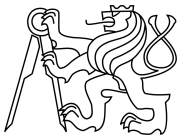
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| **Název:** | Evaluace algoritmů maticové faktorizace v kolaborativním filtrování |
| **Student:** | Bc. Tomáš Richtr |
| **Vedoucí:** | Ing. Tomáš Řehořek |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Znalostní inženýrství |
| **Katedra:** | Katedra teoretické informatiky |
| **Platnost zadání:** | Do konce letního semestru 2016/17 |

## Pokyny pro vypracování

Prozkoumejte metody maticové faktorizace používané v doporučovacích systémech. Proveďte rešerši dostupných implementací faktorizačních algoritmů pro velká data (miliony záznamů), případně implementujte algoritmus vlastní. Navrhněte postup pro porovnání algoritmů založených na faktorizaci s algoritmem nejbližších sousedů z hlediska různých metrik (recall, catalog coverage). Proveďte sadu experimentů na několika dostupných datasetech (např. MovieLens). Zhodnoťte úspěšnost algoritmů a diskutujte dosažené výsledky.

## Seznam odborné literatury

Dodá vedoucí práce.

L.S.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 18. února 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Diplomová práce

# Evaluace algoritmů maticové faktorizace v kolaborativním filtrování

***Bc. Tomáš Richtr***

Vedoucí práce: Ing. Tomáš Řehořek

6. května 2016

# Poděkování

Děkuji vedoucímu mé práce, Ing. Tomáši Řehořkovi, že si na mě udělal čas kdykoliv jsem potřeboval konzultovat probíhající práci, a děkuji mu také za četné připomínky a konstruktivní nápady, které během těchto konzultací k mé práci vznášel. Za další cenné připomínky děkuji také Ing. Pavlu Kordíkovi, Ph.D., který mě rovněž upozornil na některé zajímavé vědecké články související se zpracovávanou problematikou. Rovněž bych chtěl poděkovat Bc. Ondřeji Fiedlerovi za pomoc s rozchozením některých částí softwarové infrastruktury společnosti Recombee, s.r.o.. Mé poděkování patří i společnosti GoOut, s.r.o., že poskytla data pro účely provedení experimentů v rámci této diplomové práce. V neposlední řadě děkuji mé rodině za pochopení a podporu, které se mi z jejich strany při ne vždy lehké práci dostávalo.

# Prohlášení

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

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

V Praze dne 6. května 2016 .....

### Odkaz na tuto práci

Richtr, Tomáš. *Evaluace algoritmů maticové faktorizace v kolaborativním filtrování.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

# Abstrakt

Tato práce se zabývá metodami maticové faktorizace v kolaborativním filtrování. Po počáteční analýze metod maticové faktorizace je navržen postup pro vyhodnocení faktorizačních modelů z hlediska metrik recall a catalog coverage. Tento návrh je realizován a je provedena sada experimentů na datasetech MovieLens a GoOut s algoritmem stochastického gradientního sestupu implementovaného knihovnami LIBMF a Apache Mahout. Výsledkem práce jsou výstupy provedených experimentů a připravený proces pro efektivní vyhodnocování faktorizačních algoritmů.

**Klíčová slova** Maticová faktorizace, Singulární rozklad, Stochastický gradientní sestup, Vyhodnocování, Knihovna LIBMF, Knihovna Apache Mahout, RMSE, Catalog coverage, Recall

# Abstract

This diploma thesis is concerned with matrix factorization methods in collaborative filtering. After the initial analysis of matrix factorization methods a procedure for evaluation of factorization models in recall and catalog coverage metrics is designed. This design is implemented and the set of experiments on

the MovieLens and GoOut datasets is done with stochastic gradient descent algorithm implemented by LIBMF and Apache Mahout libraries. The results of this thesis are outputs of done experiments and prepared process for effective evaluation of factorization algorithms.

**Keywords** Matrix factorization, Singular value decomposition, Stochastic gradient descent, Evaluation, LIBMF library, Apache Mahout library, RMSE, Catalog coverage, Recall

# Obsah

# Seznam obrázků

# Úvod

Jedním ze způsobů, jak zatraktivnit různé webové portály, sociální sítě nebo internetové obchody, je přizpůsobovat jejich obsah či nabídku konkrétním uživatelům, kteří je navštěvují, aby nebyli zahlceni velkým množstvím nabízeného obsahu, ale mohli se rychle dostat k tomu, co je skutečně zajímá. K tomuto přizpůsobování obsahu lze využít modely generující personalizovaná doporučení pro jednotlivé uživatele. Používání těchto doporučovacích systémů je v dnešní době běžné už jen z důvodu snadné dostupnosti výpočetních prostředků a malých nároků na zavedení těchto systémů – zejména v případě metod kolaborativního filtrování (kdy jsou doporučení generována na základě dříve uskutečněných interakcí uživatelů s nabízeným obsahem) není třeba věnovat nasazení doporučovacích modelů velké úsilí.

Metody maticové faktorizace typu SVD mají široké možnosti použití, vhodné uplatnění ale nachází právě i v oblasti kolaborativního filtrování, kde vždy dochází ke zpracování velkého množství dat. Tato metoda umožňuje extrahovat z dostupných dat skryté informace (latentní příznaky), popisující preference jednotlivých uživatelů a charakterizující nabízené položky. Na základě těchto údajů pak mohou být efektivně určovány položky vhodné k doporučení pro konkrétní uživatele. Navíc může maticová faktorizace redukovat množství zpracovávaných dat, aniž by došlo k významné ztrátě informace, a může zohledňovat i různé dodatečné vstupy, jako třeba v čase se měnící zaujetí uživatelů. Proto se metoda maticové faktorizace stává v poslední době stále populárnější alternativou ke klasickému přístupu generování doporučení pomocí algoritmů nejbližších sousedů.

V této práci se zabývám analýzou různých algoritmů maticové faktorizace a možnostmi jejich použití v oblasti kolaborativního filtrování. Práce dále obsahuje návrh procesu pro efektivní provádění experimentů s touto metodou, tedy návrh provádění faktorizace dat používaných doporučovacími systémy a vyhodnocování kvality doporučovacích modelů, pracujících na základě těchto faktorizovaných dat. Tento návrh je realizován a je provedena sada experimentů se zvolenou metodou maticové faktorizace na dvou vybraných data-

setech. Prezentace výsledků těchto experimentů představuje převážnou část čtvrté kapitoly. V závěru práce je pak porovnána kvalita doporučovacích modelů používajících maticovou faktorizaci s existujícím algoritmem nejbližších sousedů a jsou diskutovány důvody dosažených výsledků.

Kapitola 1

# Cíl práce

Zadání této práce pochází od společnosti Recombee s.r.o., která vyvíjí a provozuje doporučovací systém, s jehož pomocí nabízí generování personalizovaných doporučení jako B2B cloudovou službu. Celý proces funguje tak, že zákazníci (např. internetové obchody) nejprve poskytnou této společnosti data o svých uživatelích a o nabízených produktech (položkách). Poté posílají na server společnosti Recombee požadavky na vygenerování doporučení konkrétních produktů pro toho uživatele, který si právě prohlíží stránky jejich obchodu. Doporučení jsou generována v reálném čase a odesílána zpět k zákazníkovi, který je na svých webových stránkách zobrazí cílovému uživateli.

Uvedená společnost používá dnes pro generování doporučení různé metody kolaborativního filtrování, historicky využívala hlavně algoritmus nejbližších sousedů (zejména verzi User k-NN). Tento algoritmus byl po dobu existence společnosti značně optimalizován, aby generoval doporučení v co nejkratším možném čase. Algoritmus User k-NN ovšem odvádí veškerou práci právě v průběhu vybavování (generování doporučení), kdy hledá nejbližší sousedy napříč velkou množinou dostupných dat (případně napříč všemi daty), což ho pro generování doporučení v reálném čase nečiní nejvhodnějším. Dalším problémem je pak paměťová náročnost, kdy v případě velkého objemu dat není snadné nahrát všechna tato data do operační paměti.

Právě z důvodu připravenosti na velká data (miliony a více záznamů) se společnost Recombee rozhodla obrátit svou pozornost také k dalším metodám kolaborativního filtrování, zejména k těm, které předzpracovávají poskytnutá data již v průběhu učení doporučovacích modelů (jedná se například o předpočítávaný Item k-NN model nebo Asociační pravidla). Předzpracováním dat v průběhu učení je redukováno množství operací prováděných během vybavování a proces generování doporučení tak oproti algoritmu User k-NN urychlují. Jednou z těchto metod, která získala v poslední době na popularitě (například díky jejímu úspěchu v soutěži Netflix Prize), je metoda maticové faktorizace typu SVD. Tato metoda faktorizuje předložená data ve formě matice (typicky velké, ale řídké) na dvě matice výrazně menších dimenzí. Tyto matice se tak

snáze vejdou do paměti a také jejich zpracování během procesu vybavování může být výrazně rychlejší.

Cílem této práce je tedy prozkoumat metody maticové faktorizace v doporučovacích systémech odvozené od metody SVD a nalézt vhodného kandidáta, se kterým budou provedeny potřebné experimenty pro získání prvních zkušeností s metodami maticové faktorizace z hlediska různých metrik (recall, catalog coverage). Výsledky těchto experimentů mohou pomoci nejen společnosti Recombee v jejím dalším rozhodování o praktickém nasazení těchto algoritmů, ale mohou být zajímavé i pro další akademické výzkumníky. Pro provedení vyhodnocení zvoleného algoritmu bude potřeba tento algoritmus implementovat, nebo lépe, nalézt existující a již odladěnou implementaci pro faktorizaci velkých dat, která by byla volně poskytována například ve formě knihovny pro použití v dalších aplikacích. Cílem je dále navrhnout postup pro provádění experimentů a tento návrh realizovat, a konečně provést s faktorizačním algoritmem sadu experimentů na datasetech z akademické sféry i z průmyslu, které svým rozsahem odpovídají očekávanému množství dat zpracovávaných v uvedené společnosti.

Účelem plánovaných experimentů je zejména ověření schopnosti faktorizačních algoritmů dosahovat dostatečně malé chyby při predikci hodnot trénovacích dat, tedy schopnosti maximálně se přiblížit optimálnímu řešení faktorizace. Dále je účelem experimentů zjištění časové náročnosti faktorizace velkých dat a zjištění hodnot metrik recall a catalog coverage dosahovaných při doporučování na testovacích datech. Tyto výsledky poslouží nakonec ke vzájemnému porovnání doporučovacích modelů založených na maticové faktorizaci s existujícím doporučovacím modelem, používajícím algoritmus nejbližších sousedů ve verzi User k-NN, a to právě z hlediska metrik recall a catalog coverage.

Kapitola 2

# Analýza metod maticové faktorizace

Při tvorbě doporučení pracují modely vždy se dvěma typy subjektů, které spolu nějakým způsobem interagují. Těmito subjekty jsou obvykle uživatelé (users) a položky (items), přičemž položky mohou představovat zboží v internetovém obchodě, videa či filmy v internetových televizích, nebo třeba příspěvky na sociálních sítích. Stejně tak interakce mohou být různého druhu, například v případě internetového obchodu to může být zakoupení položky uživatelem nebo třeba jen detailní prohlédnutí položky při výběru zboží. Uvedené příklady, kdy je zachycováno chování uživatele, představují tzv. implicitní zpětnou vazbu. Dalším klasickým typem interakce je explicitní uživatelské ohodnocení položky – např. počtem hvězdiček z dané škály. To už dává systému přímou informaci o vhodnosti konkrétní položky pro daného uživatele a v tomto případě hovoříme o explicitní zpětné vazbě.

Metody pro tvorbu doporučení spadají podle [1] do dvou kategorií – metody filtrování obsahu a metody kolaborativního filtrování. V případě metod filtrování obsahu vyžaduje model explicitní informace, které charakterizují a popisují dané subjekty, může jít třeba o atributy konkrétních položek a atributy preferované jednotlivými uživateli. Explicitní informace musí být modelu dopředu poskytnuty, například formou manuálního vyplnění profilu každého jednotlivého subjektu, a na základě těchto informací model určuje, které položky jsou pro dané uživatele relevantní a mají jim být doporučeny.

Naproti tomu metody kolaborativního filtrování explicitní informace nevyžadují a pro tvorbu doporučení využívají zejména informace implicitní, nejčastěji právě historii interakcí mezi subjekty. Hlavní přístupy, které spadají mezi metody kolaborativního filtrování, jsou metody prohledávání okolí a modely latentních příznaků [1].

Nejznámějším představitelem metod prohledávání okolí je algoritmus nejbližších sousedů (k-NN), který v datech vyhledává podobné uživatele nebo položky (varianty User k-NN nebo Item k-NN), přičemž podobnost je určována

na základě známých interakcí. Například ve variantě User k-NN algoritmus předpokládá, že podobní uživatelé (nazývaní sousedé) budou preferovat stejné položky, takže danému uživateli doporučí ty položky, se kterými interagovali jeho sousedé. Algoritmus nejbližších sousedů obvykle v průběhu učení pouze ukládá data a dále je zpracovává až v průběhu vybavování, což proces vybavování zpomaluje. Výhodou tohoto přístupu je ale skutečnost, že algoritmus při tvorbě doporučení vždy pracuje se všemi dostupnými daty, včetně nejnovějších záznamů.

Modely latentních příznaků (např. modely maticové faktorizace) naopak zpracovávají poskytnutá data zejména v průběhu učení, kdy se snaží odvodit takzvané latentní příznaky (latent features) pro jednotlivé uživatele a položky, kterými se snaží vysvětlit známé interakce mezi subjekty. Latentní příznaky jsou obvykle vektory reálných čísel menší dimenze, než je původní prostor interakcí. Například to mohou být vektory obsahující od 20 do 100 hodnot [1], přičemž ale dimenze závisí často na komplexitě konkrétního problému. Latentní příznaky představují algoritmem odvozené charakterizace subjektů, v případě internetového obchodu mohou třeba příznaky položek definovat abstraktní druh zboží (tento druh zboží nemusí odpovídat kategorizaci, kterou obchod skutečně používá) a příznaky uživatelů preference jednotlivých abstraktních druhů. Korespondující latentní příznaky uživatele a položek (respektive vektory těchto příznaků) pak určují relevantní položky k doporučení, přičemž korespondující latentní vektory lze jednoduše najít pomocí jejich vzájemného vynásobení. Výhodou těchto modelů je rychlé vybavování, neboť charakterizace uživatelů a položek jsou dopředu známé, naopak ale vyžadují delší dobu pro učení a s nově přicházejícími daty je nutné učení opakovat, tak aby latentní příznaky odpovídaly i nejnovějším datům.

## 2.1 Metody maticové fakorizace

Existuje více možností jak hledat latentní příznaky, podle [1] jsou však jedněmi z nejúspěšnějších modelů latentních příznaků metody využívající maticovou faktorizaci typu SVD. Vstupem pro učící algoritmus těchto modelů je matice interakcí, kde typicky řádky představují uživatele a sloupce položky, hodnoty matice pak udávají interakci mezi uživatelem a položkou. Matice mohou být unární (např. zaznamenávající uskutečněné prodeje), binární (např. zpětná vazba uživatele, zda se mu daná položka líbí nebo nikoliv), nebo mohou obsahovat reálná čísla (např. explicitní hodnocení položky uživatelem). Typ zpracovávané matice má přitom zásadní vliv na zvolený způsob faktorizace – například minimalizace chyby RMSE (viz. dále) na unární matici pozbývá smyslu, neboť optimálním řešením jsou triviálně nalezitelné matice dávající v součinu výslednou matici jedniček. Společnou vlastností všech uvedených matic je jejich značná velikost, kdy nejsou výjimkou miliony uživatelů i položek. Zároveň však matice obsahují pouze nízký počet pozorovaných hodnot,

neboť každý uživatel typicky interaguje jen se zlomkem nabízených položek. Kvůli vysokému počtu neznámých hodnot bývají matice interakcí často označovány za řídké matice.

Metody maticové faktorizace se snaží najít latentní příznaky rozkladem (faktorizací) matice interakcí $\mathbf{A}$ na matice $\mathbf{U}$ a $\mathbf{V}$, tak aby ideálně platilo

$$\mathbf{A} = \mathbf{U}\mathbf{V}^T$$

přičemž pokud má matice $\mathbf{A}$ rozměry $m \times n$, pak matice $\mathbf{U}$ má rozměry $m \times k$ a matice $\mathbf{V}$ má rozměry $n \times k$. Řádky $\mathbf{p_u}$ matice $\mathbf{U}$ představují vektory latentních příznaků uživatelů a řádky $\mathbf{q_i}$ matice $\mathbf{V}$ představují vektory latentních příznaků položek. Parametr $k$ je dimenze těchto latentních vektorů, takže platí $\mathbf{p_u}, \mathbf{q_i} \in \mathbb{R}^k$.

Základní metodou pro tvorbu modelů latentních příznaků je podle [2] metoda SVD (singular value decomposition), která má ale pro praktické využití některá omezení, jako problémy s neznámými hodnotami matice $\mathbf{A}$ nebo obtížnější paralelizaci. Proto existuje řada dalších, z metody SVD odvozených algoritmů, které hledají matice příznaků $\mathbf{U}$ a $\mathbf{V}$ odlišnými způsoby. Například jde o algoritmus gradientního sestupu nebo metodu ALS (Alternating least squares). Tyto algoritmy dokážou nalézt rozklad matice $\mathbf{A}$ s určitou malou chybou, takže pro vektory $\mathbf{p_u}$ a $\mathbf{q_i}$ platí vztah

$$\hat{r}_{u,i} = \mathbf{p_u}\mathbf{q_i}^T$$

kde $\hat{r}_{u,i}$ je aproximace známé hodnoty $r_{u,i}$ matice $\mathbf{A}$. Zásadní skutečností je, že se znalostí latentních příznaků všech subjektů lze pomocí uvedeného vztahu odhadovat také všechny neznámé hodnoty $r_{u,i}$ matice $\mathbf{A}$ a tyto hodnoty použít k určení vhodných položek k doporučení.

## 2.2 Metoda SVD

Jak udává [3], každou matici reálných čísel $\mathbf{A}$ o rozměrech $m \times n$, jejíž hodnost je rovna hodnotě $r$, lze rozložit na součin tří matic:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

kde $\mathbf{U}$ je ortogonální matice o rozměrech $m \times m$, jejíž sloupce jsou vlastními vektory matice $\mathbf{A}\mathbf{A}^T$, $\mathbf{V}$ je ortogonální matice o rozměrech $n \times n$, jejíž sloupce jsou vlastními vektory matice $\mathbf{A}^T\mathbf{A}$, a $\mathbf{S}$ je diagonální matice o rozměrech $m \times n$, která má $r$ nenulových prvků $\sigma_i$, pro něž platí $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. Hodnoty $\sigma_i$ jsou druhými odmocninami vlastních čísel matic $\mathbf{A}\mathbf{A}^T$ i $\mathbf{A}^T\mathbf{A}$ a nazývají se singulární hodnoty matice $\mathbf{A}$. Podobně sloupce matic $\mathbf{U}$ a $\mathbf{V}$ jsou označovány jako singulární vektory.

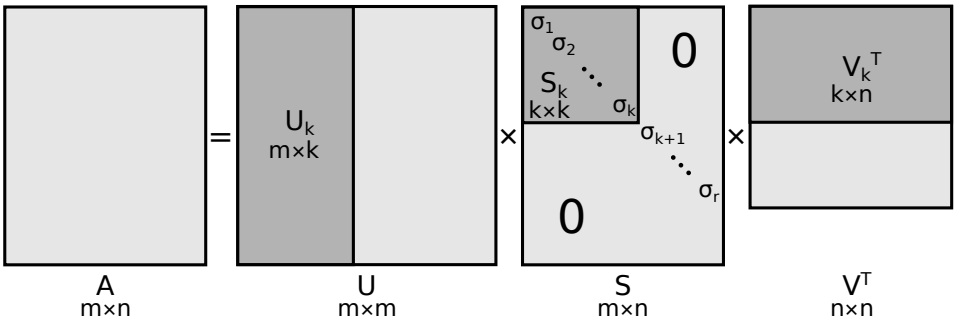
Pomocí singulárního rozkladu (SVD) lze nalézt přesné numerické řešení matic $\mathbf{U}$,$\mathbf{S}$ a $\mathbf{V}^T$. Podle [4] je pro nalezení matice $\mathbf{U}$ třeba vypočítat matici

$\mathbf{AA}^T$ a určit vlastní čísla $\lambda_i$ této matice. Dále je třeba najít vlastní vektory $\mathbf{u_i}$ matice $\mathbf{AA}^T$ příslušné k vlastním číslům $\lambda_i$ a nalezené vektory normovat na jednotkovou velikost. Výsledné vektory tvoří sloupce matice $\mathbf{U}$. Se znalostí vlastních čísel $\lambda_i$ můžeme dále sestavit matici $\mathbf{S}$. Matici $\mathbf{V}$ lze nalézt obdobným způsobem jako matici $\mathbf{U}$, přičemž se ale vychází z matice $\mathbf{A}^T\mathbf{A}$.

Protože pro hodnost matice $\mathbf{S}$ platí $h(\mathbf{S}) = h(\mathbf{A}) = r$, a zároveň platí $r \leq min(m, n)$, nemá smysl uchovávat matice $\mathbf{U}$, $\mathbf{S}$ a $\mathbf{V}$ v celé své původní velikosti. Odstraněním nulových řádků či sloupců matice $\mathbf{S}$, posledních $(m-r)$ sloupců matice $\mathbf{U}$, a posledních $(m-r)$ sloupců matice $\mathbf{V}$, se řešení nezmění. Případná další redukce dimenzí má již za následek určitou ztrátu informace, nicméně pro účely tvorby doporučení není typicky nutné mít k dispozici přesné řešení. Další redukce lze dosáhnout tak, že si ponecháme pouze prvních – a tudíž nejvýznamnějších – $k$ singulárních hodnot v matici $\mathbf{S}$ a řádky a sloupce se zbývajícími singulárními hodnotami odstraníme [4]. Spolu s odstraněnými singulárními hodnotami můžeme odstranit i příslušné sloupce matic $\mathbf{U}$ a $\mathbf{V}$, jde vždy o posledních $(r-k)$ sloupců. Výsledkem redukce dimenzí je soustava

$$\mathbf{A_k} = \mathbf{U_k}\mathbf{S_k}{\mathbf{V_k}}^T$$

kde matice $\mathbf{U_k}$ má rozměry $m \times k$, matice $\mathbf{S_k}$ má rozměry $k \times k$ a matice $\mathbf{V_k}$ má rozměry $n \times k$. Matice $\mathbf{A_k}$ je blízkou aproximací původní matice $\mathbf{A}$, přičemž tato aproximace je nazývána low-rank aproximací [2]. Výsledný redukovaný prostor má výhodu rychlejšího zpracování při tvorbě doporučení, navíc podle [2] odstraněním nejméně významných singulárních hodnot dojde zároveň k odstranění nežádoucího šumu obsaženého v původní matici $\mathbf{A}$.




Obrázek 2.1: Singulární rozklad matice $\mathbf{A}$ na matice $\mathbf{U}$, $\mathbf{S}$ a $\mathbf{V}^T$. Tmavší barvou je znázorněna redukce dimenzí při zachování $k$ nejvýznamnějších singulárních hodnot – výsledkem jsou matice $\mathbf{U_k}$, $\mathbf{S_k}$ a ${\mathbf{V_k}}^T$.

Metoda SVD má však tu nevýhodu, že pro výpočet rozkladu matice $\mathbf{A}$ je třeba znát všechny hodnoty této matice, takže před samotnou faktorizací musí být neznámé hodnoty matice $\mathbf{A}$ vyplněny. Jednou z možností je vyplnit

chybějící hodnoty nulami, vhodnější je ale podle [5] vyplnit chybějící hodnoty průměry ze známých hodnot, přičemž lze volit mezi řádkovými nebo sloupcovými průměry. Z praktického hlediska je však tento postup nepoužitelný, neboť produkuje datově neúměrně velké záznamy, které lze stěží uložit např. do operační paměti počítače.

### 2.2.1 Algoritmy SVD

Pro výpočet matic $\mathbf{U_k}$, $\mathbf{S_k}$ a $\mathbf{V_k}$ z původní matice $\mathbf{A}$ existuje více možných algoritmů [6]. Snahou je vždy najít singulární čísla matice $\mathbf{A}^T\mathbf{A}$ bez nutnosti počítání této matice samotné. Předpokladem vždy je, že pro rozměry matice $\mathbf{A}$ platí $m \geq n$, pokud tato podmínka neplatí, lze vycházet z matice $\mathbf{A}^T$. Mnohé z algoritmů pro SVD pracují podle [6] na principu redukce matice $\mathbf{A}$ na bidiagonální matici a následném výpočtu singulárních čísel a singulárních vektorů z této bidiagonální matice. Jiné algoritmy iterují přímo nad maticí $\mathbf{A}$ a vyžadují explicitní zadání maximální požadované chyby $\varepsilon$ pro kontrolu ukončení iterování. Platí přitom, že faktorizace, která má nalézt přesné numerické řešení, je výpočetně velmi náročná.

Protože matice interakcí $\mathbf{A}$ se v průběhu času mění (například přicházejí noví uživatelé), doporučovací systémy musí typicky faktorizaci periodicky opakovat, aby také matice $\mathbf{U_k}$, $\mathbf{S_k}$ a $\mathbf{V_k}$ (respektive pouze $\mathbf{U_k}$ a $\mathbf{V_k}$, viz. dále) sloužící k tvorbě doporučení zůstávaly aktuální. Alternativou k tomuto častému spouštění výpočetně náročného kroku je inkrementální SVD algoritmus představený v [2]. Algoritmus sice nedokáže aktualizovat vektory již existujících uživatelů a položek v maticích $\mathbf{U_k}$ a $\mathbf{V_k}$ podle nových hodnot v matici $\mathbf{A}$, umí ale vypočítat příslušné vektory nových uživatelů nebo položek.

Výsledkem inkrementálního algoritmu není přesný SVD model, přesto nabízí podle [2] dostatečnou kvalitu a zejména má výhodu lepší škálovatelnosti. Algoritmus využívá možnosti inkrementálního výpočtu modelu tak, že nejprve faktorizuje výchozí matici $\mathbf{A}$ nebo submatici vhodné velikosti a následně používá projekční metodu známou jako folding-in k přidání nových uživatelů nebo položek do prostoru již redukované dimenze. V případě nového uživatele je podle [7] možné zobrazit jeho vektor interakcí $\mathbf{n_u}$ dimenze $n$ do vektoru $\mathbf{p_u}$ dimenze $k$ pomocí projekce definované vzorcem

$$\mathbf{p_u} = \mathbf{n_u}\mathbf{V_k}{\mathbf{S_k}}^{-1}$$

Podobně také v případě nové položky je možné zobrazit její vektor interakcí $\mathbf{n_i}$ dimenze $m$ do vektoru $\mathbf{q_i}$ dimenze $k$ pomocí projekce dané vzorcem

$$\mathbf{q_i} = {\mathbf{n_i}}^T\mathbf{U_k}{\mathbf{S_k}}^{-1}$$

Výsledné vektory $\mathbf{p_u}$ a $\mathbf{q_i}$ lze pak přidat jako nové řádky k maticím $\mathbf{U_k}$, respektive $\mathbf{V_k}$. V reálném nasazení je pak logickým krokem kombinace obou přístupů, tedy prodloužit intervaly mezi opakováním kompletní faktorizace

všech dostupných dat a v mezidobí pouze dopočítávat latentní vektory nově příchozích subjektů.

### 2.2.2 Doporučování na základě SVD

Existuje více možných způsobů, jak na základě znalosti matic $\mathbf{U_k}$, $\mathbf{S_k}$ a $\mathbf{V_k}$ generovat doporučení pro konkrétní uživatele. Jedním z možných způsobů je vyhledávání podobných uživatelů v matici $\mathbf{U_k}$ a doporučení položek, které tito uživatelé nejlépe hodnotili. Jedná se vlastně o upravenou verzi algoritmu User k-NN, s tím rozdílem, že algoritmus nevyhledává podobné uživatele v původní matici interakcí $\mathbf{A}$, ale právě v redukované matici $\mathbf{U_k}$ obsahující charakterizace uživatelů, díky čemuž může být vyhledávání mnohem rychlejší. Navíc díky redukci dimenzí a odstranění šumu může podle [2] dojít k tomu, že podobnými se stanou nejen uživatelé, kteří hodnotily shodné produkty, ale také ti, kteří hodnotily různé, ale vzájemně podobné produkty.

Klasickým postupem generování doporučení při využití maticové faktorizace je ale predikce hodnot interakcí daného uživatele se všemi položkami (respektive se všemi, se kterými daný uživatel dosud neinteragoval) a doporučení těch položek, kde je predikovaná hodnota nejvyšší. Predikce hodnoty $r_{u,i}$, která představuje například hodnocení položky $i$ uživatelem $u$, může být generována vypočítáním kosinové podobnosti mezi uživatelem a danou položkou. Podle [2] lze tuto hodnotu určit vynásobením příslušného $u$-tého vektoru matice $\mathbf{U_k}\sqrt{\mathbf{S_k}}^T$ (vektor $\mathbf{p_u}$) s $i$-tým vektorem matice $\sqrt{\mathbf{S_k}}\mathbf{V_k}^T$ (vektor $\mathbf{q_i}$), kde uvedené matice představují latentní příznaky uživatelů a položek. Vzhledem k tomu, že hodnoty matice $\mathbf{A}$, které se snažíme predikovat, byly již před faktorizací vyplněny hodnotami řádkových nebo sloupcových průměrů, je vhodné tyto hodnoty k výsledné predikci připočítat. Celý vztah pro predikci hodnoty prvku $r_{u,i}$ matice $\mathbf{A}$ tak vypadá následovně [2]:

$$\hat{r}_{u,i} = \bar{r}_u + \left(\mathbf{U_k}\sqrt{\mathbf{S_k}}^T(u)\right)\left(\sqrt{\mathbf{S_k}}\mathbf{V_k}^T(i)\right)$$

přičemž $\bar{r}_u$ je v tomto případě právě řádkový průměr, který nicméně při generování doporučení nemá žádný efekt.

Často dochází k situaci, kdy má algoritmus vygenerovat doporučení pro nového uživatele, pro kterého dosud neexistuje vektor dimenze $k$ v matici $\mathbf{U_k}$. V takovém případě je možné využít projekci folding-in popsanou výše k zobrazení nového uživatele do vektoru dimenze $k$.

## 2.3 Metoda gradientního sestupu

Jak je uvedeno výše, metoda SVD ve své základní podobě má některé nevýhody, zejména nutnost znát všechny hodnoty matice $\mathbf{A}$. Tím vzniká nejen problém doplnění vhodných hodnot, ale také se z řídké matice stane matice plná, čímž se samotná faktorizace stává časově velmi náročnou. Mimo jiné

z tohoto důvodu vznikla řada dalších algoritmů, které se snaží najít rozklad matice **A** na matice **U** a **V** jinak, než exaktními matematickými metodami, a odstranit tak některé nevýhody algoritmu SVD.

Jednou z těchto metod je stochastický gradientní sestup (stochastic gradient descent, SGD), který zpopularizoval Simon Funk [8] když s ním dosáhl jistých úspěchů v uznávané soutěži známé jako Netflix Prize, v níž byla vypsána finanční odměna za zlepšení doporučovacích algoritmů. Metoda gradientního sestupu má výhodu jednoduché implementace a zároveň nižší výpočetní náročnosti, která vede k rychlejšímu běhu algoritmu v porovnání s metodou SVD.

Algoritmus začíná tím, že inicializuje matice $\mathbf{U_k}$ a $\mathbf{V_k}$, kde matice $\mathbf{U_k}$ o rozměrech $m \times k$ je maticí latentních příznaků uživatelů a matice $\mathbf{V_k}$ o rozměrech $n \times k$ je maticí latentních příznaků položek. Na rozdíl od metody SVD je tedy matice interakcí **A** aproximována jako $\mathbf{A_k} = \mathbf{U_k}\mathbf{V_k}^T$ a diagonální matice $\mathbf{S_k}$ je v tomto případě zahrnuta v uvedených maticích. V případě gradientního sestupu obecně nezáleží na způsobu, jakým budou obě matice inicializovány – lze je inicializovat konstantami, stejně jako náhodnými hodnotami, druhý způsob je nicméně v situaci, kdy nevíme kde se nachází minimum, obecně používanější přístup.

Po inicializaci matic $\mathbf{U_k}$ a $\mathbf{V_k}$ algoritmus tyto matice iterativně upravuje tak, aby se matice $\mathbf{A_k}$ co nejvíce blížila originální matici **A**, a aby tak maximálně snížil celkovou chybu (energii) modelu. Chyba modelu se nejčastěji vyjadřuje pomocí RMSE (root mean square error), počítanou jako odmocnina z průměru čtvercových chyb na všech známých hodnotách matice **A**:

$$RMSE = \sqrt{\frac{1}{|\,\kappa\,|} \sum_{(u,i)\in\kappa} (r_{u,i} - \hat{r}_{u,i})^2}$$

kde $\kappa$ je množina všech dvojic $(u, i)$, pro které je známa hodnota $r_{u,i}$ matice **A** a $\hat{r}_{u,i}$ je predikce těchto hodnot.

Algoritmus v každém kroku (iteraci) prochází všechny známé prvky $r_{u,i}$ matice **A**, pro každý takový prvek vyhodnotí chybu $e_{u,i}$ a následně upraví hodnoty příslušných vektorů $\mathbf{p_u}$ matice $\mathbf{U_k}$ a $\mathbf{q_i}$ matice $\mathbf{V_k}$ proti směru gradientů, čímž sníží chybu na daném prvku [9]. Pro možnost gradientního sestupu je potřeba najít parciální derivace chyby mocněné na druhou podle $\mathbf{p_u}$ a podle $\mathbf{q_i}$. Chyba je dána jako

$$e_{u,i} = r_{u,i} - \hat{r}_{u,i}$$

kde $\hat{r}_{u,i}$ je predikce dané hodnoty získaná vztahem $\hat{r}_{u,i} = \mathbf{p_u}\mathbf{q_i}^T$. Parciální derivace podle $\mathbf{p_u}$ pak vypadá následovně:

$$\frac{\partial e_{u,i}^2}{\partial \mathbf{p_u}} = 2e_{u,i}\frac{\partial e_{u,i}}{\partial \mathbf{p_u}}$$

Protože $r_{u,i}$ je konstanta, derivace chyby $e_{u,i}$ podle $\mathbf{p_u}$ je vlastně derivací hodnoty $-\hat{r}_{u,i} = -\mathbf{p_u}\mathbf{q_i}^T$, což je rovno $-\mathbf{q_i}$. Platí tedy:

$$\frac{\partial e_{u,i}^2}{\partial \mathbf{p_u}} = 2e_{u,i}\frac{\partial e_{u,i}}{\partial \mathbf{p_u}} = 2e_{u,i}(-\mathbf{q_i})$$

Podobně vypadá také derivace podle $\mathbf{q_i}$:

$$\frac{\partial e_{u,i}^2}{\partial \mathbf{q_i}} = 2e_{u,i}\frac{\partial e_{u,i}}{\partial \mathbf{q_i}} = 2e_{u,i}(-\mathbf{p_u})$$

Nyní můžeme udělat krok o velikosti $\eta$ proti směru gradientu, výsledky derivací tedy násobíme hodnotou $-\eta$ a můžeme zanedbat konstantu 2. Dostaneme tím vzorce pro úpravu vektorů $\mathbf{p_u}$ a $\mathbf{q_i}$, které vypadají následovně [9]:

$$\mathbf{p_u}(t) = \mathbf{p_u}(t-1) - \eta\frac{\partial e_{u,i}^2}{\partial \mathbf{p_u}} = \mathbf{p_u}(t-1) + \eta\, e_{u,i}\, \mathbf{q_i}(t-1)$$

$$\mathbf{q_i}(t) = \mathbf{q_i}(t-1) - \eta\frac{\partial e_{u,i}^2}{\partial \mathbf{q_i}} = \mathbf{q_i}(t-1) + \eta\, e_{u,i}\, \mathbf{p_u}(t-1)$$

Parametr $\eta \in \langle 0, 1\rangle$ je koeficient rychlosti učení (learning rate).

Obecným nebezpečím gradientního sestupu je vždy přeučení na trénovací data. Tomu se i v tomto případě snažíme předejít, neboť výsledný model chceme používat především pro predikci dosud neznámých hodnot. Podle [1] lze přeučení zabránit pomocí regularizovaného modelu, kdy algoritmus upravuje matice $\mathbf{U_k}$ a $\mathbf{V_k}$ tak, aby minimalizoval regularizovanou čtvercovou chybu na známých hodnotách $\mathbf{A}$. Funkce, kterou algoritmus během trénování minimalizuje (a která je používána i dalšími metodami, např ALS – viz. dále), má tedy podobu:

$$\min_{\mathbf{p}*,\mathbf{q}*} \sum_{(u,i)\in\kappa} (r_{u,i} - \mathbf{p_u}\mathbf{q_i}^T)^2 + \lambda(\| \mathbf{p_u} \|^2 + \| \mathbf{q_i} \|^2)$$

kde $\kappa$ je množina všech dvojic $(u, i)$, pro které je známa hodnota $r_{u,i}$ matice $\mathbf{A}$, a parametr $\lambda \in \langle 0, 1\rangle$ určuje rozsah regularizace. Pro minimalizaci chybové funkce během učení je třeba zavést regularizaci ve vzorcích pro úpravu vektorů $\mathbf{p_u}$ a $\mathbf{q_i}$. Tyto vzorce mají tedy podle [8] podobu:

$$\mathbf{p_u}(t) = \mathbf{p_u}(t-1) + \eta\Big(e_{u,i}\, \mathbf{q_i}(t-1) - \lambda\mathbf{p_u}(t-1)\Big)$$

$$\mathbf{q_i}(t) = \mathbf{q_i}(t-1) + \eta\Big(e_{u,i}\, \mathbf{p_u}(t-1) - \lambda\mathbf{q_i}(t-1)\Big)$$

Z uvedeného popisu gradientního sestupu je zřejmé, že tento algoritmus při hledání matic $\mathbf{U_k}$ a $\mathbf{V_k}$ pracuje pouze se známými hodnotami řídké matice $\mathbf{A}$. Výsledkem je skutečnost, že iterace gradientního sestupu jsou mnohem rychlejší než metoda SVD a také paměťové nároky pro uložení řídké matice jsou

mnohem nižší, než pro uložení plné matice v případě SVD. Obě tyto vlastnosti dělají z tohoto algoritmu vhodného kandidáta pro faktorizaci matic velkých dimenzí. Naopak nevýhodou metody gradientního sestupu je nebezpečí uváznutí v lokálním minimu nebo potenciální možnost přeučení na trénovací data. Nepříjemnou vlastností je také velké množství parametrů, jakými jsou hodnoty $\eta$ a $\lambda$, ale také počet trénovacích iterací nebo způsob inicializace matic $\mathbf{U_k}$ a $\mathbf{V_k}$. Nalezení optimálních parametrů tak samo o sobě představuje nelehký problém.

### 2.3.1 Zaujetí jako další možnost zpřesnění modelu

Jak uvádí [1], v případě explicitní zpětné vazby mohou hodnoty matice $\mathbf{A}$ záviset i na dalších skutečnostech, než je pouhá interakce daného uživatele s položkou. Udělená uživatelská hodnocení zde mohou mimo spokojenosti uživatele s položkou záviset také na jeho tendenci být spíše kritickým hodnotitelem, nebo naopak. Podobně některé položky jsou oblíbenější nebo kvalitnější než jiné, takže i jejich hodnocení se jedním nebo druhým směrem odchylují od průměru. Součástí každé hodnoty $r_{u,i}$ matice $\mathbf{A}$ je tedy takzvané zaujetí (bias) $b_{u,i}$, které vzniká kombinací zaujetí daného uživatele a příslušné položky [10]. Toto zaujetí je definováno jako:

$$b_{u,i} = \mu + b_u + b_i$$

kde $b_u$ představuje zaujetí daného uživatele a $b_i$ zaujetí dané položky, přičemž platí $b_u, b_i \in \mathbb{R}$. Hodnota $\mu$ je globálním průměrem ze známých hodnot matice interakcí $\mathbf{A}$.

Pokud známe hodnoty zaujetí uživatelů a položek, můžeme je zohlednit při predikci neznámých hodnot $r_{u,i}$, což může dále zlepšit kvalitu a přesnost výsledného doporučovacího modelu. Vzorec pro predikci má v takovém případě následující podobu:

$$\hat{r}_{u,i} = \mu + b_u + b_i + \mathbf{p_u}\mathbf{q_i}^T$$

Při použití metody gradientního sestupu lze hodnoty zaujetí uživatelů a položek hledat podobným způsobem, jako hodnoty vektorů latentních příznaků. Hledané hodnoty zaujetí jsou inicializovány na počáteční hodnotu a během iterací gradientního sestupu jsou upravovány tak, aby byla minimalizována chybová funkce. Tato funkce má nyní podle [1] podobu

$$\min_{\mathbf{p}*,\mathbf{q}*} \sum_{(u,i)\in\kappa} (r_{u,i} - \mu - b_u - b_i - \mathbf{p_u}\mathbf{q_i}^T)^2 + \lambda(\| \mathbf{p_u} \|^2 + \| \mathbf{q_i} \|^2 + {b_u}^2 + {b_i}^2)$$

Jedná se vlastně o shodnou chybovou funkci jaká byla přestavena výše, pouze rozšířenou právě přidáním zaujetí, jako dalších optimalizovaných parametrů.

Výše uvedený přístup pracuje se zaujetím uživatelů a položek jako se statickými hodnotami. V reálném světě se ale preference či chování jednotlivých uživatelů, stejně jako oblíbenost konkrétních položek, obvykle v průběhu času

mění. Proto může být výhodnější nahradit statické hodnoty zaujetí za funkce času, tedy nahradit hodnoty $b_u$ a $b_i$ funkcemi $b_u(t)$ a $b_i(t)$. Podle [10] je toto další krok ke zlepšení přesnosti doporučovacího modelu, vyvstává však nový problém s definicí uvedených funkcí. V případě funkce $b_i(t)$ nabízí autor řešení v podobě rozdělení známých hodnocení položky $i$ do několika binů podle okamžiku udělení těchto hodnocení. Funkce $b_i(t)$ je pak definována jako kombinace statického a v čase se měnícího zaujetí: $b_i(t) = b_i + b_{i,Bin(t)}$. V případě funkce $b_u(t)$ je situace složitější, neboť nemusí být k dispozici dostatek hodnocení od každého uživatele, aby se dalo použít rozdělení do binů jako v případě položek. Nabízí se pak řada různých možností, například použití lineární funkce k zachycení možné postupné změny uživatelského zaujetí. V takovém případě je určena časová vzdálenost okamžiku, kdy mají být predikována hodnocení, od průměrného okamžiku udělení hodnocení uživatelem a na základě této vzdálenosti a koeficientu $\alpha_u$ je upravena statická hodnota uživatelského zaujetí [10].

## 2.4 Metoda ALS

Metoda ALS (Alternating Least Squares) byla vymyšlena v souvislosti s problémem faktorizace matic obsahujících implicitní zpětné vazby namísto explicitních hodnocení. Takové matice mohou totiž podle [11] obsahovat mnohem více známých hodnot, v krajním případě se může jednat i o plné matice, kdy takové prvky matice, kde nedošlo k žádné interakci mezi uživatelem a položkou, jsou vyplněny nulami. Metoda gradientního sestupu, která v každé iteraci prochází všechny známé prvky matice $\mathbf{A}$, by se v takovém případě stala časově značně neefektivní.

Hlavní myšlenka metody ALS spočívá v úpravě chybové funkce, jejíž hodnota je minimalizována – tato funkce je shodná jako v případě algoritmu gradientního sestupu:

$$\min_{\mathbf{p}*,\mathbf{q}*} \sum_{(u,i)\in\kappa} (r_{u,i} - \mathbf{p_u}\mathbf{q_i}^T)^2 + \lambda(\| \mathbf{p_u} \|^2 + \| \mathbf{q_i} \|^2)$$

Vzhledem k tomu, že jak vektory $\mathbf{p_u}$, tak vektory $\mathbf{q_i}$ jsou neznámé, chybová funkce není konvexní a nelze tak jednoduše nalézt globální minimum. Pokud ale jeden z těchto vektorů zafixujeme jako neměnný, nekonvexní chybová funkce se stane kvadratickou a globální minimum může být snadno nalezeno [1]. Principem metody ALS je tedy střídající se zafixování matic $\mathbf{U_k}$ a $\mathbf{V_k}$. Je-li v dané iteraci zafixována matice $\mathbf{U_k}$, může být vzhledem k hodnotám této matice vypočítána metodou nejmenších čtverců optimální hodnota matice $\mathbf{V_k}$ a naopak, přičemž každou takovou iterací je podle [11] garantováno snížení hodnoty chybové funkce.

Mimo výhodu efektivnější faktorizace plných nebo téměř plných matic oproti metodě gradientního sestupu má metoda ALS ještě další výhodu ve

snadné masivní paralelizaci. Výpočet optimálních hodnot každého uživatelského vektoru $\mathbf{p_u}$ je totiž nezávislý na všech ostatních uživatelských vektorech, a stejně tak výpočet každého vektoru položek $\mathbf{q_i}$ je nezávislý na všech ostatních vektorech položek [1]. Teoreticky je tak možné zpracovávat každý vektor latentních příznaků samostatně speciálním vláknem, což může v případě nasazení na masivně paralelních architekturách, jakými jsou třeba grafické karty, znamenat výrazné zrychlení celého procesu faktorizace.

## 2.5 Metriky pro vyhodnocování modelů

Abychom mohli porovnávat různé algoritmy nebo různé implementace shodných algoritmů maticové faktorizace, je třeba definovat metriky pro vyhodnocování modelů získaných těmito algoritmy. V případě doporučovacích systémů se používá několik různých metrik a není dán přesný standard, podle čeho výsledné modely vyhodnocovat [12].

Pro určení kvality každého modelu je třeba rozdělit dostupná data na trénovací množinu, která slouží k naučení modelu, a na testovací množinu, s jejíž pomocí je úspěšnost výsledného modelu vyhodnocována. Robustní metodou pro vyhodnocování kvality modelu je metoda křížové validace (cross validation), kdy jsou dostupná data rozdělena do několika stejně velkých částí (foldů). Jedna z částí pak slouží jako testovací data a ostatní části jako trénovací data, přičemž celý proces se opakuje tolikrát, kolik existuje uvedených částí, tak aby se postupně všechny tyto části vystřídaly v roli testovacích dat. Výsledné hodnoty sledovaných metrik jsou potom průměrem z hodnot získaných v jednotlivých iteracích křížové validace. Tento postup je sice pomalý, dává nám ale možnost vyhodnotit kvalitu algoritmu generujícího model na všech dostupných datech.

K vyhodnocování úspěšnosti doporučovacího modelu jsou používány metriky precision a recall. Jde o metriky tzv. top-k doporučování, tedy doporučování $k$ položek s očekávaným nejvyšším hodnocením. Předpokladem pro výpočet hodnot těchto metrik je rozdělení položek pro každého testovacího uživatele do množin relevantních a irelevantních položek. Relevantní položky jsou doporučovacímu modelu skryty a ten se do nich snaží pomocí $k$ doporučení strefit. Precision je pak definována vzorcem:

$$precision(u) = \frac{\mid r(u) \cap t(u) \mid}{\mid r(u) \mid}$$

kde $r(u)$ je množina $k$ položek doporučených testovaným modelem danému uživateli $u$ a $t(u)$ je množina relevantních položek, se kterými tento testovací uživatel skutečně interagoval. Tato metrika tedy udává s jakým procentuálním množstvím položek, které byly danému testovacímu uživateli doporučeny, tento uživatel skutečně interagoval, a jak se tedy model strefuje do skutečných

preferencí daného uživatele. Naproti tomu recall je definovaný vzorcem:

$$recall(u) = \frac{\mid r(u) \cap t(u) \mid}{\mid t(u) \mid}$$

takže tato metrika udává, jaké procentuální množství položek, se kterými daný testovací uživatel interagoval, mu bylo skutečně doporučeno.

Často sledovaným parametrem je také pokrytí množiny uživatelů (user coverage) a pokrytí množiny položek (catalog coverage) generovanými doporučeními. Systém pro tvorbu doporučení nemusí být obecně schopen generovat doporučení pro každého uživatele (k této situaci může dojít, pokud například nemá k dispozici pro některé uživatele dostatečné množství trénovacích dat), proto je důležitým ukazatelem veličina user coverage, definovaná jako procentuální množství uživatelů, pro které vygeneruje systém alespoň jedno doporučení [12]. Podobná situace panuje v případě položek, kde typicky požadujeme, aby byla doporučována co největší škála dostupných položek namísto doporučování jen několika nejpopulárnějších. Ke sledování této vlastnosti doporučovacího modelu slouží veličina catalog coverage, definovaná jako procentuální množství položek, které jsou systémem doporučovány [12]:

$$catalog\ coverage(U_{test}) = \frac{\mid \bigcup_{u \in U_{test}} r(u) \mid}{\mid I \mid}$$

kde $U_{test}$ je množina testovacích uživatelů, funkce $r(u)$ vrací množinu položek doporučených uživateli $u$ a $I$ představuje množinu všech položek.

Kapitola 3

# Návrh a realizace procesu provádění experimentů

## 3.1 Dostupné implementace faktorizačních algoritmů

Po analýze výše uvedených algoritmů maticové faktorizace jsem provedl rešerši dostupných implementací těchto algoritmů s cílem zjistit možnosti jejich použití a případně vybrat vhodné kandidáty pro provedení experimentů s faktorizací velkých dat.

Modely latentních příznaků, zejména faktorizační modely, jsou u odborníků spravujících doporučovací systémy značně oblíbené, hlavně z toho důvodu, že nabízejí dvě zásadní vlastnosti, které jsou od doporučovacích systémů požadovány. Je to produkce kvalitních personalizovaných doporučení a vysoká rychlost vybavování, která umožňuje zpracovávat množství požadavků v reálném čase [2]. Díky této oblíbenosti existuje množství dostupných implementací faktorizačních algoritmů odvozených od metody SVD. Tyto implementace jsou nabízeny ve formě knihoven nebo frameworků, a to obvykle zdarma ve formě Open source, například pod některou z licencí kompatibilních nebo odvozených od licence GNU GPL.

Zatímco knihovny nabízejí většinou implementaci jednoho konkrétního algoritmu pro maticovou faktorizaci, rozsáhlejší frameworky často implementují celý doporučovací systém, přičemž obsahují širokou škálu algoritmů pro tvorbu doporučení, z nichž algoritmy maticové faktorizace tvoří jen část. Nabízejí také metody pro tvorbu doporučení nad výsledky faktorizace, takže koncový uživatel může vytvořit a používat doporučovací systém prostřednictvím několika málo řádek kódu. Knihovny či frameworky pro maticovou faktorizaci jsou obvykle připraveny na efektivní ukládání řídkých matic a mnohdy řeší také paralelizaci algoritmů tak, aby byla pro zpracování velkých dat využita všechna dostupná výpočetní kapacita. Mezi víceúčelové frameworky nebo knihovny po-

pulární v komunitě uživatelů doporučovacích systémů patří například LensKit, Mahout a MyMediaLite [12]. Specializovanou knihovnou, nabízející propracovanou implementaci pouze jednoho faktorizačních algoritmu, je například knihovna LIBMF.

### 3.1.1 LensKit

LensKit [13] je šířen zdarma jako Open source pod licencí GNU Lesser General Public license verze 2.1 a pozdější. Jedná se o nástroj určený k vytváření, výzkumu a studování doporučovacích systémů, který je připraven například pro použití ve webových aplikacích. Celý framework je napsán v programovacím jazyku Java a vzhledem k probíhajícímu vývoji jsou poměrně často vydávány nové verze, aktuálně poslední nabízená verze je 2.2.1. LensKit nabízí v současné době pouze základní algoritmy, jako User k-NN, Item k-NN, a z algoritmů maticové faktorizace je to FunkSVD, což je implementace stochastického gradientního sestupu podle [8]. Uživatel může volit obvyklé parametry, jako je počet latentních příznaků, koeficient rychlosti učení (learning rate) a hodnota regularizačního parametru $\lambda$, dále je to podmínka zastavení gradientního sestupu nebo výchozí hodnota latentních příznaků.

Podle výsledků vyhodnocování v [12], kde jsou vzájemně srovnávány frameworky LensKit, Mahout a MyMediaLite na různých datasetech, mají faktorizační modely produkované frameworkem LensKit nejvyšší chybu RMSE, což značí menší schopnost algoritmu naučit se na trénovací data. Přitom ale rozdíly v hodnotách RMSE nejsou tak velké, aby to znamenalo nefunkčnost algoritmu, o čemž svědčí i dobré výsledky z hlediska dalších metrik, jakými jsou user coverage nebo catalog coverage. Rovněž v případě výpočetního času dosahuje LensKit dobrých výsledků.

### 3.1.2 Apache Mahout

Projekt Mahout [14] společnosti Apache Software Foundation má za cíl umožnit komukoliv rychlé vytváření škálovatelných a výkonných aplikací pro strojové učení, k čemuž nabízí velké množství algoritmů pro oblast kolaborativního filtrování a vytěžování znalostí z dat (Data mining). Mnohé z těchto algoritmů jsou přitom připraveny pro distribuované výpočty s využitím technologií Hadoop MapReduce, Spark a dalších. Mahout může být použit jako prostředí pro pro práci s daty prostřednictvím příkazů zadávaných v terminálu, poskytuje však i knihovny pro použití implementovaných algoritmů nebo celého doporučovacího systému v dalších uživatelských aplikacích. Stejně jako LensKit je i Mahout šířen jako Open source, tentokrát pod licencí Apache License verze 2.0 [15] kompatibilní s GPL v3, a stejně jako LensKit je Mahout implementován v programovacím jazyku Java. V současné době stále probíhá vývoj, který vylepšuje existující algoritmy nebo přidává nové, obvykle v rozmezí několika

měsíců jsou tak průběžně vydávány nové verze, přičemž poslední dostupnou verzí je v době vzniku této práce verze 0.11.2.

Z algoritmů pro maticovou faktorizaci nabízí Mahout stochastický gradientní sestup, metodu ALS a algoritmus SVD++, přičemž první dva z uvedených algoritmů mohou využít princip MapReduce pro distribuci výpočtů v případě velkého množství dat. Uživatelé mohou opět nastavovat množství parametrů, podobně jako v případě frameworku LensKit. Mahout dosahuje podle [12] dobrých výsledků z hlediska rychlosti algoritmu, zejména v případě velkých dat, která zpracovává paralelně, dobře si však vede i z hlediska ostatních parametrů (RMSE, recall a catalog coverage).

### 3.1.3 MyMediaLite

MyMediaLite [16] je další z projektů nabízejících širší škálu algoritmů pro oblast kolaborativního filtrování. Jedná se kompaktní knihovnu napsanou v jazyku C#, která je přenositelná mezi různými platformami prostřednictvím multiplatformního .NET frameworku Mono. Také MyMediaLite je poskytována zdarma jako Open source, tentokrát přímo pod licencí GNU GPL. Vývoj knihovny zřejmě stále probíhá, ale nové verze jsou v poslední době vydávány jen zřídka, aktuální verze je 3.11.

Tato knihovna se od předchozích dvou liší tím, že odděluje predikci ratingů konkrétních položek od doporučování top-k množiny položek na základě pouze pozitivní zpětné vazby. V obou případech pak nabízí řadu různých doporučovacích algoritmů, z oblasti maticové faktorizace nabízí MyMediaLite stochastický gradientní sestup a algoritmus SVD++. Knihovna však není v základu připravena na distribuované výpočty, což ji nečiní moc vhodnou pro zpracování velkých dat a jak ukazují výsledky v [12], ze tří porovnávaných frameworků je při faktorizaci skutečně nejpomalejší. Na druhou stranu produkuje modely s nejmenší chybou RMSE a dosahuje dobrých výsledků i z hlediska parametru user coverage. Naopak v případě hodnoty catalog coverage dosahuje podle [12] v některých případech jen několika málo procent.

### 3.1.4 LIBMF

Knihovna LIBMF [17] v aktuální verzi 2.01 byla vyvinuta pro možnost rychlé paralelní faktorizace řídkých matic metodou stochastického gradientního sestupu. Umožňuje faktorizovat matice reálných čísel, stejně jako matice binární, přičemž důraz je kladen na rychlý paralelní výpočet na vícejádrových procesorech v systémech se sdílenou pamětí. Knihovna LIBMF je implementována v programovacím jazyku C++ a pro další zrychlení výpočtu využívá také speciální instrukce procesoru urychlující vektorové operace (např. SSE instrukce nebo modernější AVX instrukce). Další vývoj probíhá v poslední době zřejmě pouze sporadicky. Knihovna je poskytována zdarma jako Open source pod licencí BSD [18].

Použití LIBMF je možné dvěma způsoby – buď mohou být použity zdrojové kódy jako knihovna v uživatelských projektech, nebo je-li k dispozici soubor s trénovacími daty, lze zkompilovanou knihovnu spustit jako samostatnou aplikaci, která si sama načte trénovací data a výsledky faktorizace opět uloží do souboru na disku. Uživatelé mohou volit řadu klasických parametrů, jako je počet latentních příznaků, hodnota regularizace $\lambda$, ztrátová funkce (čtvercová chyba RMSE, absolutní chyba a další), koeficient rychlosti učení (learning rate) či počet trénovacích iterací. Dále mohou volit počet vláken pro výpočet faktorizace nebo povolit použití disku jako cache pro případ opravdu velkých dat, která se nevejdou do operační paměti počítače.

## 3.2 Volba použitých implementací faktorizačních algoritmů

Na základě provedené analýzy metod maticové faktorizace a na základě rešerše dostupných implementací faktorizačních algoritmů jsem postupně vybral dvě knihovny, s nimiž jsem dále provedl sadu experimentů na testovacích datasetech. Soustředil jsem se přitom zejména na algoritmus gradientního sestupu, který se vzhledem ke své jednoduchosti a rychlosti výpočtu zdá být vhodným kandidátem pro faktorizaci velkých dat. Ve prospěch tohoto algoritmu hovoří také úspěchy jeho autora, Simona Funka, v soutěži Netflix Prize i jeho celková oblíbenost mezi uživateli spravujícími doporučovací systémy. Tuto oblíbenost potvrzuje i ta skutečnost, že je algoritmus SVD implementován ve všech mnou zkoumaných knihovnách nabízejících faktorizační algoritmy (viz. výše).

K provedení experimentů jsem nejprve zvolil knihovnu LIBMF, která umožňuje jednoduché použití a je prezentována jako specializovaná pro rychlý a paralelizovaný výpočet faktorizace, což se jeví jako silný argument pro její použití na velkých datech. Provedené experimenty skutečně potvrdily schopnost rychlého výpočtu a minimalizace chyby RMSE, která je důležitým ukazatelem úspěšnosti algoritmu v naučení modelu na trénovací data (viz. kapitola 4). Horší výsledky ale vykazovaly modely z hlediska parametrů recall a catalog coverage. Proto jsem se rozhodl vyzkoušet ještě další z nabízených knihoven, abych mohl relevantněji posoudit, zda jde o vlastnost zvoleného algoritmu nebo pouze jedné konkrétní implementace tohoto algoritmu. Jako další jsem tedy zvolil projekt Apache Mahout, který je rovněž připraven na distribuované zpracování velkých dat a podle [12] dosahuje dobrých výsledků z hlediska RMSE nebo catalog coverage. Navíc jde o framework s probíhajícím vývojem a širokou komunitou uživatelů.

## 3.3 Existující softwarová infrastruktura společnosti Recombee

Společnost Recombee vyvinula pro účely svého podnikání komplexní softwarovou architekturu, která slouží k přijímání požadavků na doporučení, generování a odesílání výsledných doporučení a k dalším potřebným účelům. Celá infrastruktura se skládá z několika prvků, pro mou práci jsou však nejdůležitější komponenta sloužící k vlastnímu generování doporučení a vyhodnocovací framework, pomocí něhož lze doporučovací modely testovat a vyhodnocovat z hlediska různých parametrů. Komunikaci mezi oběma těmito prvky zajišťují další části softwarové infrastruktury.

S výše uvedenými prvky infrastruktury jsem se blíže seznámil, tak abych je mohl použít při realizaci svých experimentů. Tím odpadne část práce nutná při implementaci vlastních metod pro vyhodnocování faktorizačních algoritmů a uspořený čas mi umožní více se soustředit na provádění experimentů se zvolenými algoritmy. Navíc použitím existujícího vyhodnocovacího frameworku bude zajištěna kontinuita ve vyhodnocování doporučovacích modelů, založených na maticové faktorizaci, s dřívějším vyhodnocováním existujících modelů ve společnosti Recombee, což umožní jejich objektivní vzájemné porovnání.

### 3.3.1 Komponenta generující doporučení

Komponenta generující doporučení je klíčovou částí celé softwarové infrastruktury. Je implementována v programovacím jazyku Java a obsahuje již několik algoritmů z oblasti kolaborativního filtrování, s nimiž realizuje různé doporučovací modely. Tyto modely lze různě parametrizovat a kombinovat – například je lze sekvenčně skládat za sebe tak, aby následný model mohl vygenerovat doporučení v případě, že je předchozí model z nějakého důvodu (např. nedostatek dat) vygenerovat nedokáže. Doporučovací komponentu lze také dále rozšiřovat implementací nových algoritmů, k čemuž jsem v této práci také přistoupil – viz. dále.

Jako perzistentní úložiště pro data potřebná ke generování doporučení je používána relační databáze. Odtud si je doporučovací komponenta načítá a po dobu běhu je uchovává ve vlastních datových strukturách v operační paměti. Tím je zajištěn rychlý přístup k datům pro možnost generování doporučení v reálném čase. Jak jsem již uvedl v úvodu této práce, právě tato skutečnost vede k problémům s uchováním stále většího množství dat v operační paměti a byla jedním z hlavních impulzů pro zahájení experimentů s metodami maticové faktorizace.

### 3.3.2 Vyhodnocovací framework

Vyhodnocovací framework je další částí softwarové infrastruktury a je používán pro vyhodnocování a porovnávání implementovaných doporučovacích mo-

delů. Společnost Recombee totiž není ryze komerční firmou, ale zabývá se také akademickým výzkumem a vyhodnocovací framework slouží právě ke generování výzkumných výstupů. V kontextu mé práce může být tento framework použit pro vzájemné porovnání algoritmů maticové faktorizace, stejně jako pro srovnání těchto algoritmů s již implementovaným algoritmem nejbližších sousedů. Použitím tohoto frameworku mám totiž jistotu vyhodnocování všech porovnávaných algoritmů shodným způsobem.

Framework pro vyhodnocování doporučovacích modelů je napsán v programovacím jazyku Python. Umožňuje rozdělit dostupná data načítaná z databáze na trénovací množinu (trénovací uživatele) a testovací množinu (testovací uživatele), přičemž trénovací množinu poskytne příslušnému algoritmu pro naučení modelů a s pomocí testovací množiny výsledné modely vyhodnocuje. Tento postup rozdělení dat se nazývá Split validation a uvedený způsob vyhodnocování modelů je využíván zejména v případě algoritmů nejbližších sousedů. Tehdy jsou všechny modely, lišící se jednotlivými parametry, vytvořeny pouhým uložením trénovacích dat a dané varianty modelů se realizují aplikací příslušných parametrů až v průběhu vybavování.

Framework je připraven také na vyhodnocování algoritmů metodou křížové validace (Cross validation). V tomto případě sice také načte z databáze dostupná data, ale neprovádí již rozdělení na trénovací a testovací množiny, nýbrž načte ze souboru na disku předem daný seznam testovacích uživatelů pro jednotlivé foldy. Tento postup předpokládá, že, na rozdíl od algoritmů nejbližších sousedů, vyprodukuje každá kombinace parametrů algoritmu speciální model, a že trénování příslušných modelů vyžaduje delší dobu. Tím, že je rozdělení dat na trénovací a testovací množiny známo předem, mohou být modely naučeny již před spuštěním vyhodnocování. O vygenerování souboru testovacích uživatelů se však musí postarat jiná komponenta softwarové infrastruktury.

Samotné vyhodnocování modelů probíhá tak, že framework rozdělí množinu známých interakcí každého testovacího uživatele do dvou disjunktních podmnožin. První z těchto podmnožin, označovanou jako observation, odešle framework doporučovací komponentě spolu s požadavkem na vrácení určitého počtu doporučení od vyhodnocovaného doporučovacího modelu. Druhá, testovací podmnožina, je určena ke srovnání s obdrženými doporučeními. Framework zde pracuje metodou leave-one-out křížové validace, kdy testovací množinu tvoří vždy jediná skrytá položka a model se do ní má strefit na základě observation tvořené všemi zbývajícími položkami. Tento postup se opakuje tolikrát, dokud se všechny (nebo daný počet) položek nevystřídají v roli testovací podmnožiny. Framework také umožňuje iterování přes různé hodnoty parametrů doporučovacího modelu takovým způsobem, že požaduje doporučení po všech takto parametrizovaných modelech najednou. Tím je možno vyhodnotit různé doporučovací modely (například modely natrénované s odlišnými parametry) aniž by bylo nutné spouštět proces vyhodnocování opakovaně.

Všechny potřebné údaje – množina observation a doporučení získaná po-

mocí modelů naučených s příslušnými parametry – jsou ukládány na disk. Na konci procesu vyhodnocování jsou pak použity k výpočtu hodnot sledovaných metrik, z hlediska této práce jsou to zejména metriky recall a catalog coverage. Uložené výsledky vybavování mohou být ale použity i pro výpočet hodnot dalších metrik, jako je precision nebo user coverage, a to bez nutnosti opakování celého vybavování, které je v případě velké množiny testovacích uživatelů časově značně náročné.

Uvedený postup vyhodnocování algoritmů se mírně liší od postupu používaného například frameworkem Mahout nebo dalších, které mají metody pro vyhodnocování algoritmů rovněž implementovány. V jejich případě jsou dostupná data rozdělena na trénovací a testovací množinu tak, že část známých interakcí každého uživatele je určena jako trénovací a zbytek interakcí jako testovací. Při požadavku na doporučení je pak předán modelu identifikátor cílového uživatele, jehož uživatelský vektor $\mathbf{p_u}$ je modelu znám. Naproti tomu při použití popisovaného vyhodnocovacího frameworku je cílový uživatel z pohledu doporučovacího modelu definován pouze množinou observation a jeho uživatelský vektor musí být nejprve vytvořen. Takový postup je používán z toho důvodu, že v praktickém použití musí model velmi často generovat doporučení pro nové, neznámé uživatele, a je tedy ověřováno chování modelu v těchto případech. Navíc je tímto postupem zaručeno, že doporučovací model během trénování testovacího uživatele vůbec neviděl, což je i z teoretického hlediska správnější postup. Výsledky vyhodnocování se však mohou lišit od některých výsledků prezentovaných například v teoretických pracích zabývajících se algoritmy maticové faktorizace, pokud je zde použit odlišný postup vyhodnocování.

## 3.4 Návrh provádění experimentů

Poté, co jsem se seznámil s fungováním frameworku pro vyhodnocování doporučovacích modelů a s komponentou sloužící k realizaci těchto modelů a generování doporučení, navrhl jsem postup provádění jednotlivých experimentů. Pro vyhodnocování jsem zvolil metodu křížové validace, která umožňuje použít pro trénování modelů postupně všechna dostupná data. Navíc prostřednictvím této metody nevyhodnocujeme kvalitu jednoho konkrétního modelu, ale spíše algoritmu, který tyto modely vytváří (díky průměrování výsledků dosažených během několika iterací křížové validace), což je cílem mé práce.

Pro možnost provedení křížové validace je nejprve potřeba rozdělit dostupná data do $n$ částí (foldů) a vygenerovat tak trénovací a testovací data pro každou z $n$ iterací křížové validace. K tomu jsem navrhl implementaci vlastní metody, která s ohledem na funkci vyhodnocovacího frameworku rozdělí dostupná data rozdělením množiny uživatelů do těchto $n$ foldů. Metoda dále vygeneruje na disku $n+1$ souborů, kde prvních $n$ souborů představuje trénovací data (množinu interakcí trénovacích uživatelů) pro $n$ foldů křížové vali-

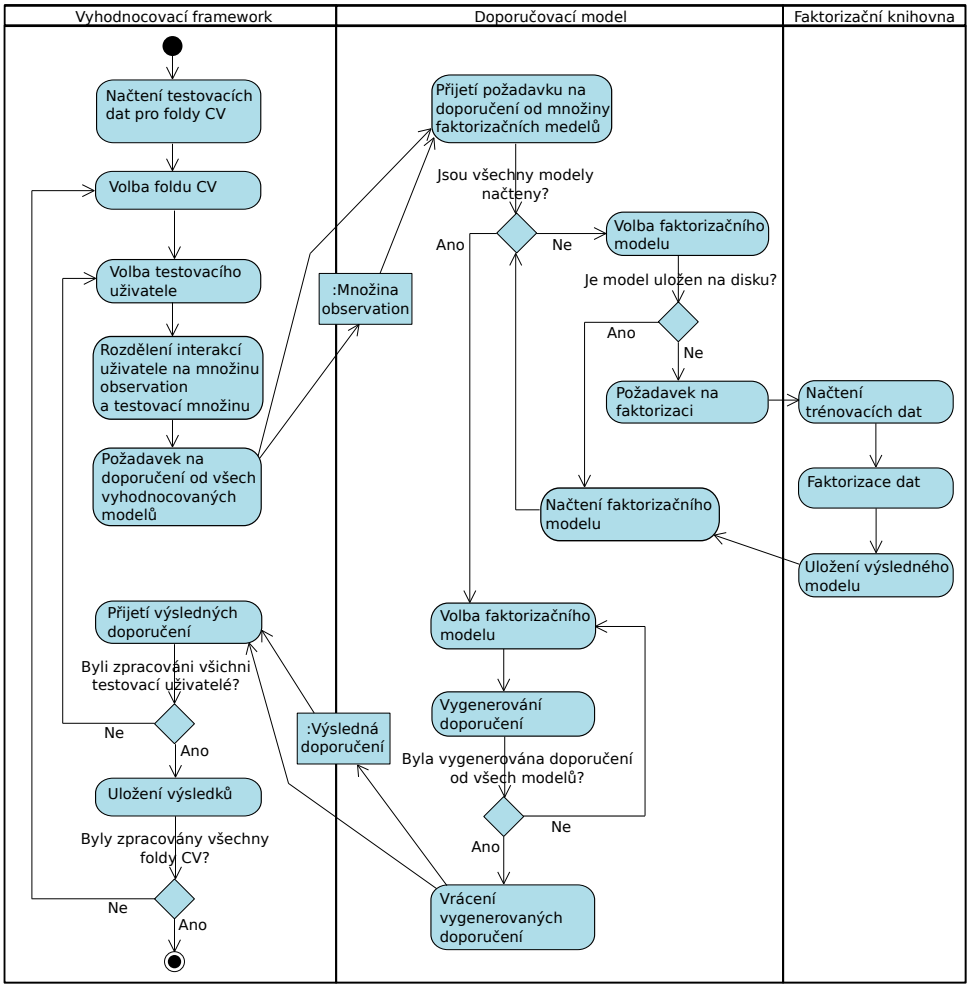
dace a poslední soubor představuje seznam testovacích uživatelů pro všechny jednotlivé foldy.

Dalším krokem každého experimentu je faktorizace trénovacích dat jednotlivých foldů křížové validace. K faktorizaci jsem použil zvolenou knihovnu, respektive připravený faktorizační algoritmus, jehož implementaci daná knihovna nabízí. V rámci každého experimentu lze také najednou vyhodnotit kvalitu různých faktorizačních modelů, tedy modelů získaných při různých hodnotách parametrů faktorizačních algoritmů. V takovém případě je třeba vygenerovat pro každou iteraci křížové validace všechny tyto modely a nechat vyhodnocovací framework iterovat přes zvolené parametry. Framework pak bude požadovat doporučení po všech těchto modelech najednou a není nutné pro každý parametr vyhodnocování znovu ručně spouštět.

Výsledkem každé faktorizace je faktorizační model obsahující plné matice $\mathbf{U_k}$ a $\mathbf{V_k}$, které představují vektory latentních příznaků uživatelů ($\mathbf{p_u}$) a vektory latentních příznaků položek ($\mathbf{q_i}$). Protože každý výpočet faktorizace je typicky časově i paměťově náročný, přistoupil jsem opět k ukládání faktorizačních modelů na disk, čímž se uvolní paměť pro provádění další faktorizace a navíc mohou být uložené faktorizační modely opakovaně používány v doporučovacích modelech. Vzhledem k tomu, že během vybavování dostává doporučovací model od vyhodnocovacího frameworku pouze množinu observation charakterizující cílového uživatele a nikoliv jeho identifikátor, není matice $\mathbf{U_k}$ při tvorbě doporučení využívána (viz. dále). Při ukládání faktorizačního modelu na disk tak stačí pro účely provádění experimentů uložit pouze hodnoty matice $\mathbf{V_k}$.

Pro dokončení experimentu je třeba načíst uložené faktorizační modely a zajistit možnost vybavování, tedy generování doporučení na základě těchto načtených modelů. Za tímto účelem jsem navrhl implementaci vlastního doporučovacího modelu. Faktorizační knihovny nabízejí sice ve většině případů vlastní metody pro vybavování nad výsledky faktorizace, jejich implementace se ale obvykle vzájemně liší a navíc často požadují identifikátor cílového uživatele a neumí pracovat s množinou observation. Implementací vlastní vybavovací metody proces vybavování sjednotím a navíc snížím provázání celého experimentu s konkrétní faktorizační knihovnou. Tím zajistím také možnost snadného vyměnění této knihovny za jinou. Vlastní doporučovací model jsem navrhl jako rozšíření stávající doporučovací komponenty společnosti Recombee, tak abych využil existující softwarové infrastruktury umožňující komunikaci vyhodnocovacího frameworku s doporučovacími modely.

Vlastní doporučovací model jsem navrhl tak, aby při inicializaci (nebo při prvním požadavku na doporučení – viz. dále) načetl příslušný faktorizační model uložený na disku (nebo více modelů). Tím načte matici $\mathbf{V_k}$, tedy latentní příznaky položek. Pokud by příslušný faktorizační model na disku neexistoval, doporučovací model může nejprve spustit proces faktorizace a teprve po dokončení výpočtu a uložení výsledků faktorizace na disk může tato data načíst. Doporučovací model dále obsahuje metodu pro tvorbu doporučení, kde



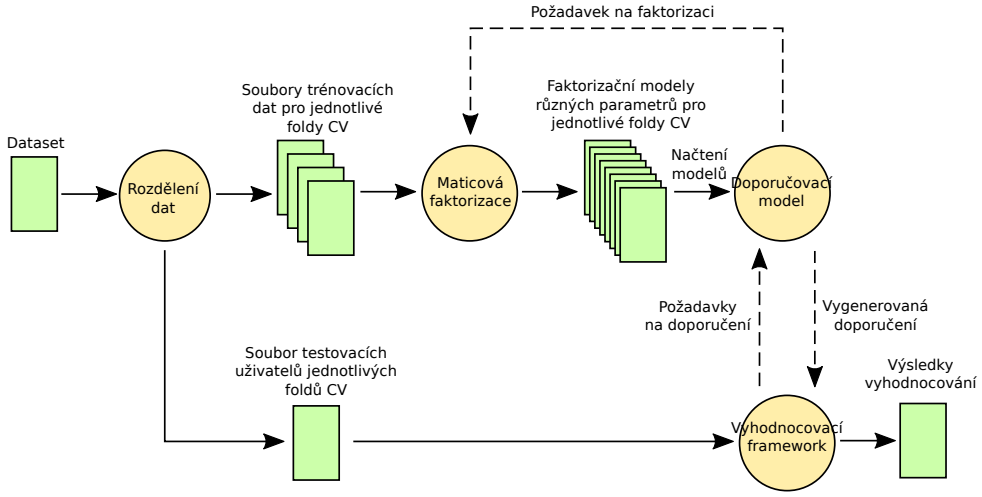

Obrázek 3.1: Diagram průběhu vyhodnocování faktorizačních modelů metodou křížové validace.

parametrem této metody je množina známých interakcí (observation) charakterizující cílového uživatele. Metoda pro tvorbu doporučení nejprve provede projekci množiny observation do prostoru dimenze $k$, čímž získá vektor latentních příznaků $\mathbf{p_u}$ cílového uživatele. Projekce je provedena metodou folding-in (viz. kapitola 2):

$$\mathbf{p_u} = \mathbf{n_u}\mathbf{V_k}$$

kde $\mathbf{n_u}$ je vektor observation. Matice $\mathbf{S_k}$ zde není použita, neboť v případě algoritmů provádějících rozklad pouze na matice $\mathbf{U_k}$ a $\mathbf{V_k}$ (jakým je i metoda SGD) je již začleněna v těchto maticích. Se znalostí vektoru $\mathbf{p_u}$ může nyní metoda predikovat hodnoty $\hat{r}_{u,i}$ jako $\mathbf{p_u}\mathbf{q_i}^T$, a to pro všechny položky $i$, kde je tato hodnota neznámá. Na základě predikovaných hodnot vybere daný počet nejrelevantnějších položek a tyto vrátí jak výsledná doporučení.

Uvedený postup získání vektoru $\mathbf{p_u}$ je důvodem, proč není třeba pro účely vyhodnocování ukládat matici $\mathbf{U_k}$. Jak je totiž z uvedených informací zřejmé, vybavovací metoda v této podobě ji vůbec nepoužívá, což koresponduje s používaným způsobem vyhodnocování modelů, kdy se předpokládá časté doporučování pro neznámé uživatele (viz. 3.3.2). V praxi přesto mohou nastat situace, kdy je identifikátor cílového uživatele známý. Může k tomu dojít například na webových portálech, které umožňují, nebo přímo vyžadují, registraci uživatelů, takže každý uživatel je pak spojen s konkrétním profilem. V takovém případě je možné předat doporučovacímu modelu identifikátor cílového uživatele a model může místo provádění projekce folding-in použít přímo příslušný vektor $\mathbf{p_u}$ nalezený maticovou faktorizací (pokud se nejedná o nového uživatele a vektor $\mathbf{p_u}$ je znám). Samozřejmě v tomto případě je třeba zajistit také ukládání matice $\mathbf{U_k}$.




Obrázek 3.2: Znázornění toku dat při použití metody křížové validace pro vyhodnocování doporučovacího modelu založeného na faktorizačních modelech různých parametrů.

Posledním krokem experimentu je souběžné spuštění doporučovacího modelu a vyhodnocovacího frameworku. Doporučovací model načte všechny potřebné faktorizační modely, jejichž seznam získá z prvního požadavku na doporučení zaslaného vyhodnocovacím frameworkem. Tento seznam je dán množinou vyhodnocovaných modelů a konkrétním foldem křížové validace, který vyhodnocovací framework právě zpracovává. Framework následně iteruje přes všechny foldy křížové validace a přes všechny testovací uživatele a po doporučovacím modelu požaduje pro každého testovacího uživatele doporučení vygenerovaná na základě všech vyhodnocovaných faktorizačních modelů. Výsledky vybavování ukládá framework na disk a na závěr vyhodnocovacího procesu

z nich vypočítá hodnoty sledovaných metrik kvality doporučovacích modelů (recall, catalog coverage).

Navržený postup předávání trénovacích dat a výsledných faktorizací ve formě souborů uložených na disku má ještě další výhodu v tom, že lze poměrně jednoduše vyměnit používanou faktorizační knihovnu za jinou, aniž by byly změnou dotčeny ostatní části experimentu – generování trénovacích a testovacích dat a doporučovací model s metodou pro vybavování. Pro nasazení jiné knihovny stačí zajistit správné načtení trénovacích dat a jejich předání faktorizačnímu algoritmu a po skončení faktorizace získat výsledné vektory latentních příznaků položek (případně i uživatelů) a opět je ve správném formátu uložit na disk. Samotná faktorizace dokonce ani nemusí proběhnout na stejném výpočetním stroji jako vyhodnocování výsledků. Je například možné vyhodnotit faktorizační modely dodané někým, kdo se zabývá jinými metodami maticové faktorizace, než je mnou zvolená metoda stochastického gradientního sestupu. Jedinou podmínkou je přitom naučení těchto modelů na srpávné množině trénovacích dat.

## 3.5 Realizace procesu provádění experimentů

Z výše navrženého postupu provádění experimentů vyplynula potřeba implementace dvou samostatných komponent celého procesu. První z nich je metoda pro rozdělení dostupných dat do foldů křížové validace a vygenerování souborů trénovacích a testovacích dat pro jednotlivé foldy. Druhou komponentou je doporučovací model, schopný načíst jeden nebo více faktorizačních modelů (pro účely vyhodnocování) a generovat doporučení na základě těchto modelů. Obě tyto komponenty jsem implementoval podle návrhu uvedeného v 3.4.

Jak jsem již uvedl v návrhu, doporučovací model jsem implementoval jako rozšíření stávající doporučovací komponenty společnosti Recombee, což je nutný předpoklad pro možnost použití vyhodnocovacího frameworku. Výsledný doporučovací model tak nemá podobu samostatně spustitelného programu, ale jedná se o jednu část komplexního systému. K implementaci jsem z uvedeného důvodu použil programovací jazyk Java, v němž je napsána celá komponenta generující doporučení. Zdrojové kódy implementovaného modelu jsou k nahlédnutí na přiloženém CD, viz. příloha B. Pro jednoduchost čtení dat z databáze a pro možnost využití datových struktur pro uchovávání interakcí mezi uživateli a položkami jsem jako součást doporučovací komponenty implementoval i metodu pro rozdělení dostupných dat a vygenerování trénovacích a testovacích souborů pro foldy křížové validace.

Vzhledem ke zpracování velkých dat a dostupnosti více výpočetních jader na testovacím serveru (viz 4.2) bylo žádoucí zajistit maximální možnou paralelizaci algoritmů pro co největší urychlení celého procesu. To se v mém případě týká zejména doporučovacího modelu, který při každém generování doporučení provádí množství výpočtů pro predikci všech neznámých hodnot

matice **A** pro daného uživatele. Doporučovací model jsem proto implementoval tak, že při generování doporučení se spouští předem daný počet vláken, kde v rámci každého vlákna jsou predikovány hodnoty pouze pro konkrétní podmnožinu položek. Každé vlákno pak ze zpracovávaných položek vybere daný počet nejrelevantnějších (nejlépe hodnocených) a z nich je na závěr vybráno potřebné množství globálně nejrelevantnějších položek, které jsou doporučeny cílovému uživateli.

K faktorizaci dat jsem v souladu s 3.2 použil knihovny LIBMF a Mahout, obě připravené na paralelní zpracování dat. Obě knihovny načítají trénovací data ze souborů na disku a knihovna LIBMF ukládá na disk také výsledné latentní vektory uživatelů a položek. V případě použití knihovny Mahout bylo potřeba ukládání výsledných latentních vektorů na disk doimplementovat, což jsem opět provedl v rámci doporučovací komponenty, která tyto faktorizace spouští.

### 3.5.1 Postup provádění experimetů

Provedení každého experimentu se skládá z několika kroků, které mohou, ale nemusí být zcela zřejmé z předchozích kapitol. Proto zde uvádím konkrétní postup provedení jednoho experimentu:

1. Příprava každého experimentu zahrnuje zejména nahrání testovacích dat do databáze. Odtud si později načte data metoda pro rozdělení dostupných dat do foldů křížové validace a vygeneruje z nich soubory trénovacích dat a testovacích uživatelů pro jednotlivé foldy. Hlavním důvodem nahrání dat do databáze je však skutečnost, že odtud čerpá potřebné údaje vyhodnocovací framework. Ten sice dostává na vstup soubor testovacích uživatelů, informace o interakcích uživatelů s položkami však obsahem tohoto souboru nejsou, a framework je načítá právě z databáze.

2. Další krok je nutný z důvodu jistých omezení používané softwarové infrastruktury. Při spuštění každé doporučovací komponenty je třeba předat jí název jedné nebo více databází, jejichž data mají sloužit pro generování doporučení, přičemž těmito názvy databází jsou dané instance zároveň identifikovány. Při návrhu infrastruktury se nepočítalo s možností spouštět nad jednou databází více doporučovacích komponent, mezi kterými by se při požadavku na generování doporučení dalo explicitně volit. Lze sice spustit nad jednou databází více instancí doporučovací komponenty, ale všechny tyto instance jsou identifikovány shodným názvem databáze a infrastruktura se v takovém případě sama rozhoduje, na kterou instanci požadavky na doporučení delegovat, čímž se snaží o rovnoměrné vyvažování zátěže. Z toho vyplývá, že není možné spustit nad jednou databází více doporučovacích komponent tak, aby každá odpovídala pouze na požadavky předem dané iterace křížové validace.

Řešením tohoto problému je rozkopírování dané databáze na $n$ nových databází, které se v názvu liší právě pořadovým číslem iterace křížové validace. Poté lze spustit $n$ doporučovacích komponent s modely pro tvorbu doporučení – každou komponentu nad svou databází – a vyhodnocovací framework může při zpracovávání dané iterace křížové validace požadovat doporučení pouze po tom konkrétním modelu, který se naučil na správných trénovacích datech. Nicméně vzhledem k tomu, že doporučovací modely načítají faktorizační modely ze souborů na disku a rovněž faktorizační knihovny načítají trénovací data ze souborů na disku, není nutné kopírovat do nových databází celý obsah datasetu – pro možnost spuštění doporučovací komponenty postačuje při současném řešení prostá existence těchto databází.

3. Třetím krokem je již rozdělení dostupných dat do částí křížové validace a vygenerování příslušných souborů trénovacích dat a souboru testovacích uživatelů pro jednotlivé foldy. Tento proces lze provést jednorázovým spuštěním doporučovací komponenty s mnou implementovanou metodou pro rozdělení dat, a to nad originální databází obsahující celý dataset.

   Uvedené kroky 1–3 je nutné provést pouze při zahajování experimentů nad novým datasetem, při provádění dalších experimentů na shodném datasetu je není nutné opakovat.

4. Následuje spuštění faktorizace trénovacích dat pomocí zvolené faktorizační knihovny. Například při použití knihovny LIBMF, kde lze faktorizaci matice spustit jako samostatný proces, může být jednoduchým skriptem spuštěno trénování všech potřebných faktorizačních modelů ještě před spuštěním vyhodnocování. V tomto kroku je třeba vygenerovat faktorizační modely pro všechny foldy křížové validace a v případě vyhodnocování vlivu různých parametrů faktorizačního algoritmu je třeba vygenerovat pro každý fold všechny faktorizační modely naučené s danými parametry.

   Tento krok je za současného řešení také možné přeskočit a ponechat spouštění faktorizací na mnou implementovaném doporučovacím modelu. Ten může faktorizaci spustit v případě, že na disku neexistuje soubor faktorizačního modelu, který se na základě požadavku na doporučení od vyhodnocovacího frameworku snaží načíst. V neposlední řadě je v rámci tohoto kroku také možné dodání hotových faktorizačních modelů od dalších lidí, kteří se zabývají například jinými faktorizačními metodami, než je mnou zvolený a vyhodnocovaný stochastický gradientní sestup.

5. Pro vyhodnocování doporučovacích modelů z hlediska sledovaných metrik je třeba spustit doporučovací komponentu s mnou implementovaným doporučovacím modelem, a to nad každou rozkopírovanou databází, tedy

pro každý fold křížové validace. Doporučovací modely pak čekají na požadavky na doporučení a při obdržení prvního takového požadavku jsou načteny potřebné faktorizační modely (případně, při přeskočení kroku č. 4, je nejprve spuštěna faktorizace dat s danými parametry, tak jak je znázorněno na obrázku 3.1). Na základě faktorizačních modelů jsou pak generována o odesílána výsledná doporučení.

6. Klíčovým krokem každého experimentu je spuštění vyhodnocovacího frameworku nad původní databází obsahující celý dataset a s dříve vygenerovaným souborem testovacích uživatelů pro jednotlivé foldy křížové validace. Prostřednictvím konfiguračního souboru je možné určit, jaké doporučovací modely má framework vyhodnocovat, respektive jaké faktorizační modely a s jakými parametry mají být použity k doporučování. Framework pak načte příslušná data a postupně iteruje přes jednotlivé foldy křížové validace a jednotlivé testovací uživatele, dělí dostupné interakce na množinu observation a testovací množinu a požaduje doporučení po příslušných doporučovacích modelech spuštěných v předchozím kroku. Výsledky vybavování ukládá na disk pro závěrečné vypočítání hodnot sledovaných metrik. Tato část experimentu je časově značně náročná a v závislosti na množství vyhodnocovaných modelů a testovacích uživatelů (lze vyhodnocovat všechny testovací uživatele nebo jen jejich část) může trvat i několik dní.

7. Posledním krokem experimentu je vypočítání hodnot sledovaných metrik z výsledků vyhodnocování uložených na disku. Jak jsem již popsal výše, důvodem k oddělení tohoto kroku od procesu vyhodnocování je umožnění výpočtu různých metrik kvality doporučovacích modelů bez nutnosti časově náročného opakování celého vybavování. Výpočet hodnot zvolených metrik je opět prováděn vyhodnocovacím frameworkem, spuštěným ale s příslušnými parametry. Konkrétní metriky, jejichž hodnoty mají být spočítány, jsou určeny konfiguračním souborem.

Kapitola 4

# Experimenty s maticovou faktorizací

## 4.1 Volba datasetů

### 4.1.1 MovieLens

Uznávanými a často používanými datasety pro vyhodnocování doporučovacích algoritmů jsou datasety MovieLens, které poskytuje výzkumná laboratoř GroupLens univerzity v Minnesotě. Datasety obsahují reálná anonymizovaná data z projektu MovieLens, který pomáhá zaregistrovaným uživatelům vyhledávat relevantní filmy ke zhlédnutí, k čemuž používá zpětnou vazbu od uživatelů, tedy jejich hodnocení již zhlédnutých snímků [19]. S tím, jak v systému rostlo množství uživatelů a množství známých hodnocení, byly postupně publikovány čtyři datasety – 100K, 1M, 10M a 20M, kde dané číslo udává přibližné množství explicitních hodnocení v datasetu.

Pro účely vyhodnocení mnou zvolených algoritmů maticové faktorizace jsem vybral dataset MovieLens 10M [20], který odpovídá požadavku zadání mé práce na otestování faktorizačních algoritmů na datech s miliony záznamů. Tento dataset obsahuje 71 567 uživatelů, 10 681 filmů a celkem 10 000 054 explicitních hodnocení filmů těmito uživateli, přičemž každý uživatel hodnotil nejméně 20 filmů. Hodnocení filmů jsou z rozsahu hodnot $\langle 0.5, 5 \rangle$, přičemž pro účely vyhodnocování faktorizačních algoritmů jsem tyto hodnoty normalizoval do rozsahu $\langle -1.0, 1.0 \rangle$.

### 4.1.2 GoOut

Jako další testovací dataset jsem zvolil reálná data zpracovávaná společností Recombee, konkrétně data společnosti GoOut, s.r.o., která laskavě souhlasila s použitím svých dat v rámci výzkumu realizovaného v této práci. Tato společnost provozuje stejnojmenný webový portál, kde shromažďuje informace

o chystaných a probíhajících kulturních akcích. Ty lze na webu procházet a filtrovat podle podle různých parametrů. GoOut dále nabízí jednoduchý a jednotný způsob zakoupení vstupenek na konkrétní akce. Webový portál umožňuje také registraci uživatelů, přičemž GoOut poté přizpůsobuje zobrazovaný obsah podle preferencí přihlášených uživatelů.

Data z projektu GoOut, která jsem použil pro provedení experimentů, obsahují přibližně 2 469 000 uživatelů či identifikátorů anonymních sessions a 54 000 položek. Na rozdíl od datasetů MovieLens v projektu GoOut uživatelé nehodnotí nabízené položky. Hodnotami matice interakcí tak není explicitní zpětná vazba od uživatelů, ale implicitní zpětná vazba, která zachycuje jejich chování. Zachycované události jsou tří typů – zobrazení stránky s detailními informacemi o položce (o konkrétní akci), uložení této stránky do záložek a nakonec zakoupení vstupenek. Použitá data obsahují konkrétně 7 970 000 záznamů o detailním prohlédnutí položek, 113 000 záznamů o uložení stránky mezi záložky a 193 000 záznamů o uskutečněných nákupech provedených v roce 2014. Pro účely vyhodnocování všechny tyto události implikují hodnotu 1.0 příslušného prvku matice interakcí (všechny události jsou brány jako stejně důležité), a to i v případě, kdy uživatel interaguje s konkrétní položkou více způsoby.

Problémem implicitních interakcí a výše uvedeného kódování je skutečnost, že všechny známé prvky matice interakcí mají nyní stejnou hodnotu 1.0. Výsledkem faktorizace tak může být triviálně nalezitelný model, který bude pro všechny (známé i neznámé) prvky matice interakcí predikovat tuto hodnotu. Takový faktorizační model nám však nijak nepomůže při generování doporučení. Ke správné funkci je třeba dát systému najevo, že existují i jiné, nižší hodnoty interakcí (například 0), které znamenají nezájem uživatele o položku. Vzhledem k tomu, že realizované interakce jsou v původním systému vždy zaznamenávány a ukládány do matice, mohu předpokládat, že všechny, nebo alespoň velká většina neznámých hodnot matice interakcí, jsou právě nulové hodnoty. Podobný přístup zastává také Harald Steck v [21], který říká, že neznámé hodnoty nechybí náhodně, ale že hodnoty 0 chybí mnohem častěji, než hodnoty 1. Nulové hodnoty tak můžeme do matice zapsat s vědomím, že se nedopustíme velké chyby a naopak že zajistíme správnou funkci faktorizačního a doporučovacího algoritmu. Aby se však z řídké matice nestala matice plná (čímž by dramaticky klesla výkonnost metody gradientního sestupu), rozhodl jsem se nahradit nulami pouze část neznámých hodnot. Pro experimenty s datasetem GoOut jsem tedy přidal mezi známá data pouze takové množství nul, které je rovno polovině množství známých interakcí, abych neúměrně nezvětšil velikost trénovacích dat.

## 4.2 Testovací prostředí

Všechny výzkumné experimenty jsem spouštěl na serveru, který k tomuto účelu poskytla společnost Recombee. Jedná se o server s dvanácti výpočetními jádry a s dostatečným množstvím operační paměti pro zpracování velkých dat. Experimenty běžely na uvedeném serveru pod operačním systémem GNU Linux, vzhledem k použitým multiplatformním programovacím jazykům softwarové infrastruktury společnosti Recombee (Java, Python) i mnou zvolených faktorizačních knihoven (C++, Java) je však možné provádět navržený experiment i na dalších operačních systémech.

Jak jsem již uvedl v předchozích kapitolách, během návrhu a realizace experimentů jsem kladl důraz na možnost paralelního provádění celého experimentu (jak faktorizace, tak vyhodnocování modelů), takže výsledný proces mohl naplno využít dostupný výkon hardwaru. Konkrétně jsem tak nechával běžet faktorizace i generování doporučení na osmi až dvanácti jádrech při současném přidělení 40-60 GB operační paměti. Přesto bylo provádění experimentů časově náročné, což způsobovala jednak velikost datasetů, zejména ale velké množství vyhodnocovaných faktorizačních modelů (množství testovaných hodnot parametrů faktorizačních algoritmů) násobené ještě daným počtem iterací křížové validace. V závislosti na těchto parametrech tak běžel každý experiment i několik dní.

## 4.3 Experimenty na datasetu MovieLens

### 4.3.1 Experimenty s knihovnou LIBMF

Jako první jsem pro experimenty s maticovou faktorizací použil knihovnu LIBMF 3.1.4 a jí implementovaný algoritmus stochastického gradientního sestupu – volba knihovny byla zdůvodněna v kapitole 3.2. Již při prvních pokusech se ukázalo, že množství parametrů algoritmu – jakými jsou počet latentních příznaků, hodnota regularizačního parametru $\lambda$ nebo i počet trénovacích iterací – znemožňuje pouhé naivní spuštění faktorizace a dosažení uspokojivých výsledků, nýbrž že je třeba věnovat určité úsilí nalezení optimálních hodnot těchto parametrů. Proto jsem zpočátku prováděl velké množství experimentů s různými hodnotami parametrů a ověřoval chování výsledných faktorizačních modelů. Lze přitom předpokládat, že optimální hodnoty uvedených parametrů se mohou v případě různých datasetů i produkčních dat do jisté míry lišit. Tomuto procesu se tak alespoň v omezené míře bude nutné věnovat i v případě reálného nasazení algoritmu gradientního sestupu, případně dalších faktorizačních algoritmů, pokud budou podobně parametrizovány.

Když jsem zhruba stanovil oblasti hodnot, kde by se měly příslušné parametry pohybovat, přistoupil jsem k provedení experimentů na množině všech testovacích uživatelů. Abych zjistil, jak se vyvíjí dosažené hodnoty metrik catalog coverage a recall během učení, nechal jsem vyhodnotit faktorizační modely

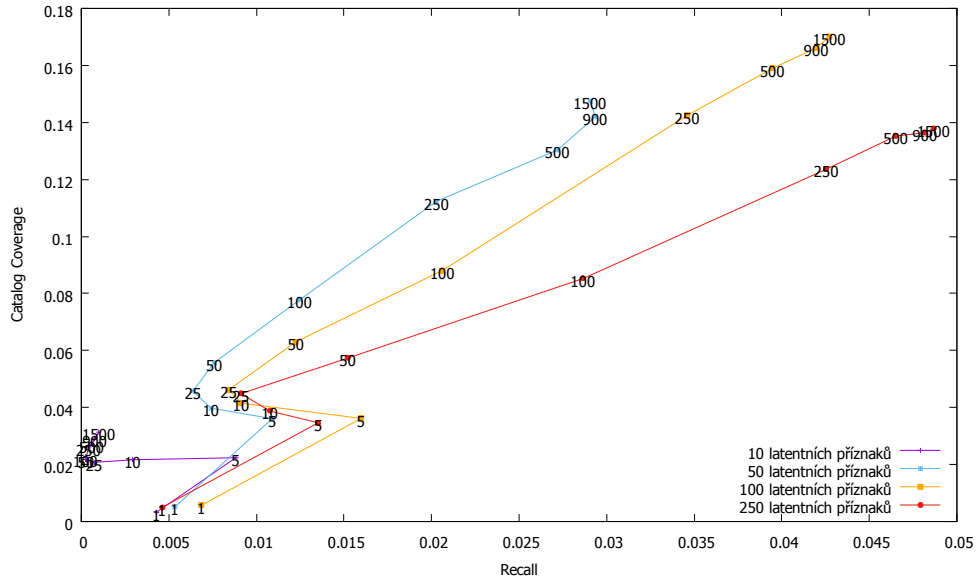
získané postupně vzrůstajícím počtem trénovacích iterací. Toto vyhodnocení jsem provedl pro různé velikosti vektorů latentních příznaků, přičemž jsem zvolil regularizační parametr $\lambda = 0.01$. Výsledný graf je zobrazen na obrázku 4.1. Knihovna LIBMF při faktorizaci dat vypisuje do terminálu hodnoty chyby RMSE pro jednotlivé iterace gradientního sestupu. Tato skutečnost mi umožnila vybrat data o vývoji chyby RMSE při faktorizaci trénovacích dat jedné části křížové validace a také je vynést do grafu. Tento graf vývoje RMSE je zobrazen na obrázku 4.2.

Předpokládal jsem, že s větší dimenzí vektorů latentních příznaků se podaří dosáhnout menší chyby RMSE, neboť více hodnot latentních příznaků může lépe vysvětlit informace obsažené v původní matici interakcí. Tento předpoklad plyne i z úvahy, že nejlepších výsledků by se dosáhlo s dimenzí latentních vektorů rovnající se větší z hodnot počtu uživatelů nebo položek, kde by například pro každou položku byl vyhrazen jeden příznak. To by však pochopitelně nevedlo k žádné redukci množství dat a nemělo by smysl takovou faktorizaci provádět, navíc by byl výsledný model přeučený a poklesla by jeho schopnost generalizace. Jak ukazuje graf vývoje RMSE, dosažené výsledky tento předpoklad potvrzují. Srovnáme-li ale vývoj RMSE s dosaženými hodnotami metrik catalog coverage a recall při odpovídajícím počtu latentních příznaků, zjistíme, že generování doporučení na základě faktorizačního modelu s nejmenší chybou RMSE nevede vždy k nejlepším výsledkům ve všech sledovaných metrikách.

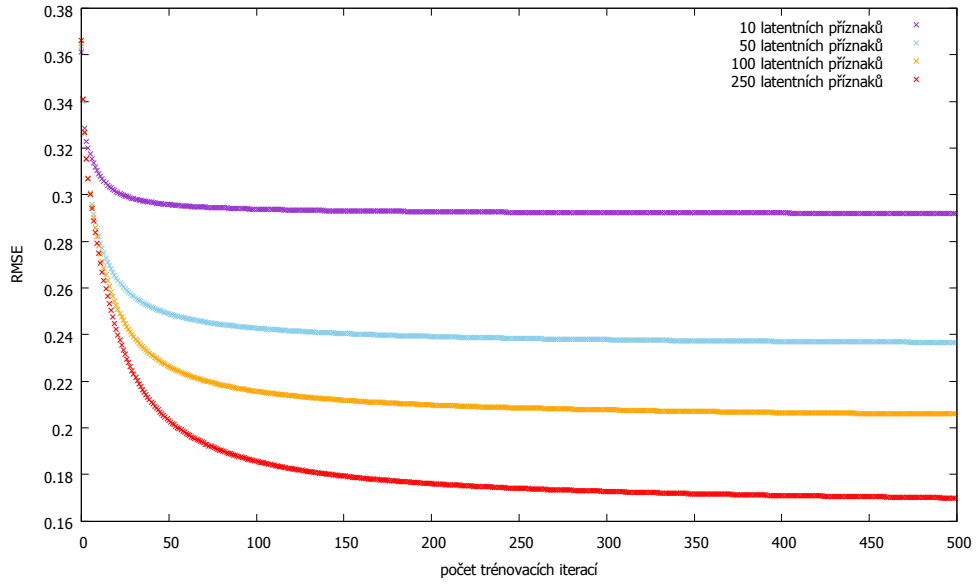
Z výsledků experimentu s různým počtem trénovacích iterací (obrázek 4.1) se také zdá, že nejvyšších hodnot metriky catalog coverage je na datasetu MovieLens dosaženo při použití faktorizačních modelů se 100 latentními příznaky, zatímco při použití modelů s větším počtem latentních příznaků je dosaženo vyšších hodnot metriky recall, byť rozdíl mezi těmito hodnotami není příliš vysoký. Abych mohl relevantněji určit vliv dimenze latentních vektorů na dosažené výsledky, provedl jsem další experiment, kde jsem porovnal faktorizační modely se širší škálou dimenzí latentních vektorů. Zároveň jsem nechal vyhodnotit chování algoritmu pro různé hodnoty regularizačního parametru $\lambda$. Při provádění tohoto experimentu jsem nastavil počet trénovacích iterací ve všech případech na hodnotu 1 500, kdy jsou zjevně produkovány nejlepší faktorizační modely. Dosažené výsledky jsou prezentovány v grafu 4.3.

Výsledky experimentu potvrdily, že při použití faktorizačních modelů s latentními vektory dimenze 100 dosahuje doporučovací model nejlepších výsledků z hlediska metriky catalog coverage. S větší dimenzí latentních vektorů sice mírně roste hodnota recall, ale catalog coverage klesá. Tyto výsledky jsou v souladu s předchozími zjištěními – s větší dimenzí vektorů latentních příznaků klesá chyba RMSE, ta však nic neříká o kvalitě doporučovacího modelu. Nemělo by tedy být překvapením, že se hodnoty metrik recall a catalog coverage mohou vyvíjet jinak, než bychom si přáli.

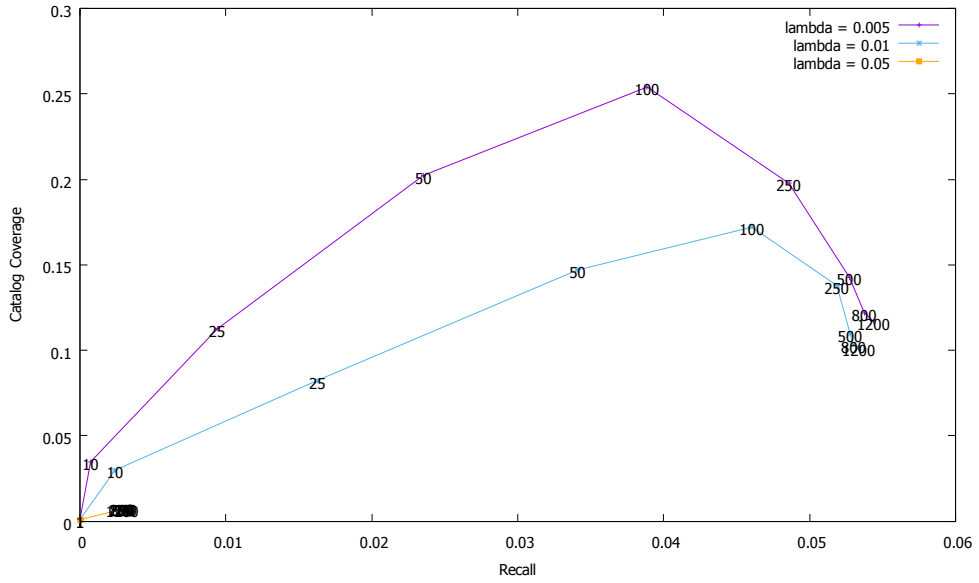
Jak z experimentu dále vyplynulo, algoritmus gradientního sestupu implementovaný knihovnou LIBMF je velmi citlivý na hodnotu regularizačního parametru. Je zřejmé, že výchozí hodnota parametru $\lambda$, která je v případě




Obrázek 4.1: Vliv různého množství trénovacích iterací metody SGD implementované knihovnou LIBMF na dosažedné hodnoty metrik recall a catalog coverage na datasetu MovieLens při doporučování 10 položek. Hodnoty na křivkách udávají počet trénovacích iterací, použitá hodnota regularizace $\lambda = 0.01$.




Obrázek 4.2: Hodnoty chyby RMSE dosažené metodou SGD knihovny LIBMF při různém počtu latentních příznaků a hodnotě regularizace $\lambda = 0.01$.



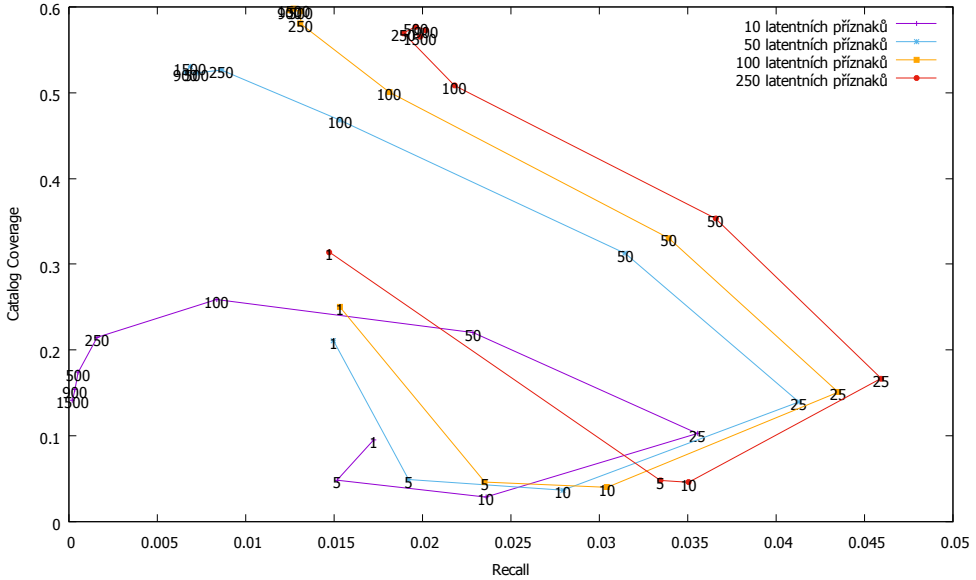

Obrázek 4.3: Dosažené výsledky pro faktorizační modely s různými dimenzemi latentních vektorů při použití knihovny LIBMF na datasetu MovieLens při doporučování 10 položek. Hodnoty na křivkách udávají dimenzi latentních vektorů. Všechny faktorizační modely byly naučeny 1 500 trénovacími iteracemi.

knihovny LIBMF nastavena na 0.1, by v případě datasetu MovieLens znamenala vygenerování pouze takových faktorizačních modelů, se kterými by bylo dosaženo prakticky nulových hodnot v obou sledovaných metrikách. Toto zjištění jen potvrzuje domněnku, že v případě nasazení algoritmu gradientního sestupu na faktorizaci reálných dat s úmyslem získání co nejkvalitnějšího faktorizačního a následně doporučovacího modelu se nelze spoléhat na výchozí hodnoty parametrů faktorizačního algoritmu, nýbrž že bude třeba věnovat vždy jisté úsilí zjištění optimálních hodnot těchto parametrů, případně tento proces automatizovat.

### 4.3.2 Experimenty s knihovnou Mahout

Provedené experimenty s knihovnou LIBMF ukázaly, že doporučovací modely naučené touto faktorizační knihovnou dosahují poměrně nízkých hodnot ve sledovaných metrikách, zejména v metrice recall. Abych mohl určit, zda jde o vlastnost algoritmu SGD, nebo jen konkrétní implementace tohoto algoritmu knihovnou LIBMF, rozhodl jsem se provést obdobné experimenty ještě s některou další existující implementací algoritmu SGD. Jak jsem již uvedl v kapitole 3.2, zvolil jsem k tomuto účelu knihovnu vytvořenou a dále vyvíjenou v rámci projektu Apache Mahout.

Podobně jako v případě knihovny LIBMF jsem nejprve vyzkoušel, jak se vyvíjí hodnoty catalog coverage a recall při použití faktorizačních modelů naučených postupně vzrůstajícím počtem trénovacích iterací. Experiment jsem provedl se shodnými parametry, jako v případě knihovny LIBMF, tedy pro shodné dimenze vektorů latentních příznaků a stejnou hodnotu regularizačního parametru $\lambda = 0.01$. Dosažené výsledky jsou zobrazeny v grafu 4.4.




Obrázek 4.4: Vliv různého množství trénovacích iterací metody SGD implementované knihovnou Mahout na dosažené hodnoty metrik recall a catalog coverage na datasetu MovieLens při doporučování 10 položek. Hodnoty na křivkách udávají počet trénovacích iterací, použitá hodnota regularizace $\lambda = 0.01$.

Výsledky experimentu ukázaly, že s použitím jiné implementace může být dosaženo výrazně odlišných hodnot. Zejména v případě metriky catalog coverage bylo za použití faktorizačních modelů produkovaných knihovnou Mahout dosaženo až třikrát vyšší hodnoty, než bylo dosaženo při použití modelů produkovaných při shodném regularizačním parametru knihovnou LIBMF. Naopak v případě metriky recall vykázaly obě knihovny podobné výsledky, takže lze usuzovat, že jde v tomto případě spíše o vlastnost samotného algoritmu gradientního sestupu, přesněji o vlastnost gradientního sestupu optimalizovaného na hodnotu RMSE.

Zajímavý je ale odlišný tvar křivek znázorňujících vývoj hodnot catalog coverage a recall v průběhu učení. Zatímco v případě knihovny LIBMF přináší stoupající počet trénovacích iterací lepší výsledky v obou metrikách catalog coverage i recall, v případě knihovny Mahout se se stoupajícím počtem trénovacích iterací zlepšují pouze hodnoty catalog coverage, zatímco dosažené hodnoty metriky recall od určitého počtu trénovacích iterací klesají.

Abych mohl vysvětlit příčinu tohoto odlišného chování obou knihoven, pokusil jsem se analyzovat jejich zdrojové kódy, které jsou vzhledem k šíření knihoven jako Open source volně k dispozici. Obě knihovny provádějí gradientní sestup způsobem popsaným v 2.3, nicméně v obou implementacích jsou určité rozdíly. Například knihovna Mahout trénuje vedle hodnot latentních příznaků také hodnoty zaujetí jednotlivých uživatelů a položek (verze statických hodnot zaujetí – viz. 2.3.1), zatímco knihovna LIBMF s hodnotami zaujetí uživatelů a položek vůbec nepracuje.

Poměrně zásadním rozdílem obou knihoven je práce s koeficientem rychlosti učení $\eta$ (learning rate) v průběhu iterací gradientního sestupu. Teoretická úvaha říká, že po neinformované inicializaci matic $\mathbf{U_k}$ a $\mathbf{V_k}$ se algoritmus nalézá daleko od oblasti globálního optima, takže během počátečních trénovacích iterací může dělat velké kroky proti směru gradientu, aby se k tomuto optimu rychle přiblížil. Jak se však vzdálenost od globálního optima snižuje, může být vhodné velikost těchto kroků snižovat (snižovat hodnotu $\eta$), jinak se může stát, že vzhledem k délce kroku bude algoritmus optimum stále „přeskakovat“. Knihovna Mahout proto implementuje metodu simulovaného žíhání reprezentovaného parametrem $d \in \langle 0, 1 \rangle$, pomocí něhož je hodnota learning rate v průběhu iterací snižována. Během počátečních experimentů s knihovnou Mahout (při ověřování vhodných rozsahů parametrů) jsem zkoušel volit i několik málo různých hodnot parametru $d$, nicméně s žádnou vyzkoušenou hodnotou jsem nedosáhl lepších výsledků, než v případě konstantní hodnoty learning rate. Proto jsem ve všech experimentech s knihovnou Mahout pracoval s $d = 1$, a to i přesto, že se pravděpodobně (z prosté logické úvahy) nejedná o optimální hodnotu.

Naopak knihovna LIBMF umožňuje volit pouze výchozí hodnotu parametru $\eta$ a možnost ovlivnit vývoj tohoto parametru v průběhu trénovacích iterací uživateli nedává. Hodnotu learning rate totiž nastavuje sama, a to pro každý vektor $\mathbf{p_u}$ a každý vektor $\mathbf{q_i}$ zvlášť, v závislosti na průměrné hodnotě gradientů všech latentních příznaků daného vektoru. Použitá adaptivní technika se nazývá Reduced Per-coordinate Schedule (RPCS) a jde o metodu odvozenou autory knihovny LIBMF z již dříve představené a úspěšné, avšak výpočetně náročné techniky PCS, která zvlášť upravuje hodnotu learning rate v závislosti na gradientu pro každý jednotlivý latentní příznak [22]. Je tedy zřejmé, že jednotlivé knihovny produkují odlišné faktorizační modely a s použitím těchto modelů tak může být dosahováno odlišných výsledků.

S knihovnou Mahout jsem zopakoval i druhý experiment, tedy vyhodnocení doporučovacího modelu využívajícího faktorizační modely různých dimenzí latentních vektorů a naučené s různými hodnotami regularizačního parametru $\lambda$, přičemž jsem opět nechal všechny faktorizační modely naučit 1 500 trénovacími iteracemi. Výsledky, prezentované v grafu 4.5, zde ukazují podobné chování faktorizačních modelů, jako v případě knihovny LIBMF. V souladu s předchozími zjištěními ohledně vysokého počtu trénovacích iterací knihovny Mahout je zde dosaženo nižších hodnot recall oproti LIBMF. Potvrdilo se také,

že použití hodnoty regularizačního parametru $\lambda = 0.05$ vede k faktorizačním modelům, s nimiž je dosahováno jen velmi nízkých hodnot v obou sledovaných metrikách. Přitom shodně s knihovnou LIBMF je i v případě knihovny Mahout výchozí hodnotou $\lambda = 0.1$. Při použití hodnot $\lambda = 0.01$ a $\lambda = 0.005$ se však mezi příslušnými faktorizačním modely projevily pouze minimální rozdíly.

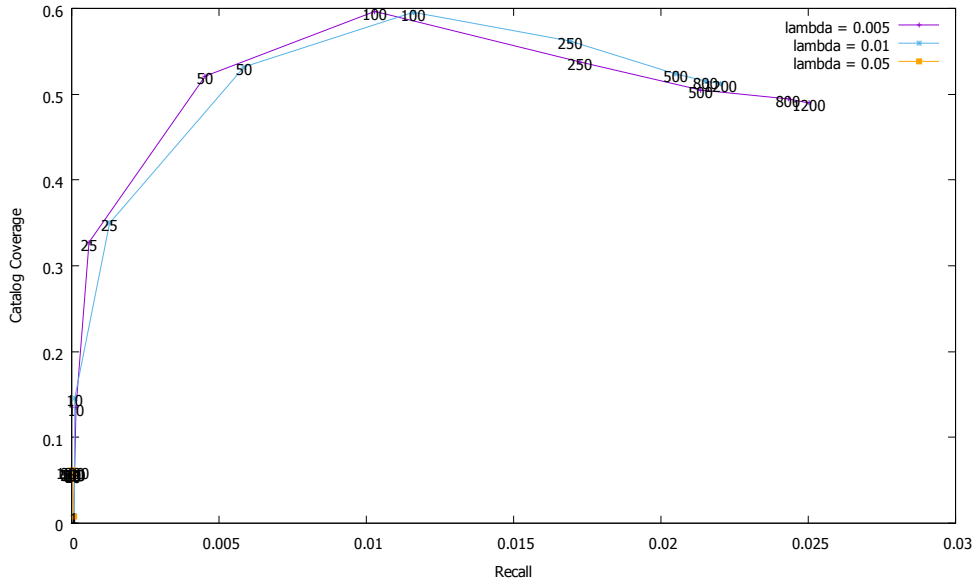
Použití dvou různých knihoven, nabízejících různé implementace shodného faktorizačního algoritmu, mi také umožnilo porovnat obě implementace z hlediska času potřebného k provedení shodných faktorizačních úloh. Obě knihovny jsem porovnal na stejných trénovacích datech jednoho foldu křížové validace, a to při faktorizaci 1 500 trénovacími iteracemi a pro různý počet latentních příznaků (různou dimenzi latentních vektorů). Protože jsou obě knihovny připraveny na paralelní zpracování dat, každý proces faktorizace jsem spouštěl s osmi vlákny na shodném počtu výpočetních jader. Naměřené výsledky, zobrazené v grafu 4.6, jsou průměrné časy ze tří opakovaných spuštění každé faktorizační úlohy, u obou knihoven byly nicméně rozdíly v naměřených hodnotách při opakovaném spuštění maximálně v řádu jednotek sekund.

Výsledky porovnání časové náročnosti obou implementací při faktorizaci shodných dat ukazují, že knihovna LIBMF potřebuje pro provedení faktorizačních úloh několikanásobně méně času, než potřebuje pro stejné úlohy knihovna Mahout. Tato skutečnost dokládá splnění cílů, které si stanovili autoři knihovny LIBMF, totiž implementovat algoritmus stochastického gradientního sestupu schopný faktorizovat data v co nejkratším čase a při co nejefektivnějším využití všech dostupných výpočetních jader procesoru. K dosaženým výsledkům přispěla volba programovacího jazyka, kterým je jazyk C++, zatímco knihovna Mahout je implementována v jazyku Java, kde jsou zdrojové kódy místo do binárního kódu překládány do bytecodu zpracovávaného při běhu virtuálním strojem. Rychlosti knihovny LIBMF jistě pomáhá i zmiňované využití různých speciálních instrukcí procesoru (viz. 3.1.4).

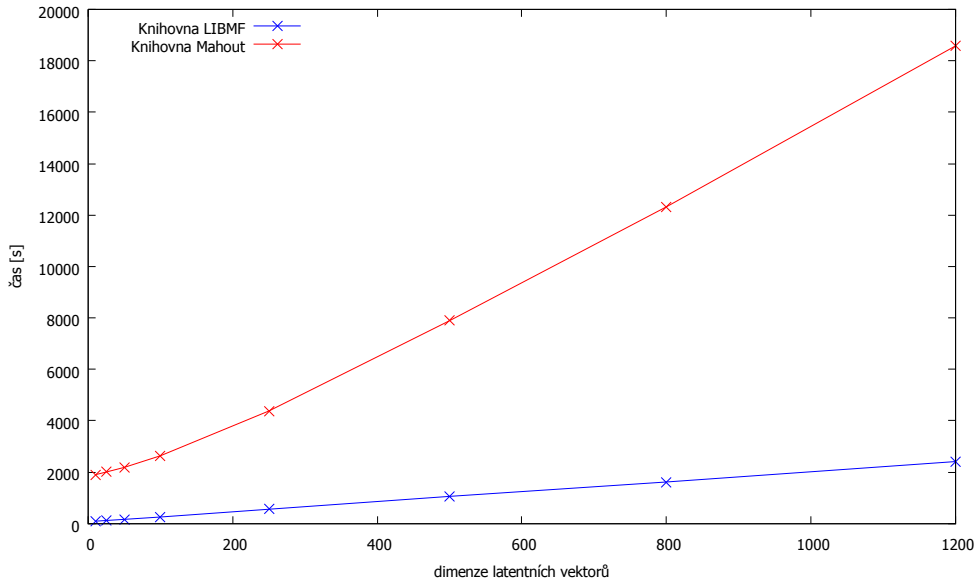
Nicméně ani v případě knihovny Mahout netrvá faktorizace dat, velikostně odpovídajících datasetu MovieLens 10M, neúměrně dlouho, alespoň tedy při provádění faktorizace na více výpočetních jádrech a při rozumné volbě dimenze vektorů latentních příznaků. Časové nároky pro faktorizaci těchto dat tedy nejsou překážkou pro praktické nasazení metody maticové faktorizace, kdy je třeba faktorizaci měnících se dat v určitých časových intervalech periodicky opakovat.

## 4.4 Experimenty na datasetu GoOut

Po provedení experimentů na datasetu MovieLens jsem stejné experimenty zopakoval na datasetu GoOut, abych ověřil chování faktorizačních algoritmů na různých datech. Vzhledem k tomu, že dataset GoOut obsahuje více než 30 krát tolik uživatelů, kolik jich obsahuje dataset MovieLens (viz. 4.1.1 a




Obrázek 4.5: Dosažené výsledky pro faktorizační modely s různými dimenzemi latentních vektorů při použití knihovny Mahout na datasetu MovieLens. Hodnoty na křivkách udávají dimenzi vektorů latentních příznaků. Všechny faktorizační modely byly naučeny 1 500 trénovacími iteracemi.




Obrázek 4.6: Porovnání knihoven LIBMF a Mahout z hlediska doby běhu algoritmů na shodných faktorizačních úlohách při provádění 1 500 trénovacích iterací.

4.1.2), rozhodl jsem se provádět vyhodnocování pouze na jedné desetině testovacích uživatelů, což lze realizovat volbou příslušného parametru při spuštění vyhodnocovacího frameworku. V opačném případě – při vyhodnocování na všech dostupných uživatelích (a při současném provádění křížové validace) – by totiž jeden experiment netrval v řádu hodin nebo dnů, ale v řádu týdnů.

Vyhodnocováním doporučovacího modelu na podmnožině dostupných uživatelů je sice uměle snižována hodnota metriky catalog coverage, pro vzájemné porovnání algoritmů maticové faktorizace s algoritmem nejbližších sousedů (viz. dále) to ale není podstatné, pokud je i tento algoritmus vyhodnocován na stejné podmnožině uživatelů.

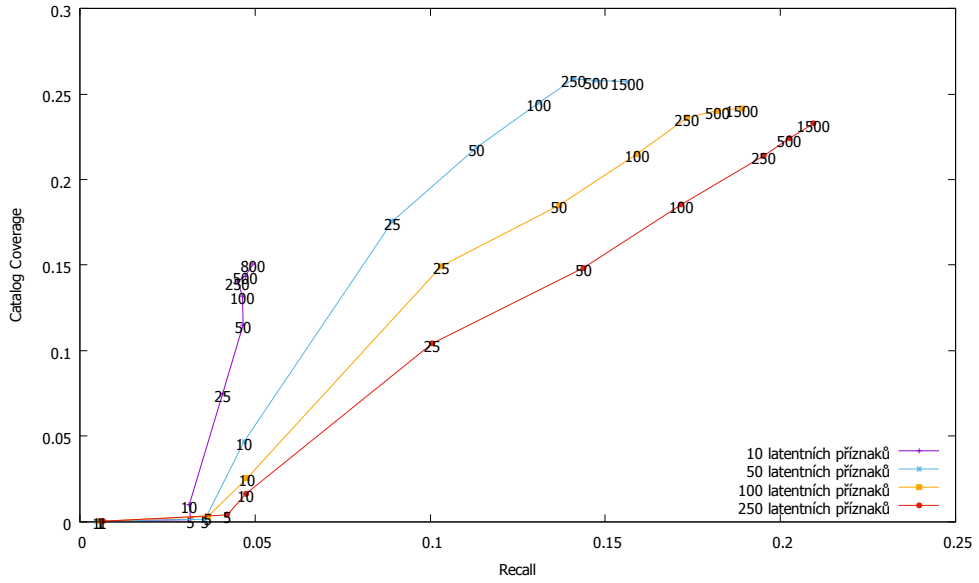
### 4.4.1 Experimenty s knihovnou LIBMF

Stejně jako v případě datasetu MovieLens jsem nejprve provedl experimenty s knihovnou LIBMF, přičemž jsem použil stejné hodnoty parametrů faktorizačního algoritmu, jako v předchozích experimentech. Nejprve jsem opět ověřil, jak se vyvíjí dosažené hodnoty recall a catalog coverage při vzrůstajícím počtu trénovacích iterací a při různých dimenzích vektorů latentních příznaků – výsledný graf je zobrazen na obrázku 4.7. Dále jsem provedl experiment s faktorizačními modely různých dimenzí latentních vektorů, kdy jsem všechny tyto modely nechal naučit 1 500 trénovacími iteracemi a při použití tří různých hodnot regularizačního parametru $\lambda$ – dosažené výsledky jsou vyneseny v grafu 4.8.

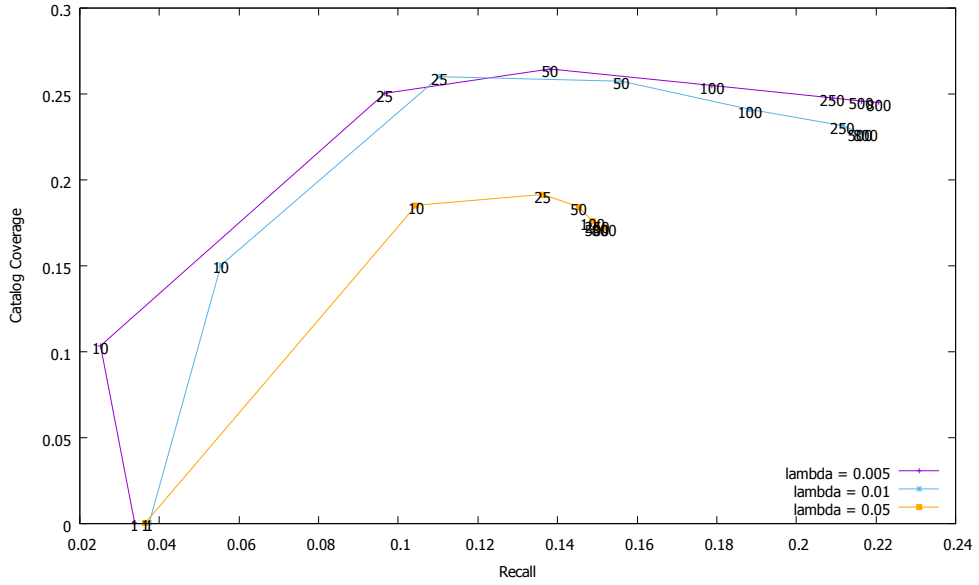
Naměřené výsledky prvního experimentu potvrzují schopnost knihovny LIBMF dosahovat se vzrůstajícím počtem trénovacích iterací lepších výsledků v obou sledovaných metrikách – recall i catalog coverage. Oproti datasetu MovieLens jsou navíc dosažené hodnoty v obou metrikách výrazně vyšší. Podobně si vede na datasetu GoOut také knihovna Mahout i algoritmus User k-NN (viz. dále). Lze tedy říci, že tento dataset s implicitní zpětnou vazbou, která je kódována binárními hodnotami v matici interakcí, lze těmito algoritmy snáze zpracovat a výsledné modely mají lepší schopnost generalizace.

Dosažené výsledky potvrzuje i druhý experiment, který také ukazuje, že větší dimenze latentních příznaků zde nevede k takovému propadu catalog coverage, jako na datasetu MovieLens. Možným vysvětlením je výrazně vyšší počet uživatelů a položek datasetu GoOut oproti MovieLens, čímž vzniká mnohem rozměrnější matice interakcí. Tuto větší matici interakcí potom stejně dobře vysvětlují i faktorizační modely s větší dimenzí matic $\mathbf{U_k}$ a $\mathbf{V_k}$. Všimnout si lze také odlišného vývoje oproti datasetu MovieLens při použití hodnoty regularizačního parametru $\lambda = 0.05$. Zatímco v případě MovieLens bylo při použití této hodnoty dosahováno téměř nulových hodnot v obou sledovaných metrikách, v případě datasetu GoOut je situace výrazně příznivější. Stále však platí, že výchozí hodnota $\lambda = 0.1$ zde není vhodnou volbou.

Při sledování dosažených výsledků je také třeba si uvědomit, že hodnoty metriky catalog coverage, byť jsou vyšší než na datasetu MovieLens, jsou




Obrázek 4.7: Vliv různého množství trénovacích iterací metody SGD implementované knihovnou LIBMF na dosažené hodnoty metrik recall a catalog coverage na datasetu GoOut. Hodnoty na křivkách udávají počet trénovacích iterací, použitá hodnota regularizace $\lambda = 0.01$.

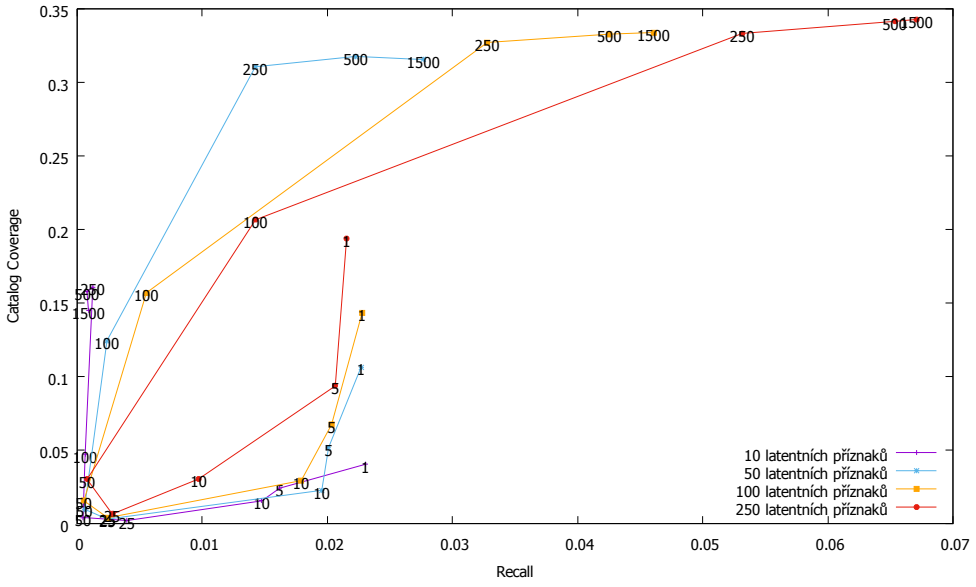



Obrázek 4.8: Dosažené výsledky pro faktorizační modely s různými dimenzemi latentních vektorů při použití knihovny LIBMF na datasetu GoOut. Hodnoty na křivkách udávají dimenzi vektorů latentních příznaků. Všechny faktorizační modely byly naučeny 1 500 trénovacími iteracemi.

negativně ovlivněny provedením experimentů pouze na desetině testovacích uživatelů. Lze předpokládat, že při vyhodnocení doporučovacího modelu na množině všech dostupných uživatelů by se dosažená hodnota metriky catalog coverage ještě zvýšila. Ve srovnání s datasetem MovieLens pracuje tedy faktorizační algoritmus na datasetu GoOut dokonce výrazně lépe.

### 4.4.2 Experimenty s knihovnou Mahout

Také na datasetu GoOut jsem zopakoval oba experimenty i s knihovnou projektu Apache Mahout. Výstupem jsou opět dva grafy s naměřenými výsledky – graf na obrázku 4.9 zobrazuje vliv různého množství trénovacích iterací na dosažené hodnoty metrik catalog coverage a recall, druhý graf na obrázku 4.10 pak prezentuje hodnoty dosažené při použití faktorizačních modelů různých dimenzí vektorů latentních příznaků naučených 1 500 trénovacími iteracemi při použití různých hodnot regularizace.




Obrázek 4.9: Vliv různého množství trénovacích iterací metody SGD implementované knihovnou Mahout na dosažené hodnoty metrik recall a catalog coverage na datasetu GoOut při doporučování 10 položek. Hodnoty na křivkách udávají počet trénovacích iterací, použitá hodnota regularizace $\lambda = 0.01$.

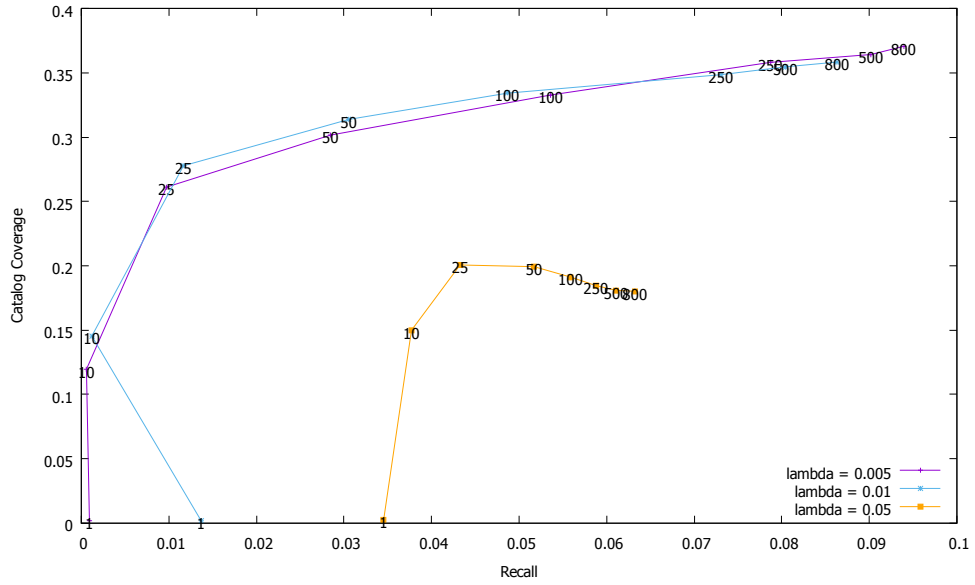
Výsledky experimentů ukazují, že vývoj dosahovaných hodnot recall a catalog coverage během trénování na datasetu GoOut je zcela odlišný oproti vývoji během trénování na datasetu MovieLens. Důvodem může být jiná povaha faktorizovaných dat, kdy v případě datasetu GoOut je faktorizována binární matice namísto matice reálných čísel datasetu MovieLens (ostatně i algoritmus User k-NN vykazuje na datasetu GoOut odlišné chování oproti

datasetu MovieLens – viz grafy 4.11 a 4.12). Chování knihovny Mahout na datasetu GoOut je spíše podobné chování knihovny LIBMF. V čem se však obě knihovny liší (ať už na datasetu GoOut, nebo MovieLens) je dosažená hodnota catalog coverage a recall při provedení pouze malého počtu trénovacích iterací. Zatímco knihovna Mahout dosahuje v těchto případech poměrně vysokých hodnot (zejména v metrice catalog coverage), knihovna LIBMF dosahuje při malém počtu trénovacích iterací v obou metrikách takřka nulových hodnot, což spíše odpovídá očekávaní.

Rozdíl v tomto chování obou knihoven očividně nelze vysvětlit používáním zaujetí v případě knihovny Mahout nebo rozdílnými způsoby práce s koeficientem rychlosti učení $\eta$. Ty by se totiž po provedení například jediné trénovací iterace neměly na dosažených hodnotách tak výrazně projevit. Pro objasnění této skutečnosti jsem znovu analyzoval zdrojové kódy obou knihoven, abych zjistil v čem se obě implementace gradientního sestupu liší, že dosahují na začátku trénovacího procesu tak výrazně odlišných výsledků. Provedená analýza zdrojových kódů potvrdila můj předpoklad, že každá knihovna používá jiný způsob inicializace počátečních hodnot matic $\mathbf{U_k}$ a $\mathbf{V_k}$. Zatímco knihovna LIBMF inicializuje matice pseudonáhodnými hodnotami z rovnoměrného rozdělení na intervalu $[0, 1]$, knihovna Mahout inicializuje obě matice pseudonáhodnými hodnotami z normálního (Gaussova) rozdělení se střední hodnotou $\mu = 0$ a směrodatnou odchylkou $\sigma = 1$. Pokud tedy, v případě knihovny LIBMF, mají některé položky „štěstí“ na vysoké hodnoty latentních příznaků, pak (bez provedení dostatečného množství trénovacích iterací) jsou zřejmě doporučovány všem uživatelům bez ohledu na jejich latentní vektory a hodnoty catalog coverage i recall zůstávají nízké. V případě knihovny Mahout však při predikci neznámých hodnot záleží také na znaménku jednotlivých latentních příznaků, takže nelze říci, že některé položky budou doporučovány všem uživatelům. Bez dostatečného trénování jsou doporučovány takřka náhodné položky – rozptyl těchto položek je větší a úměrně tomu je vyšší i hodnota catalog coverage. Navíc doporučováním náhodných položek se algoritmus častěji strefí do skutečných preferencí uživatele, než když všem doporučuje tytéž položky, proto je oproti knihovně LIBMF vyšší i hodnota recall.

Jak je uvedeno v 2.3, při použití metody gradientního sestupu na způsobu inicializace matic $\mathbf{U_k}$ a $\mathbf{V_k}$ obecně nezáleží, což je jistě pravda při provádění většího počtu trénovacích iterací, kdy by měl algoritmus z libovolného počátečního stavu dospět do oblasti globálního optima. Při provádění velmi malého počtu trénovacích iterací však počáteční inicializace dosažené výsledky značně ovlivňuje, což se projevilo právě v mém experimentu při vyhodnocování faktorizačních modelů získaných malým počtem trénovacích iterací. Toto zjištění však není žádnou překážkou praktickému nasazení, neboť v praxi by se vždy provádělo dostatečné množství trénovacích iterací na to, aby bylo dosaženo globálního optima.

Výsledky experimentu s různými dimenzemi latentních vektorů a různými hodnotami $\lambda$ jsou zde podobné výsledkům knihovny LIBMF – mám tím na




Obrázek 4.10: Dosažené výsledky pro faktorizační modely s různými dimenzemi latentních vektorů při použití knihovny Mahout na datasetu GoOut. Hodnoty na křivkách udávají dimenzi vektorů latentních příznaků. Všechny faktorizační modely byly naučeny 1 500 trénovacími iteracemi.

mysli například chování algoritmu při volbě $\lambda = 0.05$. Vlastnost knihovny Mahout produkovat faktorizační modely, s nimiž je dosahováno vyšších hodnot catalog coverage a naopak nižších hodnot recall ve srovnání s modely knihovny LIBMF, zůstává zachována. I ve výsledcích tohoto experimentu je dosažená hodnota catalog coverage negativně ovlivněna vyhodnocením na pouhé desetině testovacích uživatelů. Pro vzájemné porovnání algoritmů vyhodnocovaných na stejné množině dat to nepředstavuje problém, při vyhodnocení na všech testovacích uživatelích by se však hodnota catalog coverage jistě dále zvýšila.

## 4.5 Porovnání maticové faktorizace s algoritmem nejbližších sousedů

Výsledky provedených experimentů mi dále posloužily ke srovnání použité metody maticové faktorizace, tedy faktorizace metodou stochastického gradientního sestupu, s algoritmem nejbližších sousedů ve verzi User k-NN, a to opět z hlediska metrik recall a catalog coverage. K získání výsledků algoritmu nejbližších sousedů jsem použil implementaci User k-NN, která je již přítomna v doporučovacím systému společnosti Recombee. Tento algoritmus jsem pomocí vyhodnocovacího frameworku vyhodnotil stejným způsobem, jako do-

poručovací modely, pracující na základě maticové faktorizace, a to vždy na stejných datech, tedy například na desetině testovacích uživatelů v případě datasetu GoOut. Nejlepší výsledky dosažené algoritmem nejbližších sousedů i metodou maticové faktorizace při použití obou knihoven jsem zobrazil ve společném grafu, aby byly jednoduše porovnatelné. Graf 4.11 tak zobrazuje hodnoty metrik recall a catalog coverage dosažené na datasetu MovieLens a graf 4.12 zobrazuje hodnoty obou metrik dosažené na datasetu GoOut. Všechny faktorizační modely byly naučeny 1 500 trénovacími iteracemi a s příslušnou regularizací $\lambda$, parametr beta v případě algoritmu User k-NN udává míru penalizace bestselerů.

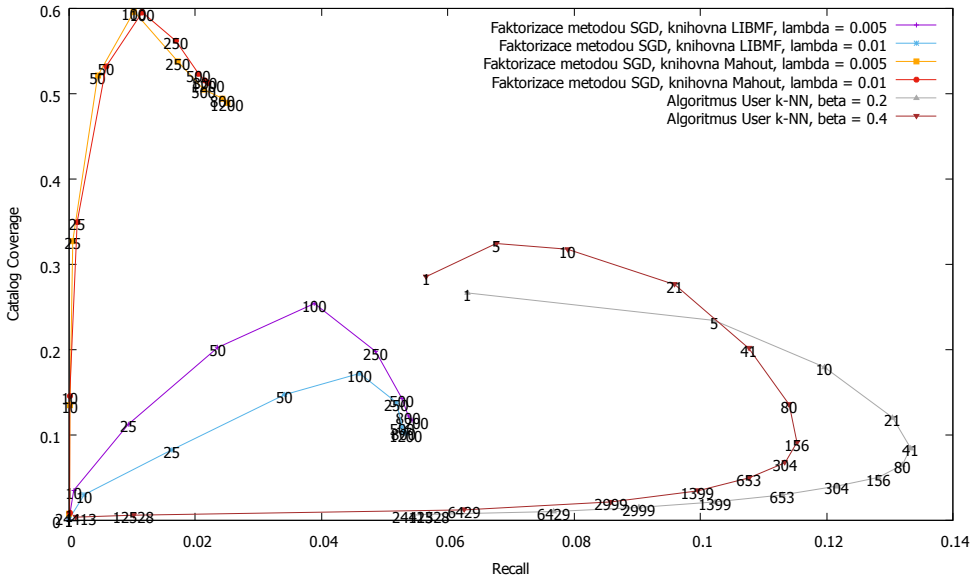
Z výsledků porovnání je zřejmé, že na datasetech MovieLens i GoOut dosahují doporučovací modely, pracující na základě faktorizačních modelů optimalizovaných na hodnotu RMSE, výrazně nižších hodnot metriky recall, než modely používající algoritmus nejbližších sousedů. Z hlediska metriky catalog coverage již situace tak jednoznačná není. Na datasetu MovieLens bylo při použití knihovny LIBMF dosaženo srovnatelných výsledků jako s algoritmem nejbližších sousedů, při použití knihovny Mahout jsou dokonce dosažené výsledky výrazně lepší. Znamená to, že doporučení generovaná na základě maticové faktorizace celkem dobře pokrývají množinu nabízených položek, v porovnání s algoritmem User k-NN je však testovacím uživatelům doporučováno procentuálně méně položek z těch, se kterými skutečně interagovali.

Naopak na datasetu GoOut, obsahujícím implicitní hodnoty interakcí, dosáhl algoritmus nejbližších sousedů lepších výsledků v obou metrikách. Je možné, že hodnoty recall, dosažené modely pracujícími na základě maticové faktorizace, by ještě mírně vzrostly při použití faktorizačních modelů s větší dimenzí latentních vektorů. Nicméně v takovém případě by již faktorizační modely obsahovaly výrazně více hodnot než původní řídká matice interakcí, takže by nedošlo k žádné redukci množství dat a i proces vybavování se může zpomalit. Proto takové experimenty nemají velký smysl a sám jsem je (také z důvodu časové náročnosti) již neprováděl.

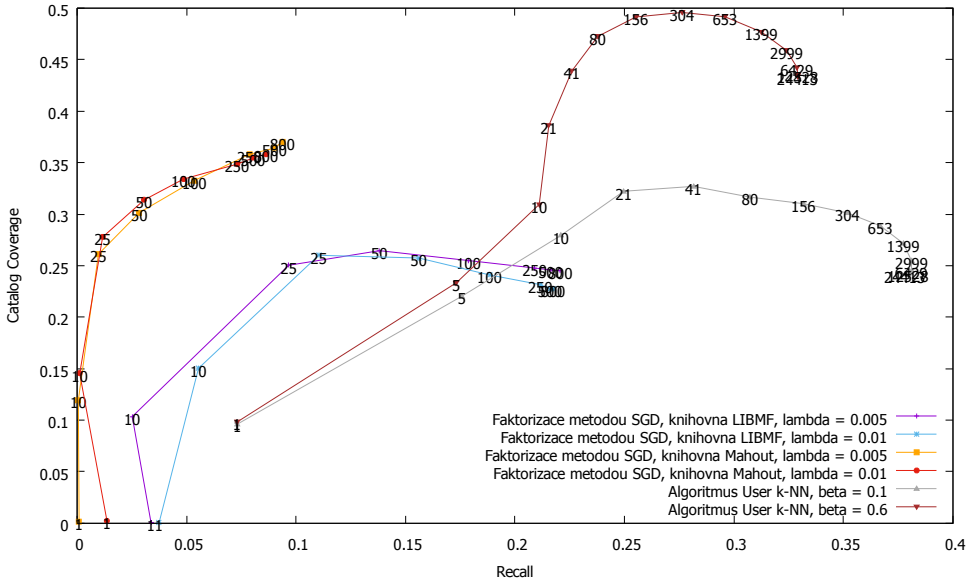
## 4.6 Diskuse dosažených výsledků

V předchozích kapitolách jsem popsal prováděné experimenty a prezentoval dosažené výsledky, zároveň jsem se snažil odůvodnit rozdíly v dosažených hodnotách při použití dvou různých implementací téhož algoritmu, tedy knihoven LIBMF a Mahout. Na závěr této práce se pokusím zhodnotit úspěšnost použitého faktorizačního algoritmu a vysvětlit důvody odlišných výsledků při srovnání a algoritmem User k-NN.

Provedené experimenty potvrdily, že zvolená faktorizační metoda, tedy metoda stochastického gradientního sestupu, dokáže faktorizovat i velká data o milionech záznamů v přijatelném čase, který umožňuje praktické nasazení faktorizačního algoritmu a periodické opakování faktorizace měnících se dat




Obrázek 4.11: Nejlepší výsledky dosažené maticovou faktorizací a algoritmem nejbližších sousedů na datasetu MovieLens. Hodnoty na křivkách udávají dimenzi vektorů latentních příznaků (v případě maticové faktorizace) a hodnotu parametru $k$ – počet sousedů (v případě algoritmu User k-NN).




Obrázek 4.12: Nejlepší výsledky dosažené maticovou faktorizací a algoritmem nejbližších sousedů na datasetu GoOut. Hodnoty na křivkách udávají dimenzi vektorů latentních příznaků (v případě maticové faktorizace) a hodnotu parametru $k$ – počet sousedů (v případě algoritmu User k-NN).

tak, aby vektory latentních příznaků uživatelů i položek, tedy matice $\mathbf{U_k}$ a $\mathbf{V_k}$, zůstávaly aktuální. Ani paměťové nároky této metody nejsou větší, než je nezbytné – metoda potřebuje uchovávat známé hodnoty matice interakcí (typicky ve formě řídké matice) a dále plné matice $\mathbf{U_k}$ a $\mathbf{V_k}$, tedy pouze ta data, která jsou pro výpočet faktorizace nezbytná a jejichž množství nelze dále redukovat. Například knihovna LIBMF navíc nabízí i možnost použití pevného disku jako další paměti pro faktorizaci opravdu velkých dat, kdy už se všechny nezbytné informace do operační paměti nevejdou, takže velikost dat, která lze faktorizovat, není teoreticky nijak limitována.

Výsledky, prezentované například v grafu 4.2, potvrdily rovněž schopnost algoritmu stochastického gradientního sestupu naučit se s dostatečně malou chybou na trénovací data. Uvedený graf demonstruje, že algoritmus během trénovacích iterací skutečně směřuje do oblasti minima z hlediska chyby RMSE, a že tedy správně optimalizuje faktorizační model tak, aby výsledná matice $\mathbf{A_k}$, získaná jako $\mathbf{A_k} = \mathbf{U_k}\mathbf{V_k}^T$, co nejpřesněji aproximovala původní matici interakcí $\mathbf{A}$.

Nicméně, jak rovněž prokázaly dosažené výsledky, optimalizace na RMSE nevede vždy k nejlepším dosaženým výsledkům z hlediska metrik recall a catalog coverage. Jak je patrné z několika výše uvedených grafů, zejména z grafů ukazujících vývoj dosažených hodnot recall a catalog coverage v průběhu rostoucího množství trénovacích iterací, dochází často k situaci, kdy sice klesá hodnota chyby RMSE, ale dosažené hodnoty z hlediska metrik recall a catalog coverage se zhoršují. Dobře patrné je to například v grafu na obrázku 4.1, kdy při srovnání s grafem 4.2 vidíme, že hodnota RMSE je v případě faktorizačního modelu s 250 latentními příznaky nejnižší, ale dosažené hodnoty catalog coverage jsou horší než v případě faktorizačního modelu se 100 latentními příznaky, jehož RMSE je vyšší. Zřejmé je to také v případě grafu 4.4 zobrazujícího vývoj hodnot catalog coverage a recall při použití knihovny Mahout na datasetu MovieLens. Již ze samotného tvaru křivek je zřejmé, že s klesající chybou RMSE (která zde není vizualizována, je ale podstatou fungování algoritmu gradientního sestupu) od určitého okamžiku zároveň klesá i dosažená hodnota recall, což je opakem požadovaného chování. V ostatních případech k tomuto jevu sice nedochází, přesto jsou ale dosažené hodnoty metriky recall nižší ve srovnání s těmi, kterých dosahuje algoritmus nejbližších sousedů ve verzi User k-NN. To sice nutně neznamená, že doporučovací model, pracující na základě těchto faktorizačních modelů, generuje nevhodná doporučení, tato doporučení však nemusí být pro cílové uživatele tak zajímavá, jako doporučení generovaná algoritmem User k-NN.

Jak je už výše naznačeno, hlavní příčinou problému nízké hodnoty recall je skutečnost, že faktorizační modely jsou během trénování optimalizovány na jinou metriku, než z hlediska které jsou později vyhodnocovány. Chcemeli zlepšit dosahovanou hodnotu metriky recall, bude třeba upravit chybovou funkci minimalizovanou algoritmem gradientního sestupu tak, aby byly faktorizační modely místo na RMSE optimalizovány alespoň částečně i na recall.

Problémem je však již skutečnost, že hodnota recall by musela být během trénování neustále vyhodnocována, což by samo o sobě proces trénování neúnosně zpomalovalo.

Tímto problémem se zabýval také Harald Steck, který ve své práci zavádí novou metriku ATOP, již lze s úspěchem použít k optimalizaci hodnot precision a recall [21] během gradientního sestupu. Také tato metrika je však pro účely trénování výpočetně velmi náročná (ostatně vychází z metriky recall), takže není vhodná k praktickému použití. Proto autor zavedl také upravenou chybovou funkci, s níž lze údajně dosáhnout uspokojivých výsledků při současném zachování výpočetní náročnosti algoritmu gradientního sestupu. Hlavní myšlenkou je trénování modelu nejen na známých hodnotách matice interakcí, ale i na hodnotách neznámých, které jsou pro účely trénování vyplněny zvolenou konstantou $r_m$, například i nulou. Velikosti chyb vznikajících na všech těchto hodnotách jsou potom váženy podle toho, zda se jedná o známou nebo uměle vloženou hodnotu. Optimalizační funkce má tedy tvar:

$$\min_{\mathbf{p}*,\mathbf{q}*} \sum_{(u,i)\in\kappa} w_{u,i}(r_{u,i}^{o\&i} - \mathbf{p_u}\mathbf{q_i}^T)^2 + \lambda(\parallel \mathbf{p_u} \parallel^2 + \parallel \mathbf{q_i} \parallel^2)$$

kde $r_{u,i}^{o\&i}$ jsou známé i uměle vložené (observed & imputed) hodnoty matice interakcí a $w_{u,i}$ je váha určující jak se bude daná chyba podílet na úpravách příslušných latentních vektorů. Hodnota $w_{u,i}$ je 1 v případě, že $r_{u,i}^{o\&i}$ je známá hodnota, a nějaká malá konstanta, pokud je $r_{u,i}^{o\&i}$ uměle vložená hodnota $r_m$ – například autorovi vyplynula z experimentů jako vhodná hodnota 0.002 [21]. Tím, že jsou při procesu učení zvýhodňovány známé hodnoty matice interakcí a v případě všech neznámých hodnot je model tlačen k tomu, aby predikoval spíše nižší hodnoty, měly by být při použití výsledného faktorizačního modelu doporučovány ve větší míře ty položky, se kterými testovací uživatelé skutečně interagovali, čímž se hodnota recall zvýší. Nevýhodou tohoto postupu je ale skutečnost, že původně řídká matice interakcí se pro účely trénování stane plnou, což metodu gradientního sestupu zpomalí. Potom je třeba zvážit například použití metody ALS namísto metody gradientního sestupu, neboť metoda ALS by měla být v případě plných matic efektivnější (viz. 2.4).

# Závěr

V rámci této práce jsem analyzoval metodu singulárního rozkladu matice (SVD), která představuje základ mnohých dalších metod maticové faktorizace používaných v oblasti kolaborativního filtrování. V návaznosti na zjištěné nevýhody praktického použití metody SVD jsem analyzoval dvě další metody, které jsou již pro praktické nasazení vhodnější – algoritmus stochastického gradientního sestupu (SGD) a algoritmus alternujících nejmenších čtverců (ALS). Právě algoritmus SGD jsem poté zvolil pro provedení závěrečných experimentů.

Dále jsem v rámci této práce provedl rešerši dostupných implementací faktorizačních algoritmů, zejména zvolené metody SGD. Ze zkoumaných implementací jsem vybral dvě volně dostupné knihovny (LIBMF a Apache Mahout), abych s nimi provedl potřebné experimenty a posoudil případné rozdíly v dosažených výsledcích při použití různých implementací. Nakonec jsem se seznámil se softwarovou architekturou společnosti Recombee a zjistil možnosti jejího použití při vyhodnocování faktorizačních modelů produkovaných oběma knihovnami.

Na základě těchto analýz jsem navrhl postup pro provádění experimentů s maticovou faktorizací. Z navrženého postupu vyplynula potřeba implementace dvou prvků nezbytných pro provedení těchto experimentů. Prvním z nich je metoda pro rozdělení dostupných dat na trénovací a testovací data pro jednotlivé foldy křížové validace a vygenerování příslušných souborů těchto dat. Druhým prvkem je doporučovací model schopný generovat doporučení na základě výsledků faktorizace, kdy tento model bylo třeba realizovat jako rozšíření stávající doporučovací komponenty, aby mohl být použit vyhodnocovací experiment. Oba tyto prvky jsem implementoval, abych mohl realizovat navržený postup provádění experimentů.

Po přípravě potřebného softwaru na testovacím serveru jsem provedl rozsáhlou sadu výpočetně náročných experimentů na dvou datasetech (MovieLens a GoOut) za účelem zjištění úspěšnosti doporučovacích modelů, pracujících na základě faktorizovaných dat, a to z hlediska různých metrik, především však

recall a catalog coverage. Hodnoty těchto metrik, dosažené v rámci experimentů s maticovou faktorizací, jsem porovnal s hodnotami dosaženými při použití algoritmu nejbližších sousedů ve verzi User k-NN. Zároveň s prováděním experimentů jsem analyzoval zdrojové kódy obou použitých knihoven, abych zdůvodnil rozdíly ve výsledcích, které se projevily při použití těchto dvou implementací algoritmu gradientního sestupu. Na závěr jsem provedl diskusi dosažených výsledků a nastínil možnosti zlepšení hodnoty recall.

Výstupy této práce jsou dva. Prvním z nich je navržený a realizovaný proces vyhodnocování faktorizačních algoritmů, který umožňuje jejich efektivní vyhodnocení z hlediska různých parametrů. Na základě výsledků vyhodnocování lze pak tyto algoritmy vzájemně srovnávat, stejně jako je lze srovnávat s dalšími metodami kolaborativního filtrování. Druhým výstupem této práce jsou samotné výsledky experimentů provedených s algoritmem stochastického gradientního sestupu na výše uvedených datasetech. Tyto experimenty prokázaly, že algoritmus SGD je schopen faktorizovat data v čase umožňujícím jeho praktické použití a že úspěšně optimalizuje hodnotu chyby RMSE. Zároveň však ukázaly, že při použití faktorizačních modelů optimalizovaných na RMSE nedosahují doporučovací modely tak dobrých výsledků (zejména z hlediska metriky recall), jako dosud používaný algoritmus User k-NN.

## Možnosti pokračování

Pokud by bylo možné ponechat proces vyhodnocování běžet dostatečně dlouhou dobu, bylo by jistě zajímavé nechat vyhodnotit algoritmus gradientního sestupu na všech testovacích uživatelích datasetu GoOut. Užitečné výsledky by mohly přinést také další experimenty, na které již v této práci z časových důvodů nedošlo – například experimenty s různými hodnotami parametru $\eta$, tedy koeficientu rychlosti učení.

Hlavní možnosti pokračování této práce však byly naznačeny již v závěrečné diskusi dosažených výsledků. Jedná se zejména o snahu zlepšit dosaženou hodnotu recall, tak aby se faktorizační algoritmus stal kvalitní alternativou k algoritmům nejbližších sousedů. Možné cesty k dosažení tohoto cíle jsou v podstatě dvě.

První možností je upravit chybovou funkci současného algoritmu stochastického gradientního sestupu tak, aby tento algoritmus neoptimalizoval faktorizační modely pouze na hodnotu chyby RMSE při současném ignorování vyhodnocovaných metrik catalog coverage a recall. Nejjednodušší možností, která vlastně úpravy zdrojových kódů ještě nutně nevyžaduje, je přitom vložení nul namísto neznámých hodnot matice interakcí daného datasetu. Podobný postup jsem sice již aplikoval při provádění experimentů na datasetu GoOut, v tomto případě jsem však (mimo jiné z důvodu časově náročné faktorizace) vložil mezi trénovací data pouze takové množství nul, které je rovno polovině množství známých hodnot. Přitom [21] doporučuje vložit nové hod-

noty místo všech neznámých hodnot matice interakcí. Druhou možností, jak potenciálně zvýšit hodnotu recall, je opustit algoritmus stochastického gradientního sestupu a vyzkoušet některé další algoritmy, které jsem v této práci neanalyzoval. Takovým algoritmem může být například SVD++, který je také implementován některými knihovnami popisovanými výše.

V současné době několik dalších studentů zkouší faktorizovat data alternativními metodami odlišnými od základní podoby algoritmu SGD prezentované a použité v této práci. Pro vyhodnocení jejich výsledných faktorizací bude použit mnou připravený proces vyhodnocování faktorizačních algoritmů, který je jedním z výstupů této práce. Na výsledcích mé práce tak bude i nadále stavěno při pokračujícím hledání optimálního způsobu faktorizace dat z pohledu dosahovaných hodnot metrik recall a catalog coverage.

# Použité zdroje


[1] KOREN, Y., R. BELL a Ch. VOLINSKY: *Matrix factorization techniques for recommender systems* [online]. IEEE Computer Society, 2009 [cit. 2016-03-06]. Dostupné z: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.147.8295&rep=rep1&type=pdf

[2] SARWAR, B., G. KARYPIS, J. KONSTAN a J. RIEDL: Incremental singular value decomposition algorithms for highly scalable recommender systems. In: *Fifth International Conference on Computer and Information Science* [online]. 2002 [cit. 2016-03-11]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.7894&rep=rep1&type=pdf

[3] KLEMA, V. C. a A. J. LAUB: *The singular value decomposition: Its computation and some applications* [online]. Automatic Control, IEEE Transactions on, 1980 [cit. 2016-03-11]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.527.2157&rep=rep1&type=pdf

[4] BAKER, K.: *Singular value decomposition tutorial* [online]. The Ohio State University, 2005 [cit. 2016-03-11]. Dostupné z: http://lsa-svd-application-for-analysis.googlecode.com/svn-history/r120/trunk/LSA/Other/LsaToRead/SVDTut.pdf

[5] JULIÀ, C., A. D. SAPPA, F. LUMBRERAS, J. SERRAT a A. LÓPEZ: Predicting missing ratings in recommender systems: Adapted factorization approach. *International Journal of Electronic Commerce* [online]. 2009 [cit. 2016-03-11]. Dostupné z: http://refbase.cvc.uab.es/files/JSL2009b.pdf

[6] CLINE, A. K. a I. S. DHILLON: Computation of the singular value decomposition. In: HOGBEN, L., ed. *Handbook of linear algebra.*

CRC Press, 2006, Dostupné také z: http://www.cs.utexas.edu/users/inderjit/public_papers/HLA_SVD.pdf.

[7] BERRY, M. W., S. T. DUMAIS a G. W. O'BRIEN: *Using linear algebra for intelligent information retrieval* [online]. SIAM review, 1995 [cit. 2016-03-18]. Dostupné z: http://languagelog.ldc.upenn.edu/myl/ldc/LSA_SIAM.pdf

[8] FUNK, S.: *Netflix update: try this at home* [online]. 2006 [cit. 2016-01-16]. Dostupné z: http://sifter.org/~simon/journal/20061211.html

[9] FUNK, S.: *Netflix SVD derivation* [online]. 2007 [cit. 2016-03-18]. Dostupné z: http://sifter.org/~simon/journal/20070815.html

[10] KOREN, Y.: Collaborative filtering with temporal dynamics. *Communications of the ACM* [online]. 2010 [cit. 2016-03-18]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.379.1951&rep=rep1&type=pdf

[11] HU, Y., K. YEHUDA a Ch. VOLINSKY: Collaborative Filtering for Implicit Feedback Datasets. In: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference* [online]. IEEE, 2008 [cit. 2016-04-04]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.167.5120&rep=rep1&type=pdf

[12] SAID, A. a A. BELLOGÍN: Comparative recommender system evaluation: benchmarking recommendation frameworks. In: *Proceedings of the 8th ACM Conference on Recommender systems* [online]. ACM, 2014 [cit. 2016-03-25]. Dostupné z: http://ir.ii.uam.es/~alejandro/2014/recsys-benchmarking.pdf

[13] LensKit Contributors: *LensKit 2.2.1* [software]. [přístup 25.ledna 2016]., Dostupné z: http://lenskit.org/.

[14] The Apache Software Foundation: *Apache Mahout 0.11.1* [software]. [přístup 25.ledna 2016]., Dostupné z: http://mahout.apache.org/.

[15] The Apache Software Foundation: *Apache License, Version 2.0* [online]. Leden 2004 [cit. 2016-04-12]. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0

[16] GANTNER, Z., S. RENDLE, L. DRUMOND a Ch. FREUDENTHALER: *MyMediaLite Recommender System Library 3.11* [software]. [přístup 26.ledna 2016]., Dostupné z: http://www.mymedialite.net/index.html.

[17] JUAN, Y.-C., W.-S. CHIN, Y. ZHUANG, B.-W. YUAN, M.-Y. YANG a Ch.-J. LIN: *LIBMF: A Matrix-factorization Library for Recommender Systems 2.00* [software]. [přístup 11.prosince 2015]., Dostupné z: `https://www.csie.ntu.edu.tw/~cjlin/libmf/`.

[18] JUAN, Y.-C., W.-S. CHIN, Y. ZHUANG, B.-W. YUAN, M.-Y. YANG a Ch.-J. LIN: *The LIBMF Project Copyright* [online]. 2014-2015 [cit. 2015-12-14]. Dostupné z: `https://www.csie.ntu.edu.tw/~cjlin/libmf/COPYRIGHT`

[19] HARPER, F. M. a J. A. KONSTAN: The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)* [online]. 2015 [cit. 2016-04-05]. Dostupné z: `http://files.grouplens.org/papers/harper-tiis2015.pdf`

[20] UNIVERSITY OF MINNESOTA. GroupLens Research.: *MovieLens 10M Dataset* [online]. 2009 [cit. 2016-01-12]. Dostupné z: `http://grouplens.org/datasets/movielens/10m/`

[21] STECK, H.: Training and Testing of Recommender Systems on Data Missing Not at Random In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* [online]. ACM, 2010 [cit. 2016-04-26]. Dostupné z: `http://users.cs.fiu.edu/~lzhen001/activities/KDD_USB_key_2010/docs/p713.pdf`

[22] CHIN, W.-S., Y. ZHUANG, Y.-Ch. JUAN a Ch.-J. LIN: A Learning-rate Schedule for Stochastic Gradient Methods to Matrix Factorization. In: *Advances in Knowledge Discovery and Data Mining* [online]. Springer International Publishing, 2015 [cit. 2016-04-18]. Dostupné z: `https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/mf_adaptive_pakdd.pdf`

Příloha **A**

# Seznam použitých zkratek

**ALS** Alternating Least Squares

**CV** Cross Validation

**k-NN** $k$-Nearest Neighbours

**GNU GPL** GNU General Public License

**RMSE** Root Mean Square Error

**SGD** Stochastic Gradient Descent

**SVD** Singular Value Decomposition

Příloha **B**

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... zdrojové soubory práce
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `img` ..... grafy s výsledky experimentů ve formátu PDF
- `text` ..... text práce
  - `DP_Richtr_Tomáš_2016.pdf` ..... text práce ve formátu PDF