Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Aplikace machine learning algoritmu na doporučení zboží ke koupi

Bakalářská práce

Autor: Štěpán Kameník
Studijní obor: Aplikovaná informatika

Vedoucí práce: prof. Ing. Vladimír Bureš, Ph.D., MBA

Odborný konzultant: Ing. Jan Novotný
FG Forrest a.s.

Hradec Králové Srpen 2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 21.8.2020 Štěpán Kameník

Poděkování:

Děkuji prof. Ing. Vladimíru Burešovi Ph.D., MBA za osobní přístup, metodické vedení a hlavně trpělivost při tvorbě této práce. Dále děkuji Ing. Janu Novotnému za konzultace, zkušenosti a odborné rady při studování problematiky. Zároveň děkuji firmám FG Forrest a.s. a J&H online s.r.o. za podporu a poskytnutí výzkumných dat pro vytvoření této práce.

Anotace

Tato bakalářská práce se zabývá problematikou doporučovacích systémů a jejich zapojením do existujících či nově vzniklých e-commerce webových projektů. Teoretická část této práce je věnována obecnému seznámení s metodami doporučování, které se nejčastěji využívají v úrovni menších a středních e-commerce. V teoretické části je nejdříve popisován obecný problém doporučování a personalizace obsahu, následně pak typy a nejčastěji využívané algoritmy, které zadané problémy řeší. Poslední úsek teoretické části je věnován popisu složitějších situací, které řeší jednotlivé implementace s doplněním o pokročilé techniky a způsob testování jednotlivých typů systémů. Praktickým cílem této práce je samotná implementace řady doporučovacích algoritmů, které se mohou využít v situaci doporučování na základě "uživatelského košíku". První část praktického řešení obecně zavádí implementační situaci. Následně je detailně popsána samotná implementace algoritmů a výsledky testování na poskytnutých anonymizovaných datech z e-shop projektu signal-nabytek.cz a nejmenovaného e-shop projektu s potravinami. Poslední část této práce je věnována vysvětlení vzniklých problémů v implementaci, testování a interpretování výsledků s návrhy řešení.

Klíčová slova: doporučování, doporučovací systém, e-commerce, e-shop, personalizace, přizpůsobení obsahu, kolaborativní filtrace, kontextové přizpůsobení

Annotation

Title: Application of machine learning algorithms on purchase recommendation

This bachelor thesis deals with the issue of recommending systems and their involvement in existing or newly created e-commerce web projects. The theoretical part of this work is devoted to general acquaintance with the methods of recommendation that could be used at the level of small and medium e-shops. The theoretical parts describe the general problem of recommending, personalizing content and frequently used algorithms that solve the given problems. The last section of the theoretical parts is marked by a description of more complex situations that solves specific algorithms with the addition of advanced techniques and testing individual types of systems. The practical objective of this work is the actual implementation of a number of recommending algorithms that can be used in a recommendation situation based on a "user cart". The first part of the practical solution generally introduces the implementation situation. Subsequently is described in detail the implementation of algorithms itself and the results of testing on the provided anonymous data from the e-shop project signal-nabytek.cz and an unnamed e-shop project with food. The last part of this work explains problems in the implementation, testing and interpretation of results with proposed solutions.

Keywords: recommendation, recommendation systems, e-commerce, e-shop, personalization, content customization, collaborative filtering, contextual customization

Obsah

1 Úvod

V současné internetové době, kdy takřka každá rodina vlastní alespoň jedno chytré zařízení a většina domácností je vybavena internetovým připojením (Český statistický úřad, 2018), jsou častým tématem odborníků, ale i obecné veřejnosti, osobní údaje a obecně význam lidské bytosti ve světě internetu. Téma osobních údajů je velmi komplexní a skýtá obrovské množství neduhů, které společnost považuje za nežádoucí a ve většině případů uživatelé nechtějí svá osobní data rozšiřovat více, než je nezbytně nutné (parafráze dat: Český statistický úřad, 2018). Z hlediska bezpečnosti je tato praktika úspěšná, ale osobní údaj ať již charakterizující či popisující, nemusí být vždy mimo svůj hlavní účel nebezpečný. V dané situaci, kdy existuje okolo 200 milionů aktivních webových stránek (techjury.com, 2020) je najít přesně to, co uživatel žádá – velmi obtížné (Pandey, Pradhan, 2018). Nebylo by tomu tak v případě, kdyby se uživateli zobrazily konkrétní možnosti, které nějakým daným způsobem odpovídají jeho hledání. Právě v tuto chvíli přichází doporučovací systémy, které na základě známých dat o dotazujícím uživateli a podobných uživatelích systému, mohou doporučit to nejrelevantnější k dané situaci (Aggarwal, 2016). Tento problém se však neomezuje pouze na vyhledávání napříč webovými stránkami. Dalším příkladem může být obyčejné hledání restaurace – vyhledává-li uživatel restaurace a nachází se v Praze, tak ho pravděpodobně restaurace v jiných městech nezajímají. Velmi podobná situace je i na platformách e-commerce.

V e-commerce u velmi malých systémů (kde můžeme nalézt do desítky produktů) není problém zobrazit přesně to, co uživatel chce vidět. Problém nastává u středně velkých systémů (kde se naskýtá v rozmezí od sto do tisíc produktů), protože málokterý uživatel bude v tomto velkém systému listovat všemi stránkami produktů a vybírat přesně to, pro co opravdu přišel. Částečně tento problém řeší kategorizace produktů, ale ne zcela a ne tak, jak by uživatel uvítal pro nejrychlejší a nejpohodlnější nákup (Dennis, Merrilees, Jayawardhena, Wright, 2009). Hlavní otázkou k řešení tohoto problému je: Co vlastně uživatel na e-shopu doopravdy hledá/chce nakoupit?

Zde se dostáváme znovu k tématu osobních údajů, konkrétně osobních dat, které popisují zadaného uživatele. Nakupování v kamenných obchodech je totiž velmi situačně podobné – přijdete do obchodu, vyberete sekci, která vás zajímá a následně již vybíráte konkrétní produkt – sám nebo s pomocí obsluhy obchodu, která vám s výběrem poradí.

V internetovém obchodě však obsluha být fyzicky nemůže, a proto se jí vývojáři těchto systémů snaží nahradit doporučovacím systémem, který stejně jako obsluha v kamenném obchodě zhodnotí situaci a následně k ní vybere nejlepší možnou variantu. Příkladem by mohla být situace ženy, která vybírá obuv – obsluha zhodnotí věk zákaznice, určitý styl obuvi, kterou má právě na sobě, může zakomponovat i aktuální módní trend – na tomto základě s určitou mírou interakce, dokáže obsluha odhadnout, která konkrétní obuv by se ženě mohla líbit. V situaci internetového obchodu je situace velmi podobná. Uživatel vstoupí do sekce/kategorie obuvi a systém by mu měl, na základě osobních dat, též adekvátně doporučit produkt. Zde se již můžeme setkat s komplexními otázkami (Konstan, Ekstrand, 2016) a to například: Podle jakých údajů se rozhodovat o správném produktu? nebo více technická otázka: Jak systém zjistí, že uživateli dané doporučení pomohlo a jak efektivitu doporučení měřit?

Tato práce by měla odpovídat nejen na všechny z výše uvedených otázek, ale i osvětlit obecné řešení problematiky doporučení v různých odvětvích. S růstem internetového prostoru se doporučovací systémy velmi rozšířily, a to převážně kvůli vzrůstajícímu počtu dostupných osobních dat o konkrétním uživateli a následně tak i počtu dat/informací, které mohou být danému uživateli nabídnuty. Tento účel řadí domény doporučovacích systémů do filtračních systémů – v problematice by měla obecně vyfiltrovat jakoukoliv informaci, která je pro koncového uživatele relevantní (Aggarwal, 2016). Výše jsou popsány situace jako doporučení produktů či filtrace webových stránek, ale aplikovatelné jsou tyto algoritmy na veškerý obsah, který se nyní v počítačovém světě nachází – ať již newslettery, doporučení přátel či skupin na sociální sítích nebo dokonce řazení aplikací v nabídkách mobilních zařízení (Konstan, Ekstrand, 2016).

Oblastí, kde nyní uživatelé jakýchkoliv chytrých zařízení mohou najít doporučovací systémy je nesčetně, a právě proto byla tato problematika vybrána jako téma této bakalářské práce. Sám autor po studiu těchto doporučovacích technik změnil způsob, jakým na internetu pracuje – pomáhá těmto systémům k lepším výsledkům – ne ani tak pro zlepšení aplikací samotných, ale hlavně pro komfort a ulehčení, které nabízejí.

2 Cíl práce

Cílem této práce v teoretické části je popsat situaci, ve které jsou použity doporučovací systémy a zároveň se seznámit s nejpoužívanějšími metodami a algoritmy pro samotné doporučení. V rámci tohoto úkolu by měl čtenář pochopit vlastnosti, efektivitu i úskalí jednotlivých algoritmů a následně pomocí popsaných metrik efektivity určit nejlepší řešení pro konkrétní situaci. Součástí praktické části je následně samotná implementace nejpoužívanějších a nejreprezentativnějších algoritmů různých metod doporučovacích systémů a zároveň použití na reálném případu e-commerce řešení pro doporučování v košíku.

3 Metodika zpracování

3.1 Metodika implementačního řešení

V rámci implementačního řešení by se měla prokázat vlastnost, že ne všechny algoritmy doporučení jsou vhodné pro použití v obecném případě doporučování kdekoliv v prostoru řešení. Dále by se mělo prokázat jaká z metod doporučování je možná pro využití v rámci konkrétního doporučování v košíku a jaký z algoritmů je zde nejefektivnější dle používaných metrik.

3.2 Prokazované hypotézy v implementačním řešení

Pracovní hypotézy:

Hypotéza č. 1: V situacích doporučování v košíku na základě právě objednávaných produktů není možné bez větší abstrakce či odhadu aplikovat pokročilé druhy algoritmů metody faktorizace matic s použitím strojového učení.

Hypotéza č. 2: Doporučování musí být na jednotlivých e-shop aplikacích odlišně nastaveno a nedá se prokázat, že by jediný algoritmus či metoda, mohla být použita beze změn a úprav v řešení napříč celou platformou.

3.3 Způsob ověřování hypotéz

Ověření hypotéz bude prováděno na dvou sadách dat z reálných e-commerce řešení signal-nabytek.cz a nejmenového projektu prodávajícího potraviny. Pro ověření i samotný cíl práce bude implementováno několik nejpoužívanějších algoritmů, které by pomocí daných metrik měly prokázat správnost hypotéz i správnost vlastní implementace s případnými návrhy řešení v situaci, kdy implementované řešení v kombinaci s poskytnutými daty nebude dostatečně efektivní pro přímé napojení.

4 Teoretická část práce

Samotný systém doporučování sahá svým principem daleko do minulosti, do dob, kdy ještě lidé samotní neexistovali. Tento princip je nejlépe demonstrovatelný například na mravencích (Konstan, 2016). Ve standardním mraveništi žijí obvykle milióny mravenců a většinu z této populace tvoří tzv. dělnice. Dělnice mají za úkol obstarat potravu a materiál pro stavbu mraveniště, a to typickým způsobem – dělnice se ze svého mraveniště vydá do větší vzdálenosti a hledá potencionální zdroj potravy či materiálu. V případě, že daný zdroj najde, zapamatuje si jeho polohu, vrátí se zpět do kolonie a uvědomí první jednotky dělnic o svém nálezu. Informace o nalezení nového zdroje se poté kaskádově šíří mraveništěm od jedné dělnice k druhé. Po získání dostatečného množství dělnic ke zpracování nalezeného zdroje každá z nich pro zdroj samotný vyrazí. Tento princip na první pohled s doporučováním vůbec nesouvisí, ale paradoxně je jeho hlavní součástí.

Přeneseně můžeme tvrdit, že první dělnice, která nalezla zdroj, objevila objekt doporučení a vše co následně udělala je "doporučení" ostatním dělnicím. Ty pak samy zváží relevanci jejího doporučení a pokud je relevance natolik velká, že v ostatních potencionálních zdrojích zvítězí (doporučený zdroj je největší, nejblíž, nejpotřebnější) je objekt doporučení převeden na objekt zájmu. Tomuto druhu doporučení se ve světě online doporučování říká kolaborativní filtrace (en. collaborative filtering) a je jedním z vícero možností, jak doporučení samotné provést. O seznámení s těmito technikami pojednává tato teoretická část práce.

4.1 Úvod do situace použití doporučovacích systémů

Ve webovém prostoru e-commerce platforem je velmi těžké přesně vydefinovat obecnou situaci - tzn. "kdy a jak" k doporučení má dojít – a to jak z technického hlediska, tak i z hlediska uživatelské přívětivosti (UX). Kvůli tomuto problému vznikly čtyři obecné principy stavby recommender systémů (Aggarwal, 2016), prvním způsobem řešení takového systému je pomocí obecné souhrnné statistiky. S tímto řešením se nejspíše setkal každý uživatel, který alespoň jednou navštívil některý z e-shopů. V nejjednodušší formě se jedná o sekce webu, které jsou nazvané jako "nejprodávanější", "nejoblíbenější" nebo "právě v trendu". Pomocí těchto metod však můžeme o doporučovaném objektu zjistit i skryté obecné relace (Fátima, Bruno, 2016) k ostatním objektům. Tento způsob aplikace je

tzv. nepersonalizovaný, to znamená, že není nijak ovlivněn samotným uživatelem, pro kterého je doporučování tvořeno. Právě kvůli této vlastnosti se používá pouze jako podpůrný nástroj koupě, jako je třeba filtrace, méně již do sekcí jako jsou “doporučeno” nebo “přesně pro vás”.

Ostatní způsoby, které jsou více popsány od kapitoly 4.3. využívají opačný způsob doporučení a obecně řečeno se snaží uživateli poskytnout seznam objektů, které s největší pravděpodobností (z celku daných produktů) kopírují uživatelské preference. Tento způsob s sebou ale nese dva hlavní nedostatky, první z nich je, že uživatel musí systému co možná nejdetailněji popsat své preference, případně je systém musí bez větší odchylky zjistit sám. Druhým nedostatkem se pak stává fakt, že ve většině případů objekty/produkty jako takové systém zná jen v omezeném kontextu. Každý ze způsobů řešení personalizovaných doporučení řeší výše uvedené problémy trochu jiným způsobem, ale jedno mají společné – potřebují o uživateli zjistit co nejvíce informací – co “má rád” nebo alespoň jaké jsou objekty jeho zájmu.

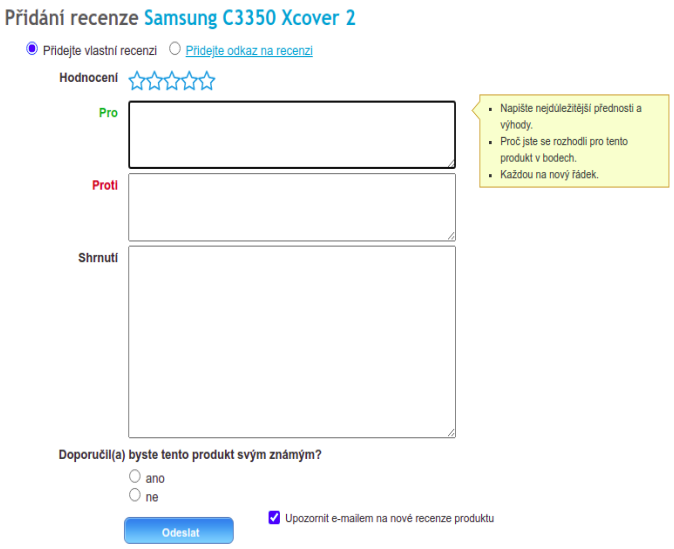
4.1.1 Typy použitelných dat

Každý uživatel webu je povětšinou v něčem odlišný v hodnocení (Aggarwal, 2016), jeden z nich zhlédne film a na fóru, k tomu určeném, ho ohodnotí pěti hvězdičkami, protože se mu moc líbil. Druhému se však může též líbit, ale udělí filmu pouze tři hvězdičky, protože další dvě jsou vyhrazené pro jeho nejoblíbenější série filmů. Třetí uživatel hodnocení filmu vůbec neudělí. Různorodé chování uživatelů úlohu o definování uživatelova zájmu dále komplikuje a hlavním se v této úloze stává porozumění signálům, které uživatel na zvoleném webu může systému předávat/zanechat.

4.1.1.1 Explicitní zpětná vazba

První z druhů těchto signálů se nazývá Explicitní zpětná vazba (en. Explicit feedback) a je definována jako vědomé uživatelské hodnocení objektu (Jawaheer, Szomszor, Kostková, 2010). Pod tímto hodnocením se mohou skrývat různé varianty ratingů, od nejčastějších jako je hodnocení filmu nebo hotelu hvězdičkami na škále 0-5 až po vytvořené formuláře pro specifický segment objektů či uživatelů.

Tento způsob hodnocení má v praxi dvě hlavní negativa, s kterými systém musí počítat, prvním z nich, jak již bylo zmíněno, je různá hodnotící škála uživatele. Uživatel z menšího města, který navštěvuje pouze lokální restaurace, může ohodnotit zvolenou restauraci všemi hvězdičkami, protože byla jedna z nejlepších, ve které kdy byl. Ale v případě, že přijede do stejné restaurace například cizinec, zvyklý na nejlepší restaurace Londýna, ohodnotí restauraci pravděpodobně záporně, protože v jeho měřítku jsou restaurace daleko lepší. Dalším z faktorů je poté obecná míra hodnocení, kdy jeden uživatel může místo hodnocení 0-5 využít na většinu hodnocení pouze škálu 0-3 - buď z důvodu, že žádný z hodnocených objektů nenaplnil jeho očekávání, nebo protože má plný počet hvězdiček (5) spojen třeba jen s jednou restaurací. Naopak se musí počítat i s tím, že nemalý segment uživatelů shlédne film a v případě, že se jim líbil – mu udělí hodnocení, ale v opačném případě, kdy už o filmu nikdy nechtějí slyšet – zaklapnou počítač a na hodnocení se už nikdy nedostanou.



Obr. 1 Tvorba recenze na produkt / explicitní zpětné vazby

Zdroj: výstřižek autora z webové stránky heureka.cz

4.1.1.2 Implicitní zpětná vazba

Druhým typem zpětné vazby je vazba nepřímá, tzv. implicitní (en. implicit feedback), kterou uživatel přímo nesděluje, ale je výsledkem jeho kroků, které udělal v systému, či aplikaci. Například: logicky můžeme usoudit, že uživatel, který na e-shopu vybírá telefon značky Apple, zobrazí si 5 variant modelů produktu této značky v černé barvě a v určitém rozsahu ceny, nebude mít zájem o typ telefonu, který patří do nižší třídy se systémem Android a naopak. Tento způsob "dolování" preferencí je možné využít na velkém množství případů. Můžeme jej například využít na zmíněném zobrazování produktu, výběru kategorie v které se nachází dané produkty, vložením do košíku, vložením do porovnání cen, parametrů nebo přímo v konečném objednání konkrétního zboží.

Na rozdíl od explicitní zpětné vazby zde máme pouze binární informaci o tom, zda uživatel konkrétní krok učinil, nebo ne. Velká nevýhoda tedy vzniká v tom, že bez explicitní vazby není možné zjistit, zda se objednaný produkt uživateli vůbec nakonec líbil. Situace se může o něco zlepšit tím, že implicitní data zkombinujeme a vytvoříme svou vlastní škálu preferencí a to tak, že například definujeme úspěšnost produktu dle toho, zda uživatel produkt viděl a zakoupil stejný, případně podobný (např. ve stejné kategorii, či obdobnou variantu). Tímto procesem implicitní data můžeme převést z binárních čísel do reálných čísel. Tím můžeme i měřit ve škále relevanci, která bude mít své opodstatnění hlavně v konkrétní implementaci systému.

4.1.1.3 Predikce vs. doporučení

Po seznámení se vstupními daty doporučovacích systémů je ještě nutné vymezit výstupní data. Jako výstup z konkrétních aplikací lze použít dva typy objektů – predikce (předpověď) nebo doporučení.

Predikce ve většině případů udává míru (na specifické škále), jak moc se uživateli bude objekt líbit. Vzhledem k dříve uvedeným faktorům při explicitním zpětným vazbám je tato míra přetransformována na škálu uživatele a může být použita při prohledávání katalogu objektů na platformě či detailu samotného objektu jako jakýsi pomocný prvek pro výběr (Konstan, Ekstrand, 2016).

Obr. 2 Predikce hodnocení telefonu

Zdroj: výstřižek autora z webové stránky heureka.cz

Doporučení jsou následně na predikcích založena a tvoří tzv. n-listy. N-listy jsou sekce na webu, na kterých je zobrazeno prvních n objektů (většinou 5-10) o kterých si systém myslí, že jsou pro uživatele nejrelevantnější, či nejzajímavější (může být založeno na nejvyšší predikční hodnotě). Na rozdíl od predikcí jsou doporučení tvořena jako plný nástroj výběru, místo pouhé podpory výběru při hledání či procházení webu, to dovoluje doporučením tvořit sekce v zájmových oblastech po boku stránky – například s bannery – a upoutat tak uživatelům zájem.

Obr. 3 Doporučení produktů

Zdroj: výstřižek autora z webové stránky signal-nabytek.cz

4.2 Využití metod pro nepersonalizované doporučení

Jak již bylo zmíněno v předchozí kapitole, prvním způsobem, kterým se dá problematika doporučení řešit je nepersonalizovaný způsob využití souhrnné statistiky. Tento způsob je četně využíván napříč všemi platformami pro e-commerce a jeho přímá aplikace je jedna z nejjednodušších. Konkrétně se v této kategorii vyskytují algoritmy pro tvorbu například filtrací "nejoblíbenější", "nejprodávanější", "právě v trendu" nebo i v některých případech sekce "související produkty".

4.2.1 Metoda průměrného hodnocení

Za první z algoritmů této metody se dá považovat prostý průměr všech dostupných hodnocení uživatelů. Tento algoritmus je jedním z nejjednodušších a vyjadřuje obecný názor na objekt na základě explicitního hodnocení či frekvence v případě implicitní vazby.

4.2.2 Asociační metody

Technika asociačních metod je velmi používaná nejen v rámci e-commerce webových projektů ale i například v marketingu a propagaci produktů obecně (Fátima, Bruno, 2016). Částečně se může považovat za personalizovanou úlohu. Úloha vychází z vazeb, které jsou určeny vstupními daty. Dle algoritmu, který využívá, dokáže na základě například objednávek určit, že produkt X byl často nakupován s produktem Y a proto jsou vzájemně nějakým způsobem produkty propojené. Obecně řečeno tato technika, resp. algoritmy, které ji využívají, vytváří malý kontext v rámci jednotek objektů, které spolu v rámci nějakých dodaných dat souvisí/mohou souviset.

Díky této definici můžeme říci, že tato metoda je již částečně personalizovaná, protože pro příklad: při doporučování u produktu jídelního stolu na základě předchozích objednávek, algoritmus určí, že v rámci kontextu jídelního stolu je nejblíže jídelní židle, kvůli tomu, že spolu tyto produkty uživatelé nakupují často. Metoda však abstrahuje jakékoliv historické chování uživatele a hledá "související" produkty pouze v rámci kontextu daného produktu na který se uživatel dívá právě teď.

Základem asociační metody je počet hodnocení dvou objektů v rámci nějaké události, příkladem by mohla být událost nákupu, kdy asociační metoda sčítá výskyty dvou objektů v jedné objednávce. Tento mechanismus však nepracuje s hodnotou samotného hodnocení – tedy v případě, že se jedná o explicitní hodnocení nedokáže v základním tvaru reflektovat míru "spokojenosti" ale pouze vypočítaný vztah. Tomuto jevu můžeme zamezit tak, že použijeme součet údajů hodnocení pouze nad určenou hranicí, například pouze hodnocení nad 2.5 z 5 (Fátima, Bruno, 2016), případně použitím komplexnějších způsobů aplikace, jakými jsou například hybridní systémy.

4.3 Obsahová filtrace

Dalším ze způsobů řešení doporučovacích systémů je obsahová filtrace (content-based filtering), která na rozdíl od ostatních metod nezávisí na ostatních uživatelích systému, ale pouze na zjištěních, které byly vypočteny z interakcí s konkrétním uživatelem samotným.

Hlavní myšlenkou obsahové filtrace je fakt, že každý objekt, který by mohl být doporučován má své vlastnosti, například – složení. Uživatel, který nakoupí sadu těchto objektů, může být na základě těchto vlastností objektů identifikován, respektive mu může být vytvořen tzv. profil (Zisopoulos, Karagiannidis, Demirtsoglou, Antaris; 2008). Tento profil zakládá na tom, že uživatel má pro dané vlastnosti (př. obsah cukru, soli, mouky, tuků, sacharidů a barviv) nějaký svůj názor nebo lépe – přesnou hodnotu preference. Máme-li konkrétní příklad, objednávku uživatele – z 8 produktů: 6 bez lepku, 2 bez cukru a všechny bez tuků – analýza preferencí tohoto uživatele by tedy měla vyvodit výsledek, že uživatel má velmi zápornou preferenci mouky a vyšší negativní preferenci lepku, naopak všechny produkty obsahují sacharidy, proto uživatel tyto preference bude mít pozitivní, barviva neobsahuje ani jeden z objednaných produktů, proto budou preference neutrální. Na základě těchto dat, respektive profilu konkrétního uživatele, můžeme posléze najít takovou skupinu zboží, která se svými hodnotami vlastností přibližuje těm uživatelským.

Komplikace v tomto způsobu "dolování" uživatelského profilu nastává v případě, že většina produktů například obsahuje sůl – uživatel ji tedy s velkou pravděpodobností bude mít i častokrát ve svých objednávkách a vytvoří se tím i velmi silná preference v jeho profilu. Na tomto zjištění poté systém zkreslí doporučení, protože upřednostní produkty, které obsahují sůl, i když některý z produktů bez soli by se vzhledem k uživatelskému profilu, při zanedbání preference soli, líbil uživateli více.

Tento problém řeší metodika hodnocení relevance zvaná TF-IDF (term frequency – inverse document frequency) (Konstan, Ekstrand, 2016), která je často používaná mimo jiné i ve vyhledávačích a v knihovnách jako je Apache Lucene. Základem (zjednodušenou formou) této metodiky je vytvoření vektoru, který ke každé vlastnosti v objektu přiřadí počet výskytů (TF) a současně vytvoří vektor vlastností pro všechny objekty (počet výskytů globálně). Z globálního vektoru vlastností následně vytvoříme IDF (často pomocí log(N/t), kde N je počet objektů a t počet výskytů (více v praktické části), TF v tomto algoritmu značí počet výskytů v rámci objektu, kdy v některých případech může nabývat pouze hodnot 0,1

v případě, že objekt vlastnost má, či nemá, v opačném případě (jako například u dokumentů) je TF počet výskytů konkrétního slova. IDF naproti tomu signalizuje globální výskyt slova a tím vzniká tzv. "váha slova/vlastnosti", pomocí které následně můžeme normalizovat preferenci ke konkrétní vlastnosti, přesně jak bylo uvedeno v předchozím případě soli, která by váhu (IDF) měla velmi nízkou, a proto by méně ovlivnila výsledky celého výpočtu.

4.3.1 Podmínky použití

Z předchozí kapitoly vyplývá, že metoda TF-IDF je velmi komplexní a lehce upravitelná. Právě díky upravitelnosti vzniklo velmi mnoho různých způsobů výpočtů, avšak základním stavebním kamenem celé metody je znalost objektů v prostoru doporučování (Aggarwal, 2016). V rámci vlastního průzkumu dat napříč menšími i středními e-commerce projekty však bylo zjištěno, že i přes to, že projekt již stabilně funguje roky, nemá u částí sortimentu vyplněné žádné z vlastností. Tento fakt je hlavně způsoben tím, že některé produkty vlastnostmi popsat ani relevantně nelze (například kuchyňské nádobí), u některých by popsání nebylo k povaze věci přínosné (například okrasná svíčka, kde se rozhodujeme podle obrázku, a ne podle daných vlastností barvy a objemu) nebo dokonce nejsou relevantní k danému produktu a proto nemá přínosný význam s nimi počítat při doporučování (například plyšová hračka - většinou se nevybírá dle konkrétní velikosti). Tyto produkty jsou dále v systému brány jako přirozený šum a pro korektní práci se musí upravit, případně vymezit z prostoru doporučení.

4.4 Kolaborativní filtrace

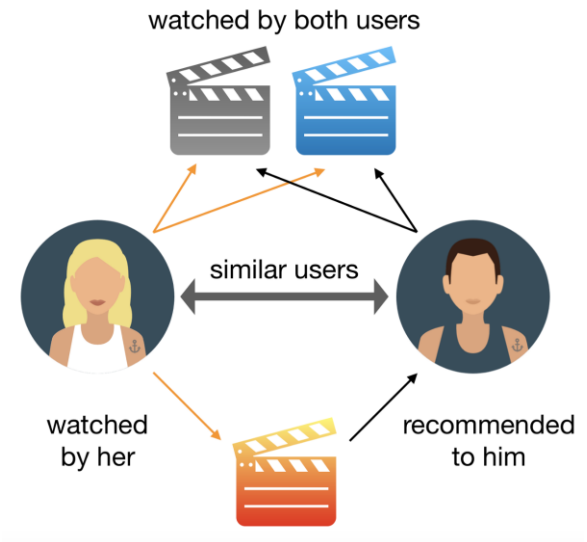
Kolaborativní filtrace (collaborative filtering) je jednou z nejpoužívanějších metod doporučení napříč systémy. Jedná se o metodu, zakládající se na komunitě lidí, kteří systém (e-commerce) používají a spoléhá na podobnosti uživatelů, nebo produktů v rámci této komunity. Nejjednodušším příkladem této metody může být doporučení mezi známými, kdy přátelé vědí, že mají podobný vkus například na filmy určitého žánru. Když první z těchto přátel uvidí film a bude se mu líbit, s velkou pravděpodobností film doporučí druhému, protože ví, že jsou ve svém vkusu k filmům podobní. Na tomto principu zakládá i kolaborativní filtrace, obecně řečeno – metoda vyhledá uživatelé, kteří mají stejný názor/vkus na zboží jako uživatel S a doporučí mu objekty, které zakoupili (nebo kladně ohodnotili) uživatelé s velmi podobným vkusem jako má zmíněný uživatel S.

4.4.1 Metody nejbližšího souseda

Na příkladu minulé kapitoly je prakticky přesně popsána technika, jak funguje tzn. metoda nejbližšího souseda, jen s tím rozdílem, že tato metoda hledá N nejbližších/nejpodobnějších sousedů se stejným vkusem, proto je metoda někdy označována jako K-NN (K - nearest neighbor).

4.4.1.1 Uživatelská filtrace

Prvním z konkrétních aplikací metody nejbližšího souseda je uživatelská filtrace (user-based filtering). Tento druh filtrace spočívá ve výpočtu podobnosti uživatelů mezi sebou, respektive podobností jejich interakcí k jednotlivým produktům v systému (Desarkar, Sarkar; 2012). K tomuto měření podobnosti se používají různé techniky, jakou je například kosinova podobnost nebo Pearsonův koeficient. Na základě N nejpodobnějších uživatelů je dále tvořen seznam koncových doporučovaných produktů na základě váhy uživatele, kdy pro každý produkt systému algoritmus vyhledá počet výskytů u podobných uživatelů a (nejčastěji) vynásobí samotnou mírou podobnosti mezi uživateli. Tímto způsobem vznikne seznam produktů, které se nejčastěji objevily v interakcích uživatelů, kteří jsou koncovému uživateli nejpodobnější. V případě hodnotící škály nejsou brány v potaz pouze výskyty, ale hodnota samotného hodnocení vynásobená touto mírou, která zajistí normalizované hodnocení vůči koncovému uživateli.

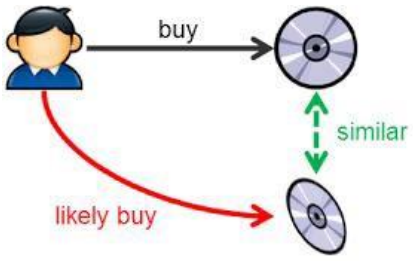


Obr. 4 Styl doporučení v User-based alg.
Zdroj: Seelan,2019

Mezi hlavní klady tohoto řešení patří převážně relevance a popsatelnost těchto doporučení (Konstan, Ekstrand, 2016), protože je rychle dopočitatelné – jací uživatelé jsou danému podobní, co přesně zmínění uživatelé nakoupili nebo ohodnotili. Tato výhoda je sice na první pohled prostá, ale klíčovou se stává ve chvíli použití s ostatními metodami kolaborativní filtrace, které takto snadno již dohledatelné nejsou. Druhým z kladů této metody je časová nenáročnost doporučení, resp. všechny údaje, které jsou k doporučení potřeba již existují v databázi ve chvíli výpočtu modelu a je tedy možné předběžně vypočítat i výsledné doporučení. Tato vlastnost se pojí ke skutečnosti, že uživatelé mění svůj zájem jen málo a stejně tak se v krátkém časovém rozmezí nezmění celý sortiment zboží.

4.4.1.2 Objektová filtrace

Objektová filtrace (item-based filtering) je, dá se říct, opačnou metodou user-based filtrace. Podobnost se v této metodě nepočítá mezi uživateli, ale mezi objekty samotnými. Principem této filtrace je tedy získání vektoru objektu – hodnoty vektoru tvoří samotné hodnocení uživatelů. Na základě tohoto vektoru je následně možné změřit podobnost (vzdálenost) dvou objektů a ustanovit nejpodobnější objekty k právě zpracovávanému objektu.



Obr. 5 Styl doporučení v Item-based alg.
Zdroj: Seelan, 2019

V případě explicitního hodnocení (ratingu) je v rámci tohoto zpracování nutné normalizovat hodnoty jak vstupní (rating uživatelů dle jejich biasu/průměru) tak i výstupní, vložením do normativní škály koncového uživatele. Naopak v případě implicitní zpětné vazby (rankingu) je již škála daná, resp. škálou jsou pouze 0 / 1 a tedy nemusí dojít k normalizaci hodnot a výpočet prostorové vzdálenosti vychází pouze z binárních hodnot. Tento fakt můžeme využít ve srovnání s metodou asociační, kdy "podobnost" objektů počítáme na základě počtu jejich společných výskytů v košíku. Výsledek tohoto srovnání je ten, že

výsledky obou metod budou v některých případech podobné či shodné, a právě díky tomuto zjištění se tyto dvě metody mohou navzájem překrývat, případně upravovat (Konstan, Ekstrand, 2016). Obecně řečeno je ale metoda objektové filtrace daleko variabilnější než je tomu u asociační metody, která v základním tvaru podporuje pouze úpravy ve výpočtu frekvence ale ne výpočet samotný, objektová filtrace dovoluje naproti tomu variaci úprav před výpočtem podobnosti, těmito úpravami tak můžeme dosáhnout rozdílnějších výsledků díky prostorovému posunutí vektoru (Aggarwal, 2016).

Jak již vypovídá z popisu algoritmu, objektová filtrace je úspornější ve výpočtu modelu a následně velikosti modelu, díky faktu, že objektů je většinou méně než uživatelů, proto výpočet, který se tvoří pouze hledáním objektů k objektům (O(p*p)) není tak výpočetně náročný jako u uživatelů, kde se v pozdější fázi výpočtu modelu ještě pro každého uživatele tvoří jeho relevantní seznam (O(u*u+u*p)). Fakt, že je výpočet úspornější je v obecné implementaci správný a mimo jiné – vytvořený model, který obsahuje seznam objektů a k nim seznam objektů relevantních s mírou podobnosti, poskytuje také podporu pro doporučení i uživatelům o kterých nemáme informace ve chvíli tvorby modelu. Na druhou stranu ale v případě, že uživatelé jsou ve většině přihlášeni, a tedy známe jejich minulost (třeba u služeb iTunes, Spotify) je výhoda tohoto modelu zbytečná. Můžeme tedy přesunout výpočet samotných doporučovaných seznamů zpět do tvorby modelu, čímž sice zvětšíme velikost modelu a čas výpočtu, ale získáme čas výpočtu doporučení, který je většinou zpracováván na produkčních strojích, které poskytují i jiné služby napříč webovým řešením, model však může být vypočítán na jiném node a pouze přesunut a načten do paměti.

4.4.2 Faktorizace matic

Ve velkých systémech (například Netflix, Spotify, Amazon) je využití nejbližších sousedů neefektivní a to hned z několika důvodů. Prvním z důvodů je skutečnost, že část sortimentu platformy může být natolik oblíbena jedním segmentem uživatelů, že mimo tento sortiment nikdy doporučení nevznikne, protože žádný z uživatelů v segmentu nekoupí jiný výrobek (Shah, 2018). Prakticky by se tento úkaz dal demonstrovat na e-shopu s počítači, kde existuje část uživatelů, která vždy nakoupí pouze počítač od značky Apple, v případě, že je konkrétní uživatel právě v této skupině, bude s největší pravděpodobností v této skupině i většina jeho "sousedů". Tedy jediný sortiment, který může být doporučen (resp. byl kdy zakoupen), bude znovu pouze od značky Apple. Tento princip může být na jednu stranu správný, protože v některých případech, jako je nejspíše přesně tento, uživatelé od jiné značky počítač prostě nekoupí, ale stejným způsobem to třeba u filmů nefunguje.

Druhým ze zmíněných problémů je ten, že uživatelé mohou mít částečně stejný vkus například v jednom žánru hudby, ale v druhém nikoliv, a i přes to, že v systému existuje stovky tisíc uživatelů ani jeden z nich věrně nereprezentuje právě vkus zmíněného.

Tato problematika vedla firmy a odborníky k vývoji nové formy recommender systémů a tím se stala sada metod používající faktorizaci matic (matrix factorization) (Shah, 2018). Základem této sady metod je matice interakcí uživatele a objektu (User x Item matrix), která je z podstaty věci velmi řídká (doslova z angl. překladu používaného výrazu sparse – vzhledem k počtu polí je vyplněných jen velmi málo). Cílem této metody je umožnit vytvoření matice, která na základě matice interakcí vyplní chybějící hodnoty uživatelů, kteří daný objekt neohodnotili hodnotou nejlepší aproximace (Takács, Pilászy, Németh, Tikk, 2008).

4.4.2.1 Metoda SVD

Pro splnění výše uvedeného cíle je využíváno mnoho metod z lineární algebry, či pravděpodobnostní teorie (Zhang, Liu, 2014). Jednou z nejznámějších je právě metoda SVD (singular value decomposition), která se řadí do metod zmíněné lineární algebry a jejím úkolem je vytvoření tzv. "latentních faktorů", které nejlépe "popisují" matici interakcí (Shah, 2018).

V předešlých kapitolách, při vysvětlování metody content-based filtrací jsme tvořili dvě matice, jednu s objekty a jejich vlastnostmi, která reprezentovala typ objektu dle jeho vlastností a druhou s uživatelskými profily, která reprezentovala hodnotu spojení uživatele

s vlastností, respektive se zájmem uživatele o konkrétní vlastnost. Metoda SVD, potažmo další z metod faktorizace matic, se zakládá na podobném mechanismu – při výpočtu faktorizace hledá "latentní faktory", které reprezentují vlastnosti daných objektů (Shah, 2018). Jinými slovy faktorizací celé matice interakcí můžeme dostat výslednou matici, která reprezentuje vypočítané ("virtuální"/"latentní") vlastnosti objektů a k nim, stejně jako u obsahové filtrace - profily uživatelů dle toho, zda latentní faktor je v zájmu uživatele nebo ne. Latentní faktory / vlastnosti nemůžeme na rozdíl od content-based přístupu jakkoliv popsat či vysvětlit, protože výsledky početní operace mezi maticemi nereprezentují žádnou určitou vlastnost objektu ani uživatele (Shah, 2018). Proto je i v implementacích obtížnější ověření správnosti a vysvětlení samotného doporučení. Výsledkem faktorizace jsou tři oddělené menší matice – matice uživatelů (user-feature affinity matrix, velikost #users X #latentFactors), matice objektů (item-feature relevance matrix, velikost #items X #latentFactors) a nakonec matice vah latentních faktorů / vlastností, která je ortogonální a u každé z vlastností reprezentuje, jak moc vlastnost ovlivňuje prostor interakcí. Matematicky je tento princip zapsán jako vyřešení rovnice (Konstan, Ekstrand, 2016)

$$R = P\Sigma Q^T \quad \textbf{(1)}$$

, kde: R = interakční matice (u x i)

P = user-feature matrice (u x l)

Σ = diagonální matice vah latentních faktorů (l x l) (SVD – singulární čísla)

Q = item-feature matice (i x l)

Způsob této faktorizace je velmi účinný, protože po výpočtu těchto matic stačí pro samotnou predikci hodnocení vynásobení konkrétních členů matic mezi sebou – tím vznikne původní hodnota v interakční matici bez nutnosti udržet v paměti velkou matici interakcí.

Další z výhod této metody je možnost jednoduché aproximace, kdy matice latentních vlastnosti obsahuje vlastnosti "seřazené", tedy první v řádcích matice jsou ty nejrelevantnější vlastnosti. Princip seřazených faktorů můžeme využít ke zmenšení (truncate) této matice o dimenze posledních vlastností (Shah, 2018). Tento způsob řešení se nazývá low-rank approximation a dovoluje nejen uvolnit další místo, protože o stejný počet dimenzí zmenšíme i matice uživatelů a objektů, ale i odstraní šum v hodnocení, který klasicky uživatelé webových aplikací generují například koupí pro člena rodiny z jiné generace, nebo jako

ojedinělý dárek. Výsledkem tohoto zmenšení matice je ale abstrakce od některých méně důležitých detailů a zpětné elementární vynásobení nedosáhne původní matice R, ale pouze její aproximace, kterou nazýváme Rk (k je počet latentních faktorů po jejich omezení).
Jak již bylo nastíněno v minulém odstavci, snížením počtu latentních vlastností sice ztratíme část informací a výsledky nebudou věrné, ale dle průzkumu skupiny Grouplens na datech ze systému Movielens (Konstan, Ekstrand, 2016) tento počin samotné doporučení dokonce zefektivní, právě kvůli faktu, že uživatelé generují šum dat, který odstraněním latentních faktorů potlačíme.

Přesná implementace této faktorizační metody má však jedno velké úskalí a tím je podmínka plné matice interakcí, ze které se faktorizace aplikuje (Vozalis, Markos, Margaritis; 2009). To v případě doporučovacích systému ale není prakticky v žádném systému uspokojeno, protože v případě, že by naprosto každý uživatel ohodnotil naprosto každý produkt by doporučení samotné pozbylo efektu.

Tento problém se v rámci SVD řeší různými metodami (Aggarwal, 2016), první z možných variant je doplnění neznámých hodnot o průměrné hodnoty daného uživatele nebo objektu, druhým z možných řešení je vyplnění hodnot nulami a tím je v algoritmu ignorovat. Tento způsob se jeví jako jeden z nejefektivnějších, ale stejně jako způsob průměrů vygeneruje velmi datově náročnou matici, která je v případě velkého množství uživatelů i objektů z hlediska operační paměti neúnosná. Dalším ze způsobů řešení je tzv. Strojové učení s učitelem - resp. Stochastický gradientní sestup, který se místo klasické faktorizace, která vygeneruje latentní faktory, pokouší faktory odhadnout a minimalizovat chybu mezi předpovědí z odhadnutých vlastností a existujícím hodnocením v interakční matici (Ilic, Kabiljo; 2015).

4.4.2.2 SGD

V předchozí kapitole byla popsána základní varianta SVD metody faktorizace matic, která je efektivně použitelná na interakčních maticích o přiměřených rozměrech (statisíce záznamů v matici), vzhledem k velikosti matic ve velkých systémech (desítky tisíc uživatelů a tisíce objektů - 10 mil záznamů v interakční matici) je metoda velmi pomalá a vzhledem k vytížení operační paměti až nerealizovatelná. Z tohoto důvodu je ve faktorizaci matic používána pro výpočet faktorů metoda strojového učení nazvaná gradient descent – gradientní sestup – tato metoda se obecně řadí mezi optimalizační metody strojového učení,

které zjednodušeně hledají nejlepší možnou hodnotu (hodnoty) pro daný problém na základě chybové funkce (Ilic, Kabiljo; 2015).

Přesněji tato metoda neinicializuje matici latentních faktorů, ale pouze matice uživatelských preferencí a objektových latentních vlastností jako jednotkové matice, či spíše jako matice s náhodnými hodnotami dle normálního rozložení (Ekstrand, Riedl, Konstan; 2011). Následně z těchto matic vypočítává přímo hodnocení a porovnává s hodnocením v původní interakční matici (většinou pomocí RMSE (Ekstrand, Riedl, Konstan; 2011) - viz kapitola 4.5.2.2, či pouze jako $r_i - prediction(i)$ pro konkrétního uživatele). Velikost chyby, kterou algoritmus naměří, může dále zpracovat jako koeficient nutného posunu v maticích pro zlepšení výsledků. Tento, nyní již koeficient, použije do funkce aktualizující matice uživatelů i objektů tak, aby jejich hodnoty byly co nejblíže optimu (resp. aby jejich elementární produkt co nejvíce odpovídal reálným hodnotám).

Tyto aktualizační funkce pro matice P (matice uživatelských preferencí k faktorům) a Q (matice objektových hodnot pro latentní vlastnosti) jsou v případě SGD a uživatelského prostoru: $\frac{\partial e^2}{\partial p}$, respektive i objektového prostoru: $\frac{\partial e^2}{\partial q}$ (Ekstrand, Riedl, Konstan; 2011). Derivace těchto funkcí, konkrétně vypočtený směr tečny, udává směr pohybu změnové funkce a tím i pokles chybové složky. Derivační vzorec dále upravíme pro možnost definování velikosti generované změny a též přidáme normalizační prvek. Ten zajistí, že relace uživatele a objektu neovlivní vlastnost moc velkou mírou, resp. v případě, že by daná relace velkou mírou vlastnost ovlivňovala, nově získané informace by nemohly danou změnu již zvrátit – například časově změněný trend. Po těchto úpravách mohou změnové rovnice vypadat například takto (Ekstrand, Riedl, Konstan; 2011):

$$\triangle q_{il} = \lambda * (error * p_{ul} - \gamma * q_{il}) \quad \textbf{(2)}$$

$$\triangle p_{ul} = \lambda * (error * q_{il} - \gamma * q_{ul}), \text{ kde}$$

$\triangle q_{il}$= vypočítaná změna pro danou latentní vlastnost v daném objektu

$\triangle p_{ul}$= vypočítaná změna pro danou latentní vlastnost u daného uživatele

λ= učící koeficient – jak velké změny se mají v jedné změně provést (obvykle okolo 0.001)

γ= normalizační prvek, který zamezuje příliš velké změně díky jednomu objektu

Tento proces optimalizace hodnot algoritmus provede na velkém množství produktů a je možné specifikovat přesný počet iterací algoritmu nebo průměrnou chybu po které již prohlásíme, že algoritmus je na globálním optimalizačním minimu (Ng, 2016).

Pro samotné doporučení dále stačí pouze elementárně vynásobit matice Q a P, protože matice latentních faktorů již v tomto výpočtu není potřeba.

Metoda gradientního sestupu není jediná použitelná a je jen jednou z mnoha, které poskytuje strojové učení, dalšími jsou například metoda ALS (alternating least squares) nebo expectation maximization. V některých případech se můžeme setkat i s pokročilými metodami postavenými na teorii pravděpodobnosti např. PMF (probabilistic matrix factorization) (Ekstrand, Riedl, Konstan; 2011). Tyto metody jsou však natolik pokročilé a specifické že v rámci této práce nebudou více popisovány a zůstávají jako další možné rozšíření implementované knihovny.

4.4.2.3 Technika folding-in

Princip fungování metody SVD umožňuje jednu z užitečných výhod tohoto algoritmu, která se nazývá "folding-in". Tato technika se zakládá na zjištění, že v rovnici (1) jsou matice P a Q ortogonální, a tedy můžeme vzorec upravit do podoby: $RQ\Sigma^{-1} = P$ (Konstan, Ekstrand; 2016). Tento upravený vzorec dovoluje vypočítat vektor P pro jakéhokoliv nového uživatele i poté, co již byl základní model vytvořen. Obrovský benefit, který z tohoto principu můžeme získat, je ten, že velmi málo algoritmů doporučování dokáže takto "lehce" model samotný upravit – bez nutnosti přepočítání výpočtu základního modelu, který běžně trvá jednotky až desítky hodin. Dokáže tak přidat v rámci runtime aplikace nové uživatele i ve chvíli, kdy právě prochází webovou platformou. Tato technika funguje však pouze do doby, dokud matice P a Q budou ortogonální. Tuto algoritmickou úpravu však není možné použít u techniky gradientního sestupu, ani jeho stochastické úpravy, protože uplatňuje v rámci výpočtu sestupu normalizační faktor, který diagonalitu vyvrací. V několika odborných studiích však vznikla teorie, že v případě malého faktoru změny jsou matice s mírnou odchylkou ortogonální, tedy je možné na ně způsob přidání prvků aplikovat. Tento postup je však velmi závislý na samotné datové sadě a nelze jej (prozatím) prokázat v obecné rovině (Cotter; 2013).

4.5 Metriky hodnocení

Jak již bylo naznačeno v kapitolách o samotných metodách doporučení, metody jsou většinou velmi snadno upravitelné a parametrizovatelné. Pro ověření správnosti a také míry zlepšení jednotlivých metod vznikly i metriky, kterými můžeme jednotlivé metody měřit jak v oblasti chyby predikce, tak v oblasti správného doporučení. Samotné doporučení je velmi subjektivní záležitost, protože každý z uživatelů systému i sám autor konkrétní implementace, má rozdílný vkus na rozdílné produkty, proto mají i tyto metriky svoje nevýhody a měly by být brány v potaz v rámci implementace celé aplikace. V rámci analýzy studií reálných nasazení se velmi málo spoléhá na samotné metriky a jsou spíše podpůrným nástrojem při ověření správnosti, případně výkonnosti. V akademických pracích jsou nicméně měřitelné a spolehlivé v detekci hlídaných údajů.

4.5.1 Příprava dat

Testování samotných algoritmů doporučovacích metod musí probíhat na datech, která nebyla součástí systému ve chvíli, kdy byl vytvářen model. V případě, že by data byla součástí již při samotné tvorbě, algoritmus by tato uložená data zahrnul v modelu – tím by rozhodl/doporučil naprosto správně, protože by již data znal nebo nedoporučil známá data vůbec, protože v rámci implementace by vyfiltroval již známé (ohodnocené) objekty. Z tohoto důvodu se používá při testování dvojí rozdělení dat, jeden segment, příkladem 80 % vložíme do trénovacího algoritmu pro vytvoření modelu – tento segment nazveme trénovací data, druhý segment, zbylých 20 % ponecháme až na samotné testování – nazveme testovací sada dat (Aggarwal, 2016).

Tento způsob se v případě použití strojového učení ještě zdokonaluje tím, že data rozdělíme na N segmentů a následně sloučíme do původního poměru (80 %/20 %), tento běh opakujeme 5x, kdy každý ze segmentů alespoň jednou bude testovacím. Výsledky metrik následně zprůměrujeme, případně hlídáme extrémní změny hodnot. Tento způsob evaluace se nazývá cross-validace a aktuální set se pojmenovává cross-validation set (Ng, 2016).

4.5.2 Detekce chyby

První z kategorií metrik je kategorie detekce chyb, která funguje na principu porovnání predikcí s reálnými hodnotami hodnocení. Tento způsob měření je efektivní v případě explicitního hodnocení a je tedy relevantní pro měření algoritmů uzpůsobených

pro hodnocení v určitém rozsahu. Tyto metriky ale nelze použít pro implicitní zpětnou vazbu, protože rozdíl mezi hodnotami na škále [0,1] by neměl hodnotnou informaci.

4.5.2.1 MAE

MAE – Mean Absolute Error – je nejjednodušší metrika chyby predikce, která vypočítává absolutní chybu v předpovězeném hodnocení vzhledem k reálné hodnotě hodnocení a je definována následujícím vzorcem (Konstan, Ekstrand; 2016):

$$MAE(i) = \Sigma_{i=1}^{n} |\hat{r_i} - r_i| / n \quad \textbf{(3)}$$

kde: r_i =skutečná hodnota hodnocení z testovacích dat

$\hat{r_i}$= predikce hodnoty

n = počet testovaných hodnocení

4.5.2.2 RMSE

Alternativou k metrice MAE jsou metriky MSE a RMSE, kde MSE (mean squared error) počítá odchylku predikce a reálné hodnoty v druhé mocnině, čímž umocní velmi vzdálené hodnoty predikce, ale naruší škálu hodnot – výsledky z této metriky nejsou jednoduše reprezentovatelné. Naproti tomu RMSE (root mean squared error) výsledky dále odmocní, proto vrátí výsledkům normalizovaný tvar a je tedy definován následujícím vzorcem (Konstan, Ekstrand; 2016):

$$RMSE(i) = \sqrt{\Sigma_{i=1}^{n} (\hat{r_i} - r_i)^2 / n} \quad \textbf{(4)}$$

kde: r_i =skutečná hodnota hodnocení z testovacích dat

$\hat{r_i}$= predikce hodnoty

n = počet testovaných hodnocení

4.5.3 Pravděpodobnostní hodnocení

Na rozdíl od metrik měřících chyby v predikci hodnocení využívají pravděpodobnostní metriky povětšinou pouze výskyty, a ne přesné hodnocení. Právě proto jsou použitelné jak pro implicitní zpětnou vazbu, tak i pro explicitní, často též nazývanou "Rank-Aware Top-N Metrics" (Shani, Asela; 2011). Metriky se snaží přiblížit reálné situaci tím, že sledují doporučení pro konkrétní uživatele a hledají, zda v testovacím setu některá

z doporučení existuje. Například v případě, že uživateli byl doporučen produkt A, B a C hodnotí metrika míru pravděpodobnosti, že doporučené tři produkty uživatel koupil, nebo obráceně, nakolik pravděpodobné je, že produkty, které zakoupil, budou doporučené.

4.5.3.1 Precision

Precision je jednou ze základních metrik všech doporučovacích algoritmů a vyjadřuje právě procento doporučení, které se potvrdilo existencí v testovacím setu, tedy – doporučené položky, které uživatel v budoucnosti reálně zakoupil. Tato metrika se tedy vypočítá následujícím vzorcem (Shani, Asela; 2011):

$$Precision(u) = \frac{|\, r(u) \cap t(u) \,|}{|\, r(u) \,|} \quad \textbf{(5)}$$

kde: $r(u)$ =sada doporučených objektů

$t(u)$= sada zakoupených/ohodnocených objektů

4.5.3.2 Recall

Recall je velmi často spojován s metrikou precision a tvoří spolu jakýsi standardní metrický způsob hodnocení algoritmu, protože recall staví ohodnocení algoritmu na podobném principu jako precision, jen s tím rozdílem, že nevypočítává procento doporučených proti reálným, ale obráceně. Počet reálně objednaných proti doporučeným a tím stanový procento objektů, které uživatel koupil/ohodnotil a které zároveň byly součástí doporučení. Rovnice výpočtu je v případě recall metriky následující (Shani, Asela; 2011):

$$Recall(u) = \frac{|\, r(u) \cap t(u) \,|}{|\, t(u) \,|} \quad \textbf{(6)}$$

kde: $r(u)$ =sada doporučených objektů

$t(u)$= sada zakoupených/ohodnocených objektů

4.5.3.3 MAP

Mean average precision je jednou z dalších častěji používaných metrik, která měří stejné hodnoty jako precision, ale koncovou hodnotu dále dělí počtem dotazovaných objektů, tedy menší z proměnných top-K nebo velikostí testovacích objektů - resp. objektů, které uživatel zakoupil v testovacím setu.

$$MAP(u) = \frac{\sum_{i=1}^{I} \boldsymbol{Precision(i)}}{I} \quad \textbf{(7)}$$

kde: $Precision(i)$ =výstup z metriky Precision

I= sada zakoupených/ohodnocených objektů

(Konstan, Ekstrand; 2016)

4.5.3.4 Reciprocal rank

Reciprocal rank (vzájemná hodnost) je rozdílná od předchozích metrik tím, že samotná neměří výskyty samotné, ale vyjadřuje relevantnost objektů dle jejich pořadí. Toto vyhodnocení je měřeno nastavením konstant pro každé pořadí (1/pořadí) v seznamu, tedy pro první objekt v pořadí nastaví hodnotu 1/1, pro druhý 1/2, pro další ⅓ a tak dále. V případě, že se objekt nacházel v seznamu nákupů je hodnota jeho pořadí přidána do součtu a stoupá či klesá jeho relevance. Výsledkem této metriky je údaj znázorňující míru relevance objektů na místech. Tedy čím více objektů, které uživatel opravdu zakoupil bude umístěno mezi prvními, tím lepší hodnocení metrika vypočítá (Konstan, Ekstrand; 2016).

$$RR(u) = \frac{1}{|I|} * \sum_{i=1}^{|I|} \frac{1}{rank_i} \quad \textbf{(8)}$$

kde: $rank_i$ = hodnota pořadí na které se daný objekt nacházel

I= sada zakoupených/ohodnocených objektů uživatelem u

4.5.4 Online hodnocení

Dříve představené metriky hodnocení mají jedno velké úskalí a tím je relevantnost samotných metrik. Představme si například novomanžele nakupující vybavení kuchyně a mající v košíku talíře, misky a vařečky. Doporučovací systém jim logicky nejspíše doporučí nákup příborů a je to i správně, protože další uživatelé, kteří nakoupili stejný sortiment nakoupili i příbory. Daní novomanželé by ale tento příbor s velkou pravděpodobností koupili i bez toho, aby jim to systém doporučil. Tedy: Neměl by v takovýchto případech systém doporučit zboží jiné? Jak systém může rozpoznat takovou situaci? Tento problém je velice komplexní a nelze odpovědět jedním konkrétním řešením. V posledních letech se od klasických metrik pro použití v reálně nasazených recommender systémech ustupuje a zavádí se tzv. online metriky (Konstan, Ekstrand; 2016). Online metriky nelze testovat na

datech samotných a lze je jen mírně automatizovat, protože se jedná o testování samotnými uživateli systému a rozpoznávání jejich reakcí k určitým stylům doporučení.

4.5.4.1 A/B testování

Jedním z těchto online hodnocení je tzv. A/B testování, které je využíváno napříč softwarovým světem ale i například marketingovým (Shani, Asela; 2011). Jedná se o princip segmentace uživatelů – části uživatelů systému je nabídnut jeden typ doporučení, řekněme item-based doporučení, zatímco druhé části uživatelů jsou poskytovány doporučení pomocí user-based metody. Systém v provozu uloží vypočtené a nabídnuté doporučené objekty konkrétnímu uživateli. Výsledkem je zhodnocení, kolik z doporučených objektů uživatel na základě doporučení skutečně zakoupil nebo alespoň zobrazil. V této chvíli už se můžou použít základní metriky jako je Precision a za nějakou přiměřenou dobu (3 týdny - 2 měsíce) vyhodnotí, který z algoritmů měl lepší výsledky, případně jak by se dal dále vylepšit.

V rámci takovéhoto testování je nutné vědět kompletní kontext doporučení pro reprodukci, jak chyb nebo i samotného zlepšení algoritmu, proto systém doporučení musí ukládat a s ním i vše, podle čeho se pro takové doporučení rozhodl (položky košíku, předchozí zobrazení uživatele,..). V případě algoritmů se statickým modelem (kromě faktorizace matic) je možné ukládat pouze datum doporučení a vytvořit při dohledávání model nový z dat noci před doporučením. V případě faktorizace je nutné provádět hotspot zálohy, které se mohou použít i v případě, že by systém zkolaboval a nebylo by možné obnovit celý obraz modelu.

4.6 Speciální a pokročilé metody

Po seznámení s metodami samotného doporučení je snadno viditelné, že metody se dají velmi razantně upravovat a jejich výsledek závisí pouze na jejich samotné implementaci. Základ těchto metod je však prakticky ve všech implementacích podobný a některé stinné stránky konkrétních metod nelze jednoduše odstranit. Právě pro tento případ vznikají nové, komplexnější systémy, které například kombinují několik metod doporučení (hybridní), nebo ukládají výsledek doporučení a následně se snaží metodu vylepšit přímo online (learn-to-rank) (Aggarwal, 2016). Tyto systémy se často používají pro komplexní úlohy doporučení nebo ve velkých systémech, kdy se doporučující systém používá hned v několika různých částech webového prostoru.

4.6.1 Hybridní systémy

Jedním z komplexnějších typů systémů je typ hybridní, který se velmi často používá v případě, že existuje větší množství rozdílných produktů nebo větší množství zdrojů, které nelze aplikovat do jednoho z doporučovacích systémů. Hybridní systém je postaven na jedné či více technikách hybridizace a obsahuje většinou více jiných doporučovacích algoritmů, které pouze skládá v celek.

Příkladem hybridizačních metod může být určení váhy algoritmů, kdy hybridní systém nastavuje (například pomocí A/B testování) váhu nejlepšímu z recommender algoritmů a toho tím prioritizuje v konečných výsledcích (Aggarwal, 2016). V případě, že ani jeden z7algoritmů prozatím nedosáhl svého efektivního optima může dále hybridní systém sloužit pro hodnocení jednotlivých výstupů a k sloučení pouze nejrelevantnějších produktů do finálního doporučení. Dále se může jednat o spojení přímo na datovém stupni, kdy hybridní systém vyhledá uživateli produkty pomocí faktorizace matic, ale dále tyto produkty použije pro doporučení asociativním algoritmem. Využití těchto systémů je nespočetně, a právě proto je v dnešní době můžeme nalézt prakticky ve většině velkých systémů (vlastní výzkum).

5 Praktická část

Pro praktické představení a aplikaci metod uvedených v teoretické části byla v rámci této práce vytvořena knihovna Kera (K-based E-commerce Recommendation Algorithms library) obsahující základní implementace algoritmů doporučení z každé kategorie. Implementace samotná je stavěna na principu rozšiřitelnosti pro snadnější úpravu jednotlivých algoritmů. Jak již bylo vysvětlováno v předešlých částech práce – každý jednotlivý případ nasazení je většinou specifický a pro něj nutné algoritmus adekvátním způsobem upravit dle již nasbíraných dat. I přes tyto fakta je knihovna pokrytá unit testy včetně implementací metrik a může být uplatněna jak pro otestování jednotlivých algoritmů proti novým datasetům, tak i pro reálné nasazení.

Knihovna je vyvinuta pomocí programovacího jazyku Java ve verzi 11 a využívá buildovací systém Maven. Pro účely vývoje byl dále použit podpůrný nástroj Lombok, knihovny Apache commons a json serializéry Jackson. Pro specifické matematické funkce dále Apache commons math3 a knihovna Jama.

Všechny třídy včetně metod jsou dokumentované pomocí javadoc se standardním formátováním.

Pro účel zjednodušení operativních úkolů byl pro knihovnu využit verzovací nástroj git, respektive služba jej poskytující - Gitlab.com (https://gitlab.com/kamest/kera) - v rámci tohoto nástroje byly použity i možnosti tzv. “CI/CD”, které umožňují po nahrání nové verze programu automaticky spouštět funkcionální testy. Možný další vývoj této knihovny a případné optimalizace tedy mohou být od úkonů testování a manuálního vydávání verzí oproštěny. Při publikaci této práce též proběhlo první vydání knihovny s verzí 1.0.

5.1 Zavedení dat a předzpracování

Pro sjednocení práce s daty systému, který knihovnu může používat je v knihovně vytvořeno několik modelových tříd

- KeraItem
    - Reprezentuje objekt systému, který je potencionálně možné doporučit nebo mohl být zakoupen/ohodnocen
    - "produkt"
- KeraOpinion
    - Reprezentuje obecnou interakci uživatele s objektem
    - "koupě", "ohodnocení"
- KeraResultItem
    - Reprezentuje výstupní objekt z knihovny, který obsahuje referenci na původní KeraItem i předpovězenou hodnotu algoritmem
- KeraUser
    - Reprezentuje objekt uživatele systému, který může být hodnotícím/nakupujícím, případně objektem tvorby doporučení

Dále jsou pro vstup informací do knihovny vytvořeny rozhraní poskytovatelů: ItemsProvider, OpinionsProvider a UsersProvider, které implementující systém musí poskytovat pro dodání instancí výše uvedených modelových tříd.

Uživatelské i objektové (resp. výsledkové) modelovací třídy dědí z třídy IdProvider jenž poskytuje atribut externalId. Tento identifikátor je čistě externím unikátním identifikátorem daného uživatele nebo objektu a pro účely knihovny se přemapuje do interního id objektu. Tento princip omezuje velikost dat uložených v maticích a dovoluje použití interních id jako referencí na řádky či sloupce konkrétní matice.

5.2 Implementace algoritmů

Samotná implementace algoritmů je co nejvíce odstíněna od nutných operativních kroků kvůli koncepci knihovny. Celá knihovna v ideálním případě (pokud není potřeba razantně zasahovat do funkcionality) poskytuje veřejné rozhraní pomocí servisní třídy Kera, která poskytuje jednotlivé rozhraní z externích systémů a zároveň drží jednotlivé použité algoritmy včetně práce s jejich modelem a jeho zálohami na disk. Všechny úkony, které jsou

potřeba k použití této knihovny by tedy měly být interpretovány přes veřejné rozhraní právě této třídy.

Dále je pro zjednodušení práce v knihovně zavedena abstraktní třída KeraComputer, která je předkem všech implementací algoritmů a poskytuje právě například mapování externích identifikátorů na interní, api pro práci s vnitřním modelem, uchovává bias data uživatelů, objektů (odchylky průměrného hodnocení pro normalizaci predikcí) atp.

Algoritmy jsou koncipovány pro uplatnění ve dvou fázích, první z nich je fáze tvorby modelu/trénování a následně již samotné doporučení. Veřejné api k těmto úkonům poskytuje samotný předek KeraComputer pomocí abstraktních metod:

- List<KeraResultItem> recommend(String userId, int n, boolean forbidKnownItems)
    - Poskytuje n doporučení pouze na základě historických dat uživatele (userId) se specifikací, zda známé objekty (byly již ohodnoceny/koupeny) mohou být v seznamu doporučení
- List<KeraResultItem> recommend(String userId, int n, List<KeraItem> referenceItems, boolean forbidKnownItems)
    - Poskytuje n doporučení na základě referenčních produktů (např. produktů v košíku) - pokud tento způsob algoritmus dovoluje
- KeraMatrixModel computeModel(Boolean isRanking)
    - Vypočítá základní model algoritmu z poskytnutých dat
- void computeExtModel(Boolean isRanking)
    - Speciální metoda pro algoritmy, které využívají externích knihoven kdy výstupem těchto použití není standardizovaný model (KeraMatrixModel)

Dalším z přidaných funkcionalit knihovny je možnost využití tzv. masterId, které lze uvést v objektu KeraItem. Tento atribut signalizuje předka produktu například v případě variantních produktů a pro samotné doporučení je použit pouze první (nejrelevantnější) předmět obsahující konkrétní masterId, ostatní jsou vyfiltrovány.

5.2.1 Metoda průměrného hodnocení (ItemMeanPopularityComputer)

Implementace metody průměrného hodnocení je založena na aplikaci základní jednoduché rovnice průměru pro explicitní hodnocení:

$$prediction(i) = \frac{\sum_{u=0}^{U} r_{ui}}{|U|} \quad \textbf{(9)}$$

a pro implicitní (ranking) pouze samotný součet výskytů

$$prediction(i) = \sum_{u=0}^{U} r_{ui} \quad \textbf{(10)}$$

Doporučení je následně provedeno pouhým seřazením a filtrací referenčních objektů.

5.2.2 Asociační metoda (ItemAssociationComputer)

Asociační metoda je z implementačního hlediska též jednodušší, protože jejím základem je následující vzorec (Fátima, Bruno, 2016):

$$A(i,j) = \frac{|Ui \cap Uj|}{|Ui|} \quad \textbf{(11)}$$

, kde U_i = počet hodnotících uživatelů objektu i

U_j = počet hodnotících uživatelů objektu j

Implementováno v knihovně Kera následovně:

```
// For each item computes similar items
itemUsers.forEach((key, value) → {
    Map<Integer, Double> itemScores = new HashMap<>();
    itemUsers.forEach((key1, value1) → {
        AtomicInteger countOfCommonUsers = new AtomicInteger( initialValue: 0);
        // Compare iu matrix and get count of users that rated / bought both items
        countOfCommonUsers.addAndGet((int) value.stream().filter(value1::contains).count());
        // "Normalize" for avoiding situation that one
        // item is highly bought and one almost never but they has same users
        itemScores.put(key1, countOfCommonUsers.get() / (double) value.size());
    });

    // Put only high valued tom n items
    keraMatrixModel.put(
            key,
            itemScores
                    .entrySet() Set<Map<K, V>.Entry<Integer, Double>>
                    .stream() Stream<Map<K, V>.Entry<Integer, Double>>
                    .sorted(Collections.reverseOrder(Comparator.comparingDouble(Map.Entry::getValue)))
                    .limit(neighborhoodSize)
                    .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue)));

});
```

Obr. 6 Implementace objektové asociace v knihovně Kera

Zdroj: Kameník, 2020

Doporučení v tomto případě opět vypočítáme pouze seřazením, filtrací a omezením n členů.

5.2.3 TF-IDF (TFIDFComputer)

Při implementaci jedné z algoritmů content-based metody bylo zapotřebí rozšířit modelové třídy o některé dodatečné informace. Konkrétně byla do třídy KeraItem přidána vlastnost List<String> tags, která poskytuje seznam štítků navázaných na daný produkt.

Implementace v rámci vytváření modelu se dělí na výpočet TF, výpočet IDF a následnou normalizaci výstupního vektoru s následujícími definicemi (Konstan, Ekstrand, 2016):

TF:

$$TF(i,t) = \frac{|N_{it}|}{|N_i|} \quad \textbf{(12)}$$

, kde i = právě zpracovávaný objekt

t = aktuálně zpracovávaný štítek

N_{it} = počet výskytů štítku t v objektu i

N_i= počet všech unikátních štítků v objektu i

```
// Compute TF vector
List<KeraItem> items = itemsProvider.getObjects();
items.forEach(item → {
    Map<Integer, Double> tagSum = new HashMap<>();
    Set<Integer> addedTags = new HashSet<>();
    List<String> tags = Optional.ofNullable(item.getTags())
            .orElse(Collections.emptyList());
    tags
            .stream()
            .map(this::addTagToIndex)
            .forEach(tagIndex → {
                tagSum.put(tagIndex, tagSum.containsKey(tagIndex) ? tagSum.get(tagIndex) + 1.0 : 1.0);
                if (!addedTags.contains(tagIndex)) {
                    addedTags.add(tagIndex);
                    df.put(tagIndex, df.containsKey(tagIndex) ? df.get(tagIndex) + 1.0 : 1.0);
                }
            });
    tagSum.entrySet().forEach(i→i.setValue(i.getValue() / tags.size()));
    itemVectors.put(addItemToIndex(item.getExternalId()), tagSum);
});
```

Obr. 7 Implementace Term frequency části v rámci algoritmu TF-IDF

Zdroj: Kameník, 2020

IDF:

$$IDF(t) = log(\frac{|I|}{|I_t|}) \quad \textbf{(13)}$$

, kde t = aktuálně zpracovávaný štítek

I = počet všech objektů v databázi

I_t= počet objektů s přiřazeným štítkem t

Výsledek rovnic (12) a (13) nakonec vynásobíme mezi sebou, čímž získáme konečnou hodnotu TF-IDF. Vektor nakonec ještě znormalizuje, pomocí Euklidovské vzdálenosti – pro ušetření paměti a využití v kosinově podobnosti, převod do jednotné škály.

Implementováno v knihovně Kera následovně:

```
// Compute IDF
df.entrySet().forEach(e → e.setValue(Math.log(items.size() / e.getValue())));

KeraMatrixModel modelData = new KeraMatrixModel();
itemVectors.forEach((key, value) → {
    Map<Integer, Double> tv = new HashMap<>(value);
    tv.entrySet().forEach(e → e.setValue(e.getValue() * df.get(e.getKey())));

    // Normalize tag vec
    double sumSquares = VectorUtils.sumOfSquares(tv);
    double norm = Math.sqrt(sumSquares);
    tv.entrySet().forEach(e → e.setValue(e.getValue() / norm));

    modelData.put(key, tv);
});
```

Obr. 8 Implementace IDF části v rámci algoritmu TF-IDF

Zdroj: Kameník, 2020

Doporučení v tomto algoritmu probíhá na principu tvorby vektoru uživatele, který se může vytvářet dvojím způsobem dle proměnné isWeighted. V případě váženého profilu (isWeighted = true) je výpočet uživatelského vektoru následující (Konstan, Ekstrand, 2016):

$$uv_{ut} = \sum_{t=0}^{|T|}(r_{ui} - \mu_u) * q_{it} \quad \textbf{(14)}$$

, kde T= počet všech štítků v hodnocených objektech

r_{ui} = hodnocení objektu zpracovávaným uživatelem

μ_u = průměrné hodnocení uživatele

q_{it}= výsledek metody tf-idf pro daný objekt a štítek

```java
// Get mean of users opinions
double mean = opinions
        .stream()
        .mapToDouble(KeraOpinion::getOpinion).sum() / (double) opinions.size();
opinions.forEach(i -> {
    Map<Integer, Double> itemVector = getModel().get(getItemId(i.getItemId()));
    for (Map.Entry<Integer, Double> e : itemVector.entrySet()) {

        // Normalize value and apply mean
        double weight = e.getValue() * (i.getOpinion() - mean);
        if (userVector.containsKey(e.getKey())) {
            // Add other posible cumulations
            weight += userVector.get(e.getKey());
        }
        userVector.put(e.getKey(), weight);
    }
});
```

Obr. 9 Vážené hodnocení vlastnosti v algoritmu TF-IDF

Zdroj: Kameník, 2020

V případě neváženého profilu nebo implicitních hodnocení je výpočet následující:

$$uv_{ut} = \sum_{t=0}^{|T>=Threshold|} q_{it} \quad \textbf{(15)}$$

, kde Threshold je konstanta hodnocení pro vymezení pouze kladných hodnocení ke zpracování (např. 3.5 na škále 0-5), tato konstanta je pro implicitní hodnocení zanedbána.

```java
// Gets only those opinions that are over threshold
Stream<Map<Integer, Double>> mapStream = opinions
        .stream()
        .filter(r -> isRanking() || r.getOpinion() ⩾ ratingThreshold)
        .map(r -> getModel().get(getItemId(r.getItemId())));
// Computes user - tag vector
mapStream
        .filter(Objects::nonNull)
        .flatMap(itemVector -> itemVector.entrySet().stream())
        .forEach(e -> userVector.put(e.getKey(),
                userVector.containsKey(e.getKey())
                        ? e.getValue() + userVector.get(e.getKey()) : e.getValue()));
```

Obr. 10 Hodnocení vlastnosti stylem above-threshold v algoritmu TF-IDF

Zdroj: Kameník, 2020

Samotné hodnocení objektů je pak provedeno porovnáním pomocí kosinové podobnosti:

$$cos(p_u, q_i) = \frac{\sum_{t=0}^{|T|} q_{ut} * p_{it}}{\sqrt{\sum_{t=0}^{|T|} q_{ut}^2} * \sqrt{\sum_{t=0}^{|T|} p_{it}^2}} \quad \textbf{(16)}$$

```
.forEach(item → {
    Map<Integer, Double> itemVector = getModel().get(item);
    double itemNorm = VectorUtils.sumOfSquares(itemVector);
    if (itemNorm ≠ 0.0 && userNorm ≠ 0.0) {

        // Computes item euclidean normalization value
        itemNorm = Math.sqrt(itemNorm);

        AtomicDouble userItem = new AtomicDouble( initialValue: 0.0);
        userVector
                .entrySet() Set<Map<K, V>.Entry<Integer, Double>>
                .stream() Stream<Map<K, V>.Entry<Integer, Double>>
                .filter(j → itemVector.containsKey(j.getKey()))
                .forEach(j → userItem.addAndGet( delta: j.getValue() * itemVector.get(j.getKey())));

        // Compute final score
        double score = userItem.get() / (itemNorm * userNorm);
        max.set(Math.max(score, max.get()));
        min.set(Math.min(score, min.get()));
        String itemId = getItemId(item);
        results.add(new KeraResultItem(items.get(itemId), score));
    }
});
```

Obr. 11 Doporučení na základě modelu TF-IDF

Zdroj: Kameník, 2020

Možné zlepšení:

V rámci této konkrétní implementace nemají jednotlivé štítky zadané explicitní váhy (priority) a knihovna se k nim tedy chová jako k naprosto rovnocenným. Tento způsob chování simuluje vztah produktu a parametru, kde parametry produktu nemají (většinou) dané explicitní priority. Možné (neimplementované) vylepšení by však mohlo být při předávání například stromu kategorií jako štítků s prioritou klesající po hledání ve stromu vzestupně.

5.2.4 User-based algoritmus (UserUserNNComputer)

User-based algoritmus implementovaný v knihovně je znovu rozdělen na část výpočtu modelu a na část samotného doporučení. V tomto algoritmu však veškeré výpočty počítají s informacemi, které již existují v systému v době výpočtu modelu, proto doporučovací

metoda samotná pouze řadí předpočítané výsledky a veškerý výpočet jak podobnosti uživatelů, tak samotného doporučení jednotlivých objektů provádí ve výpočtu modelu. Tento princip má jednu velkou výhodu – v rychlosti doporučení, kdy složitost doporučení závisí pouze na limitu objektů, které do modelu přenáší výpočet.

Model je vytvářen dvěma oddělenými procesy v rámci každého uživatele systému. V prvním z nich se k aktuálnímu uživateli vypočítá jeho normalizovaný uživatelský vektor a dle něj kosinova podobnost s vektory všech ostatních uživatelů (dle rovnice (16) v algoritmu TF-IDF). Tímto procesem vznikne seznam "sousedů" uživatele, kteří nakoupili, nebo ohodnotili stejné zboží. Tento seznam algoritmus seřadí dle podobnosti a omezí na zadaný počet využívaných sousedů. V druhé fázi výpočtu algoritmus prochází všechny objekty systému a pro každý z nich, v případě explicitního hodnocení, vytvoří součet součinu – hodnocení souseda * podobnost k sousedovy, čímž vznikne údaj o relevantnosti objektu samotného. V případě implicitního hodnocení se využije pouze podobnost uživatele, který objekt zakoupil. Matematicky je pak vyjádření pro explicitní hodnocení následující (Konstan, Ekstrand, 2016):

$$p_e(u,i) = \mu_u + \frac{\sum_{v=0}^{|V|} cos(u,v)*(r_{v,i}-\mu_v)}{\sum_{v=0}^{|V|} |cos(u,v)|}, \quad \textbf{(17)}$$

pro implicitní pouze: $p_i(u,i) = \sum_{v=0}^{|V|} cos(u,v)$,

kde V = podobní uživatelé aktuálnímu u, kteří interagovali s daným produktem 'i',

$r_{v,i}$ = hodnocení objektu i uživatelem 'v',

cos(u,v) = kosinova similarita uživatelů 'u' a 'v'

5.2.5 Item-based algoritmus (ItemItemNNComputer)

Princip item-based algoritmu je velmi podobný algoritmu user-based. V této metodě však nejdříve algoritmus přetransformuje interakce (opinions) do matice Map<Objekt,Map<Uživatel, Hodnocení>>. Tato proměnná algoritmu dovoluje jednoduše normalizovat vektory objektů a následně stačí pouze vypočítat kosinovou podobnost mezi samotnými objekty, čímž získá nejpodobnější objekty (nejbližší sousedy) každému ze zpracovávaných objektů systému.

```
KeraMatrixModel model = new KeraMatrixModel();
itemVectors.forEach((key1, value1) → {
    Map<Integer, Double> similarities = new HashMap<>();
    itemVectors.forEach((key, value) → {
        if (key.equals(key1)) return;
        // Computes cosine similarity
        double sumUV = VectorUtils.dotProduct(value1, value);
        double similarity = sumUV / (VectorUtils.euclideanNorm(value1) * VectorUtils.euclideanNorm(value));
        // Only if items are at least a bit similar, we don't want to recommend upon negativity (
        if (similarity > 0) {
            similarities.put(key, similarity);
        }
    });
    if (!similarities.isEmpty()) {
        model.put(
                key1,
                similarities
                        .entrySet() Set<Map<K, V>.Entry<Integer, Double>>
                        .stream() Stream<Map<K, V>.Entry<Integer, Double>>
                         .sorted(Collections.reverseOrder(Comparator.comparingDouble(Map.Entry::getValue)))
                         .limit(getNeighborhoodSize())
                        .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue)) Map<Integer, Double>
        );
    }

});
```

Obr. 12 Implementace tvorby modelu algoritmu objektové filtrace

Zdroj: Kameník, 2020

Doporučení je následně oproti user-based technice složitější, v případě doporučení pouze na základě historie nákupů, vyhledá nakupované objekty a z vypočítané matice převezme jejich nejbližší sousedy. Tyto sousedy následně porovná (s přiřazením váhy podobnosti stejně jako u user-based) a sečte vypočítané skóre pro jednotlivé sousedy produktů. Matematicky (Aggarwal, 2016):

$$prediction(i,u) = \mu_i + \frac{\sum_{j=0}^{|I_u|}(r_{uj}-\mu_j)*w_{ij}}{\sum_{j=0}^{|I_u|}|w_{ij}|} \quad \textbf{(18)}$$

```
List<KeraResultItem> results = new ArrayList<>();
itemsProvider.getObjects().forEach(i -> {
    Integer itemExtId = getItemId(i.getExternalId());
    // Cannot recommend unknown item
    if (opinions.containsKey(itemExtId)) return;
    // Get similar to this item
    Map<Integer, Double> itemNeighbors = getModel().get(itemExtId);

    if (itemNeighbors == null) return;
    Set<Map.Entry<Integer, Double>> entrySet = itemNeighbors.entrySet();
    AtomicDouble weightSum = new AtomicDouble( initialValue: 0);
    AtomicDouble ratingSum = new AtomicDouble( initialValue: 0);
    entrySet
            .stream()
            .filter(j -> opinions.containsKey(j.getKey()))
            .sorted(Collections.reverseOrder(Map.Entry.comparingByValue()))
            .limit(neighborhoodSize)
            .forEach(j -> {
                Double weight = j.getValue();
                // Weight neighbors with known opinion from this user
                ratingSum.addAndGet(isRanking() ? weight : weight * opinions.get(j.getKey()));
                weightSum.addAndGet(weight);
            });
    if (ratingSum.get() == 0) return;
    // Normalize score in case of ratings , in case of ranking only push ratingSum
    double score =
            isRanking() ? ratingSum.get()
                    : Optional.ofNullable(itemMeans.get(itemExtId)).orElse( other: 0.0) + ratingSum.get()
                        / weightSum.get();

    results.add(new KeraResultItem(i, score));
});
```

Obr. 13 Doporučení na základě modelu z alg. objektové filtrace

Zdroj: Kameník, 2020

5.2.6 Faktorizace matic

Jak již bylo nastíněno v teoretické části práce, faktorizace matic může být velmi specifická a upravitelná dle jednotlivých implementací a použití. Z tohoto důvodu je v knihovně použita pouze základní varianta SVD (respektive SVD bez použití SGD). Knihovna je však plně připravena pro možné doimplementování těchto algoritmů pomocí implementace třídy KeraComputer, která poskytuje možnost uložení a obsluhy matic vycházejících z těchto komplexnějších postupů.

5.2.6.1 SVD (JamaSVDComputer)

Způsob výpočtu základního modelu ALS je velmi časově a paměťově náročný, jednotlivé implementace proto musí být co nejefektivnější. Z těchto důvodů a z důvodu časové náročnosti samotné implementace a optimalizace byl v knihovně po nezdařených pokusech použit výpočet pomocí knihovny Jama (možné použít i Apache commons math3

implementaci). Pro toto použití bylo zapotřebí implementovat mechanismus baseline, který udržuje uživatelské, objektové i globální odchylky hodnocení a je jej možné použít pro normalizaci vstupních dat do SVD. Data samotná se následně vyfaktorují pomocí Jama implementace, která vrací tři matice (Jama.Matrix) - U, V a singularValues. Tyto matice po výpočtu omezí na počet nastavených latentních faktorů v proměnné latentFactorCount.

```
// Create user-item matrx
Matrix matrix = Matrix.identity(userIndex.size(), itemIndex.size());
opinionsProvider.getObjects() List<KeraOpinion>
        .stream() Stream<KeraOpinion>
        .filter(i → !isRanking() || (userIndex.get(i.getUserId()) ≠ null &&  userIndex.get(i.getUserId()).isRegistered()))
        .forEach(i → {
            int itemId = addItemToIndex(i.getItemId());
            int userId = addUserToIndex(i.getUserId());
            // Fill with normalized value from biases
            // Again rankings are already "normalized"
            double value = isRanking()
                    ? i.getOpinion()
                    : i.getOpinion() - getGlobalBias() - getUserBias().get(userId) - getItemBias().get(itemId);

            matrix.set(userId, itemId, value);
        });

// Construct SVD and then truncate to only count of latent factors stored in var latentFactorCount
SingularValueDecomposition svd = new SingularValueDecomposition(matrix);
Matrix um = svd.getU();
Matrix im = svd.getV();

// Truncate latent factors and store UM, IM, Weights
setUserMatrix(um.getMatrix( i0: 0,  i1: um.getRowDimension() - 1,  j0: 0,  j1: latentFactorCount - 1));
setItemMatrix(im.getMatrix( i0: 0,  i1: im.getRowDimension() - 1,  j0: 0,  j1: latentFactorCount - 1));
setWeight(new ArrayRealVector(Arrays.stream(svd.getSingularValues()).limit(latentFactorCount).toArray()));
```

Obr. 14 Implementace použití SVD knihovny JAMA

Zdroj: Kameník, 2020

Doporučení následně vypočítáme elementárním vynásobením matic a v případě explicitního hodnocení dále součtem s bias daty (odchylkami).

```
List<KeraResultItem> results = new ArrayList<>();
itemsProvider.getObjects().forEach(i → {
    if (forbiddenItems.contains(i.getExternalId())) return;
    RealVector weights = new ArrayRealVector(getWeight());
    Integer itemId = getItemId(i.getExternalId());
    // Item not known
    if (itemId == null) return;
    // Get item vector from matrix
    RealVector itemVector = new ArrayRealVector(getItemMatrix().getArray()[itemId]);
    // Eval dot product of UV ** W ** IV
    double prediction = userVector.ebeMultiply(weights).dotProduct(itemVector);
    // Only in case of rating, normalize
    if (!isRanking())
        prediction += getGlobalBias() + getUserBias().get(intUserId) + getItemBias().get(itemId);
    results.add(new KeraResultItem(i, prediction));

});
```

Obr. 15 Doporučení v případě použití faktorizace matic

Zdroj: Kameník, 2020

5.2.7 Implementace metrik

Pro úplnost knihovny a měření efektivity implementovaných algoritmů byla dále implementována i řada metrik uvedených v teoretické části. Tyto metriky mají za předka třídu Evaluator, která poskytuje výsledky měření metodou

$$double\ eval(Map\ testSet, Map\ recommendations)$$

Metriky jsou rozděleny do dvou package – error a precision. Evaluátory v package error jsou podmíněny použitím na datech s explicitním hodnocením, a proto jsou i v testech takto opodmínkovány, naopak v rámci balíčku precision jsou metriky využívány pro každý algoritmus.

Vysvětlení samotných implementací již bylo připraveno v teoretické části práce, proto zde kvůli nižší složitosti implementace již nebudou vysvětlovány konkrétními příklady.

6 Shrnutí výsledků

Jak bylo prezentováno v teoretické části práce, algoritmy, jejichž cílem je doporučení, mohou být velmi výrazně upraveny a mít odlišné funkce pro měření (například podobnosti). Kvůli této funkcionalitě je velmi obtížné samotné algoritmy testovat a ověřovat. Většinu z implementovaných algoritmů je možné ověřit alespoň početně, s čímž pracují vytvořené unit testy v knihovně – tím ověřují správnost implementací funkcí. Testování efektivity samotných algoritmů však kvůli nedostatku zkušeností autora muselo probíhat ve stylu porovnání autorových metrik s jinou knihovnou, která dané algoritmy či jejich alternativy též implementuje.

Porovnání bylo provedeno na stejné sadě dat movielens 100k (Harper, Konstan, 2015), která je v oboru doporučovacích systémů jednou z nejpoužívanějších testovacích sad. Sada vznikla ze stejnojmenného projektu (Movielens), kterým je webový prohlížeč filmového katalogu. Samotná organizace Grouplens, již je vedena odborníky na doporučovací systémy z University of Minnesota a spravuje webovou prezentaci, vyvíjí na tomto webu nové techniky doporučování.

Jako porovnávané knihovny byly ze začátku vybrány dvě – Lenskit a Librec. Hlavním důvodem tohoto výběru byl programovací jazyk java, na kterém obě z knihoven závisí a který autorovy umožňuje lepší orientaci v rámci unit testů v samotné knihovně. Naskýtá také možnost využití pokročilých technik doporučování z této knihovny v některé z dalších prací v tomto tématu. Knihovna Lenskit však na podzim roku 2018 ukončila svůj vývoj a byla kompletně přepsána do programovacího jazyka Python. Při průzkumu dalších možných kandidátů jako alternativ implementované knihovny byly vyhledány desítky projektů v jazyce java, které postupem času přestaly mít aktivní vývoj. Z větších projektů zůstala pouze výzkumná knihovna Librec a knihovna Spark Mllib, která velmi úzce souvisí s platformou Spark a není proto jednoduše implementovatelná do jednodušších projektů.

6.1 Porovnání výsledků s knihovnou Librec

Prvním z porovnávaných metrik jsou metriky chyby. Výsledky jsou vždy zaokrouhleny na 5 desetinných míst a testovány zvlášť pro explicitní i implicitní hodnocení. Způsob implementace algoritmu TF-IDF pro vlastnosti objektů (též využíváno pro text dokumentů) není v rámci knihovny Librec implementován, proto jsou uvedeny výsledky pouze z knihovny Kera.

Item association algoritmus:

Ranking:

	PRECISION	RECALL	RR	MAP
Kera	0.64617	0.05339	0.85302	0.55164
Librec	0.26808	0.16578	0.58561	0.20669

Tabulka 1: Srovnání algoritmu Item association s implicitními daty

Zdroj: Kameník, 2020

TF-IDF algoritmus:

Rating:

	MAE	MSE	RMSE
Kera	0.82463	0.97307	0.98644

Tabulka 2: Hodnocení algoritmu TF-IDF s explicitními daty

Zdroj: Kameník, 2020

Ranking:

	PRECISION	RECALL	RR	MAP
Kera	0.82795	0.10346	0.96539	0.75380

Tabulka 3: Hodnocení algoritmu TF-IDF s implicitními daty

Zdroj: Kameník, 2020

Item – based algoritmus:

Rating:

	MAE	MSE	RMSE
Kera	0.52678	0.51002	0.71415
Librec	0.75062	0.99369	0.99684

Tabulka 4: Srovnání Item-based algoritmu s explicitními daty

Zdroj: Kameník, 2020

Ranking:

	PRECISION	RECALL	RR	MAP
Kera	0.26496	0.01526	0.52059	0.15816
Librec	0.29798	0.17961	0.54916	0.21726

Tabulka 5: Srovnání Item-based algoritmu s implicitními daty

Zdroj: Kameník, 2020

User – based algoritmus:

Rating:

	MAE	MSE	RMSE
Kera	0.48440	0.40999	0.64030
Librec	0.86047	1.19024	1.09098

Tabulka 6: Srovnání User-based algoritmu s explicitními daty

Zdroj: Kameník, 2020

Ranking:

	PRECISION	RECALL	RR	MAP
Kera	0.79095	0.07686	0.97495	0.73585
Librec	0.27815	0.17745	0.58745	0.21842

Tabulka 7: Srovnání User-based algoritmu s implicitními daty

Zdroj: Kameník, 2020

SVD algoritmus:

Rating: (alternativa librec – SVD++)

	MAE	MSE	RMSE
Kera	0.46494	0.37714	0.61412
Librec	0.71427	0.8278	0.90983

Tabulka 8: Srovnání SVD algoritmu s explicitními daty

Zdroj: Kameník, 2020

Ranking: (alternativa librec – ALS)

	PRECISION	RECALL	RR	MAP
Kera	0.75719	0.06510	0.93994	0.68541
Librec	0.32810	0.21345	0.64782	0.26075

Tabulka 9: Srovnání SVD algoritmu s implicitními daty

Zdroj: Kameník, 2020

6.2 Výsledky testování na reálných datech

V rámci této bakalářské práce byly též osloveny dvě společnosti provozující e-shop s žádostí o poskytnutí dat pro ověření použitelnosti i na českých reálných prostředích. Tyto data jsou anonymizované sady produktů v objednávkách s referencí na objednávku i anonymizovanou referencí na uživatele. Z pohledu doporučovacích algoritmů se tedy jedná pouze o implicitní data simulující reálnou aplikaci knihovny v košíku uživatele dle zadání práce.

6.2.1 Data e-shop projektu s potravinami

Tento datový set obsahuje 2711 produktů s 86 914 ohodnoceními, resp. nakoupenými položkami.

	PRECISION	RECALL	RR	MAP
Item association	0.36589	0.59556	0.62750	0.49786
TF-IDF	0.25802	0.46326	0.62608	0.38060
Item – based	0.04886	0.05728	0.17464	0.03713
User – based	0.42139	0.47795	0.97867	0.55441
SVD	0.37295	0.35406	0.85066	0.43427

Tabulka 10: Srovnání využití algoritmů na datech e-shopu s potravinami

Zdroj: Kameník, 2020

6.2.2 Data e-shop projektu Signal-nabytek.cz společnosti J&H online s.r.o.

Tento datový set obsahuje 69 767 produktů s 97 172 ohodnoceními, resp. nakoupenými položkami. E-shop Signal-nabytek.cz dále obsahuje funkcionalitu popsanou v praktické části – master-variant. Tato funkcionalita dovoluje na platformě vytvořit podprodukty produktů, které mají společnou většinu parametrů až na variantní (barva, velikost). Obsahují nicméně i různé kódy, a tedy jsou z pohledu externích systémů produkty odlišné. Z tohoto důvodu bylo nutné doplnit do metrik výpočet pouze na základě id nadřazeného produktu, protože vzhledem k objemu produktů by metriky se zvolením samostatných id variantních produktů nikdy nedosahovaly žádných výsledků.

	PRECISION	RECALL	RR	MAP
Item association	0.09934	0.93632	0.74074	0.77440
TF-IDF	0.06428	0.57189	0.44936	0.60537
Item - based	0.00744	0.05656	0.02506	0.02617
User - based	0.08884	0.63461	0.69747	0.96061
SVD	0.04495	0.33844	0.24920	0.26612

Tabulka 11: Srovnání využití algoritmů na data e-shopu signal-nabytek.cz

Zdroj: Kameník, 2020

7 Závěry a doporučení

Cílem této bakalářské práce byla implementace a návrh algoritmu, který na základě anonymizovaných dat nákupů ostatních uživatelů doporučí uživateli v jednom z kroků košíku další potencionální produkty ke koupi.

V rámci teoretické části této práce byly zavedeny základní problémy týkající se doporučování a obecný úvod do tohoto sektoru, dále byly představeny a prakticky vysvětleny základní mechanismy výpočtu doporučení a jejich modelu s hlavními metodami měření efektivity a správnosti implementací.

V praktické části práce byla následně implementována knihovna Kera jako ukázková aplikace představených metod s jejich matematickým vysvětlením. Závěrem této práce je shrnutí výsledků aplikace jednotlivých metrik na implementované algoritmy.

Vzhledem k výsledkům v porovnání knihoven Librec a Kera je možné s použitím mírné odchylky prohlásit implementace v této práci za funkční. Výsledky dokazují, že implementace v knihovně Kera dosahuje dobré efektivity u metrik Precision, Reciprocal rank a Mean average precision. Zatímco v metrice Recall dosahuje výsledků horších nežli knihovna Librec. Skutečnost horších výsledků můžeme vysvětlit jako nedostatek zobecnění algoritmu, kdy implementované algoritmy nedokážou doporučit zboží, které se nachází mimo segment jeho sousedů. Měřením na poskytnutých datech e-shopu s potravinami, můžeme zhodnotit vyšší použitelnost algoritmu objektové asociace, protože metriky splnil nejlépe z implementovaných algoritmů. Naopak item-based algoritmus měl na těchto datech výsledky spíše špatné a dle následného studia dospěl autor k zjištění, že tento problém souvisí s chybou v předešlém měření, kde algoritmus vyhledá okolí objektu, ale objekty v tomto okolí jsou si natolik podobné, že algoritmus nikdy nedoporučí objekt mimo tento segment a tím všechny ostatní objekty, které mohou být v testovacím setu, vyřadí.

Data z e-shop systému signal-nabytek.cz byla od ostatních velmi odlišná, částečně protože e-shop obsahuje produkty ve vztahu master-variant a též protože poměr produktů a interakcí je velmi nízký. Algoritmy pracující se sousedstvím objektů musí počítat s velkým rozptylem sousedství. Tato skutečnost též zapříčinila v metrikách horší výsledky – na rozdíl od dat z e-shopu s potravinami jsou některé výkony až na polovině hodnoty efektivity.

Výše uvedené problémy potvrzují hypotézu č. 2, která zamítá předpoklad univerzální metody použitelné napříč aplikacemi. Uvedené výsledky sice naznačují, že metoda asociace

měla ve většině případů dobrou efektivitu, nesplňuje však podmínky plné personalizace a výkonově by nejspíše nebyla použitelná ve větším množství dat (výpočet modelu dat signal-nabytek.cz - ~25 min). Ostatní algoritmy s použitím různých datových sad výrazně oscilovaly ve svých hodnotách metrik, proto jimi hypotézu č. 2 opět potvrzujeme.

Vzhledem k definici uvedené v kapitole 4.4.2.2 o gradientním sestupu musíme částečně potvrdit i hypotézu č. 1, která zamítá předpoklad použití strojového učení pro doporučování zboží na základě aktuálního košíku uživatele. Tento předpoklad vyvrací fakt, že použitím strojového učení ztrácí matice vlastnost ortogonality. Kvůli tomuto jevu nemůžeme aplikovat techniku "folding-in", která by hypotézu vyvracela. Zanedbáváme-li experimentální výzkum s použitím techniky na částečně ortogonálních maticích (sám autor konceptu tohoto výzkumu prohlašuje velkou závislost na zdrojových datech systému) (Cotter; 2013) hypotézu č. 1 v obecné rovině potvrzujeme.

V rámci implementací byly shledány různé problematické situace, které znesnadňují obecné implementace jednotlivých algoritmů (jako je například poměr interakcí, funkcionality e-shop systému atp.). Nutno zmínit, že aktuální verze knihovny počítá pouze se základními z nich, respektive s těmi, které byly součástí zdrojových dat. Možný pokračující výzkum by ale tyto další vlastnosti systému mohl do knihovny zahrnout a parametrizovat/rozšířit jednotlivé algoritmy natolik, že samotná aplikace na jednotlivá řešení nebude vyžadovat programátorské úpravy. Dalším možným zájmem výzkumu může být samotná implementace celé faktorizace matic s technikou "folding-in" nebo použití metody strojového učení, případně experimenty po vzoru uvedené částečné ortogonality v kapitole 4.4.2.3.

Závěrem této práce stojí za zmínku role praktického testu samotných doporučování, kdy například při testování algoritmu Item-based na datové sadě e-shopu s potravinami byla uživateli, který měl v košíku produkt mouky bez lepku, doporučena naprosto totožná mouka s lepkem. Tento výsledek psychicky sráží hodnotu celého doporučení a v případě takto velkého rozporu i hodnotu samotného e-shopu. Nedá se však klasickými metodami odhalit v metrikách. Tato skutečnost umocňuje význam použití například A/B testování, explicitního vyjádření k dodaným doporučením nebo přímo v tomto případě i použití hybridního doporučení s využitím algoritmů kolaborativní filtrace a obsahové filtrace.

8 Seznam obrázků

9 Seznam tabulek

10 Seznam zdrojů

1) Tištěné zdroje

[1] Aggarwal,Charu C. 2016. Recommender Systems: The Textbook (1st. ed.). Springer Publishing Company, Incorporated. ISBN 978-3-319-29657-9

[2] Fátima Rodrigues, Bruno Ferreira - Product Recommendation based on Shared Customer's Behaviour [online]. 2016. Přístup z Internetu: http://www.sciencedirect.com/science/article/pii/S1877050916323018

[3] Jawaheer, Gawesh & Szomszor, Martin & Kostková, Patty. Comparison of implicit and explicit feedback from an online music recommendation service. [online]. 2010. Přístupné z Internetu: https://www.researchgate.net/publication/247927239_Comparison_of_implicit_and_explicit_feedback_from_an_online_music_recommendation_service

[4] Zisopoulos, Charilaos & Karagiannidis, Savvas & Demirtsoglou, Georgios & Antaris, Stefanos. [2008]. Content-Based Recommendation Systems. Přístupné z internetu: https://www.researchgate.net/publication/236895069_Content-Based_Recommendation_Systems

[5] Desarkar, Maunendra & Sarkar, Sudeshna.User based Collaborative Filtering with Temporal Information for Purchase Data. KDIR 2012 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval. [2012]. Přístup z internetu: https://www.researchgate.net/publication/273126122_User_based_Collaborative_Filtering_with_Temporal_Information_for_Purchase_Data

[6] Shah, Shalin. Introduction to Matrix Factorization for Recommender Systems. [2018]. Přístupné z internetu z internetu: https://www.researchgate.net/publication/333934237_Introduction_to_Matrix_Factorization_for_Recommender_Systems

[7] Takács, Gábor & Pilászy, István & Németh, Bottyán & Tikk, Domonkos. Investigation of Various Matrix Factorization Methods for Large Recommender Systems. Proceedings - IEEE International Conference on Data Mining Workshops, ICDM Workshops 2008. [2008]. Přístupné z internetu: https://www.researchgate.net/publication/220765749_Investigation_of_Various_Matrix_Factorization_Methods_for_Large_Recommender_Systems

[8] Zhang, Zhijun & Liu, Hong. (2014). Application and Research of Improved Probability Matrix Factorization Techniques in Collaborative Filtering. International Journal of Control and Automation. Přístupné z internetu: https://www.researchgate.net/publication/276855047_Application_and_Research_of_Improved_Probability_Matrix_Factorization_Techniques_in_Collaborative_Filtering

[9] Vozalis, Manolis & Markos, Angelos & Margaritis, Konstantinos G.. [2009]. Evaluation of standard SVD-based techniques for Collaborative Filtering. Přístup z internetu: https://www.researchgate.net/publication/253996195_Evaluation_of_standard_SVD-based_techniques_for_Collaborative_Filtering/citation/download

[10] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. [2011]. Collaborative Filtering Recommender Systems. Foundations and Trends® in Human-Computer Interaction 4(2), Přístupné z internetu: https://md.ekstrandom.net/pubs/cf-survey.pdf

[11] Andrew Cotter, Stochastic optimization for machine learning,[2013] Přístupné z internetu: https://arxiv.org/pdf/1308.3509.pdf

[12] Shani, Guy & Gunawardana, Asela. [2011]. Evaluating Recommendation Systems. Přístupné z internetu: https://www.researchgate.net/publication/226264572_Evaluating_Recommendation_Systems

[13] Pandey, Kamlesh & Pradhan, Narendra. (2018). Internet Search Engine: Performance Evaluating the Google, Yahoo and Bing Web Search Engine based on their Searching Capabilities. Přístupné z internetu: https://www.researchgate.net/publication/324482784_Internet_Search_Engine_Performance_Evaluating_the_Google_Yahoo_and_Bing_Web_Search_Engine_based_on_their_Searching_Capabilities

[14] Dennis, Charles & Merrilees, Bill & Jayawardhena, Chanaka & Wright, Len. (2009). E-consumer behaviour. European Journal of Marketing. Přístupné z internetu: https://www.researchgate.net/publication/42622087_E-consumer_behaviour

2) Internetové zdroje

[1] Konstan Joseph A.,Ekstrand,Michael D., Studijní materiály specializace Recommender systems 2016, Přístupné z internetu: https://www.coursera.org/specializations/recommender-systems

[2] Aleksandar Ilic, Maja Kabiljo, Recommending items to more than a billion people. [2015] https://engineering.fb.com/core-data/recommending-items-to-more-than-a-billion-people/

[3] Andrew Ng, Studijní materiály kurzu Machine learning 2016, Přístupné z internetu: https://www.coursera.org/learn/machine-learning

[4] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (2015) Přístup z internetu: https://grouplens.org/datasets/movielens/

[5] Techjury.com - Statistika registrovaných domén [online]. 2020. Přístup z Internetu: https://techjury.com

[6] Heureka.cz - detail produktu telefonu iPhone SE (2020) [online]. 2020. Přístup z Internetu: https://mobilni-telefony.heureka.cz/apple-iphone-se-2020-64gb/

[7] Heureka.cz - formulář hodnocení zboží [online]. 2020. Přístup z Internetu: https://mobilni-telefony.heureka.cz/samsung-c3350-xcover-2/pridat-uzivatelskou-recenzi/?shopId=1197

[8] Signal-nabytek.cz - detail kategorie botníků [online]. 2020. Přístup z Internetu: https://www.signal-nabytek.cz/cs/botniky

[9] Seelan Guna, "Recommendation System" (obrázky) [online]. 2019. Přístup z Internetu: https://medium.com/@gunacse1996/recommendation-system-19a53913da16

[10] Český statistický úřad. Rozvoj informační společnosti v České republice a zemích EU [online].2018.Přístup z internetu: https://www.czso.cz/documents/10180/94876554/06202618.pdf/b7166a40-93fa-41a4-bb4c-ee06722d51e8?version=1.2

3) Ostatní zdroje

[1] E-shop signal-nabytek.cz - zdroj testovacích dat, společnost J&H online s.r.o.. Webové stránky: https://www.signal-nabytek.cz/

[2] Platforma Edee.one na níž byly testovací data vytvořena. Webové stránky: https://www.edee.one/

[3] Tvůrce platformy Edee.one a poskytovatel podpůrných prostředků práce. Webové stránky: https://www.fg.cz

[4] Java - programovací jazyk. Webové stránky: https://www.oracle.com/java/

[5] Apache maven project. Webové stránky: https://maven.apache.org/

[6] Apache commons. Webové stránky: https://commons.apache.org/

[7] Jackson Project. Webové stránky: https://github.com/FasterXML/jackson

[8] JAMA: A Java Matrix Package. Webové stránky: https://math.nist.gov/javanumerics/jama/

[9] LibRec. Webové stránky: https://github.com/guoguibing/librec

[10] Původní implementace knihovny Lenskit v jazyku java. Webové stránky: https://java.lenskit.org/

[11] Apache Spark MLlib. Webové stránky: https://spark.apache.org/mllib/

[12] Programovací jazyk Python. Webové stránky: https://www.python.org/

[13] Repozitář vyvinuté knihovny Kera. Webové stránky: https://gitlab.com/kamest/kera

11. Přílohy

1) Zdrojové kódy knihovny Kera s testovacími anonymizovanými daty

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Štěpán Kameník**

Studium: I1800691

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Aplikace machine learning algoritmu na doporučení zboží ke koupi**

Název bakalářské práce AJ: Application of machine learning algorithms on purchase recommendation

Cíl, metody, literatura, předpoklady:

Cílem bakalářské práce je implementace a otestování algoritmu pro doporučovací systém, který bude na základě anonymizovaných dat z předchozích nákupů - tj. seznamů společně nakoupeného zboží, doporučovat na základě aktuálního obsahu košíku další položky zboží ke koupi.

University of Washington: Collaborative filtering course materials

Machine learning for recommender systems series on Medium

Machine learning guide by ocdevel

Stanford: Machine learning course materials

Garantující pracoviště: Katedra informačních technologií, Fakulta informatiky a managementu

Vedoucí práce: prof. Ing. Vladimír Bureš, Ph.D., MBA

Oponent: Ing. Jan Hruška

Datum zadání závěrečné práce: 21.10.2014