uživatelů podle principu podobnosti preferencí a mezi těmito páry spočítá podobnost uživatelů. Míru této podobnosti si uloží pro následné váhování doporučených knih. Doporučení knih mezi dvěma uživateli potom probíhá tak, že se uživateli1 doporučí všechny knihy $uzivatel_2$, které sám $uzivatel_1$ nehodnotil. Ohodnocení takovéhoto doporučení je počítáno jako hodnocení knihy $uzivatel_2$ vynásobenou hodnotou míry podobnosti těchto dvou uživatelů. Pokud je nějaká kniha doporučená od více uživatelů současně, bere se výsledné skóre jako průměrná hodnota.

Výpočet podobnosti

Pro výpočet podobnosti dvou uživatelů A, B jsem v prvotní fáze vývoje systému používal rovnici následující rovnici převzatou z [16]:

$$sim_{A \to B} = \frac{1}{1 + \sqrt{\sum_{i \in N} (r_{Ai} - r_{Bi})^2}} \tag{4.5}$$

kde r_{Ai} je hodnota hodnocení uživatele A knihy i. Na náhodně vygenerované databázi ovšem nešlo nijak ověřit, zda takto počítaná podobnost je vhodná či nikoliv. Po nasazení systému na reálná data se ovšem ukázalo, že doporučení je nepoužitelné. Proto jsem pro výpočet podobnosti použil kosinovu podobnost dvou vektorů:

$$sim_{A \to B} = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} * \sqrt{\sum_{i=1}^{n} {B_i}^2}} \tag{4.6}$$

kde A_i je hodnota hodnocení knihy i uživatelem A. Pro takto počítanou podobnost uživatelů již doporučovací systém byl schopný dávat lepší výsledky. Doporučovací skóre bylo na škále 1-5 (shodné s uživatelským hodnocením knih). Při výpočtu podobnosti pomocí kosinovy podobnosti se do popředí dostávaly knihy, které žánrově souhlasily s uživatelovými preferencemi. Při použití rovnice 4.5 se v popředí vyskytovali nesouvisející knihy.

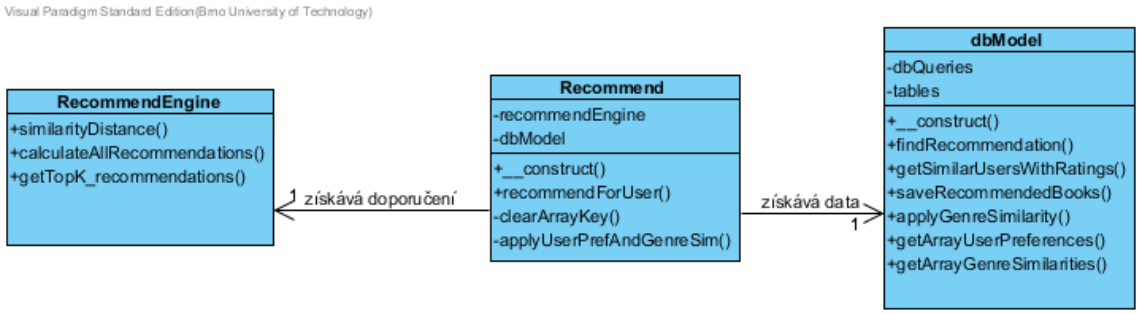
Využití uživatelských preferencí a podobnosti žánrů

Výpočet doporučených knih založený pouze na podobnosti uživatelů a jejich hodnocení by řadil doporučovací systém do kategorie kolaborativního filtrování. Abych docílil zvýšení

přesnosti doporučení, použiji informace, které jsem získal pouze o daném konkrétním uživateli - tím do systému zanesu vlastnosti filtrování založeného na obsahu. O každém uživateli budu mít informace o jeho preferencích = tři žánry, které hodnotil nejčastěji (resp. knihy navázané na tyto žánry). S využitím této informace můžu zvýhodnit doporučené knihy, které spadají do jednoho (a více) z těchto žánrů. Další zvýhodnění doporučených knih můžu provést v rámci podobných žánrů. Je jasné, že například žánry (scifi, fantasy) budou mít větší míru podoby než (scifi, ekonomie). Díky těmto zvýhodněním dostanu lepší skóre pro relevantní knihy. Doporučovací systém nyní využívá jak principů kolaborativního filtrování tak i filtrování založeného na obsahu a tím spadá do kategorie hybridních systémů.

Implementace

V další iteraci vývoje byla implementována vrstva doporučovacího systému. Schéma této iterace je zobrazeno v 4.11.



Obrázek 4.11: Konceptuální diagram tříd druhé iterace, s vazbou na třídu dbModel z první iterace

Třída **Recommend** obstarává veškerou komunikaci mezi výpočetním jádrem doporučování a databázovou vrstvou. Připomeňme fakt, že základním požadavkem celého systému je jednoduché ovládání z vnějšku systému. Pokud bude chtít konkrétní uživatel zobrazit seznam doporučených knih, otevře si ve webovém prohlížeči stránku s výpisem doporučených knih. V tomto okamžiku server cbdb.cz zpracuje požadavek, že uživatel žádá o doporučení. Zjistí jeho identifikační číslo a zavolá metodu `recommendForUser($id)`. Diagram sekvence na obrázku 4.13 zachycuje posloupnost příkazů, které vedou ke získání seznamu doporučených knih. Kód této metody je zobrazen na obrázku 4.12. Tato metoda nejdříve ze všeho ověří, zda se uživatel s daným identifikačním číslem nachází mezi aktivními uživateli - tedy, jestli ohodnotil více než dvacet knih. Pokud ne, je navráceno prázdné pole signalizující, že seznam doporučených knih je prázdný. Pokud ano, výpočt probíhá dále: metoda se nejdříve pošle požadavek do databáze, aby došlo k ověření, zda se tam již nachází vypočítaný seznam knih. Pokud ano, je tento seznam navrácen serveru cbdb.cz k výpisu. V případě, že se takovýto seznam nenachází v db (či je starší více než sedm dní) dojde k výpočtu nového seznamu. Nejdříve se zavolá metoda `getSimilarUsersWithRatings()` třídy **dbModel**, tato vrací trojrozměrné pole, které je následně předloženo doporučovacímu jádru. V tomto okamžiku proběhne výpočet nad tímto polem (výpočet je podrobně popsán v následujícím odstavci). Výsledkem tohoto doporučení je dvourozměrné pole, kde první dimenze určuje identifikační číslo doporučené knihy a druhá dimenze obsahuje informace o skóre doporučení této knihy a také její žánry. Následně jsou využity uživatelské preference a podobnosti žánrů na zvýšení skóre doporučení některých knih (detailní popis aplikace je

popsán v posledním odstavci této iterace). Následně je tento seznam s vylepšeným skórem sestupně seřazen a následně je vybráno K nejlepších doporučení. K je třídní proměnná, která se nastavuje v konstruktoru třídy. V předposledním kroku výpočtu je tento seznam K nejlepších doporučení uložen do databáze, včetně aktuálního časového razítka. Na závěr je tento seznam navrácen serveru cbdb.cz k výpisu. V tomto okamžiku vidíme, že z vnějšku server nepozná, zda data byla již vypočítána a pouze vypsána z databáze, a nebo se data musela dopočítat.

```php
public function recommendForUser($id) {
        $recommendedResult = array();
        if ($this->dbModel->isUserActive($id)) {
            $recommendation = $this->dbModel->findRecommendation($id);

            if (!$recommendation) {
                $similarUsersRatings = $this->dbModel->getSimilarUsersWithRatings($id);
                $allRecommendedBooks = $this->recommendEngine->calculateAllRecommendations($similarUsersRatings, "userId$id");

                $userPreference = $this->dbModel->getArrayUserPreferences($id);
                $genreSimilarities = $this->dbModel->getArrayGenreSimilarities($userPreference);
                $appliedPrefAndSim = $this->applyUserPrefAndGenreSim($allRecommendedBooks, $userPreference, $genreSimilarities);

                $topK_recommendedBooks = $this->recommendEngine->getTopK_recommendations($appliedPrefAndSim, $this->topK);
                $this->dbModel->saveRecommendedBooks($topK_recommendedBooks, $id);

                $recommendedResult = $this->dbModel->findRecommendation($id);
            } else {
                $recommendedResult = $recommendation;
            }
        }
        return $recommendedResult;
    }
```

Obrázek 4.12: kód metody pro zjištění seznamu doporučených knih pro uživatele $id

Díky tomu, že vypočítaný seznam doporučených knih je ukládan do databáze a jeho výpočet probíhá nejvíce nanejvýš jednou týdně, jsem server odlehčil od častého provádění složitých výpočtů.

Nejdůležitější třídou této iterace a jádrem celého doporučovacího systému je **RecommendEngine**. Metody této třídy počítají seznam doporučených knih pro konkrétního uživatele. Hlavní metodou, která obstarává výpočet doporučení je `calculateAllRecommendations()`. Má dva vstupní parametry: identifikační číslo uživatele, pro kterého budeme dělat doporučení - označme jej $uzivatel_1$; a trojrozměrné pole dat, které připraví třída **Recommend** zmíněná v předešlém odstavci. Výpočet probíhá tak, že každý kandidátní uživatel vhodný pro doporučení je porovnán s $uzivatel_1$. Porovnání uživatelů probíhá na základě podobnosti jejich hodnocení. Mějme například $uzivatel2$, který patří do kandidátní množiny uživatelů. V prvotní fázi vývoje, kdy jsem neměl přístup k reálným datům, jsem využíval rovnici 4.5. Nevýhody tohoto přístupu se projevily až v pozdní fázi vývoje, kdy jsem měl přístup k reálným datům. Zcela zásádním problémem bylo, že v seřazeném poli doporučených knih bylo skóre v průměru u 85% knih na nejvyšší hodnotě. Tedy pro seznam tvořený 500 knihy bylo 425 knih s nejvyšším možným skórem. Z tohoto důvodu jsem se rozhodl počítat pomocí kosinovy podobnosti - viz 4.6. V tomto okamžiku se již počet nejvýše ohodnocených knih razantně snížil v průměru na 15% z celkového počtu knih. A zároveň se nyní v popředí objevily i knihy žánrově odpovídající preferencím $uzivatel_1$, což byla pozitivní vlastnost. Po výpočtu podobnosti dvou uživatelů se každé hodnocení knihy, kterou hodnotil $uzivatel_2$

a zároveň tuto knihu $uzivatel_1$ nehodnotil, vynásobí mírou podobnosti těchto uživatelů.

Pokud například $uzivatel_2$ ohodnotil knihu i 4 body, a zároveň $uzivatel_1$ tuto knihu nehodnotil, vynásobím toto hodnocení mírou podobnosti těchto dvou uživatelů. Pokud je míra podobnosti například 0.85, předpovím, že $uzivatel_1$ by mohl ohodnotit knihu i známkou $4 * 0.85 = 3.4$ bodů. Pokud se najde více kandidátních uživatelů, kteří také hodnotili knihu i, předpovím skóre následovně:

$$skore_i = \frac{\sum_{b \in N} sim_{uzivatel_1 \to b} * r_{ib}}{|N|} \tag{4.7}$$

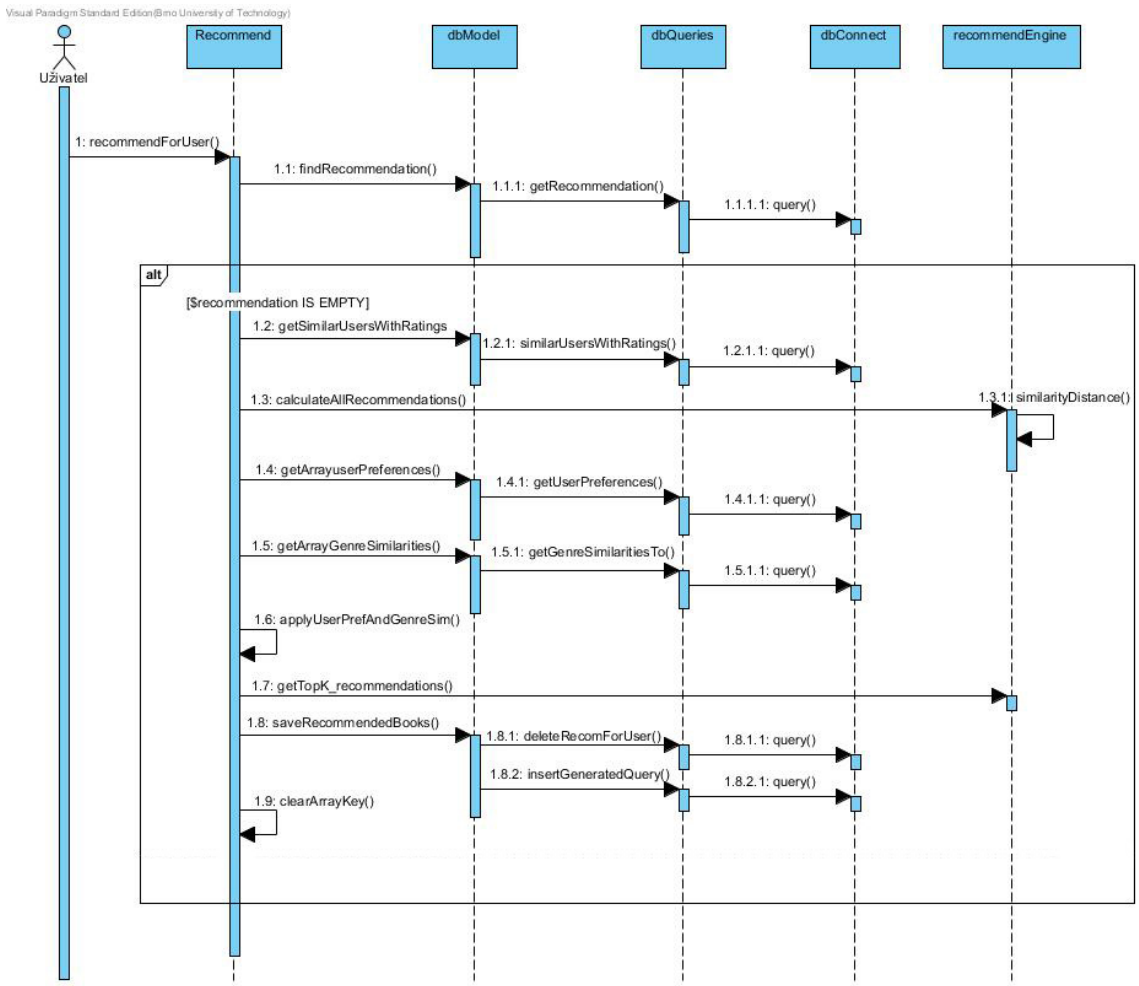
Kde N je množina kandidátních uživatelů pro porovnání s $uzivatel_1$ popsána vzorcem 4.3. Dále $sim_{uzivatel_1 \to b}$ je výpočet kosinovy podobnosti dvou uživatelů popsány rovnicí 4.6. Symbol r_{ib} vyjadřuje hodnocení knihy i uživatelem b. Po porovnání všech uživatelů s $uzivatel_1$ dostanu ve výsledku dvourozměrné pole, kde v první dimenzi mám uložené identifikační číslo knihy a druhá dimenze uchovává předpovězené skóre této knihy a zároveň všechny její žánry. Informaci o žánrech ukládám, jelikož v dalším kroku výpočtu se tyto žánry využijí na vylepšení skóre pomocí uživatelských preferencí a podobnosti zánrů.

Využití uživatelských preferencí a podobnosti žánrů

V první fázi výpočtu doporučených knih vstupují do rovnice pouze podobní uživatelé a jejich hodnocení. Výstupem je seřazená posloupnost knih podle skóre předpovědi. Toto skóre je v rozmezí 1-5. Ve druhé fázi do výpočtu vstupují uživatelovy preference. U každého uživatele si uchovávám tři jeho nejčtenější žánry, včetně poměrového podílu. Pro všechny knihy v seznamu doporučení zkontroluji jejich žánr. Pokud tento žánr souhlasí s jednou ze tří uživatelových preferencí, přičtu ke skóre knihy poměrový podíl tohoto žánru. Tedy například pokud $uzivatel1$ má zastoupení žánru sci-fi ve svém hodnocení s 15% podílem a zároveň je mu doporučovaná kniha v žánru sci-fi, tak ke skóre této knihy přičtu 0.15 bodů. Tímto způsobem se do předních příček seřazeného seznamu dostanou knihy se žánrem, které odpovídají uživatelovým prefrencím. Ve třetí fázi ještě ke skóre přičítám míru podobnosti dvou žánrů. Podobnost žánrů je počítána v offline režimu. Mějme například zmíněného $uzivatel1$ s oblíbeným žánrem sci-fi. Z databáze tedy získáme podobné žánry ke sci-fi. Dostaneme například žánr fantasy a míru podobnosti 0.2 bodů. Nyní rozlišuji dva případy: míra podobnosti žánrů je menší než míra uživatelovy preference, potom míru podobnosti žánrů přičtu bez úpravy. Ve druhém případě, pokud je míra podbnosti žánrů větší než míra uživatelovy preference, dělím míru podobnosti žánrů deseti (posouvám desetinnou čárku doleva), dokud toto číslo není menší než míra uživatelovy preference. Tímto zajistím, že podobnost žánrů bude hrát menší roli na doporučení, než oblíbené uživatelovy žánry. Po těchto fázích se na prvních příčkách doporučených knih budou pohybovat ty knihy, které jsou shodné s uživatelovými preferencemi a ty, které mají žánry podobné k uživatelovým preferencím.

4.2.4 4. iterace

Ve čtvrté, závěrečné iteraci probíhala revize kódu, refaktorizace a užití technik vkládání závislostí (anglicky dependency injection). Jelikož výsledné kódy budou implementovány do již existující aplikace a budou pod správou jiného programátora, snažil jsem se snažil kód dělit do logických vrstev, názvy funkcí vytvářet tak, aby na první pohled bylo jasné co provádí. Toto ilustruje ukázka kódu na obrázku 4.12, kde funkce, která obstarává zjištění seznamu doporučených knih, sama nic nepočítá, pouze deleguje patřičnou práci na jiné



Obrázek 4.13: Sekvence příkazů pro výpočet doporučených knih

metody různých tříd. Dále bylo potřeba odstranit všechny nepotřebné blokové a řádkové komentáře a jejich význam nahradit lepším pojmenováním proměnných a funkcí.

Vkládání závislostí

Vkládání závislostí je technika využívaná v objektově orientovaných jazycích pro vkládání závislostí mezi jednotlivými komponentami vyvíjeného programu tak, aby jednotlivé komponenty mohly využívat metody jiných komponent, aniž by měly na ně referenci v době sestavování programu. Vkládání závislostí je technika založená na principu návrhového vzoru *obrácení řízení* (anglicky inversion of control). Pokud programujeme bez VZ a náš objekt chce používat metody jiného objektu, pak zodpovídá za celý životní cyklus takovéhoto objektu (inicializace, destrukce, atd). Pokud ovšem využijeme vkládání závislostí, potom je náš objekt odstíněn od této správy. Pokud bude chtít využívat metody jiného objektu, bude objekt předpokládat existenci takovéhoto objektu a následně bude volat jeho třídy. Aby tento přístup mohl správně fungovat, je zapotřebí mít v systému nový objekt - poskytovatel - který zajistí, že všechny objekty budou mít reference na jiné požadované objekty.

Autorem tohoto přístupu je Martin Fowler, který ve svém článku [8] představil vkládání závislostí. Rozlišuje zde také tři různé přístupy, jak objektu předat referenci jiného objektu:

1. Vkládání **rozhraním** - externí modul, který je do objektu přidán, implementuje rozhraní, jež objekt očekává v době sestavení programu
2. Vkládání setter **metodou** - objekt má setter metodu, pomocí níž lze závislot vložit
3. Injekce **konstruktorem** - závislost je do objektu vložena v parametru konstruktoru již při jeho vytvoření

V mém modulu jsem se rozhodl pro využití třetí možnosti, vkládání konstruktorem. Cílem bude tedy ze všech tříd, které využívají metody jiných tříd, odstranit příkaz `new className();` a konstruktor jednotlivych tříd přeprogramovat tak, aby mohly přijímat reference na potřebné objekty. Dále bude potřeba vytvořit novou třídu, která bude zastupovat roli poskytovatele. Tato třída bude vytvářet nové objekty a v konstruktoru jim předávat všechny potřebné závislosti. Tuto úpravu popisují následující odstavce.

Implementace

Poslední novou třídou mého doporučovacího modulu je **Factory**. Tato třída bude mít za úkol tvorbu nových objektů a veškerých potřebných závislostí, které tyto objekty budou vyžadovat. Všechny dosavadní třídy, které ve svém těle vytvářejí instance jiných tříd poupravím následovně: všechny příkazy typu `new className()` odstraním; konstruktor třídy upravím tak, aby parametry volání byly právě odkazy na tyto instance objektů. Toto ilustruje obrázek 4.14. Na obrázku 4.16 vidíme, že třída **Recommend** má asociace na dvě třídy: **RecommendEngine** a **dbModel**. Pokud tedy v programu budeme chtít vytvořit instanci třídy **Recommend**, zavoláme funkci `newRecommend()`, která obstará tvorbu veškerých potřebných závislostí na obě třídy.

```php
class dbModel {

    private $dbQueries;

    function __construct() {
        $this->dbQueries = new dbQueries();
    }
    ...
}
```

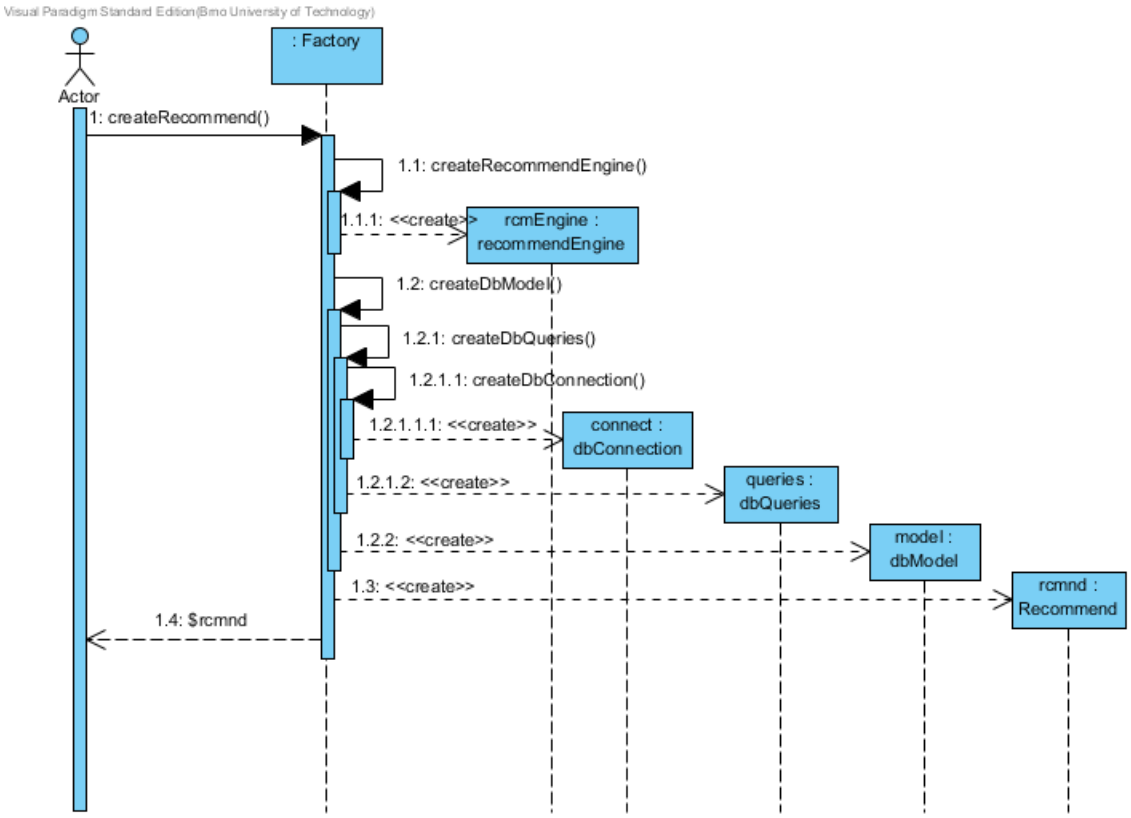
```php
class dbModel {

    private $dbQueries;

    function __construct(dbQueries $queries) {
        $this->dbQueries = $queries;
    }
    ...
}
```

Obrázek 4.14: Ukázka změny konstruktoru třídy dbModel; vlevo původní verze, vpravo po vložení závislosti

Pokud nyní budu chtít vytvořit objekt libovolné třídy v systému, stačí nyní zavolat patřičnou metodu třídy **Factory** a ta se již postará o vložení veškerých závislostí. Diagram sekvence na obrázku 4.15 zobrazuje tvorbu objektu třídy **Recommend**. V první řadě vidíme vytvoření objektů třídy **RecommendEngine** a následně třídy **dbModel**. Jelikož třída **dbModel** má další závislost na jinou třídu, je potřeba vytvořit i ji. A takto dále pokračovat dokud se nedostanem na listový uzel grafu asociací.

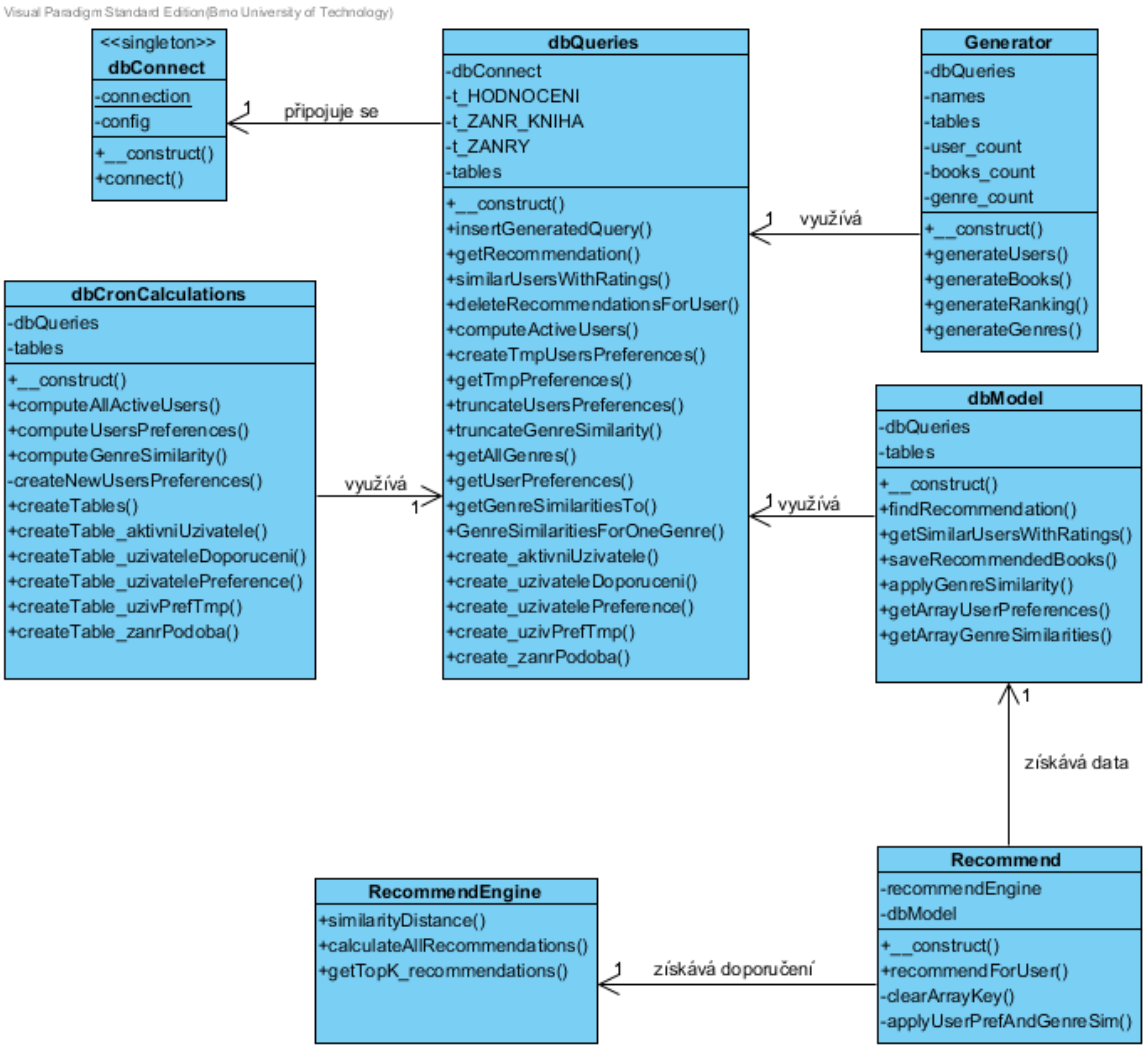


Obrázek 4.15: Sekvenční diagram zobrazující tvorbu objektu třídy **Recommend**

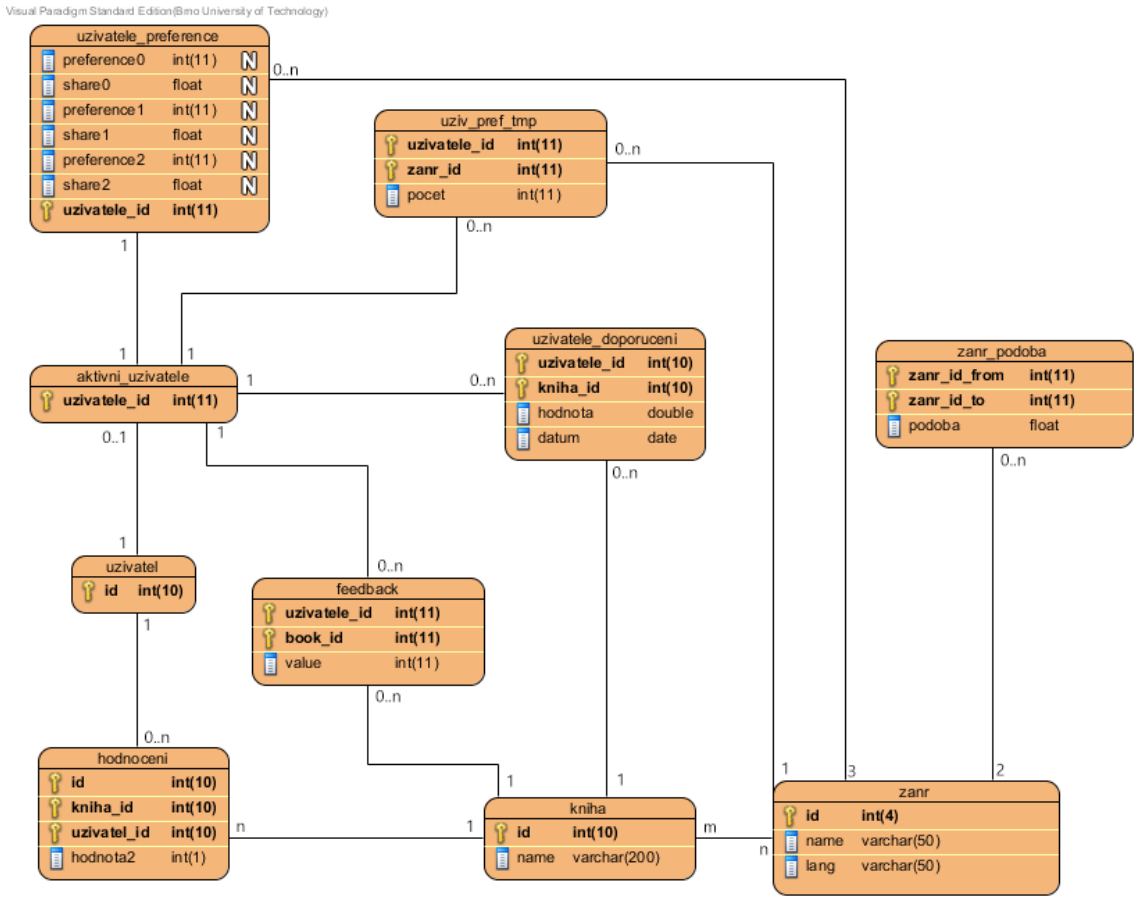
4.3 Celkový doporučovací systém

Cílem mého doporučovacího systému nebyla komplexní webová aplikace s grafickým výstupem, nýbrž zásuvný modul spolupracující se serverem cbdb.cz, který poskytne data pro doporučování. Základním požadavkem na modul bylo jednoduché ovládání. To zajistí metoda třídy **Recommend**, která při volání metody `recommendForUser()` s patřičným identifikátorem uživatele obstará veškerý výpočet doporučených knih. Dále také bylo potřeba oddělit vrstvu, kterou bude využívat CRON. Spojením všech vrstev systému do jednoho celku dostaneme jeden systém, jehož závislosti zachycuje obrazek 4.16. Obrázek zachycuje stav před poslední iterací, vynechána je nejnovější třída **Factory**, aby bylo možné názorně zobrazit mezi trídami patřičné asociace, které byly právě odstraněny v poslední iteraci.

Diagram schématu databáze na obrázku 4.17 zobrazuje nové tabulky potřebné pro doporučovací systém a zároveň také ukazuje původní tabulky serveru, které jsou využívany pro doporučení.



Obrázek 4.16: Kompletní pohled na třídy doporučovacího modulu



Obrázek 4.17: Diagram schématu databáze pro doporučovací modul

Kapitola 5

Evaluace

Po dokončení fáze implementace zdrojových kódů proběhlo nasazení na server cbdb.cz panem Jandou. Následně byl doimplementován grafický výpis, když si registrovaný uživatel otevře stránku „Doporučené knihy“. Jako hlavní zdroj správnosti doporučení jsem využil samotné uživatele. Ve výpisu seznamu doporučených knih uživatel vidí obálku knihy, její název a autora. Následně jsou také pod každou takovouto knihou dvě tlačítka „+“ a „-“. Ty umožní uživateli dát zpětnou vazbu ke každé jednotlivé knize zvlášť. Pokud si myslí, že doporučená kniha by se mu mohla líbit, zvolí „+“. V opačném případě stiskne „-“.

Každá uživatelská zpětná vazba je ukládána do tabulky *feedback* ve tvaru:

$$(kniha_i, uzivatel_j, hodnota)$$

kde $kniha_i$ je identifikační číslo právě hodnocené knihy, $uzivatel_j$ je identifikátor uživatele a $hodnota \in \{0, 1\}$ zachycuje „-“ nebo „+“.

V době psaní této práce byl doporučovací modul aktivní osm dní, když proběhla agregace a vyhodnocení uživatelské zpětné vazby. Za tu dobu bylo ohodnoceno 2 878 doporučených knih. Zpětnou vazbu poskytlo 576 uživatelů. Což z celkového počtu 8 147 aktivních uživatelů, pro které je doporučení počítáno, tvoří 7%. V průměru tedy jeden uživatel ohodnotil 5 ze 40 knih, které mu systém doporučil. Tabulka 5.1 znázorňuje rozložení pozitivního a negativního hodnocení. Vidíme, že z celkového poočtu je 1497 pozitivních hodnocení. Což tvoří 52% veškerých hodnocení.

	počet	poměr
-	1 381	48%
+	1 497	52%

Tabulka 5.1: Výsledek zpětné vazby uživatelů

Závěrem tedy ze zpětné vazby vyvodit, že uživatelé označili každou druhou doporučenou knihu jako tu, co by se jim mohla líbit. Což v samotném důsledku znamená, že by si ji i přečetli. Následně by bylo vhodné udělat další analýzu ve větším časovém horizontu, například za dobu několika měsíců. Každá doporučená kniha se ukládá do databáze, dalo by se tedy dohledat, zda uživatel ohodnotil knihu, která mu v minulosti .byla doporučena. Dále by se dalo také přihlížet na hodnotu zpětné vazby, zda si i přečetl doporučenou knihu, kterou předtím ohodnotil „-“. Pro další vývoj tohoto systému by bylo také vhodné zakomponovat tyto zpětné vazby a z nich odvodit nějaké skutečnosti. Například znovu nedoporučovat knihu, která dostala několikrát negativní zpětnou vazbu.

Kapitola 6

Závěr

Výsledkem diplomové práce je doporučovací modul, který ve spolupráci se serverem cbdb.cz doporučuje knihy. Již v tomto okamžiku mají registrovaní návštěvníci serveru možnost zobrazení seznamu doporučených knih, který na základě jejich hodnocení vybral můj modul. Nutnou podmínkou je, aby takovýto uživatel měl ohodnocených více než 20 knih. Výsledný program splňuje stanovené požadavky na jednoduché ovládání ze strany serveru a zároveň také splnil požadavek na jednoduché začlenění do stávajících kódů.

Cílem druhé kapitoly práce je prostudování a seznámení se s doporučovacími systémy. Popisuji zde jaký význam má tvorba doporučovacího systému a jaký přínos pro uživatele tyto systémy přináší. Dále jsem se zde zaměřil na typ dat, se kterými doporučovací systémy obecně pracují - položky, uživatelé a transakce. Následně byly tyto systémy rozděleny do šesti základních skupi podle přístupu k doporučování : filtrování založené na obsahu, filtrování založené na znalostech, filtrování založené na demografických informacích, kolaborativní filtrování, komunitní filtrování a hybridní doporučovací systémy. U každé kategorie jsem nastudoval jejich podstatu a význam, následně jsem ukázal pro jaké oblasti se různé přístupy používají. Jsou zde také uvedeny metriky pro měření přesnosti doporučení a také jsem v závěru kapitoly shrnu problémy, se kterými se doporučovací systémy potýkají.

Následující třetí kapitola popisuje oblast dolování z dat. V této kapitole popisuji principy předzpracování dat, jako je vzorkování a redukce dimenzionality. Následně je zde popsána klasifikace. Informace v této kapitole jsou dány do souvislosti s doporučovacími systémy.

Ve čtvrté kapitole byl popsán stav serveru cbdb.cz před samotným začátkem vývoje mého modulu. Je zde popsána datová sada, ke které mi server poskytnul přístup. Také zde popisuji funkčnost jednoduchého doporučovacího systému, který se zde již nacházel. Stěžejní částí této kapitoly je popis životního cyklu vývoje doporučovacího modulu, který byl rozdělen do čtyř iteračních cyklů. V prvním cyklu jsem se zaměřil na požadavky na celkový systém, dále na potřebná rozšíření současného stavu databáze a na použité technologie. Pro implementaci byl zvolen PHP a MySQL, byly také využity principy OOP. Druhá iterace popisuje analýzu a implementaci objektů databázové vrstvy. V této iteraci byl vyřešen hlavní problém, se kterým se potýkají doporučovací systémy - problém řídkých dat. Ve třetí iteraci proběhla analýza a implementace doporučovací vrstvy, která pro získání dat využívá služby databázové vrstvy. Je zde popsán přístup, jakým jsou knihy doporučovány jednotlivým uživatelům - využitím technik kolaborativního filtrování. A následně zde popisuji jak na doporučení hraje vliv podobnost žánrů a také uživatelských preferencí - čímž vyžívám technik filtrování založených na obsahu. Touto kombinací se celý můj doporučovací modul řadí do kategorie hybridních doporučovacích systémů.

Díky nasazení systému do reálné aplikace s tisíci uživateli bylo vhodné nasadit evaluační

metodu zpětné vazby. Každý aktivní uživatel měl možnost ohodnotit jednotlivé doporučení. Analýze těchto dat se věnuje pátá kapitola. Bylo zde popsáno jak moc se uživatelé dobrovolně zapojily do hodnocení doporučení a byly zde vyvozeny závěry. Následně zde popisuji i možné návrhy na vylepšení doporučovacího algoritmu v souvislosti se získanými daty zpětné vazby. Uživatelé řekli, že každá druhá doporučená kniha by se jim mohla líbit k přečtení. Se zmíněnými rozšířeními by mohlo dojít k nárustu spokojenosti.

Zadání práce bylo splněno ve všech bodech a bylo úspěšně nasazeno do reálné aplikace. Po nasazení se ukázalo, že systém doporučuje i knihy, které jsou součástí nějaké série knih (trilogie, pentalogie,...). Pokud tedy uživatel knihu z této série nečetl, bylo by vhodné kontrolovat, zda doporučené knihy právě nespadají do této série. Jako další rozšíření systému by mohla být úprava algoritmu výpočtu uživatelských preferencí tak, aby nejnovější hodnocené žánry měli nejvyšší váhu a nejstarší hodnocení, aby ovlivnovaly preference nejméně. Tímto by se reflektoval uživatelův růst a případný posun k jiným žánrům. V současné době je systém koncipován tak, aby jednotlivým uživatelům vypsal seznam doporučených knih, které by se jim mohly líbt, tedy doporučení uživatel - kniha. Jako rozšíření by se mohlo implementovat doporučení kniha - kniha. Tedy když by si uživatel otevřel detail nějaké knihy, uviděl by zde seznam knih, které jsou doporučené ke čtení v souvislosti s touto knihou.

Literatura

[1] Arazy, O.; Kumar, N.; Shapira, B.: Improving Social Recommender Systems. *IT professional*, ročník 11, č. 4, 2009: s. 38–44.

[2] Berkovsky, S.; Kuflik, T.; Ricci, F.: Mediation of user models for enhanced personalization in recommender systems. *User Modeling and User-Adapted Interaction*, ročník 18, č. 3, 2008: s. 245–286.

[3] Berkovsky, S.; Kuflik, T.; Ricci, F.: Cross-representation mediation of user models. *User Modeling and User-Adapted Interaction*, ročník 19, č. 1-2, 2009: s. 35–63.

[4] Bridge, D.; Göker, M. H.; McGinty, L.; aj.: Case-based recommender systems. *The Knowledge Engineering Review*, ročník 20, č. 03, 2005: s. 315–320.

[5] Burke, R.: Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, ročník 12, č. 4, 2002: s. 331–370.

[6] Burke, R.: Hybrid web recommender systems. In *The adaptive web*, Springer, 2007, s. 377–408.

[7] Felfernig, A.; Burke, R.: Constraint-based Recommender Systems: Technologies and Research Issues. In *Proceedings of the 10th international conference on Electronic commerce*, ACM, 2008, str. 3.

[8] Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern. 2004.
URL http://www.martinfowler.com/articles/injection.html

[9] Golbeck, J.: Generating predictive movie recommendations from trust in social networks. In *Trust Management*, Springer Berlin Heidelberg, 2006, s. 93–104.

[10] Gunes, I.; Kaleli, C.; Bilge, A.; aj.: Shilling Attacks Against Recommender Systems: A Comprehensive Survey. *Artif. Intell. Rev.*, ročník 42, č. 4, Prosinec 2014: s. 767–799, ISSN 0269-2821, doi:10.1007/s10462-012-9364-9.

[11] Jannach, D.; Zanker, M.; Felfernig, A.; aj.: *Recommender Systems: An Introduction*. Cambridge University Press, 2010, iSBN 978-0521493369.

[12] Meteren, R.; Someren, M.: Using Content-Based Filtering for Recommendation. University of Amsterdam.

[13] Mortensen, M.: *Design and Evaluation of a Recommender System*. Diplomová práce, University of Tromso, 2007.

[14] O'Mahony, M. P.; Hurley, N. J.; Silvestre, G. C. M.: Detecting Noise in Recommender System Databases. In *in proceedings of the international conference on intelligent user interfaces (IUI'06), 29TH-1ST*, ACM Press, 2006, s. 109–115.

[15] Resnick, P.; Iacovou, N.; Suchak, M.; aj.: GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, ACM, 1994, s. 175–186.

[16] Ricci, F.; Rokach, L.; Shapira, B.; aj.: *Recommender systems handbook*. New York: Springer, 2011, doi:10.1007/978-0-387-85820-3.

[17] Russell, S. J.: *Artificial intelligence*. New Jersey: Prentice Hall, druhé vydání, c2003.

[18] Sarwar, B.; Karypis, G.; Konstan, J.; aj.: Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, ACM, 2001, s. 285–295.

[19] Schafer, J. B.; Frankowski, D.; Herlocker, J.; aj.: Collaborative filtering recommender systems. In *The adaptive web*, Springer Berlin Heidelberg, 2007, s. 291–324.

[20] Shani, G.; Gunawardana, A.: Evaluating Recommender Systems. Technická Zpráva MSR-TR-2009-159, Microsoft Research, November 2009.
URL http://research.microsoft.com/apps/pubs/default.aspx?id=115396

[21] Sinha, R. R.; Swearingen, K.: Comparing Recommendations Made by Online Systems and Friends. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries'01*, 2001.

[22] Taghipour, N.; Kardan, A.; Ghidary, S. S.: Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*, ACM, 2007, s. 113–120.

[23] Zendulka, J.; Bartík, V.; Lukáš, R.; aj.: Získávání znalostí z databází. Studijní opora, FIT VUT, Brno, 2009.

Dodatek A

Obsah CD

- src/ - zdrojové soubory doporučovacího systému
- doc/ - zdrojoý kód a text diplomové práce