České vysoké učení technické v Praze
Fakulta elektrotechnická

Implementace algoritmu doporučování multimediálního obsahu

Bakalářská práce

Studijní program: Softwarové technologie a management
Studijní obor: Web a multimédia

Vedoucí práce: Ing. Miroslav Čepek, Ph.D.

Jitka Kubelová

Praha 2014

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze .....

Datum Jitka Kubelová

Poděkování

Chtěla bych poděkovat vedoucímu mé práce, Ing. Mirku Čepkovi, Ph.D., za jeho trpělivost, ochotu a pomoc při psaní této práce. Dále bych chtěla poděkovat fakultě za poskytnutí licencí, bez nichž by bylo zpracování zejména praktické části o dost složitější.
A samozřejmě velký dík patří mé rodině a přátelům za podporu a trpělivost.

Mnohokrát děkuji všem

Abstrakt

V této práci se zabývám metodami doporučování obsahu a modifikací jednoho z algoritmů používaných v této oblasti. V teoretické části shrnuji základní informace o systémech doporučování obsahu a jejich metodách, zvláště se pak věnuji kolaborativnímu filtrování a popisu jeho tří nejčastějších algoritmů: *k-Nearest Neighbours*, *Association rules base prediction* a *Matrix factorization*. Část také věnuji popisu dat ze soutěže *Netflix prize*, na kterých je algoritmus v praktické části implementován. V praktické části se věnuji implementaci algoritmu *k-Nearest Neihbours* v software *Wolfram Mathematica*, jeho modifikaci se zaměřením na rychlost algoritmu a přesnost doporučení a následném otestování.

Klíčová slova

Doporučování obsahu * Kolaborativní filtrování * k-NN * Doporučení na základě asociačních pravidel * Faktorizace matic * Modifikace algoritmů pro doporučování obsahu

Abstract

Aim of my bachelor thesis are recommender systems and the modification of one algorithms. In the theoretical part I summarize basic information about recommender systems, especially I concentrate on collaborative filtering and describe the three most usual algorithms: k-NN, *Association rules base prediction* and *Matrix factorization*. In next section I describe *Netflix prize* data, which are used in practical part of my thesis for testing purposes. The practical part I implement modification of *k-Nearest Neihbours* algorithm in *Wolfram Mathematica* with focus on algorithm's speed and recommendation accuracy. Last part I devote to testing of my implementation.

Key words

Recommender systems * Collaborative filtering * k-Nearest Neighbor * Association rules base prediction * Matrix factorization * Recommender systems algorithms modification

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Vytěžování dat (*Data Mining*) je technologie, jež je hlavně v posledních letech velice oblíbená. Jednou z větví, která do tohoto pomyslného stromu patří, jsou tzv. Systémy doporučování obsahu (*Recommender Systems*). Stručně řečeno by se dalo říci, že *Recommender Systems* doporučí uživateli další položku, která odpovídá jeho „vkusu“, tedy nabídne mu např. film či zboží, které by pro něj mohlo být zajímavé. Příklad použití *Recommender Systems* je např. doporučení nového filmu uživateli na základě filmů, které už viděl (*Netflix*), knížek, které by si mohl přečíst (*Amazon*), nebo elektronického zboží, které by si mohl zakoupit (*Alza*)[1].

Jednou z nejstarších metod z oblasti *Recommender Systems* je Analýza nákupního košíku (*Market Basket Analysis*) nebo též Afinní analýza (*Affinity Analysis*), postupem času byly vyvinuty i další metody, např. Doporučení na základě obsahu (*Content Based Recommendation*) nebo Kolaborativní filtrování (*Collaborative Filtering*). I když se bude většina této bakalářské práce týkat popisu a implementaci algoritmu na datech ze soutěže *Netflix prize*, *Recommender Systems* mají široké užití v různých odvětvích našeho každodenního života, ať už se jedná o doporučování různých položek, např. v e-shopech, sociálních sítích a různých dalších částech virtuálního světa, ale i např. v lékařství (v napomáhání rozeznávání nemoci podle jejích příznaků) apod.

Za cíl této práce si kladu seznámení se s problematikou systémů doporučování obsahu a metodami, které jsou zde používány. Součástí bude i popis algoritmů z této oblasti a implementace algoritmu Doporučení na základě nejbližších sousedů (*k-Nearest Neighbours*) ve *Wolfram Mathematica*.

V teoretické části vysvětlím problematiku *Recommender Systems* (kap. 2.1) a zaměřím se na dvě základní oblasti: Kolaborativní filtrování (kap. 3.1) a Doporučení na základě obsahu (kap. 4.1). V kapitole popisující Kolaborativní filtrování popíšu tři nejznámější algoritmy z této oblasti: *k-Nearest Neighbours* (kap. 3.2.1), Doporučení na základě aso-

[1] V marketingové oblasti se této úloze říká *Cross-selling*.

ciačních pravidel (známé též jako *Apriori algoritmus*, kap. 3.2.2) a Faktorizace matic (*Matrix Factorization*, kap. 3.2.3).

Protože bude praktická část implementována na datech ze soutěže *Netflix prize*, věnuji této problematice kapitolu (kap. 5.1), kde bude popsána soutěž, její výsledky a stručně i vítězné algoritmy.

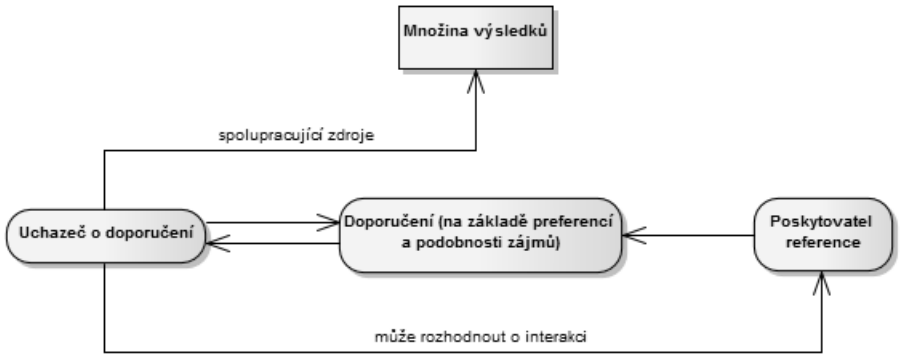
Cílem praktické části je modifikace a implementace algoritmu *k-Nearest Neighbours* (kap. 7). Popíšu zde data ze soutěže *Netflix prize* a jejich případné úpravy pro implementaci. U algoritmu k-NN uvedu seznam již existujících implementací. Součástí popisu implementace bude nastínění problémů, které bude třeba řešit, a vysvětlení modifikace algoritmu. Součástí praktické části bude také otestování algoritmu z hlediska rychlosti a přesnosti doporučení (kap. 7.4.2). Přesnost doporučení bude porovnána s přesností nejlepších řešitelů soutěže *Netflix Prize* pomocí výpočtu RMSE (kap. 5.1.1).

Kapitola 2

Metody doporučování obsahu

2.1 Systémy doporučování obsahu

Systémy doporučování obsahu, neboli *Recommender Systems* (dále jen RS), jsou často rozšířené hlavně na internetu a fungují na principu doporučování položek. Cílem RS je generovat smysluplná doporučení uživateli z množiny položek, produktů, . . . , o které by mohl mít zájem. Návrhy zajímavých knih na `Amazon.com`, filmy na `Netflix.com` apod., to jsou jedny z dnes fungujících nasazení RS v praxi. [1]



Obr. 2.1: *Model doporučovacího procesu ukazuje, jak probíhá doporučení. Vstupem jsou preference uživatele, výstupem množina výsledků, které se mu doporučí* [2].

Využití RS Jednou z nejznámějších a nejrozšířenějších funkcí RS je, že pomáhají uživateli najít a nabídnout informace, které vyhovují jeho „vkusu“. Uvedu malý příklad: uživatel systému podá osobní informace, např. o tom, které filmy už viděl a jejich ohodnocení. RS mu poté na základě informací získaných od dalších uživatelů nabídne další filmy, které uživatel ještě neviděl, ale odpovídají jeho žánru. [3]

2.1.1 Definice problému doporučování obsahu

Mějme množinu uživatelů U a množinu doporučovaných položek S. V této práci si pod množinou U můžeme představit seznam diváků, kteří sledují a hodnotí filmy, což jsou položky množiny S. Každý uživatel $u \in U$ je definován svým **uživatelským profilem**, který obsahuje informace o něm (např. *věk, příjem, rodinný stav, vkus, preference* apod.), v nejjednodušším případě obsahuje pouze `user_ID`. Podobně je definována i množina S, kde je každá položka reprezentována minimálně svým `film_ID` a případně i dalšími položkami (*název, žánr, režisér, rok natočení* apod.).

V aplikacích je velice časté, že jsou tyto množiny, uživatelů, filmů, ..., velice rozsáhlé. Stejně tomu bude i v této práci, k popisu dat se dostaneme později v kapitole. 5.1.1

		Položky					
		1	**2**	**...**	**i**	**...**	**m**
Uživatelé	**1**	5	3		1	2	
	2		2			4	
	...			5			
	j	3	4		2	1	
	...					4	
	n			3	2		
				...			

Tab. 2.1: *Příklad ohodnocení položek od uživatelů v rámci* Recommender Systems *(prázdná políčka značí, že uživatel položku nehodnotil)* [1].

Definice RS

Nechť p je funkce, která měří účinnost či prospěšnost položek množiny S pro uživatele u. Vzniká tedy množina $p : U \times S \rightarrow R$, kde R je úplně uspořádaná množina (nezáporná celá nebo reálná čísla v konečném rozsahu). [3]

Předpověď Z hlediska předpovědi RS obvykle provádí následující dvě úlohy [3]:

1. ***Rating prediction***, tedy předpověď ohodnocení, které uživatel u položce s přiřadí (užitečnost položky s pro uživatele u).

2. ***Item prediction***, což je předpověď seznamu položek, které mohou být pro uživatele u zajímavé. Interakce mezi uživateli a položkami jsou obvykle binární (např. koupil/nekoupil), mohou být i více-položkové za pomoci uživatelova hodnocení (*rating*). Následně je pak vyjádřena pravděpodobnost, s jakou uživatel položku např. koupí či nikoli.

2.1.2 Metody v RS

Existuje několik základních přístupů v RS. Každý se liší od ostatních v tom, jaká data potřebuje, jak s nimi následně zachází a jak může být v praxi využitelný. Zde je uveden seznam čtyř nejčastějších:

Content-based recommendations

Systém použije „historii“ uživatele u a na základě těchto záznamů pak doporučí položky, jež se podobají nebo nějak souvisí s těmi, o které uživatel už zájem projevil. Zaměřuje se hlavně na algoritmy pro vyhodnocení uživatelských preferencí a filtrování položek, které preferencím uživatele odpovídají (v kapitole 4.1 bude rozvedeno podrobněji).

Collaborative filtering

Systém vyžaduje vyjádření preferencí jiných uživatelů, na jejichž základě spolu se sloučením informací od uživatele u pak vyhodnotí nové hodnocení. V podstatě jde o to, že informace je získávána z porovnání s ostatními uživateli, na rozdíl od *Content-based recommendations*. Tyto systémy se zaměřují na algoritmy pro porovnávání uživatelů na základě jejich preferencí a vážení zájmů a na základě těchto informací pak vytváří doporučení. Právě této metodě se budeme věnovat v kapitole 3.1 a v praktické části.

Social data mining

Myšlenka *Social data mining* je založena na pozorování závislostí objektů na čase (např. v prostřední internetu hlavně www stránky a odkazy) a jejich opotřebení (zde je myšleno hlavně „časové opotřebení“, téměř každá informace časem ztrácí na ceně, protože se stává neaktuální). Sledují se zde také různé ukazatele, jak je např. čas strávený nad určitou částí stránky (třeba zda uživatel sleduje reklamu, která je vložena před video, nebo si raději přečte text pod videem než reklama skončí, a teprve poté obrátí pozornost zpět k videu), zda čte celé zprávy nebo jen jejich nadpisy apod.

Internet je oblast s bohatým obsahem, který je na sebe určitým způsobem závislý. Např. vyhledávač `Google.com` je založen právě na navázání odkazů mezi webovými stránkami, podobností obsahu a různými dalšími informacemi o stránkách. A asi není třeba zmiňovat úlohu *Social data mining* v sociálních sítích jako je `Facebook` či `Twitter`, kde systém těží z implicitních a výpočetních záznamů různých společenských aktivit (např. historie použití systému, citace, hypertextové odkazy apod.). [2]

Recommendation support

Recommendation support neboli Podpůrné systémy pro doporučení obsahu jsou výpočetní nástroje pro sdílení doporučení mezi lidmi. Toto se může dít za pomoci vyjádření, že se uživateli něco „líbí“ či „nelíbí“, nebo pomocí přidružení zpráv k dokumentům apod. Tyto zprávy jsou pak uloženy v databázi, odkud mohou být vybírány na základě jejich obsahu nebo spojitosti s autorem apod. Je možné uvést několik příkladů, kde je takových zpráv využíváno (např. webblogy, kde je možné články okomentovat). Např. webová stránka `about.com` obsahuje stovky tématických oblastí udržovaných odborníky. Obsah každé oblasti je doplněn o mnoho odkazů, novinek a dalších podobných užitečných funkcí.

Další důležitou zmínkou, která patří k *Recommendation Support*, je, že jejich kvalita záleží na kvalitě tzv. dělby práce. Několik dobře motivovaných lidí je zodpovědných za obsah, který spravují, zatímco ostatní návštěvníci tyto informace využívají při hledání. Tyto systémy fungují, pokud je oněch spravujících lidí dostatek a většinou je jejich motivací záliba v dané téma. [2]

Hybridní metody

Existují také metody, které kombinují některé z přístupů používaných v RS dohromady. Jsou známy jako *Hybridní metody*, obvykle se kombinují dvě až tři. Výhodou takto získané metody je, že umožňuje z kombinace metod získat vyšší výkon a potlačit nevýhody, které mají metody samostatně. V následující tabulce 2.2 je seznam některých hybridních metod:

Metoda	**Popis metody**
Weighted	Výsledek několika doporučovacích technik se spojí dohromady a vytvoří tak jedno společné doporučení.
Swithing	Na základě zadaných kritérií nebo požadavků systém „přepíná“ mezi metodami a vybírá, která je v daný okamžik nejvýhodnější.
Mixed	Vhodná pro simultánní úlohy, kdy je prezentován výsledek z jednotlivých metod současně.
Feature combination	Jsou přidány vlastnosti nebo funkce z jiných datových zdrojů a na ty se pak použije metoda.
Cascade	Metody jsou za sebou seřazeny do kaskády - jedna metoda doporučení „zjemní“ a předá ho další.
Feature augmentation	Výstup z jedné techniky se použije jako vstup do další.
Meta-level	Model naučený jednou metodou je použit jako vstup do jiné.

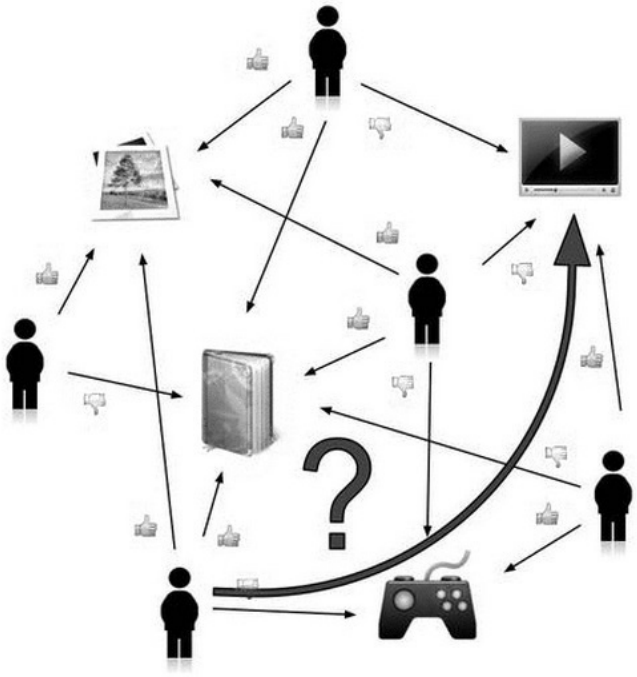
Tab. 2.2: *Přehled hybridních metod a jejich stručný popis* [4].

Kapitola 3

Kolaborativní filtrování

3.1 *Kolaborativní filtrování*

V této bakalářské práci se budu zabývat výhradně metodou Kolaborativního filtrování (*Collaborative filtering*, dále jen CF), pochopení jejího principu a popisu algoritmů.



Obr. 3.1: *Princip odhadování doporučení v reálném světě. Obrázek ukazuje záliby uživatele* u *a dalších uživatelů, kteří mají podobné zájmy. Na nich je následně založeno doporučení pro uživatele* u [5].

CF patří mezi nejstudovanější a nejrozšířenější metody používané v oblasti doporučování obsahu v praxi. Jeho aplikace zahrnuje obvykle velmi rozsáhlé datové sady a jsou také aplikovány na mnoho různých typů dat (např. geologický průzkum, finanční údaje, elektronické obchody, různé webové aplikace, ...), tedy hlavně na místech, kde je kladen důraz na práci s uživatelskými daty. [2]

Společné rysy metod CF

Ač se přístupy používané v CF navzájem liší, několik základních rysů nají společných [6]:

- **Doporučení položek** v konečném výsledku doporučí uživateli položky, které by pro něj mohly být nějakým způsobem zajímavé (i když ne vždy se zobrazí doporučení „šité na míru“, např. `Amazon.com` předpovídá položky na základě průměru).
- **Předpověď pro konkrétní položku** zobrazí k vybrané položce předpovězené ohodnocení (např. známku filmu nebo knihy), což může být náročnější než doporučení samotné položky.
- **Doporučení na základě určité množiny položek** znamená, že uživatel zadá určité parametry, které by měl výsledek doporučení splňovat (např. při návštěvě filmu: lokalita, věk diváka, délka film apod.).

3.1.1 Charakteristika CF

Klíčovou charakteristikou CF je, že předpovídá užitečnost položek pro konkrétní uživatele na základě položek dříve ohodnocených nebo zakoupených jinými podobně smýšlejícími uživateli. Často je to založeno na znalosti interakci dat *uživatel-položka*, kdy se ignorují atributy obou množin a soustředí se jen na vztahy jednotlivých položek (viz obr. 3.1). [3]

Dále shrneme některé problémy a klíčové vlastnosti CF, které je řeší.

Rozptýlenost dat

V praxi mnoho komerčních RS nabízí velice rozsáhlou množinu položek (v našem případě jednu velkou množinu filmů a jednu ještě větší množinu uživatelů). Použití CF pro doporučení položky tam, kde se nějaký vztah *uživatel-položka* vyskytuje velmi málo, může zpochybnit výsledné doporučení. To nastává třeba v okamžiku, kdy se mezi daty vyskytne nový uživatel/položka bez ohodnocení nebo jich má velmi málo – pro ně pak není možné doporučení vytvořit. Někdy se tento problém nazývá „problém nového uživatele/položky“. Pro řešení tohoto problému se využívá několik technik, např. *SVD* (singulární rozklad, viz kap. 3.2.3). Hybridní algoritmy používají k řešení toho problému externí informace o položce nebo uživateli, pokud jsou k dipozici. [7]

Následující seznam zobrazuje vztahy *uživatel-položka*, které mohou nastat [6]:

- **Velké množství položek**, kde by neměl být problém položku doporučit, záleží poté jen na frekvenci výskytu položek.
- **Málo hodnocení u položky** může způsobit, že nebude dostatek informací pro správnou předpověď nebo doporučení.

- **Více uživatelů než položek** je nejlepší případ, který může nastat, neboť pak máme dostatek informací pro doporučení.

- **Uživatel ohodnotí více položek** a pak je možné mu přesněji doporučit další, neboť předpovězení není založeno jen na jedné položce, nýbrž na více, a předpovězení je pak přesnější.

Rozšiřotatelnost dat

Po určité době se může stát, že počet uživatelů a položek vzroste do takových rozměrů, že tradiční CF algoritmy, např. s lineární složitostí Ø(n), mohou mít s takovým množstvím dat problémy. Někdy se od algoritmů RS vyžaduje téměř okamžitá reakce bez ohledu na velikost těchto množin. Existují techniky (např. již zmiňované SVD), které mohou faktorizací výpočet urychlit, avšak děje se tak za cenu předchozích výpočetních kroků, které mohou být velmi nákladné. [7]

Synonyma

Synonyma poukazují na existenci stejných nebo velmi podobných položek, které mají různé názvy či vstupy. Většina RS je schopna odhalit tento latentní vztah a zacházet s těmito produkty odlišným způsobem. Například zdánlivě různé položky „děti film“ a „děti Film“ jsou z logického hlediska stejné položky, ale některé CF systémy by mezi nimi podobnost nenašly. Některé systémy spoléhají na použití tezauru nebo podobných automatických metod, některé takové shody se však mohou významem skutečně lišit a to pak vede k degradaci výkonu a spolehlivosti doporučení. [7]

Šedá ovce

Tzv. „šedé ovce“ jsou uživatelé, jejichž názory a hodnocení se nějakým způsobem výrazně liší od ostatních uživatelů. Tento problém se většinou řeší výpočtem „střední hodnoty“ uživatelských ohodnocení, z nichž se potom konečné doporučení vyvozuje. Pokud máme k dipspozici dostatečné množství dat, pak je možné tyto extrémní záznamy ignorovat. Musíme však umět rozhodnout, kolik takovýchto „šedých ovcí“ můžeme ignorovat. [7]

Šilinkové útoky

Může nastat situace, kdy lidé dávají mnoho pozitivních hodnocení položce z důvodu vlastního profitování a naopak negativní doporučení konkurenci. Metody CF by měly myslet i na tento problém a zavést bezpečnostní opatření. [7]

3.2 Populární algoritmy CF

Níže budou popsány některé populární algoritmy CF:

- **k-Nearest Neighbor** (kap. 3.2.1)
- **Association rules base prediction** (kap. 3.2.2)
- **Matrix factorization** (kap. 3.2.3)

3.2.1 k-Nearest Neighbor

K-Nearest Neighbor (k-NN) umožňuje vytvářet předpovědi přímo. Algoritmus funguje na principu hledání podobných uživatelů nebo položek. Pokud takové nalezne, tzv. *sousedy* (uživatele s podobným „vkusem“), použije jejich ohodnocení k předpovědi preferencí cílového uživatele. Výsledná předpověď je založena na seskupení vhodných položek, které mají tito sousedé společné. [3]

Algoritmus k-NN

Typický algoritmus k-NN se kládá ze dvou základních kroků: fáze **formování sousedů** a fáze **doporučování**. V první fázi se porovnají záznamy aktivit cílového uživatele (též zvaného *visitor*) s minulými záznamy T ostatních uživatelů se snahou najít oněch k nejvíce vyhovujících sousedů s podobnými preferencemi. Mapování visitora na sousedy může být založeno na podobnosti v hodnocení položek, zájmu o podobný obsah apod. [3]

User-based k-NN

Nechť záznam (nebo také profil) cílového uživatele je u (reprezentovaný jako vektor) a záznam jiného uživatele je v ($v \in T$). Nejpodobnějších k záznamů z T je pak považováno za sousedy u. Tato podobnost může být také vyjádřena tzv. *Pearsonových korelačním koeficientem:*

$$sim(u,v) = \frac{\sum_{i \in C}(r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in C}(r_{u,i} - \overline{r}_u)^2}\sqrt{\sum_{i \in C}(r_{v,i} - \overline{r}_v)^2}}\,, \tag{1}$$

kde C je množina položek, které byly ohodnoceny oběma uživateli u a v, $r_{u,i}$ a $r_{v,i}$ jsou ohodnocení položky i uživateli u a v, a konečně $\overline{r}_u$ a $\overline{r}_v$ je průměrné hodnocení uživatelů u a v. Tím je vypočítána největší podobnost a vybráni sousedé, tedy nejpodobnější uživatelé. Není nezvyklé odfiltrovat uživatele s podobností menší než určitá hodnota, abychom nedostali příliš velkou vzdálenost od u nebo dokonce negativní korelaci. [3]

Ohodnocení uživatelů Jakmile jsou zjištěni sousedé, pro doporučovací fázi je možné použít následující vzorec pro výpočet předpovědi hodnocení položky i pro uživatele u:

$$p(u,i) = \overline{r}_u + \frac{\sum_{v \in V} sim(u,v) \times (r_{v,i} - \overline{r}_v)}{\sum_{v \in V} |sim(u,v)|} , \tag{2}$$

kde V je množina podobných uživatelů, $r_{v,i}$ je hodnocení položky i uživatelem u, $\overline{r}_u$ je průměrné hodnocení uživatele u a konečně sim(u,v) je *Pearsonova korelace* popsaná výše. Tato formule vypočítá preference všech sousedů vážené jejich podobností a výsledek je pak přidán do průměrného ohodnocení uživatele u. [3]

Item-based k-NN

Problém s *user-based* k-NN je nedostatek škálovatelnosti; vyžaduje v reálném čase srovnání cílového uživatele se všemi ostatními uživatelskými záznamy. Existuje však varianta této metody, která tento problém řeší: *Item-based* k-NN (neboli k-NN na základě položky). Díky ní můžeme předem vypočítat všechny možné páry položek na základě jejich podobnosti hodnocení přes všechny uživatele. Výsledkem je k podobných položek, které jsou spolu-hodnocené na základě podobnosti různých uživatelů. K získání míry podobnosti se obvykle používá *rozšířená kosinová podobnost*[1]:

$$sim(i,j) = \frac{\sum_{u \in U} (r_{u,i} - \overline{r}_u)(r_{u,j} - \overline{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,i} - \overline{r}_u)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \overline{r}_u)^2}} , \tag{3}$$

kde U je množina uživatelů, $r_{u,i}$ jsou hodnocení uživatele $u \in U$ položky i a konečně $\overline{r}_u$ je průměrné hodnocení uživatele. [8]

Vytvoření množiny položek Důležité je si uvědomit, že výpočet probíhá na základě párových podobnosti mezi předměty (nikoli uživateli) na základě hodnocení těchto položek pro všechny uživatele. Pro výpočet podobnosti mezi předměty vytvoříme množinu položek, které budeme k předpovídání hodnoty používat:

$$p(u,i) = \frac{\sum_{j \in J} r_{u,j} \times sim(i,j)}{\sum_{j \in J} sim(i,j)} , \tag{4}$$

kde J je množina sobě-podobných položek, $r_{u,i}$ jsou hodnocení uživatele u položky j a konečně *sim(i,j)* je podobnost mezi položkami i a j.

Opět se ignorují položky s negativní podobností s cílovou položkou. Idea této modifikace tkví v tom, že je použito vlastní hodnocení uživatele pro hledání podobných položek k odhadnutí předpovědi porovnávané položky. [3]

[1] Kosinová podobnost (cosine similarity) je míra podobnosti dvou vektorů, která se získá výpočtem kosinu úhlu těchto vektorů.

Problémy algoritmu Problém přímočaré metody k-NN tkví v tom, že velký počet dat (uživatelů nebo položek) dělá výpočet podobného uživatele nebo položky prakticky nemožným. Při porovnávání uživatelů/položek způsobem „každý s každým“ se totiž složitost algoritmu blíží k $\emptyset(n^2)$, což je pro velká čísla obrovské množství kombinací.

Existují možnosti, jak tento problém vyřešit, např. zmenšit rozsah dat apod. Tyto metody buď promítají matice uživatel-položka do méně-dimenzionálnho prostoru pomocí techniky, jako je např. Analýza hlavních komponent nebo Faktorizace matic, ve snaze získat reprezentaci s méně hodnotami, a poté identifikovat podobné uživatele (položky) v podprostoru, viz dále kapitola 3.2.3. [3]

3.2.2 *Association rules base prediction*

Market basket analysis Známá v oblasti *Association rules base prediction* (předpovězení na základě asociačních pravidel, dále ARBP) je *Affinní analýza*, nebo také *Market basket analysis* (teorie nákupního košíku), což je známý přístup pro doporučování položek. Na základě toho, co kupující v košíku už má, mu předurčíme, jakou věc by tam ještě mohl nebo měl dát [9], např.:

$$\{\textit{těstoviny, červená paprika, rajče, zakysaná smetana}\} \rightarrow \{\textit{čínské zelí}\}$$

Popis algoritmu S daty je přirozené pracovat jako s transakcemi, ze kterých mohou být asociační pravidla těžena a použita pro predikci a klasifikaci. V podstatě to znamená, že hledáme množiny položek, které se často vyskytují pospolu, tedy **jsou asociovány**.

Jde o to „vytěžit“ z transakcí asociační pravidla ve tvaru *antecedent* $\rightarrow$ *succedent*. Pokud platí *antecedent* (tedy množina položek na pravé straně) s vysokou mírou pravděpodobnosti, pak platí i *succedent* (typicky jedna položka), který se vyskytuje také. [3]

Apriori Nejznámějším algoritmem pro získávání asociačních pravidel je *Apriori* algoritmus. K jeho vysvětlení je nutné znát několik termínů, které se v oblasti ARBP používají (příklady jsou založeny na tab. 3.1 s množinou transakcí T) [9], [10]:

- ***Itemset*, množina položek (i)**: n-tice jedné či více položek, např.:

$$\{Avatar, Batman\}$$

- ***Association rule*, asociační pravidlo (r)**: pravidlo ve formě $r : Y \rightarrow X$, kde Y i X jsou množiny položek, např.:

$$\{Avatar, Batman\} \rightarrow \{Superman\}$$

- **Podpora (σ)**: počet transakcí, které obsahují stejnou množinu, např.:

$$\sigma(\{Batman, Spiderman\}) = 3$$

***Support*, procentuální podpora (s)**: procentuální vyjádření počtu transakcí obsahující množinu položek, např.:

$$s\{(Batman, Spiderman) = \frac{\sigma(Batman, Spiderman)}{|T|}\} = 3/5 = 0,6$$

- **Často se vyskytující množina položek**: frekventovaná množina položek, jejíž podpora je vyšší nebo rovna tzv. minimální podpoře (min. podpora je určena předem, slouží k vyloučení méně potřebných pravidel).

- ***Confidence*, spolehlivost (c)**: podmíněná pravděpodobnost množin X a Y, tedy pravděpodobnost, s jakou bude Y obsažena v X. Spolehlivost vyjadřuje poměr počtu transakcí, které obsahují X i Y, a počtu transakcí, které obsahují jen X, např.:

$$c = \frac{\sigma(Batman, Spiderman, Superman)}{\sigma(Batman, Spiderman)} = 2/3 = 0.67$$

`user_ID`	**Množina filmů, které `user_ID` viděl**
1	Avatar, Batman, Forrest Gump, Superman
2	Batman, Gladiátor, Spiderman, Superman
3	Avatar, Batman, Spiderman, Superman
4	Avatar, Spiderman, Superman, Gattaca
5	Batman, Gladiátor, Spiderman, Gattaca

Tab. 3.1: *Tabulka transakcí pro algoritmus* Apriori. *Každou transakci si můžeme představit jako jednoho uživatele a filmy, které viděl.*

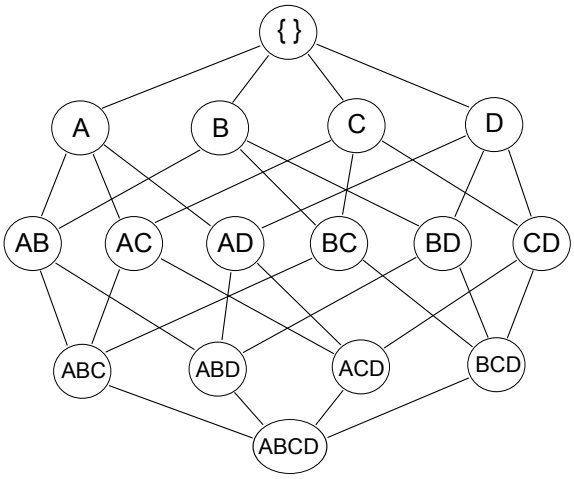
Apriori algoritmus

Následuje příklad algoritmu *Apriori*, jeho popis byl převzat z [10], doplněn z [11] a upraven. Nejprve vygenerujeme množiny častých položek. Začínáme od prázdné množiny $\emptyset$, která je častá vždy. Následně se posouváme grafem množin položek „dolů“ (viz obr. 3.2), dokud se celá množina objevuje mezi transakcemi.

Zde je důležité si uvědomit **princip monocity**: pokud není množina položek častá, není časté ani její rozšíření (např. pokud *antecedent* $\{Batman, Gattaca\}$ nemá vysokou pravděpodobnost výskytu, pak ho nemá ani *succedent* $\{Batman, Gattaca, Gladiator\}$).

Častých množin však může být velmi mnoho. Můžeme využít **redukce množiny**, díky které je možné zjednodušit množinu častých položek na maximální možnou, tedy vybrat takové, které nejsou podmnožinou žádné jiné časté množiny.
Např. pokud vybereme $L = \{AB, AC, AD, BC, ABC\}$, zredukuje se na $L = \{ABC, AD\}$.
Přesně toto vyjadřuje **princip anti-monocity**: podpora žádné z množin nepřesáhne podporu své nadmnožiny, nebo-li:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(Y) \geq s(Y) \ . \tag{5}$$



Obr. 3.2: *Apriori algoritmus - generování častých množin položek. Graf častých množin obsahuje exponenciální počet uzlů (2^n,), tedy pro $n = 4$ je to $n = 4$ $2^4 = 16$ možných kandidátů.*

Při určování asociačních pravidel platí, že hledáme v množině častých položek L všechny neprázdné množiny $f \subset L$. Ty pak splňují požadavek minimální spolehlivosti $f \to L$. Např. pokud máme množinu častých položek $L = \{ABC\}$, pak kandidáti na asociační pravidla jsou následující:

$$R_p = \{\{AB \to C\}, \{AC \to B\}, \{BC \to A\}, \{A \to BC\}, \{B \to AC\}, \{C \to AB\}\} \ . \quad (6)$$

Počet těchto kandidátů je roven $2^{|L|} - 2$ (ignoruje se $L \to \emptyset$ a $\emptyset \to L$), zde je to $2^3 - 2 = 6$ potenciálních asociačních pravidel.

Jak je z výčtu vidět, množina pravidel by se dala zredukovat. Spolehlivost pravidel generovaných na stejné množině častých položek má anti-monotónní vlastnost, což znamená, že spolehlivost pravidla, které má na pravé straně nejvíc položek, je největší. Např. u $L = \{ABC\}$ platí, že:

$$c(AB \to C) \geq c(A \to BC) \ . \quad (7)$$

Algoritmus

Příklad algoritmu *Apriori* zde bude aplikován a vysvětlen na transakcích z tab. 3.1.

A) **Množina častých položek**

Na začátek musíme zvolit minimální podporu, jakou musí množiny častých položek mít, zde zvolíme **minimální podporu 3/7**.

1) Budeme procházet jednotlivé n-tice a vybereme **množiny častých položek**, které vyhovují minimální podpoře 3/7. Zde vyhovují {*Avatar, Batman, Spiderman, Superman*} - tím jsme prošli uzly grafu na obr. 3.2 v hloubce 1. Množiny, které nedosáhly požadované podpory, dále ignorujeme.

Položka	počet výskytů	Položka	počet výskytů
Avatar	3	Spiderman	4
Batman	4	Superman	4
~~Gladiator~~	2	~~Gattaca~~	2
~~Forrest Gump~~	1		

Tab. 3.2: *Příklad častých množin položek algoritmu Apriori v hloubce 1, množiny tedy tvoří filmy samotné.*

2) V dalším kroku algoritmus hodnotí množiny v hloubce 2, tedy dvojice. Pokud jsme již některé množiny vyškrtli, nepracujeme dále ani s jejich nadmnožinami.

Položka	počet výskytů	Položka	počet výskytů
~~{Avatar, Batman}~~	2	{Batman, Spiderman}	3
~~{Avatar, Spiderman}~~	2	{Batman, Superman}	3
~~{Avatar, Superman}~~	2	~~{Spiderman, Superman}~~	2

Tab. 3.3: *Příklad častých množin položek algoritmu Apriori v hloubce 2, množiny tvoří dvojice filmů, které v hloubce jedna vyhovovaly zadané minimální podpoře 3/5.*

3) V hloubce č. 3 už nám zbývá poslední trojice. Tato trojice ovšem zadané minimální podpoře nevyhovuje, není tedy třeba pokračovat v algoritmu dále.

Položka	počet výskytů
~~{Batman, Spiderman, Superman}~~	2

Tab. 3.4: *Příklad častých množin položek algoritmu Apriori v hloubce 3. Trojice však zadané minimální podpoře nevyhovuje, algoritmus zde končí.*

B) **Nalezení asociačních pravidel**

1) Z našeho původního příkladu jsme tak získali následující **množiny častých položek**:
$L = \{\{Avatar\}, \{Batman\}, \{Spiderman\}, \{Superman\}, \{Batman, Spiderman\}, \{Batman, Superman\}\}$.

2) Množinu L **zredukujeme** o množiny obsažené v jiných množinách:
$L = \{\{Avatar\}, \{Batman, Spiderman\}, \{Batman, Superman\}\}$.

3) Nakonec máme možnost vytvořit **asociační pravidla**, u kterých spočítáme i jejich spolehlivost:

- $r_1 : \{\emptyset \rightarrow Avatar\}$, $c(r_1) = 3/5 = 0.6$
- $r_2 : \{Batman \rightarrow Spiderman\}$, $c(r_2) = 3/4 = 0.75$

- $r_3 : \{Batman \rightarrow Superman\}$, $c(r_3) = 3/4 = 0.75$
- $r_4 : \{Spiderman \rightarrow Batman\}$, $c(r_4) = 3/4 = 0.75$
- $r_5 : \{Superman \rightarrow Batman\}$, $c(r_5) = 3/4 = 0.75$

4) Podle nalezených asociačních pravidel nyní můžeme uživateli $u = \{Avatar, Batman, Gladiator, ForrestGump\}$ **doporučit množinu filmů:** $\{Spiderman, Superman\}$.

Problémy algoritmů v ARPB Jedním z problémů pro určení asociačních pravidel CF je, že systém má potíže dát doporučení v situaci, kdy je datový soubor je příliš rozptýlený, což je často případ CF (množina položek uživatele je příliš jiná vzhledem k množinách ostatních uživatelů). Důvod rozptýlenosti je, že cílový uživatel sám má vzhledem ke všem záznamům v datovém souboru příliš málo záznamů a proto je často obtížné najít dostatečný počet společných položek ve více uživatelských profilech.

Pro řešení tohoto problému mohou být použity některé standardní techniky snižující dimenzionalitu apod. Některé z neužitečných a nezajímavých položek je možno odstranit a proto se neobjeví ve finálním vzoru. Také zde existuje několik způsobů, jak tento problém řešit [3]:

- Prvním řešením je ohodnotit všechna nalezená pravidla založená na stupni shody mezi levou stranou každého pravidla a cílového uživatele a na základě této shody pak generovat nejvhodnějších k doporučení. Tento přístup tak zmírní omezení plynoucí z toho, že bychom museli získat kompletní obraz o tom, jak levá strana pravidla vypadá.
- Druhé řešení je využití CF, tj. že systém najde "blízké sousedy", kteří mají podobný zájem jako cílový uživatel a předkládá doporučení na základě analýzy těchto sousedů.

3.2.3 Matrix factorization

Myšlenka faktorizace matic (*Matrix Factorization*, dále jen MF) je mnohem mladší než předchozí dvě metody. Dalo by se říci, že hlavní myšlenka MF je rozložit matici M do několika matic $M = F_1 F_2 \dots F_n$, kde n může být jakékoli číslo, obvykle se používá 2 nebo 3. V CF získala MF popularitu v posledních letech díky vynikajícímu výkonu a to hlavně z hlediska kvality doporučení a škálovatelnosti. Nabízejí také lepší flexibilitu pro modelování různých situací z reálného života.

MF se poprvé vyskytla v soutěži *Netflix prize*, byla zde propagována jako SVD[2] (soutěž sama přispěla velkým dílem k rozvoji doporučovacích algoritmů - více viz kap. 5.1). Následný popis pochází z článku *Matrix Factorization Techniques for Recommender Systems*, jehož autorem je právě vítězný tým *BellKor* [12].

[2]SVD - *Singular value decomposition*: v lineární algebře je singulární rozklad (SVD) rozklad na reálné nebo komplexní matice s řadou užitečných aplikací ve zpracování signálu a statistice.

Základní pojmy MF patří do skupiny latentních[3] modelů. V těchto modelech se pracuje s tzv. *latentními proměnnými* (také nazývanými vlastnosti, aspekty nebo faktory), na základě nichž se pak doporučí obsah uživateli. Modely latentních faktorů ukazují alternativní přístup, jak lze charakterizovat hodnocení uživatelů a položek na např. 20-100 faktorech, vyvozených z hodnotících vzorů.

Když jsou odhadnuty během fáze učení spojení mezi latentními a pozorovanými proměnnými (uživatel, produkt, hodnocení apod.), doporučení mohou být odvozena na základě možných interakcí mezi uživatelem a každým produktem mezi latentními proměnnými. [3]

Princip SVD Aplikováno na data z *Netflix Prize*, metoda SVD rozdělí záznamy *uživatel-film-hodnocení* do dvou (či více) menších matic, které zachycují uživatelské a filmové preference z hlediska hodnocení. Tento aspekt se poté použije pro předpovězení hodnocení a následné doporučení. Stručně řečeno, pokud můžeme použít tréninková data pro nalezení oněch latentních proměnných a vztahu *uživatel-položka*, můžeme to vše použít pro předurčení uživatelského hodnocení filmů, které jím ještě hodnoceny nebyly. [3]

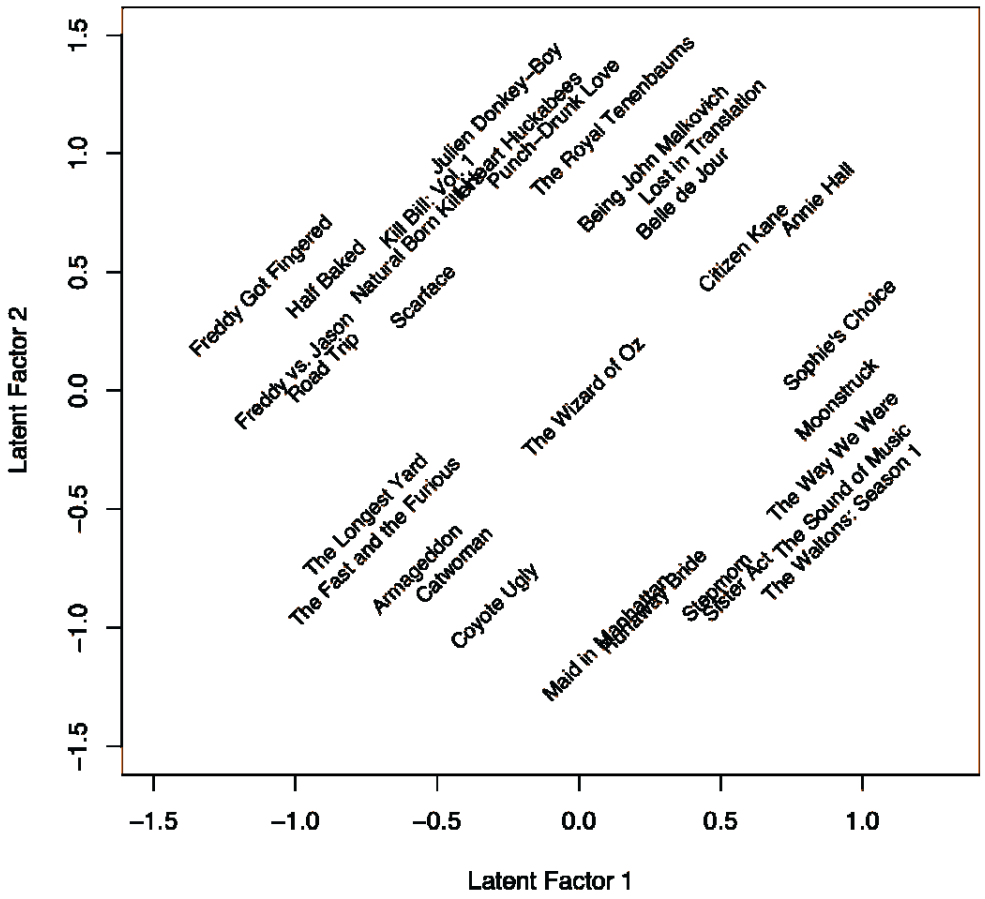
Metody MF

Jak již bylo řečeno, některé z neúspěšnějších realizací latentních faktorů jsou založeny právě na MF. V nejzákladnější formě MF charakterizuje položky i uživatele vektorem faktorů vyplývajících z hodnotících modelů. Vysoká frekvence mezi předmětem a uživatelských faktorem pak vede k vyvození doporučení.

Explicitní zpětná vazba RS spoléhají na různé typy vstupních údajů, které jsou často umístěny v matici s dvěma rozměry, v jednom směru zastupuje uživatele a v druhém předměty zájmu (viz obrázek 3.3). Nejpohodlnější vyjádření těchto dat je tzv. *explicitní zpětnou vazbou*, která obsahuje explicitní uživatelský vstup týkající se jeho zájmů. Např. společnost Netflix (více v kapitole 5.1) sbírá hodnocení hvězdičkami pro filmy, v TiVo uživatelé vyjadřují své preference pro televizní show stisknutím palec nahoru a palec dolů apod. Obvykle tato explicitní zpětná vazba zahrnuje *řídké matice*, protože většina uživatelů pravděpodobně hodnotila jen malé procento možných položek.

Implicitní zpětná vazba Jedna z výhod MF je, že umožňuje začlenění dalších informací. Když explicitní zpětná vazba není k dispozici, lze odhadovat uživatelské preference pomocí *implicitní zpětné vazby*, která nepřímo odráží stanovisko tím, že sleduje chování uživatele včetně např. historie nákupu, historie prohlížení, hledání vzorů, nebo dokonce pohyby myši. Implicitní zpětná vazba obvykle označuje přítomnost nebo nepřítomnost určité události, takže je obvykle zastoupena *plné matici*.

[3] latentní - skrytý



Obr. 3.3: *Faktorizace matic - ukázka rozdělení podle vlastností* [14].

Základní MF model

Modely MF mapují uživateli a položce společný latentní prostor f tak, aby interakce uživatele a položky byla vnitřním produktem tohoto prostoru. Proto je každá položka i asociována s vektorem $q_i \in \mathbb{R}^f$ a každý uživatel u s vektorem $p_u \in \mathbb{R}^f$. Pro dané položky i vektoru q_i chceme změřit, do jaké míry položka těmito faktory disponuje, ať už z hlediska pozitivního nebo negativního. Pro daného uživatele u vektor p_u vyjadřuje, do jaké míry se o danou položku zajímá, opět v pozitivním nebo negativním hledisku. Vznikne tak skalární součin $g_i^T p_u$ vyjadřující interakci mezi uživatelem u a položkou i. Tato aproximace uživatelova hodnocení položky i je pak odhadnuta jako:

$$r_{ui} = q_i^T p_u \tag{8}$$

Odhadnutí hodnocení Hlavním úkolem je zajistit mapování každé položky a uživatele pomocí vektorů $q_i, p_u \in \mathbb{R}$. Po dokončení tohoto mapování lze snadno odhadnout hodnocení, které uživatel u položce i dá pomocí rovnice (8). Takový model je úzce spjat s výše zmíněným singulárním rozkladem (SVD), což je osvědčený způsob pro identifi-

kaci latentních sémantických faktorů při získávání informací. Nutno však podotknout, že při použití na malém počtu záznamů je velice náchylný na přeučení.

Modelování z pozorovaných hodnocení Dřívější systémy spoléhaly na přičtení chybějících hodnocení, čímž by se tak vytvořila plnější matice hodnocení. To však může být bohužel velice nákladné, pokud se výrazně zvyšuje množství dat. Navíc, doplnění nepřesných hodnocení může výrazně data narušit. Z tohoto důvodu se navrhlo modelování přímo z pozorovaných hodnocení, aby nedocházelo k přeučení prostřednictvím uspořádaného modelu. Systém se tak snaží při učení se vektorů p_u a q_i minimalizovat chybu, která by mohla hodnocením vzniknout:

$$min(q^*, p^*) \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{9}$$

kde κ je množina dvojic (u, i) pro každé známé hodnocení r_{ui} (tréninková množina). Systém se učí model pomocí dříve pozorovaných hodnocení. Bohužel cílem je tato předchozí hodnocení zobecnit a to tak, aby z nich bylo možné vyvodit budoucí, dosud neznámá hodnocení. Z tohoto důvodu je nutné, aby se systém přeučení vyhnul.

Interakce *uživatel-položka* Bylo by však nemoudré odvozovat plnou hodnotu hodnocení prostřednictvím interakce $q_i^T p_u$. Namísto toho se systém pokusí identifikovat část těchto hodnot, které mohou být ze vztahu *uživatel-položka* vyvozeny, založené pouze na jejich interakci:

$$b_{ui} = \mu + b_i + b_u \tag{10}$$

Využití zaujatosti uživatele Vliv zaujatosti v hodnocení r_{ui} je označováno jako b_{ui}. Celkové průměrné hodnocené se označuje μ; parametr b_u a b_i indikuje hledanou zaujatost uživatele u a položky i, resp. průměru.

Např. chceme odhadnout hodnocení uživatele u filmu *Titanic*. Průměrné ohodnocení je 3,7 bodu (z $1-5$ možných). Hodnocení tohoto filmu je lepší než průměr, tedy má sklon být hodnocen o 0,5 bodu lépe než je průměr, na druhou stranu, náš uživatel může být velice kritický a může mít tendenci hodnotit o 0,3 bodu níže, než je průměr. Odhadnutí ohodnocení filmu *Titanic* tohoto uživatele by tedy mohlo být $3,9$ bodu $(3,7 + 0,5 - 0,3)$. Zaujatost odvozená z rovnice (8) by tedy mohla vypadat následovně:

$$r_{ui} = \mu + b_i + b_u + q_i^T * p_u \tag{11}$$

Ovlivnění hodnocení Zde pozorované hodnocení je sraženo dolů vlivem čtyř složek: globální průměr, zaujatost položky, zaujatost uživatele a vztah *uživatel-položka*. To umožňuje, aby každá z těchto složek ovlivňovala pouze tu část výsledku, která se jí týká. Systém

se učí minimalizovat čtvercovou chybovou funkci:

$$min(q^*, p^*, b^*) \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - p_u^T * q_i)^2 + \lambda$$
$$(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2) \tag{12}$$

Řešení z hlediska časového průběhu

Doposud prezentované modely byly statické. Ve skutečnosti se však mohou objevit situace, které mohou hodnoty dat změnit. Z výše jmenovaných čtyř složek jde o následující [12]:

- **zaujatost položky** $b_i(t)$ - oblíbenost položky se může v průběhu času měnit,
- **zaujatost uživatele** $b_u(t)$ - uživatelovo hodnocení by mohlo klesnout/stoupnout,
- **uživatelské preference** $p_u(t)$ - časem se může stát, že uživatel změní svůj „vkus“ a přesune své působení např. od psychologických filmů k filmům s historickou tematikou apod.

Naši rovnici (11) tedy upravíme s možností predikcí hodnocení v časové linii:

$$r_{ui} = \mu + b_i + b_u + q_i^T * p_u \tag{13}$$

Kapitola 4

Doporučení na základě obsahu

4.1 Analýza založená na datech

Doporučení na základě obsahu (*Content-based-recommendations*) se snaží nějakým způsobem obecně analyzovat položky, které jsou pro uživatele určitým způsobem zajímavé. Analýza je založena na datech, které tento systém získá. Pokud jsou data v databázové podobě, pak s výpočetními procesy velký problém není. Pokud se ovšem data vyskytnou v odlišné podobě (například jako text na webové stránce restaurace), zde už nastává menší problém, protože „pohodlná restaurace s výhledem na řeku“ není informace, se kterou by takový systém mohl pracovat. Postup získávání dat touto cestou je popsán na konci této kapitoly v sekci 4.1.3. [15]

4.1.1 Uživatelský profil

Uživatelský profil v podstatě popisuje:

- **Preference svého majitele** - existuje mnoho způsobů této reprezentace, avšak jednou z nejčastějších je funkce, která pro každou položku určí pravděpodobnost, s jakou se bude uživateli líbit. Tato funkce může být také určena k tomu, aby vybrala kombinaci n položek, která pak uživatele charakterizuje.
- **Historii interakcí uživatele** s RS, např. hodnocení, nákupy, zájmy, ...).

Tyto dva body jsou důležité zejména proto, že systém může zobrazit historii uživatele nebo s ní nějak pracovat, a také může filtrovat položky, se kterými se už uživatel setkal a nepředkládat mu je znova. Další důležitá vlastnost je, že takové položky se pak předloží RS a ten z nich vytvoří tzv. *uživatelský model* k naučení.

Existuje více přístupů k manuálnímu poskytování informací pomocí RS: *User Customization* a *Rule-based Recomendation Systems*.

User Customization

V podstatě jde o snahu přizpůsobit systém uživateli. Ať už je to formou zaškrtávacích položek, kde např. označí některé z možností (druhy sportů, druhy jídla apod.), nebo pomocí vstupních polí, kde uživatel sám napíše své preference (oblíbený autor, zpěvák apod.). Poté, co tyto informace zadá, databáze na to odpoví tím, že vyhledá zadaná kritéria a uživateli je zobrazí. Existuje však i zde několik omezení těchto systémů [15]:

- systém vyžaduje interakci od uživatele, tedy uživatel sám musí data ručně vyplnit, ale někdy může být obtížné, ba i nemožné tato data získat,
- zájmy uživatele se mohou měnit (v zimě se může koukat na spíše na hokej a tak ho informace o fotbalové lize, která má zrovna přestávku, nemusí zajímat),
- systémy nejsou uzpůsobeny k tomu, aby zajistily pořadí, nebo váhu, chceme-li, zadaných položek - může se tedy stát, že výsledků pak bude příliš mnoho nebo příliš málo.

Rule-based Recommendation Systems

RS má svá pravidla, na základě kterých pak doporučuje uživateli další produkty. Systém může např. obsahovat pravidlo, které doporučí pokračování filmu nebo knihy uživatelům, kteří projevili zájem o předchozí díly. Jiné pravidlo může doporučit CD interpreta, který se též objevil v uživatelově profilu. Takovéto systémy založené na pravidlech mohou zachytit několik důvodů pro vyvození doporučení, ale nemohou nabídnout až tolik detailní osobní doporučení, jaké umožňují jiné přístupy. [15] Podrobnější popis tohoto přístupu je uveden v kapitole 3.1.

4.1.2 Učení uživatelského profilu

Vytvoření modelu preferencí uživatele z jeho uživatelského profilu je forma klasifikačního učení. Trénovací data jsou rozdělena do kategorií (*classification* - třídění), např. binárních kategorií (položky, které se uživateli líbí vs. položky, které se mu nelíbí). Toto určování se děje přímo (uživatel přímo řekne, že se mu položka líbí), nebo nepřímo (pokud uživatel např. položku koupí, značí to, že se mu líbí, avšak když ji vrátí, znamená to opak).

Obecně lze říci, že explicitní způsob je přesnější, protože uživatel data zadal sám, avšak je „omezenější“ v rozsahu dat, která pak doporučí. Implicitní způsob je naopak v tomto „volnější“, ale za cenu nejistoty u doporučovaných položek. [15]

Algoritmy Zde je seznam algoritmů, které mohou být v *Rule-based Recommendation Systems* používány:

- ***Decision tree*** - vytváří rekurzivně tzv. rozhodovací strom podle klasifikací, dokud listy netvoří pouze jednotlivé třídy [16],

- ***Nearest Neighbor method*** - vybírá „vzájemně sousedící“ položky, tedy ty, které jsou si nějakým způsobem podobnější než jiné [17],
- ***Rocchio's Algorithm*** - tzv. algoritmus zpětné vazby, které umožňují uživateli ohodnotit výsledek doporučovacího procesu a na základě této zpětné vazby poté více personalizovat další kolo doporučení a snažit se jej podle potřeby vylepšit [18],
- ***Naive Bayes*** - pravděpodobnostní metoda doporučení. [19]

4.1.3 Získávání informací z textu

Někdy je potřeba získat potřebné informace z textu, pokud je nemá k dispozici od uživatele. Z tohoto důvodu se někdy využívá algoritmus, který volný text převede do strukturované reprezentace, např. pro vyhledávací systémy. Tento proces se nazývá vyplývání (*entailment*), lépe zobecňování. Jeho cílem je najít společný termín pro slova, která jsou si syntakticky podobná (jako např. *compute, computation, computer, computes, computers*...).

$$w(t,d) = \frac{tf_{t,d} \log\left(\frac{N}{df_t}\right)}{\sqrt{\sum_i (tf_{t_i,d})^2 \log\left(\frac{N}{df_{t_i}}\right)^2}}\,, \tag{14}$$

kde $w(t,d)$ je váha slova t (frekvence výskytu v dokumentu d), tf je tzv. *term-frequency*, neboli frekvence výskytu slova v dokumentu. Termíny jsou následně seřazeny podle váhy do podobné tabulky jako v databázi (např. *energy* - 0,206; *state* - 0,122; *partial* - 0,106; *alert* - 0,103; *test* - 0.088; *market* - 0,074; ... [15]

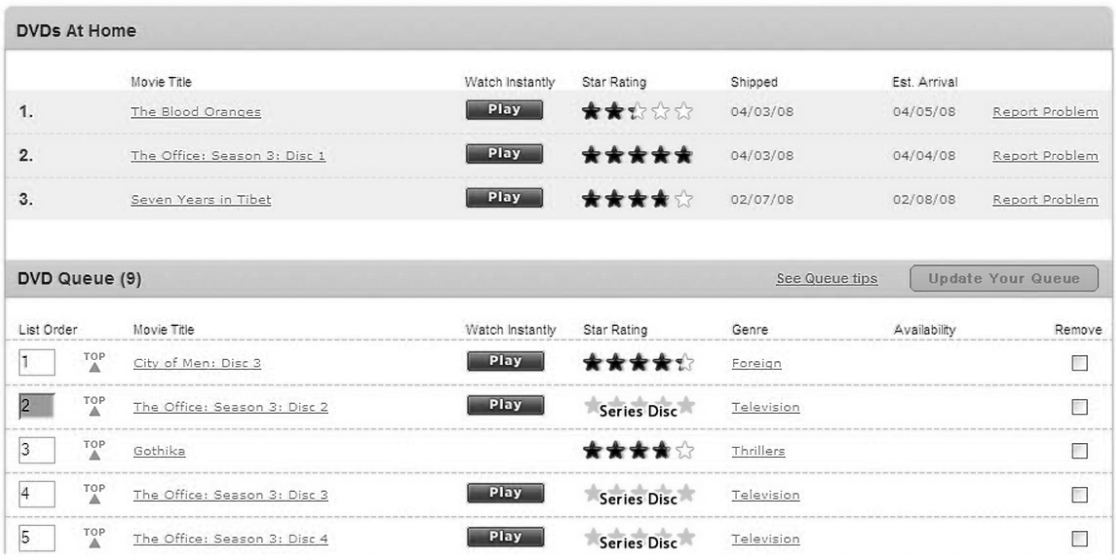
Nastává zde ovšem problém, že vyjádřením těchto vztahů ztrácíme původní vazby mezi slovy. Např. to, že text u některé restaurace obsahuje slovo „vegetariánské“ nenaznačuje, zda se jedná o restauraci vyloženě vegetariánskou nebo o restauraci, jejíž menu je z části vegetariánské, což může být pro člověka prahnoucím po kuřecím steaku značný rozdíl. Je proto důležité rozlišovat mezi technikami, které pracují se strukturovanými a které s nestrukturovanými daty. Jedním z řešení tohoto problému by bylo ukládat data jako sousloví, která lépe vystihnou význam (např. ne jen samostatná slova *electric* a *drive*, ale celé sousloví *electric drive*). [15]

Kapitola 5

Algoritmy pro doporučování obsahu

5.1 Netflix prize

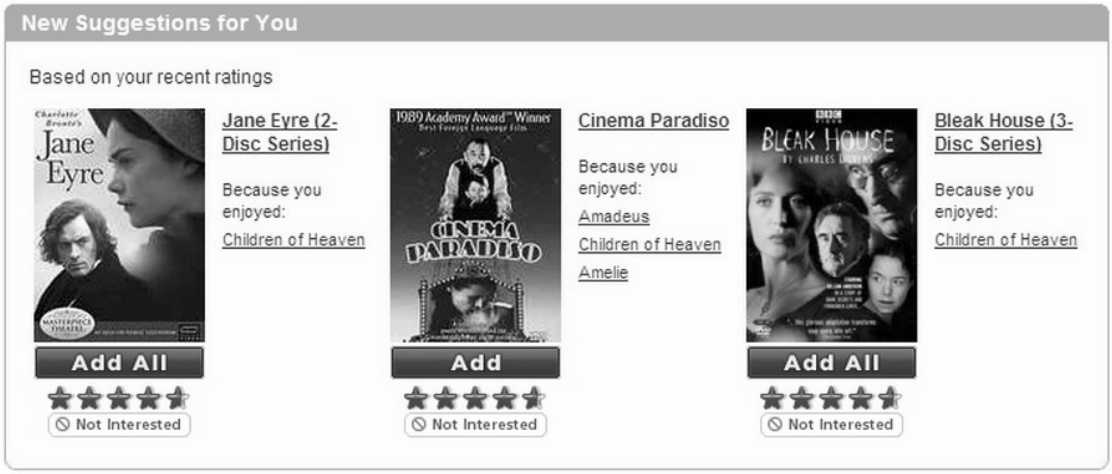
Firma Nexflix je americký poskytovatel streamovaných videí. Pracuje na principu VOD (*Video on demand*) - uživatel může na požádání sledovat video či poslouchat audio prostřednictvím zařízení jako je mobilní telefon nebo počítač, to vše v reálném čase, nebo si je může stáhnout do zařízení a přehrát je později. Netflix působí hlavně na americkém území a některých evropských státech [20].



Obr. 5.1: *Příklad ohodnocení filmu z webových stránek* Netflix [21].

Společnost Netflix měla vlastní algoritmus na doporučování obsahu. Po určité době vyhlásila soutěž s hlavní cenou 1 000 000 $ o to, kdo zlepší její algoritmus *Cinematch* o více než 10 %. Pokud by se někomu takový algoritmus podařilo prezentovat, firma předpokládala, že by

to usnadnilo práci se zákazníky a zvedlo úroveň jejího podnikání. Nabízela data, na kterých se algoritmus testoval, s tím, že zájemcům dala vždy pouze polovinu dat, druhá polovina pak byla testována před porotou, před kterou museli řešitelé svou metodu předvést a vysvětlit. [22]



Obr. 5.2: *Příklad doporučení filmu z webových stránek* Netflix [23].

5.1.1 Testovací data

Netflix poskytla k testování algoritmů všem týmům stejná, tzv. *trénovací data*. Jednalo se o soubor dat, který obsahoval 100 480 507 záznamů hodnocení, které udělilo 480 189 uživatelů 17 770 filmům. Hodnocení každého filmu bylo v rozmezí 1-5. Průměrný uživatel v těchto datech viděl více než 200 filmů a průměrný film byl hodnocen více než 5 000 uživateli. Rozptyl v datech však může být veliký, objevují se tu uživatelé hodnotící více než 17 000 filmů a napak film se 3 hodnoceními. Více také v kap. 6.2.

Během předvádění algoritmu před komisí byla použita tzv. *kvalifikační data* s 2 817 131 záznamy. Polovina těchto dat sloužila k určení vítěze, druhá polovina se použila k výpočtu žebříčku skóre.

RMSE

Root-mean-square-error neboli RMSE je často používána pro vyjádření míry mezi hodnotami předpovídané modelu a modelu skutečných hodnot, představuje směrodatnou odchylku mezi predikovanými a pozorovanými hodnotami. f_i je předpověď (*forecast*) a o_i skutečné hodnocení (*observation*) uživatele u_i. [24]

$$RMSE_{f_o} = \sqrt{\frac{\sum_{j=1}^{N}(z_{f_i} - z_{o_i})^2}{N}} \tag{15}$$

5.1.2 Výsledky soutěže

Výsledky soutěže shrnuje tabulka 5.1.

Poř.	Název týmu	RMSE	Zlepšení	Nejlepší čas
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay Un.	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

Tab. 5.1: *Výsledky soutěže Netflix Prize* [25].

26. června 2009 předvedl tým *BellKor's Pragmatic Chaos* svůj algoritmus, který dosáhl 10,05 % zlepšení oproti *Cinematch*. *Netflix Prize* poté začala odpočítávat třicetidenní „*last call*“, který měly k dispozici ostatní týmy, aby předvedly lepší řešení.

25. července 2009 tým *Ensemble* dosáhl 10,09 % zlepšení oproti *Cinematch*. O den později byla soutěž ukončena.

Konečné pořadí ukázalo, že oba týmy splňují minimální požadavky pro udělení ceny. Laureátem této ceny se měl stát tým, jehož algoritmus bude výkonnější na druhé části kvalifikačních dat, která sloužila právě pro tento účel. Algoritmus týmu *BellKor's Pragmatic Chaos* skončil o 20 minut dříve a proto se výsledkem *Netflix Prize* stal on. [25]

5.2 Algoritmy v *Neftlix Prize*

Zbývá představit původní algoritmus *Cinematch* a dva vítězné algoritmy *Neftlix Prize* týmů *BellKor's Pragmatic Chaos* a *Ensemble*.

5.2.1 *CineMatch*

Úkolem algoritmu *Cinematch* je snažit se odhadnout, který film by se uživateli líbil na základě filmů, které už viděl předtím. Snaží se tak zákazníkovi usnadnit hledání a zároveň respektovat jeho oblíbený žánr. [22] Problém je v tom, že algoritmus jako takový

se mi nepodařilo najít. Narazila jsem jen na několik článků, které popisují, jak funguje, ale jeho oficiální verze nikde k sehnání nebyla. Uvedu zde pouze informace, které jsem sesbírala, abych jen nastínila, jak algoritmus fungoval předtím, než byla ukončena Netflix Prize a implementován algoritmus *BellKor's Pragmatic Chaos*, popsaný níže.
Vstupní informace do algoritmu byly následující:

- filmy, uspořádané do skupin podle podobnosti,
- hodnocení uživatele, který si film zapůjčil,
- kombinované hodnocení všech uživatelů registrovaných na webu Netflix.

Ve [26] je uvedeno, že *Cinematch* s přesností půl hvězdičky odhadl 75 % doporučení. Zároveň by polovina uživatelů dala filmům doporučených algoritmem *Cinematch* pět hvězdiček, tedy nejvyšší hodnocení.

Při předpovídání ohodnocení je důležité zvážit následující problémy, se kterými se Netflix setkává každý den a se kterými si algoritmus musí umět poradit, ale která také mohou pomoci k optimalizaci doporučení: [27]

- Netflix disponuje několika miliardami záznamů v databázi, z nichž se hodnocení předpovídá. A každý den jich přibližně milion přibude.
- Je nutné vyřešit problém, jak prioritizovat popularitu a následné doporučení filmu. Můžeme každou hodinu, denně, týdně předkládat nová doporučení, můžeme uživatele rozdělit podle lokality či podobných metrik apod.
- Každý záznam o shlédnutí filmu také obsahuje informace jako např. délka filmu, typ zařízení, denní doba apod.
- Ke každému filmu se mohou vázat různá metadata, jako např. herec, režisér, žánr apod.
- Pokud máme záznamy o tom, na jakém typu zařízení uživatel film sledoval, můžeme také pozorovat jeho chování, např. pohyb myší po stránce, čas strávený na stránce, na konkrétní části stránky apod.
- Data ze sociální sítí, např. doporučení přátel apod.

5.2.2 Algoritmus týmu *BellKor's Pragmatic Chaos*

Pro začátek nutno říci, že bylo docela složité dát samotný algoritmus do podoby, která ho alespoň trochu vysvětlí - problém je, že tým *BellKor's Pragmatic Chaos* jej popsal někdy v polovině soutěže (přibl. v roce 2007) a v následujících dokumentech pouze vysvětloval, které metody se změnily a s jakými parametry.

Vylepšený algoritmus *k-NN* a další úpravy

Prvním krokem bylo vylepšení k-NN algoritmu, popsaného v 3.2.1. Je pravda, že sousedství a možnost lokalizace jednotlivých položek je velice užitečná, ale trochu zkreslující. V podstatě jde o to, že pokud má položka např. dvě ohodnocení $\{2, 3\}$, pak její průměr je $2, 5$. Ovšem jiná položka se stejným průměrným ohodnocením může mít jednotlivá hodnocení např. $\{2, 3, 2, 4, 3, 2, 2, 2\}$. Je vidět, že i když mají obě položky stejné ohodnocení, druhá z nich je uživateli „oblíbenější“, lépe řečeno vyhledávanější. K tomu slouží úprava tohoto algoritmu, tzv. normalizace odstraněním globálních dopadů (*Normalizing by removing global efects*).

Druhý důvod je přístup k informacím o uživatelích nebo položkách, pro které ohodnocení slouží. I když faktorizace (3.2.3) může dopomoci k odhalení jejich struktury pomocí latentních proměnných (žánr filmu apod.), jejich vyčtení z uživatelských demografií může být efektivnější. Následně je pak možné např. rozlišit uživatele, kteří preferují co nejlépe ohodnocené filmy nebo uživatele, který se raději specializuje na nějaký žánr apod.

A třetí důvod k úpravě je ten, že mohou existovat charakteristiky, jako např. datum hodnocení, které mohou vysvětlit některé odchylky ve skóre. Např. hodnocení uživatele u jednoho filmu se může časem změnit. Podobně také může hodnocení klesat s dobou, která uplynula od premiéry filmu, zatímco jiné mohou obstát proti zkoušce času lépe. [28]

Kompletní algoritmus

Celý algoritmus je velice obsáhlý a docela složitý. Po případné zájemce doporučuji zabrousit do veřejných dokumentů, které jej popisují: hrubý popis algoritmu se zveřejněnými změnami [29], vylepšená metoda k-NN [28], interpolace vah sousedů v k-NN [30] a modelování relací [31].

5.2.3 Algoritmus týmu *Ensemble*

Jako úvod k popisu tohoto algoritmu si dovolím citovat (přeložený) úvod článku, ve kterém je algoritmus *Feature-Weighted Linear Stacking* popsán:

> „Souborové metody, jako je např. skládání, jsou navrženy pro zvýšení přesnosti předpovědi smícháním předpovědí několika modelů strojového učení. Nedávné práce ukázaly, že použití meta-vlastností (přídavných vstupů popisujících každý příklad v datech), může zvýšit výkon souborových metod, avšak největší zisky pochází z nelineárních vstupů, které vyžadují množství ladění a tréninkového času.
> Zde prezentujeme lineární techniku FLWS (*Feature-Weighted Linear Stacking*), která zahrnuje meta-vlastnosti pro zlepšení přesnosti při zachování již známé

přesnosti, pokud jde o rychlost, stabilitu a interpretovatelnost. FWLS kombinuje modelové předpovědi lineárně pomocí koeficientů, které jsou samy o sobě lineární funkcí meta-vlastností.
Tato technika byla klíčovým prvkem pro získání druhého místa v nedávno uzavřené Netflix Prize soutěže. Významné zvýšení přesnosti přes standardní lineární skládání jsou demonstrovány na Netflix Prize kolaborativním filtrování datového souboru.“ [32]

Skládání „Skládání“ je technika, při které jsou předpovědi kolekcí modelů dány jako vstupy do druhé úrovně učícího algoritmu. Tato druhá úroveň je naučena kombinovat modely předpovědí co nejoptimálněji ve formě konečné sady předpovědí.

Tato technika kombinování (skládání) různých modelů pro zvýšení predikce nad úroveň přesnosti jednotlivých modelů je dnes úspěšně využívána (např. chemometrie, filtrování nevyžádané pošty apod.). Jeden z velice názorných použití síly skládání modelů pochází právě ze soutěže *Netflix Prize*, kde tým *BellKor* smíchal dohromady směs přes sto různých modelů. Ve skutečnosti to byla směs na několika úrovních, tedy směs směsí. [32]

Využití meta-vlastností Intuice nám napovídá, že spolehlivost modelu se liší v závislosti na podmínkách, při kterých se používají, a v závislosti na požadavcích. Např. při filtrování, kdy chceme předpovědět preference zákazníků pro různé produkty, se množství nashromážděných údajů může lišit v závislosti na tom, kterého uživatele/položku bereme v úvahu. Např. model A může být lepší pro uživatele, kteří ohodnotili mnoho produktů, ale naopak model B může být přesnější pro uživatele, kteří mají hodnocení jen několik. Na základě této myšlenky je právě využíváno oněch meta-vlastností.

Nejčastěji používaný algoritmus je algoritmus *lineární regrese*[1]. Pokud se však tyto meta-vlastnosti nedají vyjádřit podobně jako data, se kterými tento algoritmus pracuje, pak je neumí využít. [32]

Naopak nelineární, opakovaně trénovatelné a skládané algoritmy, které dokáží těchto metadat využít, z nich mohou vytěžit velice mnoho. Právě toho dosáhl i tým *BellKor* s jeho komplexní směsí mnoha dílčích řešení a prolnutí technik, která různá metadata používají (např. počet uživatelů a filmů, počet hodnocení na konkr. den, datum hodnocení a další interní parametry).

Téměř ve všech případech byly algoritmy využívající metadat nelineární a opakující se, např. umělá neuronová síť nebo posílený rozhodovací strom apod. Celkem bylo otestováno

[1]Lineární regrese je matematická metoda používaná pro proložení souboru bodů v grafu přímkou. O bodech reprezentujících měřená data se předpokládá, že jejich x-ové souřadnice jsou přesné, zatímco y-ové souřadnice mohou být zatíženy náhodnou chybou, přičemž předpokládáme, že závislost y na x lze graficky vyjádřit přímkou.

osm meta-vlastností, z nichž nejvíce prospěšné byly počty uživatelských hodnocení a počty hodnocení položky, které byly taky nejčastěji používané. [32]

Feature-Weighted Linear Stacking

Nechť $\mathcal{X}$ je prostor vstupů a $g_1, g_2, \ldots, g_L$ označují učící funkce předpovězení L strojových učících modelů, kde $g_i : \mathcal{X} \to \mathbb{R}, \forall i$. Kromě toho nechť $f_1, f_2, \ldots, f_M$ reprezentuje kolekci M meta-vlastností použitých při skládání. Každá meta-funkce f_i mapuje každý bod x z dat $\mathcal{X}$.

Standardní lineární regresní algoritmus usiluje o hledání skládané předpovídající funkce b následovně:

$$b(x) = \sum_i w_i g_i(x), \forall x \in \mathcal{X}, \tag{16}$$

kde každá váha naučeného modelu w_i je konstanta z $\mathbb{R}$.

Feature-weighted linear stacking (FWLS) pak modeluje skládané váhy w_i jako lineární funkci meta-vlastností:

$$w_i(x) = \sum_j v_{ij} f_j(x), \forall x \in \mathcal{X}, \tag{17}$$

pro každou naučenou váhu $vi, j \in \mathbb{R}$.

Původní rovnici pro funkci b pak můžeme upravit následovně:

$$b(x) = \sum_{i,j} v_{i,j} f_j(x) g_i(x), \forall x \in \mathcal{X}, \tag{18}$$

což nám poskytuje následující FWLS optimalizaci problému:

$$\min_v \sum_{x \in \overline{\mathcal{X}}} \sum_{i,j} \{v_{ij} f_j(x) g_i(x) - y(x)\}^2, \tag{19}$$

kde $y(x)$ je cíl předpovězení pro všechna data x a $\overline{\mathcal{X}}$ je množina $\mathcal{X}$ použitá pro skládání parametrů.

Tímto získáme funkci b, která je lineární pro volné parametry v_{ij}, a každou lineární regresi můžeme použít k odhadu těchto parametrů. Nezávislé proměnné v regresi (tj. „vstupy“ v termínu strojového učení) jsou ML produkty součinu $f_j(x)g_i(x)$ meta-vlastností a předpověď modelu pro každé $x \in \overline{\mathcal{X}}$. [32]

Kapitola 6

Popis dat

Druhá část této práce se bude věnovat implementaci a modifikaci výše popsaného algoritmu *k-Nearest Neighbor* na datech ze soutěže *Netflix prize*. Pro implementaci bylo využito software:

- *Wolfram Workbench 2.0* – implementace algoritmu,
- *MySQL Workbench 6.0* – databáze.

6.1 Data ze soutěže *Netflix prize*

Jak již bylo dříve řečeno, pro implementaci byla použita data z *Netflix prize*. Ta bylo zapotřebí nejprve ze souborů uložit do databáze (jsou připojena jako příloha na CD).

Složka *training_set* obsahuje 17 770 souborů - každý soubor je jeden film se záznamy o jeho uživatelských hodnocení. Pro nahrání dat do databáze jsem napsala program v Javě, který prochází jeden soubor za druhým a každých 10 000 hodnocení uloží jako jeden `insert` do databáze (kompletní zdrojový kód je přiložen na CD). Celkem program běžel 63 minut a 6 sekund, než všechna data do databáze nahrál.

6.2 Připrava dat v databázi

Po nahrání dat do databáze bylo třeba databázi bylo třeba naindexovat, indexace zaručí, že se data dají vyhledat mnohem rychleji.

6.2.1 Naindexování databáze

Příkazy pro naindexování databáze:

```
ALTER TABLE training_set ADD INDEX idxFilm (film_ID)
ALTER TABLE training_set ADD INDEX idxUser (user_ID)
ALTER TABLE training_set ADD INDEX idxUserFilm (user_ID, film_ID, mark)
```

```
ALTER TABLE training_set ADD INDEX idxCtFilm (COUNT(film_ID))
```

Stav po naindexování databáze:

```
SHOW KEYS in training_set
```

tabulka	index	sloupec	kardinalita	Typ indexu
training_set	idxFilm	film_ID	255422	BTREE
training_set	idxUser	user_ID	1071665	BTREE
training_set	idxUserFilm	user_ID	1159920	BTREE
training_set	idxUserFilm	film_ID	98593248	BTREE
training_set	idxUserFilm	mark	98593248	BTREE

Tab. 6.1: *Tabulka indexů v databázi.*

Pro srovnání dotaz: `SELECT * FROM training_set WHERE user_ID=12546` trval před naindexováním databáze 26,54 s, po naindexování 0,7 s.

6.2.2 Popis dat v databázi

Uvedu zde několik SQL dotazů pro představu, jak vypadají data, s kterými v dalších kapitolách budu pracovat:

- **Celkový počet dat v databázi:** 100 480 507

  ```
  SELECT COUNT(*) FROM netflix.training_set
  ```

- **Největší user_ID se záznamem v databázi:** 2 649 429

  ```
  SELECT MAX(user_ID) FROM netflix.training_set
  ```

- **Nejmenší user_ID se záznamem v databázi:** 6

  ```
  SELECT MIN(user_ID) FROM netflix.training_set
  ```

- **Rok, ve kterém bylo natočeno nejvíce filmů:** ukazuje tab. 6.2

  ```
  SELECT year, count(*) AS ct FROM list_movies GROUP BY year ORDER BY ct desc LIMIT 5
  ```

Pořadí	Rok natočení	Počet filmů
1.	2004	1 436
2.	2002	1 310
3.	2003	1 271
4.	2000	1 234
5.	2001	1 184

Tab. 6.2: *Tabulka ukazuje 5 let, ve kterých bylo natočeno nejvíce filmů.*

- **Počet uživatelů, kteří ohodnotili alespoň jeden film:** 480 189

```
SELECT user_ID, count(*) AS ct FROM tranining_set GROUP BY user_ID HAVING ct >= 1
```

- **5 uživatelů, kteří hodnotili nejvíce filmů:** ukazuje tab. 6.3

```
SELECT user_ID, count(*) AS ct FROM training_set GROUP BY user_ID ORDER BY ct desc
```

Pořadí	user_ID	Počet filmů
1.	305344	17 653
2.	387418	17 436
3.	2439493	16 565
4.	1664010	15 813
5.	2118461	14 831
6.	1461435	9 822
7.	1639792	9 767
8.	1314869	9 740
9.	2606799	9 064
10.	1932594	8 880

Tab. 6.3: *Tabulka ukazuje prvních deset uživatelů, kteří ohodnotili nejvíce fimů.*

- **10 filmů, které mají největší počet ohodnocení:** ukazuje tab. 6.4

```
SELECT film_ID, count(*) AS ct, name, year, movie_ID FROM training_set JOIN list_movies
ON (training_set.film_ID = list_movies.movie_ID) GROUP BY film_ID ORDER BY ct desc
```

Pořadí	film_ID	Počet	Jméno filmu	Natočeno
1.	5317	232 944	Miss Congeniality	2000
2.	15124	216 596	Independence Day	1996
3.	14313	200 832	The Patriot	2000
4.	15205	196 397	The Day After Tomorrow	2004
5.	1905	193 941	Pirates of Carib.: Black Pearl	2003
6.	6287	193 295	Pretty Woman	1990
7.	11283	181 508	Forrest Gump	1994
8.	16377	181 426	The Green Mile	1999
9.	16242	178 068	Con Air	1997
10.	12470	177 556	Twister	1996

Tab. 6.4: *Tabulka ukazuje prvních deset filmů, které uživatelé hodnotili nejčastěji.*

6.3 Přidané tabulky pro práci s daty

Pro práci s daty bylo nutné vytvořit tabulku navíc: `most_rated_films`. Tato tabulka obsahuje pouze dva sloupce - `film_ID` a `rated` - počet ohodnocení filmu s konkrétního `fim_ID` uživateli. Tato data není těžké získat, ale protože tuto informaci používám dále při práci s daty docela často, je jednodušší takovouto tabulku vytvořit než pokaždé vybírat potřebný údaj z databáze (příkaz pro seřazení filmů podle počtu ohodnocení: `SELECT film_ID, count(*), name, year FROM training_set JOIN list_movies ON training_set.film_ID = list_movies.movie_ID GROUP BY film_ID`).

Takovouto tabulku není těžké vytvářet nebo během změn dat obnovovat, ale snažila jsem se algoritmy implementovat s co nejméně dotazy do databáze. Navíc tento dotaz trval 4 min a 36 sec, což je čas, o který je pak algoritmus (minimálně jednou) rychlejší.

6.4 Parametry počítače

Praktická část probíhala v SW *Wolfram Mathematica* na osobním počítači Lenovo E531 s následujícími parametry:

- Procesor: Intel(R) Core(TM) i5-3230M CPU @ 2,60 GHz 2,60 GHz
- RAM: 8,00 GB
- Operační systém: Windows 8 64-bit

Velikost paměti je ve *Wolfram Mathematica* nastavena implicitně na 128 MB.

Kapitola 7

Algoritmus *k*-NN

K-NN algoritmus je nejznámější a nejpoužívanější algoritmus v oblasti CF, který už byl mnohokrát implementován v různých jazycích, ze tří výše uvedených a vysvětlených algoritmů bývá implementován nejčastěji. Zde jsou uvedeny zdroje na některé implementace:

- **Java**: aplet [34], velmi dobrá a propracovaná implementace [35], klasická implementace algoritmu [36] a [37]
- **MaLab**: implementace *fast k-NN* [38]
- **C++**: několik problémů vyskytujících se v *k*-NN [39], sekvenční a paralelní *k*-NN [40], paralelizovaný *k*-NN [41]

Co se týče implementací v *Mathematica*, podařilo se mi najít pouze dvě implementace, ovšem jednalo se o klasifikátory, nikoli algoritmy pro doporučení položek, a nelze je tedy přímo použít. První implementace pochází z [42], druhá implementace se nachází na webu univerzity v Linzi [43].

7.1 Postup vypracování

Pro implementaci *k*-NN v software *Mathematica* jsem si vybrala variantu *user-based* (porovnávání uživatelů mezi sebou). V podstatě se algoritmus skládá ze tří hlavních částí, uvedených níže.

7.1.1 Uživatel, kterému se položka doporučuje

Vždy většině příkladů se téměř vždy jedná o konkrétní `user_ID` uživatele, kterému chceme položku doporučovat. Protože se v datech vyskytují různé typy uživatelů (od uživatele, který neohodnotil žádný film, přes průměrného uživatele, který jich viděl několik, až po náruživého diváka, který ohodnotil téměř všechny), je možné vyzkoušet algoritmus opravdu na všech případech a velikostech množin ohodnocení.

7.1.2 Vybrat množinu sousedů N

Je nutné vybrat dostatečné množství co nejpodobnějších uživatelů, na jejichž základě je doporučení vykonáno. Zde však můžeme narazit na několik typů sousedů, se kterými se pojí i charakteristické problémy při doporučení položky:

- **Běžný uživatel** u je takový uživatel, pro kterého není problém najít dostatečně velkou množinu podobných uživatelů, neboli sousedů. Jednoduše řečeno, najdeme množinu uživatelů, kteří viděli tu samou nebo velice podobnou množinu filmů a všechny filmy „navíc“ pak tvoří množinu filmů, ze které se bude vybírat ten k doporučení (vysvětleno dále v kap. 7.3).

- **Extrémní uživatel** u_e je uživatel, pro kterého není možné najít dalšího uživatele s podobným ohodnocením.

  Prvním typem u_e je takový uživatel, jehož množina je příliš velká nebo příliš „jiná“ než je tomu u sousedů, a množinu N pak nelze vytvořit. Zde je řešením vyloučit několik filmů, které mají celkově nejméně hodnocení (hledáme na základě filmů, které vidělo co nejvíce uživatelů, neboť ti pak tvoří množinu N).

  Druhým typem je opačný extrém – uživatel u_n, který nemá žádné hodnocení (např, uživatel, který si právě založil účet). Protože nemáme k dispozici historii tohoto uživatele, nemáme na čem založit doporučení. Tento problém je vyřešen v kap. 7.3.3.

7.1.3 Doporučení položky

Pokud máme k dispozici množinu sousedů N, pak jen zbývá z jejich položek vybrat ty, které budou ke konečnému doporučení nejvhodnější – v závislosti na tom, zda klademe důraz na počet sousedů, kteří položku ohodnotili, nebo na ohodnocení, které film v průměru dostal. Případ u_e je řešen v kap. 7.3.1 a u_n je řešen zvlášť v kap. 7.3.3.

7.2 Problém s využitím Pearsonovy korelace a její náhrada

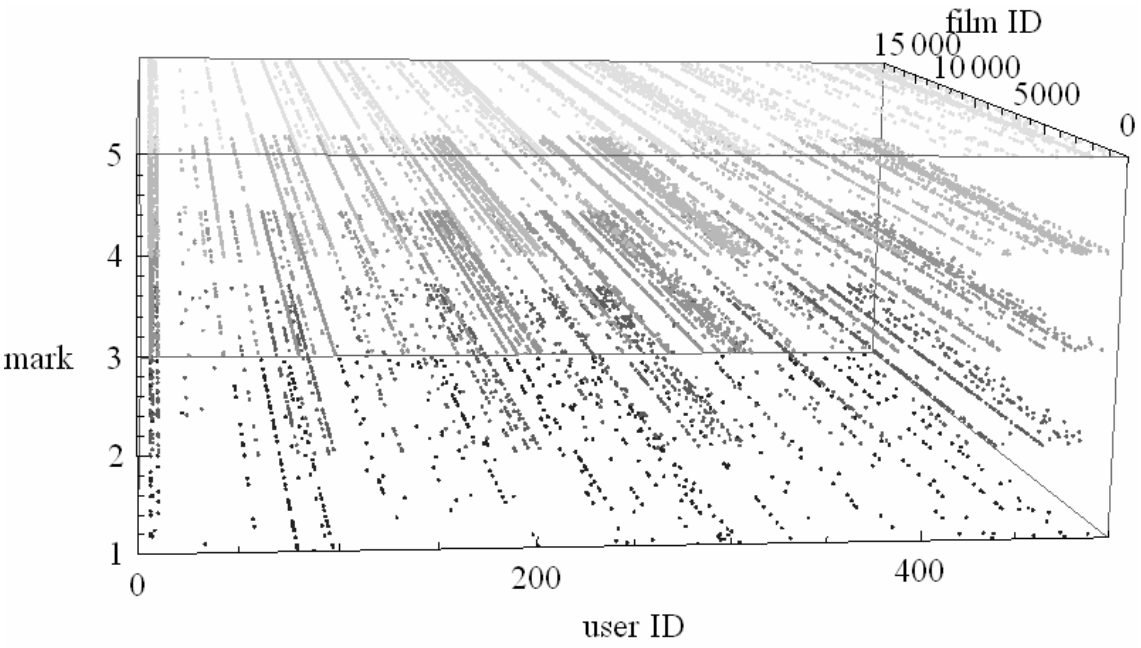
Při přesné implementaci algoritmu k-NN je nutné použít Pearsonovu korelaci (jak je uvedeno v [3]). Pearsonova korelace vyjadřuje „podobnost“ mezi sousedy:

$$sim(u,v) = \frac{\sum\limits_{i \in C}(r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum\limits_{i \in C}(r_{u,i} - \overline{r}_u)^2}\sqrt{\sum\limits_{i \in C}(r_{v,i} - \overline{r}_v)^2}}\,, \tag{18}$$

U dat z *Netflix prize* však nastává problém, že Pearsonova korelace je téměř nepoužitelná (a to se netýká jen těchto dat, ale i dalších, protože jejich struktura vztahu *uživatel-položka* bude velmi podobná). Každý film zde totiž představuje svou dimenzi, v konečném součtu

je takovýchto dimenzí 17770, což je počet, na který se Pearsonova korelace dá použít velice složitě, zda-li vůbec.

Hlavním úkolem je tedy vymyslet, jak aplikovat Pearsonovu korelaci nebo její náhradu na tato data. Hlavní myšlenka tkví v tom, jak bude uvedeno později, že namísto porovnávání sousedů Pearsonovou korelací používám jednoduchý průnik filmů uživatele u a filmů potenciálních sousedů z množiny N_p. Pokud je množina filmů uživatele u podmnožinou množiny souseda n_p (např. když se průnik množin liší jen o několik filmů), pak n_p lze považovat za souseda uživatele u a patří do množiny sousedů N.



Obr. 7.1: *Graf ukazuje vykreslení dat, na kterých je implementován algoritmus této práce, do tří dimenzí -* `user_ID` *(u tohoto příkladu pro jednoduchost jen do 500),* `film_ID` *a* `mark`*, tedy známka, kterou film obdržel v hodnocení. Z grafu je jasně vidět, že data jsou utříděna do pěti „horizontálních rovin" podle ohodnocení, a pěti set vertikálních rovin podle* `user_ID`*. Při bližším pohledu je zde možné vidět i některé extrémní případy uživatelů.*

7.3 Modifikovaný algoritmus k-NN

Modifikace algoritmu tedy spočívá hlavně v náhradě Pearsonovy korelace, jak bylo již vysvětleno výše. Další problém, na který jsem se soustředila, byla práce s daty. Doporučení musí proběhnout v relativně přípustném čase, což je na tak velkém množství dat, jako je zde, docela složité. Jedním z dalších problémů, které se zde vyskytují, je, zda filmy doporučovat spíše na základě počtu sousedů nebo jej spíše vyvážit ohodnocením, a jak vůbec doporučit film uživateli, který ještě žádné neviděl.

7.3.1 Běžný uživatel - popis algoritmu

1. **Odebereme z jeho množiny filmů U_f několik nejméně hodnocených filmů.**

   a) Pokud se jedná o extrémního uživatele u_n, který žádné filmy neohodnotil, je nutné pro něj implementovat zvláštní algoritmus - ten je uveden v kap. 7.3.3,

   b) pokud se jedná o extrémního uživatele u_e, pak je nutné filmy odebrat, jinak by nebylo možné doporučit film (podrobněji popsáno v kap. 7.1.2),

   c) určitou menší část odebírám i pokud se o u_e nejedná - na počátku jsme vybrali množinou sousedů podle toho, zda viděli nejméně ohodnocený film. Množina potenciálních sousedů, kteří viděli stejné filmy jako u, může být někdy velice malá, ne-li prázdná. Proto původní množinu zmenšíme o ty filmy, které vidělo nejméně uživatelů a proto nemají tak velký vliv na hodnocení. Najdeme tak více sousedů, kteří jsou nejen stejní, ale i „podobní“. Tato funkce pro redukci filmů je uvedena v kap. 7.3.2.

2. **Vybereme od uživatele u nejméně často ohodnocený film f_m.**

   Čím méně ohodnocení od uživatelů film má, tím méně uživatelů jej vidělo. Což znamená, že pokud budeme množinu potenciálních sousedů N_p vybírat podle f_m, vybereme tak nejmenší možnou, avšak přesto dostatečnou, množinu sousedů, se kterými budeme dále pracovat, a víme, že v ní nebudou uživatelé, u kterých není šance, že by mohli být sousedy. Znamená to tedy, že hned na začátku se tak vyloučí „nevhodní“ kandidáti.

3. **Najdeme množinu N_p uživatelů, kteří viděli také f_m.**

   Dalším krokem je získat z databáze záznamy o uživatelích, kteří viděli také f_m, ti pak tvoří množinu potenciálních sousedů N_p.

4. **Vybereme hodnocení všech uživatelů z N_p seřazených podle `user_ID`.**

   Když máme k dispozici množinu N_p, o kterých víme, že všichni viděli minimálně f_m. Vybereme tedy z databáze všechna jejich ohodnocení. Je velice výhodné seřadit si tento seznam ohodnocení hned na výstupu z databáze seřazený podle `user_ID`, algoritmus pak běží rychleji, protože jen rozděluje data do podmnožin a nemusí je prohledávat celá.

5. **Roztřídíme hodnocení jednotlivých sousedů a podle velikosti průniku s U_f rozhodneme, zda se jedná o souseda.**

   Ohodnocení sousedů z N_p roztřídíme podle jejich ID a následně pro každou množinu filmů, které tento soused viděl, uděláme pronik s filmy uživatele u. Pokud je průnik

dostatečně velký, pak můžeme uživatele $n_i \in N_p$ pro doporučení použít (opět je pro toleranci velikosti průniku použita funkce z kap. 7.3.2). Nyní máme množinu sousedů N.

6. **Vypočítáme, kolik uživatelů jednotlivé filmy vidělo a jaká je jejich průměrná známka.**

   Pro $\forall\, n \in N$ vybereme z databáze všechny filmy, které n_i viděl a ohodnotil a na pozici `film_ID` v listu L uložíme hodnoty `film_ID`, průměrné ohodnocení filmu a počet ohodnocení (počet sousedů, kteří film viděli a průměrná známka, kterou mu dali).

7. **Dle zadaných parametrů vybereme vhodný film pro doporučení.**

   Zadaným parametrem je nejnižší průměrná známka m_{min}, kterou film od sousedů dostal. Funkce nejprve vybere z L ty filmy, které viděli všichni uživatelé a vyhovují m_{min}. Pokud je množina vybraných filmů prázdná, funkce hledá filmy, které viděli všichni uživatelé kromě jednoho, kromě dvou, ..., dokud je množina prázdná.

7.3.2 Redukce množiny filmů

Funkce, která vypočítá počet filmů k odebrání, usnadňuje hledání sousedů uživatele u, v některých případech je to jediná pomoc, jak filmy doporučit. Zbývá vyřešit, kolik filmů by se mělo odebrat, aby nebyla snížena hodnota zbylých filmů k doporučení.

U_f	**1**	**2**	**3**	**5**	**10**	**20**	**30**	**50**	**100**	**200**
$\frac{x}{5}$	0.2	0.4	0.6	1	2	4	6	10	20	40
$\sqrt{x}$	1	1.41	1.73	2.23	3.16	4.47	5.47	7.07	10	14.14
$\frac{x^{1.4}}{200}$	0.005	0.013	0.023	0.048	0.126	0.331	0.585	1.195	3.155	8.325
U_f	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**10000**	**15000**	**17000**	**18000**
$\frac{x}{5}$	60	100	200	400	600	1000	2000	3000	3400	3600
$\sqrt{x}$	17.32	22.36	31.62	44.72	54.77	70.71	100	122.4	130.3	134.1
$\frac{x^{1.4}}{200}$	14.69	30.09	79.24	209.1	368.9	754.3	1990.5	3511.5	4184	4532.6

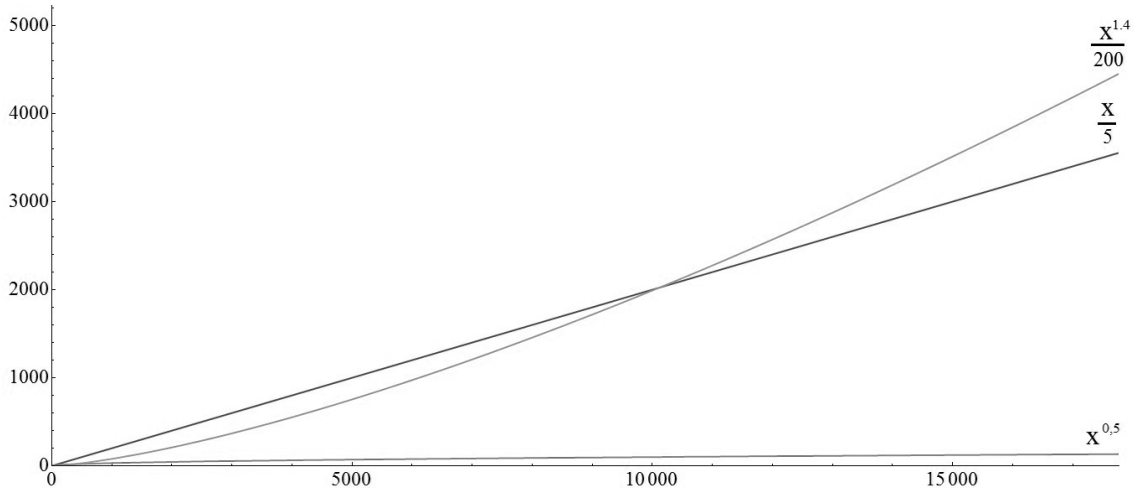
Tab. 7.1: *Tabulka hodnot jednotlivých funkcí: $y = x$, $y = \frac{x^{1.4}}{200}$ a $y = \sqrt{x}$ pro různě velké množiny filmů. V tabulce je vidět, že funkce $y = \frac{x^{1.4}}{200}$ splňuje požadavky kladené na potřebnou funkci – z malých množin odebere málo filmů a z velkých množin mnoho (v poměru k původní množině, např. v množině s 50 filmy odebere 2,39 % filmů, v množině se 17 000 filmy je to 24,6 %).*

Jedním z možných řešení je odebrat procentuální část množiny, např. 1/5 - to by pak znamenalo odebrat 1 film z 5, 4 filmy z 20... 3554 ze 17770 v extrémním případě. Ovšem odebrat jeden film z pěti je příliš velká část množiny, což naznačuje, že funkce nebude lineární. U uživatelů s malou množinou filmů nechceme odebrat žádný nebo jen několik, a naopak u uživatelů s obrovskou množinou filmů je potřeba jich odebrat větší část. To také

pramení z toho, že se mnohem častěji vyskytuje uživatel s několika desítkami filmů než ten, který jich viděl i několik tisíc.

Mezi nelineárními funkcemi by se mohla jevit jako dobrý výběr $\sqrt{n}$. Zde je však problém na obou stranách – při malém počtu filmů odebereme v poměru k původní množině uživatele příliš mnoho filmů a naopak odebrat 134 filmů z 17770 je nedostačující. Navíc, pokud bychom chtěli doporučit film uživateli, který ohodnotil jen jeden film, $\sqrt{1} = 1$, tedy odebereme mu jeho jediný film a nemáme poté podle čeho doporučovat.

Potřebujeme tedy najít takovou funkci, která vyřeší oba výše zmíněné problémy: malé množině odebere menší část, velké naopak větší část vzhledem k původní množině. Jako vyhovující jsem po empirickém zkoumání použila funkci $\frac{x^{1.4}}{200}$. Jednotlivé hodnoty pro všechny tři funkce pro porovnání ukazuje tabulka 7.1 a graf na obr. v příloze 7.2.



Obr. 7.2: *Graf ukazuje porovnání průběhů jednotlivých funkcí:* $y = x$, $y = \frac{x^{1.4}}{200}$ *a* $y = \sqrt{x}$.

7.3.3 Uživatel bez hodnocení - popis algoritmu

Při výběru uživatele, který v databázi nemá záznam o ohodnocení nějakého filmu, musí umět algoritmus doporučit film i tomuto uživateli u_n. I tento problém je tu ale vyřešen:

1. **Vybereme m filmů s jejich počtem a součtem ohodnocení.**

   Vybereme m filmů, které mají největší počet ohodnocení (pro představu je seznam 10 filmů s největším počtem ohodnocení je uveden v kap. 6.2.2).

2. **Vypočteme ze všech** *sum* `mark` **celkovou sumu $\sum_R$.**

   Pokud je to možné, je lepší vybrat počet ohodnocení i jejich součet, ve tvaru: {`film_ID`, $num_{ratings}$ a $\sum_{ratings}$}. $\sum_{ratings}$ je součet všech ohodnocení, které film dostal. Zaprvé se tak děje proto, aby filmy, které byly ohodnoceny častěji, byly vybrány s větší pravděpodobností. Zadruhé jde o to, aby film F_a, který dostal ohodnocení 5,

byl doporučen s větší pravděpodobností než film F_b, který má ohodnocení 3 nebo F_c s ohodnocením 2.

Pokud bychom vzali jedno ohodnocení od každého, pak např. F_a by byl doporučen v pravděpodobností 50 %, F_b s 30 % a F_c s 20 %. Pokud vyvážíme známku i počtem ohodnocení, výpočet vypadá následovně:

1) ohodnocení $F_a = \{5,3,4,5,2,3,5,4\}$, $F_b = \{4,3,5,2,3,4\}$ a $F_a = \{2,1,3,2,1,2\}$,
2) sumy ohodnocení: $\sum F_a = 31$, $\sum F_b = 21$ a $\sum F_c = 11$,
3) konečné pravděpodobnosti výběru: F_a 49,21 %, F_b 33,3 % a F_c 17,46 %.

Myslím si, že ač je toto řešení velice prosté, je z vyplývající logiky velice účinné. Bylo by možné přidat koeficient, který by ovlivňoval sumy podle toho, zda chceme doporučovat více na základě ohodnocení filmů nebo zda nám stačí jen počet ohodnocení filmů. Pro naše použití zatím postačí použití bez koeficientů.

3. **Vybereme náhodné číslo r od 1 do $\sum_R$ a vybereme film k doporučení.**

   Z jednotlivých sum ohodnocení spočteme sumu celkovou $\sum_R$. Ta bude sloužit k výběru náhodného čísla (s normálním rozdělením) od 1 do $\sum_R$. Když máme toto náhodné r číslo vybráno, přistoupíme k poslednímu kroku - výběru filmu podle r.

7.4 Testování

K testování jsem použila dva testy, které testují kvalitu doporučení (o tu jde především) a rychlost doporučení:

- **Přesnost doporučení**, což předpovězení známky, kterou by uživatel u filmu dal.
- **Rychlost doporučení**, jež by měla být přiměřená počtu dat.

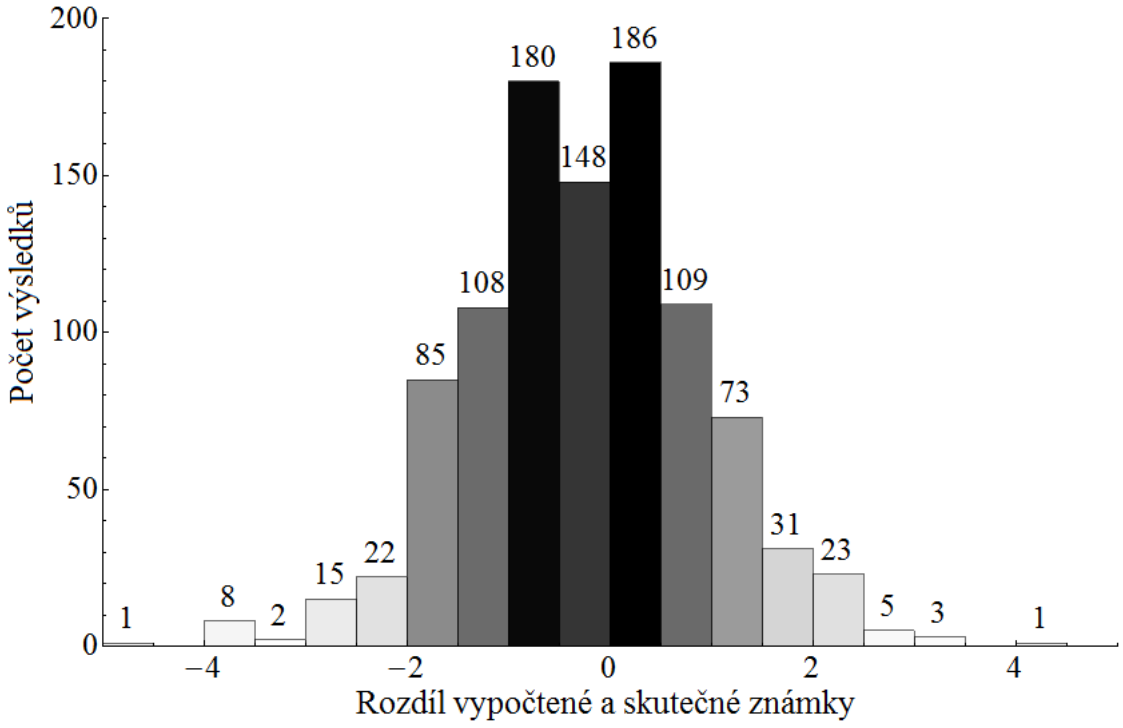
7.4.1 Přesnost doporučení

Při testování přesnosti doporučení je uživateli u z jeho množiny vymazán jeden film f. Po doběhnutí algoritmu musí výsledná množina L film f obsahovat (protože filmy, které u viděl, byly vyloučeny z počítání průměrů a na jejich místě je v průměrném hodnocení i v počtou sousedů 0).

Poté, co máme připravenou množinu L s průměry známek sousedů, jen jednoduše najdeme průměrnou známku filmu f a ta by se měla přibližně shodovat se známkou, kterou mu dal uživatel u. V množině sousedů N by měli být uživatelé, kteří mají podobnou psychologii, protože viděli podobnou množinu filmů, takže jejich průměrná známka by se měla

přibližně shodovat se známkou od uživatele u, ovšem i tak se může stát, že uživatel u ohodnotí film f známkou lišící se o více než 1 stupeň.

Test přesnosti doporučení probíhal náhodným vybráním 1000 uživatelů s neprázdnou množinou filmů a u každého byl odebrán náhodně jeden film. Průměrná známka od sousedů pak byla porovnána se známkou, kterou uživatel filmu původně dal. Histogram na obr. 7.3 ukazuje, že data rozdílů původní známky s vypočtenou napodobuje Gaussovu křivku. Průměrná známka vybraných uživatelů je 3.574, průměrná známka sousedů je 3.255. Průměrný rozdíl známek sousedů a vybraných uživatelů je -0.311, což značí, že vybraní uživatelé filmy mírně nadhodnocovali a to také odpovídá realitě, neboť i ve skutečnosti je pravděpodobnější, že uživatel dá filmu spíše známku lepší než je průměr od sousedů.



Obr. 7.3: *Graf ukazuje histogram 1000 náhodně vybraných filmů, na kterých jsem testovala, zda průměrná známka filmu odpovídá té, kterou film od uživatele skutečně dostal. Na grafu je vidět, že více než polovina výsledků (623 z 1000, tedy 62.3 %) je soustředěna kolem středu, tedy do povolené odchylky o 1 stupeň ohodnocení.*

Pro porovnání přesnosti doporučení modifikovaného k-NN s výsledky ze soutěže *Netflix prize* byl použit výpočet RMSE (popsáno v kap. 5.1.1). Výsledek výše uvedené modifikace k-NN byl 1.2042, nejlepší výsledek z soutěže byl 0.8567 (10 nejlepších výsledků je uvedeno v tab. 5.1.2). Vítězné algoritmy byly směsí několika různých metod, podložených latentními proměnnými (režisér filmu, herec apod.), které umožnily výpočet přesnějšího doporučení.

Poř.	Název týmu	RMSE
1	BellKor's Pragmatic Chaos	0.8567
2	The Ensemble	0.8567
3	Grand Prize Team	0.8582
-	má modifikace k-NN	1.2042

Tab. 7.2: *Výsledky soutěže Netflix Prize* [25].

7.4.2 Rychlost doporučení

Testování také proběhlo z hlediska porovnání rychlosti algoritmu pro 1000 různých uživatelů, probíhalo zároveň s testem doporučení. Histogram na obr. 7.3 ukazuje konkrétní časy, jaké algoritmus k doporučení potřeboval na různě velkých množinách filmů.

Histogram rychlostí 1000 náhodně vybraných uživatelů, kterým se film doporučoval, je zobrazen na obr. 7.4. Medián rychlostí algoritmu je 23.241 s, průměrný čas je docela vysoký (32.271 s), což je způsobeno hlavně tím, že ve výsledcích se vyskytují i výsledky přes 6 min (jedná se hlavně o případy s malou množinou filmů, kde bylo nalezeno v množině potenciálních sousedů N_p přes 10 000 adeptů a téměř 2/3 doby zabere načtení dat z databáze).

user_ID	poč. filmů	poč. potenc. sousedů	poč. sousedů	filmů k dopor.	čas [m:s]
65	0	-	-	10	0:14,37
907 251	12	3 085	14	9	1:04,75
6915	107	2 118	15	17	0:49,73
2 401 492	1 067	813	10	4	0:28,7
2 606 799	9 822	401	6	2	0:22,52
305 344	17 653	439	5	2	0:25,47

Tab. 7.3: *Tabulka rychlostí algoritmu k-NN v závislosti na velikosti množiny filmů. Časy jsou ovlivněny hlavně tím, že u malých množin filmů trvá dlouho porovnávání samotných sousedů a načítání dat z databáze.*

Rychlost algoritmu

V příloze B.1 je výstup z metody `Profile[]`, na které jsem otestovala, kolik času zabere algoritmus a kolik samotné nahrávání dat z databáze.

Testovaný uživatel s `user_ID` 907 251 měl ve své množině 12 filmů, algoritmus našel 3 085 potenciálních sousedů, 14 sousedů a nakonec doporučil 9 filmů ve výsledném doporučení. Celý algoritmus trval 51,187 s a samotné SQL dotazy 34,912 s, samotný algoritmus tedy trval 16,275 s, což je 31,8 % celkového času, tedy méně než třetina.

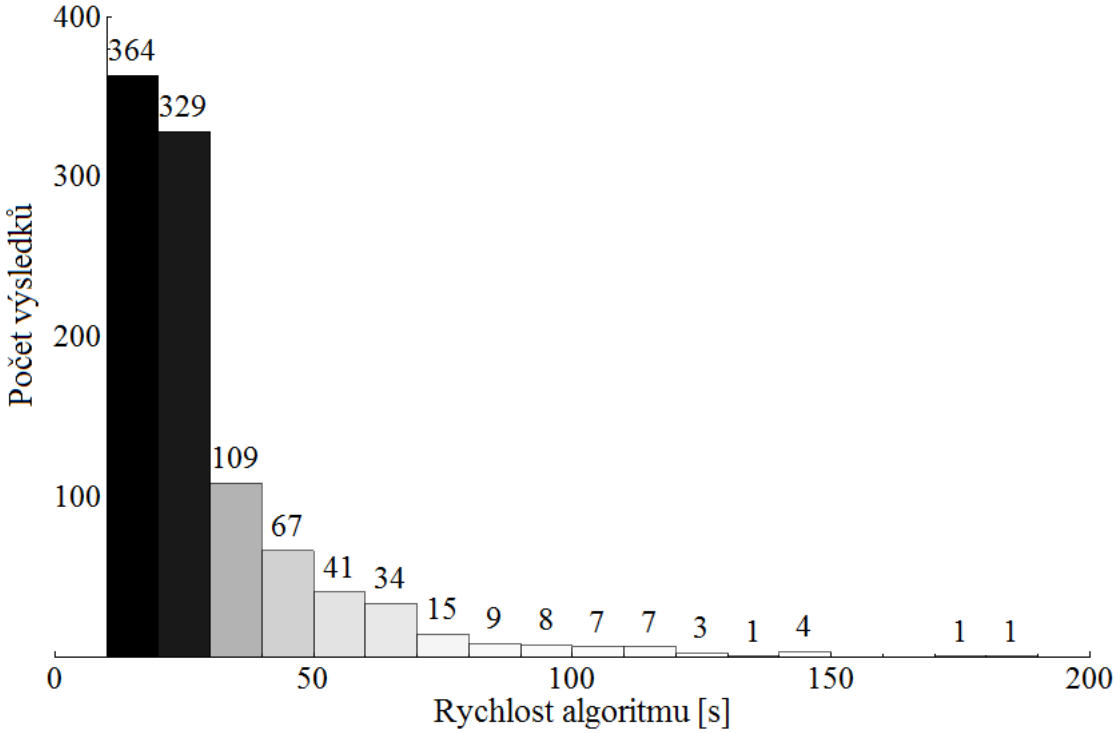
7.5 Poznámka k vypracování

Při použití knihovny *kNN* není nutné upravovat nastavení velikosti paměti *Wolfram Mathematica*, která je nastavena implicitně.

Během vývoje a testování algoritmu jsem zjistila, že existuje nelineární vztah mezi časem doporučení a kvalitou doporučení – čím přesnější doporučení chceme, tím více sousedů k jeho určení potřebujeme a tím déle trvá algoritmu jej spočítat. Zvolila jsem variantu s kvalitou doporučení, a to ze dvou hlavních důvodů:

1. Pokud jsem dbala na rychlost běhu algoritmu, v principu jsem musela dbát na to, aby bylo vybrána malá množina potenciálních sousedů N_p, ze které se pak vybrali sousedé – problém ale je, že v některých příliš malých množinách nevyhovovali žádní sousedé.

2. V dnešní době není nutné tolik dbát na dobu běhu, protože postačí algoritmus jednoduše podpořit silnějším a rychlejším strojem, který dosáhne stejných výsledků za kratší čas – ovšem kvalita doporučení je stejná. Proto jsem si dovolila dbát více na kvalitu, než na rychlost.



Obr. 7.4: *Histogram výsledků trvání algoritmu na 1000 náhodně vybraných uživatelů. V grafu je jasně vidět, že algoritmu běžel ve většině případů relativně krátce (693 z 1000 algoritmus doběhl do 25 s, 869 z 1000 algoritmus doběhl do 50 s). Době běhu je ovlivněna hlavně velikostí množiny potenciálních sousedů.*

7.6 Modifikace algoritmu

První modifikace je popsána výše, v první verzi kódu. U uživatele bez hodnocení jde zde hlavně o myšlenku, jak rozdělit pravděpodobnosti vybrání filmů mezi jednotlivé tak, aby pravděpodobnost odpovídala počtu jejich výskytů v databázi a také známce jejich ohodnocení.

Co se týče uživatele s hodnocením, zde je vyřešen problém, jak hned na začátku vybrat ty uživatele, se kterými budeme pracovat a zbytečně tak neporovnávat uživatele u s každým uživatelem v databázi, což na datech z *Netflix prize* sníží počet velikost množiny potenciálních sousedů z několika milionů na několik set. Další modifikaci tvoří funkce k redukci „nedůležitých“ filmů.

A jednou z dalších modifikací je funkce téměř na konci algoritmu, která umožňuje konečný výsledek ovlivnit minimální známkou ohodnocení, na základě které chceme filmy doporučovat.

7.6.1 Další možnosti rozšíření

Další modifikace by mohla probíhat tak, že filmy budou váženy i rokem natočení. V dnešní době je víceméně pravděpodobné, že diváka zaujme nějaký trhák, který se zrovna vysílá v kinech, než film, který byl natočen dříve, než se narodil.

Já ovšem tyto latentní vlastnosti k dipozici nemám. Pokud by bylo možné použít takovýchto dodatečných dat, bylo by možné doporučit film podle režiséra, herce, skladatele soundtracku apod. Tato myšlenka je vlastně velice podobná tomu, co využívaly týmy *BellKor* a *Ensemble* ve svých algoritmech v soutěži. Snažila jsem se ovšem použití těchto vlastností vyhnout, neboť většinou při použití tohoto algoritmu jde hlavně o to použít data, která jsou k dispozici, aniž bychom trávili čas nad tím, že bychom je nějak upravovali, a při použití těchto vlastností by byl algoritmus příliš vázán na konkrétní typ dat.

7.7 Závěr

Algoritmus k-NN jsem navrhla se zaměřením na rychlost a přesnost doporučení. Protože jsou tyto dva požadavky v protikladu, zvolila jsem kompromis mezi oběma parametry. Zrychlení algoritmu je dosaženo redukcí vstupní množiny potenciálních sousedů vybraných na základě nejméně ohodnoceného filmu uživatele. Tato redukce však nesmí být příliš velká, aby byla zachována dostatečná přesnost konečného doporučení. I přes tuto počáteční redukci dat algoritmus najde v 99 % případech dostatečný počet potenciálních sousedů, ze kterých lze poté vybrat ty nejvíce vyhovující pro relevantní doporučení. Minimální množinu potenciálních sousedů lze ovlivnit parametrem, kterým můžeme ovlivnit rychlost nebo přesnost doporučení. Tuto redukci je nutné provést, protože složitost k-NN algoritmu

při doporučneí jednomu uživateli je O(n), kde n je počet všech uživatelů v databázi. Po redukci je složitost O(k), kde k je počet potenciálních sousedů.

Např. jak je uvedeno v příloze B.2, v datech ze soutěže *Netflix prize* je 480 189 hodnotících uživatelů. V případě uživatele s ID 1 129 382 neporovnával algoritmus všech 480 189 uživatelů, ale pouze 639 potenciálních sousedů. Z nich 7 sousedů se svým výběrem filmů shodovalo a dohromady měli 140 filmů, které uživatel s ID 1 129 382 neviděl a ze kterých je možné vybrat jeden k doporučení.

Jako součást algoritmu, resp. jeho druhé části, jsem navrhla i algoritmus k doporučení pro uživatele, který žádný film ještě neviděl. Zde film vybírám z množiny několika nejčastěji hodnocených filmů. Pravděpodobnost doporučení filmu se pak odráží od počtu ohodnocení a jednotlivých známek. Cílem je častěji doporučit tzv. bestsellery.

Co se týče testování z hlediska doporučení, otestovala jsem přesnost na množině 1 000 uživatelů, kde byl každému odebrán jeden film, vypočteno doporučení a otestována známka, která byla odebranému filmu v doporučení udělena. Známka se nelišila o víc než 1 stupeň v 62.3 % případů. Pro kontrolu přesnosti doporučení jsem také vypočítala RMSE stejně jako v soutěži *Netflix prize*. Výsledek mé modifikace (tj. rozdíl rozdíl mezi skutečnou a vypočtenou známkou) byl 1.204, nejlepší výsledek v soutěži dosáhl hodnoty 0.857.

Zároveň s kvalitou doporučení jsem algoritmus otestovala i z hlediska rychlosti doporučení. Čas je z větší části závislý na tom, jak velkou množinu potenciálních sousedů algoritmus porovnává. Pak také záleží na tom, jak velké množiny filmů navíc sousedé mají, když algoritmus určuje, které filmy jsou k doporučení vhodné. Doporučení filmu trvalo max. 25 s v 69.3 % případů. Tato hodnota je docela vysoká, ovšem je potřeba si uvědomit, že většinu času zabere práce s databází. Medián potřebného času k doporučení 23,241 s. Dvě třetiny času však zabere získávání dat z databáze, medián běhu algoritmu samotného se tedy pohybuje kolem 7,5 s.

Kapitola 8

Závěr

V této práci shrnuji nejznámější a nejpoužívanější metody pro doporučování obsahu. Podařilo se mi zde vytvořit shrnutí těchto technik, na základě kterých si čtenář udělá představu o tom, co je doporučování obsahu a dvě nejznámější metody - *Content-based recommendations* a *Collaborative filtering*, a algoritmy, které se při jejich aplikaci v praxi používají. Toto shrnutí může posloužit i čtenáři, který chce principielně nahlédnout do těchto technik, neboť jejich popis není v české literatuře moc častý.

Na základě pokynů vedoucího práce jsem nastudovala problematiku algoritmu k nejbližších sousedů (k-Nearest Neighbours) a navrhla jsem a implementovala modifikaci tohoto algoritmu se zaměřením na rychlost a přesnost doporučení. Zrychlení algoritmu je dosaženo redukcí vstupní množiny potenciálních sousedů vybraných na základě nejméně ohodnoceného filmu uživatele. Díky této redukci algoritmus vyloučí uživatele, u kterých je zřejmé, že nemohou být skutečnými sousedy. Podařilo se mi tedy časovou složitost algoritmu zlepšit z $O(n)$ na $O(k)$, kde n je počet všech uživatelů a k počet potenciálních sousedů. Díky této redukci je také algoritmus méně paměťově náročný.

Algoritmus je implementován ve *Wolfram Mathematica* a realizován na datech ze soutěže *Netflix prize*. Tato data obsahovala 480 189 hodnotících uživatelů, kteří dohromady ohodnotili 100 480 507 filmů. Na takto velké množině záznamů dosáhl můj modifikovaný algoritmus medián potřebného času k doporučení 23,241 s. Dvě třetiny času však zabere získávání dat z databáze, medián běhu algoritmu samotného se tedy pohybuje kolem 7,5 s.

Co se týče přesnosti doporučení algoritmu, spočítala jsem RMSE stejně, jako jej počítali řešitelé *Netflix Prize* (porovnání skutečné a vypočtené známky filmu, který je doporučován). Výsledek nejlepšího týmu *BellKor*, jehož algoritmus byl směsí několika různých metod, byl 0,856, výsledek přesnosti doporučení mé modifikace dosáhl skóre 1,204.

Zdrojové kódy mohou být využity jako případná knihovna pro *Wolfram Mathematica*, kde jsem nenalezla žádné implementace *Recommender Systems* pomocí k-NN.

Literatura

[1] SAMMUT C., G. WEBB (ed.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30768-8, © Springer-Verlag Berlin Heidelberg, 2010, kap. 338, s 1-9. [cit. 2014-01-04]. Dostupné z: http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf

[2] TERVEEN, Loren a Will HILL. AT&T Labs - Research. Beyond Recommender Systems: Helping People Help Each Other. *HCI In The New Millennium* [online]. Addison-Wesley, 2001, s.1-21. [cit. 2013-12-10]. Dostupné z: http://files.grouplens.org/papers/rec-sys-overview.pdf

[3] LIU, Bing. *Web data mining: exploring hyperlinks, contents, and usage data*. 2nd ed. New York: Springer, c2011, xx, 622 p. ISBN 978-364-2194-597

[4] BURKE, Robin. *Hybrid Recommender Systems: Survey and Experiments* [online]. S. 1-26 [cit. 2014-01-16]. Dostupné z: http://josquin.cs.depaul.edu/~rburke/pubs/burke-umuai02.pdf

[5] Collaborative filtering [obrázek]. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2014-01-04]. Dostupné z: http://upload.wikimedia.org/wikipedia/commons/5/52/Collaborative_filtering.gif

[6] SCHAFER, J. Ben, Dan FRANKOWSKI, Jon HERLOCKER a Sen SHILAD. *Collaborative Filtering Recommender Systems*. The Adaptive Web [online]. Berlin Heidelberg, 2007, s. 298-321 [cit. 2013-11-25]. Dostupné z: http://138.100.152.2/~jbobi/jbobi/PapersRS/CollaborativeFilteringRecommenderSystems.pdf

[7] SU, Xiaoyuan a Khoshgoftaar TAGHI M. A Survey of Collaborative Filtering Techniques. *Advanced in Artificial Intelligence* [online]. 2009 [cit. 2014-11-13]. Dostupné z: http://www.hindawi.com/journals/aai/2009/421425/

[8] GOLDBERG, Ken, Theresa ROEDER, Dhruv GUPTA a Chris PERKINS. *Eigentaste: A Constant Time Collaborative Filtering Algorithm* [online]. 2000 [cit. 2013-11-25]. Dostupné z: https://www.seas.harvard.edu/courses/cs281/papers/goldberg-roeder-gupta-perkins-2001.pdf

[9] TAN, Pang-Ning, Michael STEINBACH a Vipin KUMAR. Introduction to data mining [online]. 1st ed. Harlow: Pearson Education, 2013, s. 327-414 [cit. 2014-03-04]. Pearson new international edition. ISBN 978-1-29202-615-2. Dostupné z: http://www-users.cs.umn.edu/~kumar/dmbook/ch6.pdf

[10] TAN, STEIBACH a KUMAR. *Assoc. rule mining: Apriori.* [cit. 2014-12-05]. Dostupné z: http://www.cse.cuhk.edu.hk/~taoyf/course/cmsc5724/notes/asso-apriori.pdf

[11] BORGELT, Christian a Rudolf KRUSE. Induction of Association Rules: Apriori Implementation. [online]. [cit. 2014-03-05]. Dostupné z: http://www.quretec.com/u/vilo/edu/2003-04/DM_seminar_2003_II/ver1/P01/articles/induction-of-association-rules.pdf

[12] KOREN, Yehuda, Robert BELL a Chris VOLINSKY. Matrix factorization techniques for recommender systems. *IEEE Computed Society* [online]. 2009 [cit. 2014-01-05]. Dostupné z: http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf

[13] FUNK, Simon. Netflix Update: Try This at Home. Project Sifter [online]. 2006 [cit. 2014-01-12]. Dostupné z: http://sifter.org/~simon/journal/20061211.html

[14] BELL, Robert M., Yehuda KOREN a Chris VOLINSKY. *All Together Now: A Perspective on the Netflix Prize* [obrázek]. London: Whitechapel, 2010, č. 23 [cit. 2014-01-12]. Dostupné z: http://www2.research.att.com/~volinsky/papers/chance.pdf

[15] PAZZANI, Michael J. a Daniel BILLSUS. *The Adaptive web.* Content-Based Recommendation Systems [online]. Berlin: Springer-Verlag, 2007, s. 325-341 [cit. 2014-01-20]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.130.8327&rep=rep1&type=pdf

[16] HURWICZ, Leonid. *Decision Trees* [online]. [cit. 2014-01-20]. Dostupné z: http://vserver1.cscs.lsa.umich.edu/~spage/ONLINECOURSE/R4Decision.pdf

[17] SUTTON, Oliver. *Introduction to k Nearest Neighbour Classification and condesed nearest neigbour data reduction* [online]. 2012 [cit. 2014-01-25]. Dostupné z: http://www.math.le.ac.uk/people/ag153/homepage/KNN/OliverKNN_Talk.pdf

[18] UDEN VAN, Mark. *Rocchio: Relevance Feedback in Learning Classification Algorithms* [online]. [cit. 2014-01-25]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3960&rep=rep1&type=pdf

[19] MITCHELL, Tom T. Generative and dicriminative classifiers: Naive Bayes and logistic regression. *Machine learning* [online]. 2010 [cit. 2014-01-28]. Dostupné z: http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

[20] The Netflix Prize Rules. *Netflix Prize* [online]. 2006 [cit. 2014-01-29]. Dostupné z: http://www.netflixprize.com//assets/rules.pdf

[21] *Netflix: Ratings* [obrázek]. [cit. 2014-01-29]. Dostupné z: http://www.solarsquirrel.com/MIDS/Queue.jpg

[22] *Netflix.com* [online]. [cit. 2014-02-02]. Dostupné z: http://www.netflixprize.com/

[23] *Netflix: Recommendations* [obrázek]. [cit. 2014-01-04]. Dostupné z: http://www.solarsquirrel.com/MIDS/Recommendations.jpg

[24] BARNSTON, Anthony G. Correspondence among the Correlation, RMSE, and Heidke Forecast Verification Measures; Refinement of the Heidke Score. *Notes and Correspondence* [online]. Washington D.C., 1992 [cit. 2014-02-01]. Dostupné z: http://www.nssl.noaa.gov/users/brooks/public_html/feda/papers/barnston92.pdf

[25] Netflix Prize. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2014-01-04]. Dostupné z: http://en.wikipedia.org/wiki/Netflix_prize

[26] WILSON, Tracy V a Stephanie CRAFWORD. *How Netflix works* [online]. 2007 [cit. 2014-01-04]. Dostupné z: http://electronics.howstuffworks.com/netflix2.htm

[27] AMATRIAIN, Xavier a Justin BASILICO. *Netflix Recommendations: Beyond the 5 stars (Part 2)* [online]. 2012 [cit. 2014-01-04]. Dostupné z: http://techblog.netflix.com/2012/06/netflix-recommendations-beyond-5-stars.html

[28] BELL, Robert M. a Yehuda KOREN. *Improved Neighborhood-based Collaborative Filtering* [online]. 2007 [cit. 2014-01-09]. Dostupné z: http://public.research.att.com/~volinsky/netflix/cfworkshop.pdf

[29] BELL, Robert M., Yehuda KOREN a Chris VOLINSKY. *The BellKor solution to the Netflix Prize* [online]. 2007 [cit. 2014-01-09]. Dostupné z: http://www2.research.att.com/~volinsky/netflix/ProgressPrize2007BellKorSolution.pdf

[30] BELL, Robert M. a Yehuda KOREN. *Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights* [online]. 2007 [cit. 2014-01-09]. Dostupné z: http://public.research.att.com/~volinsky/netflix/BellKorICDM07.pdf

[31] BELL, Robert M., Yehuda KOREN a Chris VOLINSKY. *Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems* [online]. 2007 [cit. 2014-01-09]. Dostupné z: http://public.research.att.com/~volinsky/netflix/NetflixKDD07.pdf

[32] SILL, Joseph, Gabor TAKACS, Lester MACKEY a David LIN. *Feature-Weighted Linear Stacking* [online]. 2009 [cit. 2014-01-04]. Dostupné z: http://arxiv.org/pdf/0911.0460.pdf

[33] WOLFRAM RESEARCH, INC. *DatabaseLink user guide* [online]. USA, 2008 [cit. 2014-01-17]. Dostupné z: http://www.wolfram.com/learningcenter/tutorialcollection/DatabaseLinkUserGuide/DatabaseLinkUserGuide.pdf

[34] ZHU, Jerry. *K Nearest Neighbor demo* [online]. 2000 [cit. 2014-02-05]. Dostupné z: http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html

[35] BATISTA, Gustavo E.A.P.A. a D.F. SILVA. Knn.java. *How k-Nearest Neighbor Parameters Affect its Performance, Proceedings of the Argentine Symposium on Artificial Intelligence* [online]. 2009 [cit. 2014-04-21]. Dostupné z: http://www.icmc.usp.br/pessoas/gbatista/knn/

[36] PAREDES, Yancy Vance. *K-Nearest Neighbor classification* [online]. 2013 [cit. 2014-02-06]. Dostupné z: http://learn.yancyparedes.net/2013/09/k-nearest-neighbor-classification/

[37] STITES, David. *K Nearest Neighbors in Java* [online]. 2010 [cit. 2014-02-08]. Dostupné z: http://afewguyscoding.com/2010/05/nearest-neighbors-java/

[38] GARCIA, Vincent, Eric DEBREUVE a Michel BARLAUD. *KNN CUDA: Fast k nearest neighbor search using GPU* [online]. 2012 [cit. 2014-01-09]. Dostupné z: http://vincentfpgarcia.github.io/kNN-CUDA/

[39] DEOKAR, S. *Weighted K nearest neighbor* [online]. 2009 [cit. 2014-04-21]. Dostupné z: http://www.d.umn.edu/~deoka001/downloads/K_Nearest_Neighbor_Algorithm.pdf

[40] Kolektiv autorů. An implementation of the k-NN algorithm on STI's Cell proc. *Cell-knn* [online]. 2010 [cit. 2014-02-10]. Dostupné z: https://code.google.com/p/cell-knn/

[41] PAN, Jia a Dinesh MANOCHA. *Fast GPU-based Locality Sensitive Hashing for K-Nearest Neighbor Computation* [online]. 2012 [cit. 2014-02-12]. Dostupné z: http://gamma.cs.unc.edu/KNN/

[42] Knn.m. *Pattern recognition package for Mathematica* [online]. 2011 [cit. 2014-02-12]. Dostupné z: https://code.google.com/p/prpackage/source/browse/t/draft/classify/knn.m

[43] *Machine Learning Demos in Mathematica* [online]. Johannes Kepler University Linz, 2012 [cit. 2014-02-15]. Dostupné z: http://www.bioinf.jku.at/software/ML-Math/

Příloha A

Příloha CD

Zde je uveden seznam všech příloh přidaného CD, popsaného po jednotlivých složkách.

BP Zdrojové soubory textů této bakalářské práce ve formátu $*.tex$.

data Data ze soutěže Netflix Prize, popsané v kapitole 6.2:

- ***movies***: soubor s názvy filmů, ID a rokem natočení,
- **movie_titles_insert.txt**: soubor s připraveným SQL příkazem `insert`, umožňující vložit data do tabulky filmů (ID, název filmu a rok natočení),
- **probe.txt**: soubor s filmy a uživateli, kteří je viděli (výběr filmů 1-9999),
- **qualifying.txt**: soubor s daty pro soutěž – film a jednotliví uživatelé, pro které bylo v soutěži počítána známka filmu,
- **README**: licence z webové stránky *Netflix prize*, odkud pochází data,
- **training_set.zip**: zazipovaný soubor se všemi daty ohodnocení.

java Zdrojové kódy pro nahrání data ze soutěže do databáze:

- **\src\bp_netflix\BP_netflix.java**: hlavní třída obsahující metodu `main` pro spuštění nahrávání dat do databáze,
- **\src\bp_netflix\Database.java**: třída obsahující připojení k databázi, čtení souborů s filmy a jejich ukládání do databáze,
- **\src\bp_netflix\MyDirectory.java**: třída sloužící k načítání souborů.

mathematica Zdrojové kódy praktické části:

- ***knn mathematica***: pomocné soubory s vykreslenými grafy:
  - **KNN_database_data_plotting.nb**: soubor s ukázkou připojení k databázi a vykreslením dat,
  - **KNN_koeficient_function.nb**: soubor s vykreslením grafu funkcí pro redukci dat,
  - **tabulka_koeficientu.xlsx**: tabulky výsledků různých funkcí při hledání funkce pro redukci dat.
- ***knn test results***: soubory s výsledky testů:
  - **KNN_profile.nb**: výstup z metody `Profile[]`,
  - **KNN_resultsQualityRecom.nb**: soubor s histogramy výstupů z testů,
  - **KNN_RMSE.nb**: spočítání RMSE pro otestování přesnosti doporučení,
  - **results.txt**: výsledky testování přesnosti doporučení ve tvaru {*ID testovaného filmu, ID uživatele, známka testovaného filmu, vypočítaný známka doporučovaného filmu, počet filmů uživatele, počet nalezených sousedů*},
  - **times.txt**: výsledky testování z hlediska rychlosti ve tvaru {*celkový čas, čas pro přípravu množiny filmů, čas potřebný pro selekci sousedů, čas potřebný pro konečné doporučení*}.
- ***wolfram workbench\CollaborativeFiltering***: zdrojové soubory modifikovaného kNN algoritmu.

zadaniBP.pdf Oficiální zadání této bakalářské práce

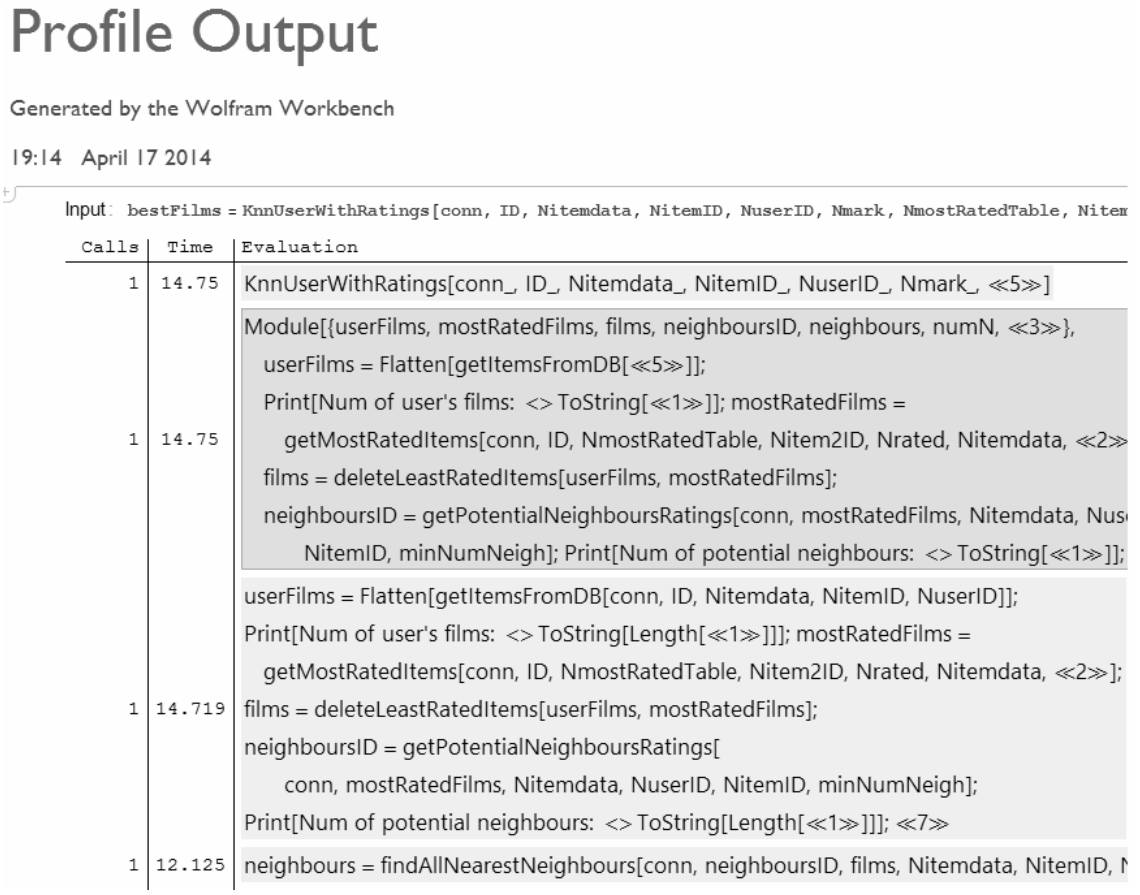
BP_Kubelova_5_2014.pdf Text bakalářské práce

Příloha B

Praktická část - přílohy

B.1 k-NN algoritmus - `Profile[]` debugging

Zde je zobrazen výstup metody `Profile[]`. Jak již bylo řečeno v 7.4.2, čas potřebný k získání dat z databáze zabere přibližně dvě třetiny času algoritmu.



Profile Output

Generated by the Wolfram Workbench

19:14 April 17 2014

Input: `bestFilms = KnnUserWithRatings[conn, ID, Nitemdata, NitemID, NuserID, Nmark, NmostRatedTable, Nitem`

Calls	Time	Evaluation
1	14.75	KnnUserWithRatings[conn_, ID_, Nitemdata_, NitemID_, NuserID_, Nmark_, ≪5≫]
1	14.75	Module[{userFilms, mostRatedFilms, films, neighboursID, neighbours, numN, ≪3≫}, userFilms = Flatten[getItemsFromDB[≪5≫]]; Print[Num of user's films: <> ToString[≪1≫]]; mostRatedFilms = getMostRatedItems[conn, ID, NmostRatedTable, Nitem2ID, Nrated, Nitemdata, ≪2≫ films = deleteLeastRatedItems[userFilms, mostRatedFilms]; neighboursID = getPotentialNeighboursRatings[conn, mostRatedFilms, Nitemdata, Nus NitemID, minNumNeigh]; Print[Num of potential neighbours: <> ToString[≪1≫]];
1	14.719	userFilms = Flatten[getItemsFromDB[conn, ID, Nitemdata, NitemID, NuserID]]; Print[Num of user's films: <> ToString[Length[≪1≫]]]; mostRatedFilms = getMostRatedItems[conn, ID, NmostRatedTable, Nitem2ID, Nrated, Nitemdata, ≪2≫]; films = deleteLeastRatedItems[userFilms, mostRatedFilms]; neighboursID = getPotentialNeighboursRatings[conn, mostRatedFilms, Nitemdata, NuserID, NitemID, minNumNeigh]; Print[Num of potential neighbours: <> ToString[Length[≪1≫]]]; ≪7≫
1	12.125	neighbours = findAllNearestNeighbours[conn, neighboursID, films, Nitemdata, NitemID, N

Obr. B.1: *Na obrázku je výstup z metody* Profile[]. *Tato metoda měří, jak dlouho trvá každá metoda, která se v ní volá a výsledky seřadí sestupně. Je to velice užitečný nástroj při psaní rychlých a úsporných kódů ve Wolfram Mathematica.*

B.2 k-NN algoritmus - Výpis testování

Výpis při testování kvality a rychlosti doporučení

```
1  12 USER ID   : 1129382
2  Num of user's films: 418
3  deleted filmID: 5601
4  Num of deleted films: 23
5  Num of potential neighbours: 639
6  Neighbours (7): {1314869, 2439493, 305344, 1129382, 387418, 2118461, 716173}
7  Num of films for recommendaion: 140
8  {5601,7,3.}
9  neighbours rating: 3.
10 {1129382,5601,3}
11 user       rating: 3
12 TOTAL TIME of recommendation:        23.8599094
13 Time user's films preparing:         0.4052694
14 Select potential and real neigbours: 22.5460303
15 Time of final recommnedation:        0.9086097
```

Popis výpisu kódu

1: ID testovaného uživatele u

2: velikost množiny F_u - počet filmů uživatele u

3: ID testovaného filmu f

4: počet filmů, o které byla množina filmů uživatele u redukována

$$(418^{1,4})/200 = 22,44$$

5: velikost množiny N_p - počet potenciálních sousedů uživatele u

6: množina N - sousedé uživatele u

7: počet filmů k doporučení (s průměrnou známkou od uživatelů z $N \geq 4.0$)

8: {ID filmu f, počet sousedů, průměrná známka sousedů}

9: vypočtená známka filmu f od sousedů uživatele u

10: {Id uživatele u, ID filmu f, známka filmu f od uživatele u}

11: známka, kterou uživatel u dal filmu f

12: celkový čas potřebný k doporučení

13: čas potřebný k redukci množiny F_u

14: čas potřebný k vybrání potenciálních a skutečných sousedů

15: čas potřebný ke konečnému doporučení