Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Doporučovací systém pro filmovou databázi

Plzeň 2015 Matěj Lochman

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2015

Matěj Lochman

Abstract

Recommendation system for movie database

This bachelor thesis explores the technologies used in recommender systems and shows how to implement such system. A framework called Apache Mahout was used to implement the system. User-based filtering algorithm was used due to the character of the input data. According to the measured evaluations, the method to compute similarity was the Euclidean distance with a neighbourhood based on a threshold of value 0.5. A web application was developed to enable simple usage of the system. Building on the contents of this thesis, it is possible to create a similar recommender system.

Abstrakt

Tato práce zkoumá technologie užívané v doporučovacích systémech a zabývá se implementací tohoto systému. K implementaci doporučovacího systému bylo využito knihovny Apache Mahout. Vzhledem k povaze vstupních dat byl k realizaci systému využit algoritmus user-based filtering. Na základě výsledků provedených experimentů byla z několika metod a parametrů vybrána metoda Euklidovské vzdálenosti s velikostí sousedstva danou prahem o hodnotě 0.5. V rámci práce byla vytvořena webová aplikace, která usnadňuje vyzkoušení systému. Na základě této práce je možné sestavit podobný doporučovací systém.

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Michalu Konkolovi za jeho ochotu, cenné rady a věcné připomínky při vedení práce.

Poděkování MetaCentru

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations"(LM2010005), is greatly appreciated.

Obsah

1 Úvod

V současné době roste množství dat vyskytujících se na internetu enormní rychlostí. Například na webové stránky YouTube.com je každou minutu nahráno 300 hodin videa[1]. Kromě videí se jedná také o knihy, hudbu, články nebo jiné produkty. Při tomto zahlcování informacemi může být z pohledu uživatele složité se v takovémto množství dat orientovat. Doporučovací systémy se cíleně snaží z množství produktů vybrat ty, které by se uživateli mohly líbit.

Doporučovací systémy patří mezi technologie zabývající se tříděním informací. Používají se zejména v rozsáhlých databázích a jsou založeny na shromažď'ování a analýze velkého množství dat. Využívají metod, pomocí kterých se snaží předpovědět vztah uživatele k nějakému předmětu. Sledují chování, aktivity a preference jednotlivých uživatelů a na základě těchto informací se pokouší předurčit, co by se jim mohlo líbit.

Cílem této práce je prozkoumat technologie užívané v doporučovacích systémech, navrhnout takový systém pro filmovou databázi a následně ho implementovat.

[1] https://www.youtube.com/yt/press/statistics.html

2 Technologie v doporučovacích systémech

Studie doporučovacích systémů je ve srovnání s jinými studiemi zabývajícími se tříděním informací (jako jsou například vyhledávače) poměrně nová. Výzkum doporučovacích systémů se datuje od 90. let [4].

Za posledních několik let získaly tyto systémy na popularitě a jsou dnes používány v několika známých internetových stránkách. Jsou jimi například Amazon, YouTube, Last.fm, Netflix nebo také sociální sítě jako Facebook či Twitter.

V této kapitole budou probrány algoritmy doporučovacích systémů. Dále budou zmíněny možnosti jejich evaluace, způsoby počítání sousedstva a také metody pro počítání podobnosti uživatelů, respektive předmětů. Nakonec budou uvedeny nástroje s jejichž pomocí je možné takovýto systém setrojit.

2.1 Algoritmy doporučovacích systémů

Doporučovací systémy je možné podle [4] rozdělit do šesti základních skupin podle toho, jakým způsobem doporučují předměty jednotlivým uživatelům. Často jsou však uváděny pouze první dva typy nebo jejich kombinace.

2.1.1 Collaborative filtering

Jednou z možných metod je collaborative filtering. Hlavní výhodou této metody je, že dokáže doporučovat vhodné předměty, aniž by o nich byly známy jakékoli informace. Pracuje pouze s uživateli a jejich vztahy k daným předmětům, jejich atributy ji nezajímají.

Pro měření podobnosti uživatelů nebo předmětů lze využít několik algoritmů, které budou zmíněny později. Tato metoda se dále dělí mezi dva následující typy user-based filtering a item-based filtering.

User-based filtering

User-based filtering je založen na podobnosti uživatelů. Předpokládá, že když má uživatel u na nějaký předmět stejný názor jako uživatel v, pak je pravděpodobnější, že uživatel u bude mít na jiný předmět i stejný názor jako uživatel v, než jakýkoli jiný náhodně zvolený uživatel [1].

Tento algoritmus (Alg. 1) nejprve projde všechny předměty, pro které uživatel u nevyjádřil žádný vztah. Pak hledá jakékoli jiné uživatele, kteří vyjádřili preferenci k tomuto předmětu. Hodnotě této preference je přidána váha na základě podobností uživatelů s. Čím více jsou si uživatelé podobní, tím je váha preference větší a naopak. Celková hodnota je přidána do váženého průměru. Nakonec je vrácen seznam předmětů s těmito průměry.

for $\forall i \nexists preference(u,i), i \in I, u \in U$ **do**
  for $\forall v(v \neq u) \exists preference(v,i), v \in U$ **do**
    $s = podobnost(u,v)$;
    $suma \mathrel{+}= s * preference(v,i)$;
    $suma\ s \mathrel{+}= s$;
  end
  $preference(u,i) = \frac{suma}{suma\ s}$;
end

Algoritmus 1: Pseudokód user-based algoritmu

Množina U je množina všech uživatelů, zatímco I je množina všech předmětů. Tento postup je pouze základní myšlenkou, reálně by však bylo časově velmi náročné procházet všechny předměty v databázi.

Zlepšení lze dosáhnout tak, že algoritmus nejprve projde všechny uživatele a vytvoří z nich sousedstvo nejbližších uživatelů nazvané N, pak jsou

procházeny pouze předměty, které jsou těmto uživatelům známé (viz Alg. 2).

for $\forall w(w \neq u), w \in U, u \in U$ **do**
  $s = podobnost(u, w)$;
end
$N = \{$množina uživatelů s nejvyšší podobností $s\}$;
for $\forall i \exists preference(w, i) \nexists preference(u, i), i \in I, u \in U, w \in N$ **do**
  for $\forall v(v \neq u)(v \in N) \exists preference(v, i), v \in U$ **do**
    $s = podobnost(u, v)$;
    $suma \mathrel{+}= s * preference(v, i)$;
    $suma\ s \mathrel{+}= s$;
  end
  $preference(u, i) = \frac{suma}{suma\ s}$;
end

Algoritmus 2: Pseudokód user-based algoritmu se sousedstvem

Jedná se o jednu z nejjednodušších a také nejrozšířenějších metod využívaných v doporučovacích systémech [2]. Je vhodné využívat tohoto přístupu v případě, že počet uživatelů je menší než počet předmětů, které jsou doporučovány. Důvodem je složitost výpočtu sousedstva, která v tomto případě závisí právě na počtu uživatelů.

Item-based filtering

Oproti user-based algoritmu, ve kterém se doporučuje na základě podobnosti uživatele s uživatelem, se v item-based algoritmu doporučuje podle vztahu předmětu s předmětem. Předpokládá se, že pokud uživatel nějak ohodnotí předmět i, bude nejspíše hodnotit podobný předmět j stejným způsobem.

Při doporučování jsou nejdříve nalezeny předměty, které uživatel u preferuje, a pak nejvíce podobné předměty, které uživatel u ještě nehodnotil, a ty jsou doporučeny.

Algoritmus (Alg. 3) se velmi podobá předchozímu algoritmu. Rozdíl spo-

čívá pouze v tom, že se váha preference odvíjí od podobnosti dvou předmětů.

for $\forall i \nexists preference(u,i), i \in I, u \in U$ **do**
  for $\forall j \exists preference(u,j), j \in I$ **do**
    $s = podobnost(i,j)$;
    $suma \mathrel{+}= s * preference(u,j)$;
    $suma\ s \mathrel{+}= s$;
  end
  $preference(u,i) = \frac{suma}{suma\ s}$;
end

Algoritmus 3: Pseudokód item-based algoritmu

Výhodou je, že na rozdíl od user-based algoritmu, ve kterém se vztah uživatele k jinému uživateli mohou měnit s průběhem času, se vztahy mezi předměty v item-based algoritmu s časem pravděpodobně měnit nebudou.

Například je možné, že názor uživatele na některé filmy se po určité době změní, nebo si zalíbí nový filmový žánr. Tím samozřejmě může být ovlivněno i sousesdsvto jeho nejbližších uživatelů. Vztah mezi dvěma filmy se však s uplývajícím časem spíše měnit nebude. Aby se tento vztah změnil, musela by většina uživatelů, která hodnotila tento film změnit svůj názor.

Z toho plyne, že sousedstvo předmětů v item-based systému lze předem vypočítat a uložit. Při doporučování předmětů už není nutné toto sousedstvo počítat, čímž dojde k urychlení doporučení [3].

Počítání sousedstva je závislé na počtu předmětů. Proto je vhodné tento algoritmus využívat v případě, že je počet předmětů menší než počet uživatelů v databázi.

Výhody

Mezi výhody collaborative filtering patří nezávislost na vlastnostech předmětů, které systém doporučuje. Pro návrh doporučení jednomu uživateli je zapotřebí pouze hodnocení od jiných uživatelů.

Další výhodou je poměrně snadná implementace těchto systémů.

Nevýhody

Jednou z hlavní nevýhod tohoto přístupu je závislost na velkém množství vstupních dat. Při nedostatku dat mohou být doporučení nepřesná nebo se vůbec nedají vypočítat. Tento problém se nazývá cold-start problem. Aby uživatel mohl dostávat dobrá doporučení, musí nejdříve ohodnotit nějaké množství předmětů.

Další slabá stránka poněkud souvisí s předchozí nevýhodou. Systém není schopen doporučit předmět, který ještě nikdo neohodnotil. Doporučení je totiž navrženo na základě oblíbených předmětů podobných uživatelů, není tedy možné aby byl navržen předmět, který nikdo neohodnotil.

2.1.2 Content-based filtering

Oproti collaborative filtering se tato metoda opírá hlavně o atributy předmětů, které doporučuje. Je tedy nutné znát vlastnosti jednotlivých předmětů a také preference uživatelů.

Pro popis předmětů se využívají klíčová slova, která reprezentují jejich konkrétní vlastnosti. Každý předmět i má vektor atributů $\mathbf{x}_i$. Data o tom, jaké vlastnosti předmětů uživatel u preferuje, jsou ukládána do jeho uživatelského profilu. Tato data jsou reprezentována váženým vektorem $\mathbf{y}_u$. Tento vektor je vypočten na základě atributů předmětů, které uživatel u ohodnotil. Jednotlivé váhy atributů indikují jejich důležitost. Následující vzorec popisuje změnu uživatelského vektoru $\mathbf{y}_u$ při ohodnocení nového předmětu x_i. Přičemž množina I_u reprezentuje předměty, které uživatel u ohodnotil a veličina r_{ui} zastává hodnocení předmětu i uživatelem u [4].

$$\mathbf{y}_u = \sum_{i \in I_u} r_{ui} \mathbf{x}_i$$

Při doporučování jsou tyto preference uživatele následně porovnány s atributy předmětů a pomocí jedné z metod počítání podobnosti (kapitola 2.4) je vyhodnoceno, do jaké míry by se mohl tento předmět uživateli líbit.

Příkladem může být uživatel, který ve svém profilu uvede, že má rád country hudbu. Doporučovací systém využívající content-based filtering tedy bude raději doporučovat hudbu z tohoto žánru, než hudbu, která spadá pod žánr, ke kterému se uživatel nevyjádřil.

Výhody

Jednou z výhod content-based filtering je nezávislost na ostatních uživatelích. K vytvoření doporučení není zapotřebí jiných uživatelů ani žádného sousedstva na rozdíl od collaborative filtering.

Tato vlastnost má za důsledek ještě jednu výhodu, a to schopnost doporučit nový předmět, který ještě nikdo neohodnotil. Jedinou potřebnou informací pro doporučení jakéhokoliv předmětu jsou totiž jeho atributy a seznam atributů, které uživatel na předmětech preferuje.

Další výhodou může být transparence doporučení. U každého doporučeného předmětu je možné identifikovat vlastnosti, které zapříčinily, že byl doporučen právě tento předmět. Uživatel pak může snadněji vyhodnotit, proč byl tento předmět doporučen a zda je toto doporučení relevantní.

Systém například doporučí uživateli film a zobrazí, že byl doporučen, protože v něm hraje jeden z jeho oblíbených herců.

Nevýhody

Tyto systémy také omezuje cold-start problem. Nedokaží pracovat s novými uživateli nebo obecně uživateli, kteří prokázali málo preferencí. Může se stát, že systém nedokáže rozlišit, jaké předměty uživatel preferuje či nikoliv [4].

Další nevýhodou je, že tento typ systému navrhuje pouze předměty, které na základě porovnání jejich vlastností s preferencemi uživatele dosahují vysokého skóre. Což může mít za příčinu jistou konzervativnost doporučení. To znamená, že uživateli pravděpodobně nebude doporučen předmět, na který by běžně nenarazil.

Například systém zjistí, že si uživatel oblíbil nějakého herce a bude mu doporučovat filmy s tímto hercem. Je však pravděpodobné, že by uživatel na tyto filmy narazil i bez doporučovacího systému. Na druhou stranu se uživatel nemusí dozvědět o podobném filmu, ve kterém se tento herec nevyskytuje.

2.1.3 Demografic filtering

Demografický doporučovací systém produkuje doporučení na základě demografických údajů o uživateli. Takovýto systém může tedy zohledňovat například věk, pohlaví, nebo národnost uživatele.

Může být výhodou například u uživatelů, kteří nemají vytvořený uživatelský účet a systém o nich nemá žádné informace. Na základě jejich IP adresy je možné zjistit odkud pocházejí, a pak doporučení přizpůsobit podle příslušné národnosti.

2.1.4 Community-based filtering

Community-based systém, jak již z názvu plyne, doporučuje na základě komunity. Tato komunita je tvořena přáteli daného uživatele. V podstatě se jedná o modifikaci collaborative filtering, ve kterém sousedsvto uživatelů reprezentují přátelé konkrétního uživatele.

Tato metoda je využívána vzhledem k současné popularitě sociálních sítí. Také zakládá na tom, že uživatelé preferují doporučení od svých přátel oproti cizím uživatelům [5].

2.1.5 Knowledge-based systémy

Knowledge-based neboli znalostní systém doporučuje na základě znalostí o předmětech a uživatelích. Systém určuje, který předmět a za jakých podmínek doporučit. Sbírá požadavky uživatele, vyhodnocuje je a na jejich základě produkuje doporučení. Pokud neumí vyhovět požadavkům uživatele, nalezne alternativní řešení.

Výhodou těchto systémů je, že netrpí cold-start problémem, nejsou tedy závislé na velkém množství vstupních dat.

Negativem je, že znalosti o vyhodnocování doporučení musí být zadány explicitně.

2.1.6 Hybridní systémy

Výše zmíněné přístupy je možné kombinovat a tím vznikají takzvané hybridní systémy. Je možné kombinovat dva nebo i více způsobů, záleží na konkrétní implementaci. Různé metody jsou spojovány tak, aby došlo k odstranění nedostatků některých z nich, nebo za prostým účelem přesnějšího doporučení. Například collaborative filtering si neumí poradit s novými předměty, je tedy možné využít vlastností content-based filtering, který tímto problémem netrpí [4].

2.2 Evaluace systému

Doporučovací systém se snaží nalézt pro daného uživatele to nejlepší doporučení. Otázkou ale je, která doporučení jsou nejlepší a jestli je daný systém produkuje?

Ideální doporučovací systém by určil, jak se uživateli bude předmět líbit, ještě předtím, než ho sám ohodnotí. Toto by se pak dalo považovat za nejlepší doporučení, ale reálně samozřejmě není možné vytvořit systém, který by předpovídal budoucnost. Ve skutečnosti se využívá metod, které na základě matematických výpočtů provedou odhad hodnocení pro všechny nebo některé předměty, které uživatel ještě neohodnotil. Každá metoda se však hodí pro jiný typ dat a v tom se odrážejí i její výsledná doporučení. Při vytváření systému je tedy potřeba několik těchto metod vyzkoušet pro konkrétní data a vybrat tu nejvhodnější z nich.

Aby bylo možné tyto metody porovnávat, je zapotřebí metriky, která by určovala kvalitu vypočtených výsledků. Podstatným krokem při hodnocení systému je také rozdělení vstupních dat na data trénovací a testovací.

2.2.1 Trénovací a testovací data

Před výpočtem jsou původní data rozdělena do dvou množin. Část dat je odložena stranou a není ve výpočtu zahrnuta. Později jsou pak tato data využita k otestování kvality výpočtu – odtud vyplývá název testovací data. Druhá část dat se nazývá trénovací a slouží k nalezení vhodných parametrů.

Tímto způsobem lze řešit zmíněný problém o předpovídání budoucnosti. Testovací data jsou totiž dopředu známa a obsahují předměty, které uživatel preferuje. Je zřejmé, že čím více těchto předmětů se zobrazí v seznamu doporučených předmětů, tím lepší výsledek bude produkován. Poměr trénovacích a testovacích dat je většinou ve prospěch dat trénovacích.

2.2.2 Metody evaluace

Na hodnocení neboli evaluaci systému je možné nahlédnout ze dvou různých pohledů. Prvním z nich je snaha zjistit, o kolik se průměrně liší hodnota navrženého doporučení s hodnocením uvedeném v testovacích datech. Druhý pak sleduje, kolik z navržených doporučení je relevantních.

Aritmetický a kvadratický průměr

V první metodě evaluace se, jak bylo již zmíněno, porovnává hodnota navrženého řešení s hodnotou z testovacích dat. Z rozdílu těchto hodnot je vypočten aritmetický průměr, který reprezentuje kvalitu systému. V tomto případě tedy platí, že čím menší je hodnota, tím lepší je výsledek.

Kromě obyčejného aritmetického průměru lze na vypočené hodnoty aplikovat kvadratický průměr, který představuje druhou odmocninu průměru druhých mocnin daných čísel. Jeho hodnota je vždy nezáporná a je větší nebo rovna aritmetickému průměru. Důsledkem umocnění hodnot je přidání větší váhy hodnotám vzdálenějším od nuly. V tomto konkrétním případě to znamená, že čím je hodnota vzdálenější od hodnoty doporučené, tím je hodnota průměru vyšší. Vnímání velikosti rozdílů těchto hodnot má za následek preciznější hodnocení systému než pomocí aritmetického průměru. Ukázka výpočtu obou průměrů je znázorněna v tabulce 2.1.

Precision a recall

Jiný pohled na evaluaci doporučovacího systému umožňuje precision a recall. Při tomto hodnocení je uvažováno, že není nezbytné provést všechna doporučení, ale stačí vrátit pouze několik nejlepších výsledků. Pro snazší pochopení následuje krátký příklad: ve filmové databázi z několika desítek tisíc filmů

	Předmět 1	Předmět 2	Předmět 3
Aktuální hodnocení	4.0	2.0	2.0
Odhad hodnocení	3.5	2.0	5.0
Rozdíl	0.5	0.0	3.0
Aritmetický průměr	$\frac{0.5+0.0+3.0}{3} = 1.1667$		
Kvadratický průměr	$\sqrt{\frac{0.5^2+0.0^2+3.0^2}{3}} = 1.7559$		

Tabulka 2.1: Ukázka výpočtu aritmetického a kvadratického průměru

ohodnotí uživatel 200 filmů. Je nejspíše zbytečené uvádět doporučenou hodnotu pro všechny filmy, stačilo by zjistit několik nejlepších doporučení. Už jen z toho důvodu, že ohodnocené filmy představují jen malý zlomek všech filmů, lze předpokládat, že ne všechna navržená doporučení musí být relevantní.

Tato metoda zjišťuje kolik výsledků je relevantních a tedy kolik doporučení má smysl počítat. Pro objasnění výpočtu precision a recall, je zapotřebí definovat několik veličin. Hodnoty navržené systémem pro uživatele u na základě trénovacích dat jsou značeny $L(u)$ (to, co by uživatele mohlo zajímat). Zatímco $T(u)$ udává všechna hodnocení uživatele u v testovacích datech (to, co uživatele opravdu zajímá).

Precision neboli přesnost určuje poměr počtu dobrých navržených doporučení ku celkovému počtu navržených doporučení. Dobrá doporučení jsou v tomto smyslu chápána jako ta, která byla systémem navržena a zároveň byla přítomna v testovacích datech.

$$Precision(L) = \frac{1}{|U|} \sum_{u \in U} \frac{|L(u) \cap T(u)|}{|L(u)|}$$

Recall, volně přeloženo jako odezva, udává počet relevantních výsledků uvedených v nejlepších navržených doporučeních. Recall je tedy poměr počtu dobrých navržených doporučení a počtu doporučení v testovacích datech.

$$Recall(L) = \frac{1}{|U|} \sum_{u \in U} \frac{|L(u) \cap T(u)|}{|T(u)|}$$

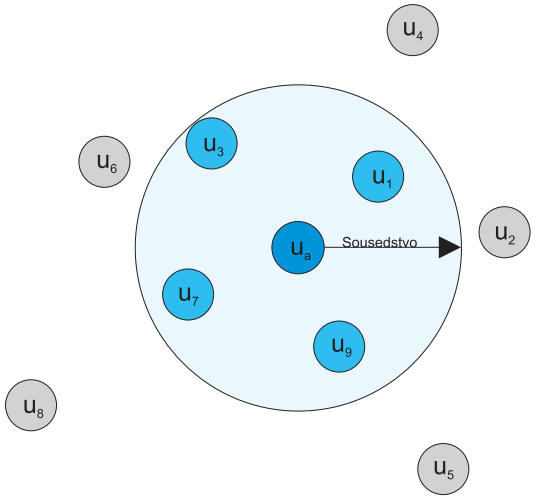
Tato hodnocení se využívají zejména v případě, když je stupnice hodnocení tvořena jen ze dvou hodnot (líbí / nelíbí). Aritmetický a kvadratický průměr je v těchto případech vždy stejný, jelikož všechny hodnoty, ze kterých je počítán, jsou stejné.

2.3 Vytváření sousedstva

Sousedstvo uživatele reprezentuje n nejbližších uživatelů, kde n je jeho velikost. Slouží ke zmenšení škály uživatelů, od kterých systém navrhuje doporučení. Existují dva základní způsoby reprezentace nejbližších sousedů.

2.3.1 Sousedstvo s pevnou velikostí

První varianta je poměrně jednoduchá, velikost sousedstva se odvíjí od předem zvolené hodnoty, takže pro $n = 100$ najde sto nejbližších uživatelů. Nabízí se otázka, jak velké n zvolit, aby bylo dosaženo nejlepšího výsledku. Čím větší je, tím více uživatelů je zahrnováno do výpočtu. To by se na první pohled mohlo zdát pozitivní, avšak nemusí tomu tak vždycky být. Když bude n menší, bude sice zahrnovat i menší počet uživatelů, ale zároveň nebude zahrnovat několik méně podobných uživatelů. Výsledná doporučení mohou být tedy přesnější, než kdyby bylo n větší. Ukázka sousedstva pro $n = 4$ je zachycena na obrázku 2.1.



Obrázek 2.1: Sousedstvo pro $n = 4$

2.3.2 Sousedstvo dané prahem

Druhý způsob přistupuje k problému jinak, nezajímá se o vlastní velikost sousedstva, ale jestli uživatel do této množiny ještě spadá, nebo již překročil nějaký práh podobnosti.

2.4 Počítání podobnosti

Nedílnou součástí doporučovacích systémů jsou techniky, pomocí kterých se počítá podobnost uživatelů či předmětů. Existuje několik metod, každá z nich má jisté výhody a nevýhody a hodí se za jiných okolností. Publikace [1] zmiňuje následující metody.

2.4.1 Pearsonova korelace

Tato metoda využívá korelace, která je používána ve statistice. Korelace určuje lineární vzdálenost mezi dvěma veličinami. Míru korelace určuje korelační koeficient nabývající hodnot v rozmezí od -1 do 1 včetně. Když se hodnota koeficientu blíží k -1, jedná se o nepřímou závislost. Čím více se jedna veličina zvyšuje, tím se druhá zmenšuje. Hodnota koeficientu 1 reprezentuje přímou závislost veličin. Nulová hodnota pak značí, že mezi veličinami není žádná korelační závislost. Korelace veličiny s tou samou veličinou je vždy rovna jedné. V doporučovacích systémech je korelace využívána k měření vzdálenosti mezi preferenčními hodnotami uživatelů.

Hlavní nevýhodou této metody je, že nebere v potaz množství předmětů, ve kterých se dva uživatelé shodují. Takže se může například stát, že výsledná hodnota bude nižší u dvou uživatelů, kteří ohodnotili 100 stejných předmětů, než u uživatelů, kteří ohodnotili pouze dva stejné předměty.

Další velkou nevýhodou je, že pokud mají dva uživatelé pouze jeden stejný předmět, pak korelace nelze vypočítat. Korelace také není definována, když by jeden uživatel ohodnotil všechny předměty stejnou hodnotou.

Jistého zlepšení je možné dosáhnout přidáním vah jednotlivým korelacím. Tím je odstraněn první nedostatek. Když je vypočtena korelace na základě více předmětů, má větší váhu, a pak je výsledná hodnota více přibližována

k -1 respektive 1.

$$s(u,v) = \frac{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)^2 \sum_{i \in I_{uv}} (r_{v,i} - \bar{r}_v)^2}}$$

Toto je vzorec pro výpočet podobnosti s mezi uživateli u a v. Množina všech předmětů, které ohodnotili oba uživatelé je označena I_{uv}. Hodnota $\bar{r}_u$ značí průměr hodnocení všech předmětů, ke kterým uživatel u vyjádřil. Podobně tak hodnota $\bar{r}_v$ udává průměr hodnocení uživatele v [6].

2.4.2 Euklidova vzdálenost

Další metoda je založena na vzdálenosti mezi uživateli. Uvažuje, že uživatelé představují body v n-dimenzionálním prostoru, kde n je určeno počtem předmětů. Vzdálenost uživatelů je pak vypočtena jako Euklidova vzdálenost mezi těmito body. Tato hodnota se vypočte jako druhá odmocnina součtu čtverců tvořených rozdílem souřadnic.

Pouze Euklidova vzdálenost pro implementaci této metody nestačí, je třeba ještě výslednou vzdálenost d upravit podle vzorce $1/(1+d)$. Pak platí, že čím větší je vzdálenost bodů, tím menší je podobnost uživatelů a naopak. Úplná podobnost uživatelů nastane, pokud je Euklidova vzdálenost rovna nule.

Oproti předchozí metodě je tato metoda funkční i v případě, když se dva porovnávaní uživatelé shodují jen v jednom předmětu. Také je opět možné přidat váhy jednotlivých hodnocení, které zohledňují množství předmětů, ve kterých se dva uživatelé shodují.

2.4.3 Cosinová vzdálenost

Tak jako v předchozí metodě je zde o preferencích uživatelů uvažováno jako bodech v prostoru, jehož dimenze je dána počtem předmětů. Nyní je možné si představit dvě přímky z počátku soustavy souřadnic k daným preferencím. Když jsou si dva uživatelé podobní, pak se budou nacházet blízko v prostoru

nebo přinejmenším stejným směrem. Úhel mezi těmito přímkami bude poměrně malý. V opačném případě, když si dva uživatelé nejsou podobní, úhel svíraný přímkami bude velký.

Tento úhel může být použit jako základ podobnosti dvou uživatelů. Výhodou je, že cosinus úhlu vždy nabývá hodnot mezi -1 a 1. Také, že cosinus malého úhlu se blíží k jedné, což reprezentuje velkou podobnost uživatelů, a naopak u velkého úhlu se hodnota blíží k -1, to znamená nízkou podobnost.

Cosinovou podobnost lze vyjádřit následujícím vztahem [6].

$$s(u,v) = \frac{\sum\limits_{i \in I_{uv}} r_{u,i} r_{v,i}}{\sum\limits_{i \in I_u} r_{u,i}^2 \sum\limits_{i \in I_v} r_{v,i}^2}$$

2.4.4 Spearmanova korelace

Základ této metody je založen na Pearsonově korelaci, ale místo původních preferencí se nejdříve vytvoří relativní pořadí preferenčních hodnot. U každého uživatele je hodnota nejméně preferovaného předmětu přepsána na jedničku, potom u dalšího nejméně oblíbeného předmětu na dvojku a tak dále, dokud nejsou všechny hodnoty nahrazeny novým číslem. Pak je na těchto hodnotách spočtena Pearsonova korelace.

Při tomto procesu dojde k určité ztrátě informace a to konkrétních hodnot jak daný uživatel předmět preferuje. Přesto zanechává údaj o pořadí preferencí.

Tato metoda se v praxi moc nevyužívá, jelikož vyžaduje netriviální výpočty a je časově velmi náročná.

2.4.5 Tanimoto koeficient

Další řešení kompletně ignoruje hodnoty preferencí vyjádřených uživateli. Zajímá se jen o to, jestli uživatel k danému předmětu vyjádřil preferenci či nikoliv. Základem této implementace je Tanimoto koeficient známý také jako Jaccardův koeficient. Je to podíl počtu předmětů, pro které vyjádřili dva uživatelé nějakou preferenci a počtu předmětů, pro které vyjádřil preferenci

alespoň jeden z nich. Když se předměty dvou uživatelů úplně překrývají, tak je koeficient roven jedné a je roven nule, když nemají ani jeden společný předmět. Aby tento přístup odpovídal svým rozsahem i ostatním implementacím, je výsledná hodnota vynásobena dvěma a následně je odečtena jednička. Tím vznikne požadovaný rozsah od -1 do 1.

Tato metoda se využívá v případě, kde se nevyskytuje stupnice hodnocení, ale pouze informace, jestli uživatel předmět preferuje.

2.4.6 Loglikelihood test

Tak jako v předchozím případě tato metoda ignoruje hodnoty preferencí vyjádřených uživateli. Zjišťuje pravděpodobnost, s jakou se dva uživatelé shodnou, na základě toho, kolik je v systému předmětů a kolik z nich oba uživatelé ohodnotili.

2.5 Existující řešení

Díky popularitě doporučovacích systémů existuje v současné době velká řada frameworků a různých nástrojů umožňující jejich vývoj, experimentování a následnou evaluaci. Jednotlivá řešení jsou realizována v odlišných programovacích jazycích a soustředí se na různé aspekty doporučovacích systémů (viz Tabulka 2.2).

2.5.1 Apache Mahout

Jedná se o knihovnu strojového učení, která umožňuje distribuci výpočtů pomocí Apache Hadoop nebo Apache Spark. Mahout[1] se hlavně zabývá třemi disciplínami – doporučovací systémy, klasifikace a shlukování. Nabízí nástroje k tvorbě systémů založených na Collaborative-filtering a také nástroje pro evaluaci.

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

Software	Jazyk	Vlastnosti
Apache Mahout	Java	Collaborative-filtering, integrace s Apache Spark a Apache Hadoop, umožňuje evaluace
Crab	Python	User-based filtering, Item-based filtering
GraphLab	C++	Collaborative-filtering, integrace s Apache Spark a Apache Hadoop, rozhraní v Pythonu
LensKit	Java	Collaborative-filtering, umožňuje evaluace
recommenderlab	R	Collaborative-filtering, umožňuje evaluace
RapidMiner	Java	Collaborative-filtering, umožňuje evaluace
Weka	Java	Collaborative-filtering, umožňuje evaluace

Tabulka 2.2: Nástroje pro tvorbu doporučovacích systémů

2.5.2 Crab

Crab[2] umožňuje tvorbu doporučovacích systémů pomocí programovacího jazyka Python. Zaměřuje se na item-based a user-based filtering, zatím nepodporuje evaluaci systému.

2.5.3 R

R[3] je programovací jazyk a nástroj pro statistické výpočty a jejich grafické zobrazení. R nabízí široké spektrum statistických pomůcek od lineárního a nelineárního modelování, statistických testů, klasifikace až po shlukování.

V R existuje knihovna zvaná recommenderlab, umožňující vývoj a testování doporučovacích systémů. Soustředí se zejména na collaborative filtering implementuje několik jeho algoritmů a poskytuje i možnost evaluace navrženého systému.

Výhodou R je, že se jedná o Free Software a je možné ho využívat na mnoha platformách (UNIX, Windows, MacOS).

[2]http://muricoca.github.io/crab/
[3]http://www.r-project.org/about.html

3 Návrh řešení

Doposud byl popisován obecný doporučovací systém, který má za úkol doporučit uživatelům nějaké předměty, ale zatím nebylo podstatné jaké. V této práci se jedná o filmovou databázi, tudíž předměty v tomto doporučovacím systému budou filmy. Tato kapitola je věnována návrhu a realizaci konkrétního doporučovacího systému.

3.1 Výběr vhodné knihovny

Jak již bylo zmíněno, existuje několik knihoven, které usnadňují tvorbu doporučovacích systémů. Většina z nich podporuje Collaborative filtering a také umožňuje evaluaci vytvořeného systému. Z uvedených nástrojů nabízí možnost distribuce výpočtů pouze Apache Mahout a GraphLab, a to oba pomocí Apache Hadoop nebo Apache Spark.

Pro tuto práci byl nakonec vybrán Apache Mahout, důvodem byla právě zmíněná možnost distribuce výpočtů, této výhody by mohlo být využito například při dalším rozšíření této práce. Další výhodou je poměrně detailní dokumence v podobě knihy [1].

3.2 Tvorba systému pomocí Apache Mahout

V této kapitole bude popsáno jak vytvořit doporučovací systém pomocí knihovny Apache Mahout. Postup pro zprovoznění Apache Mahout je popsán v příloze A.

3.2.1 Reprezentace preferencí

Preference uživatele k předmětu je v Mahoutu implementována pod názvem *GenericPreference*. Je tvořena třemi atributy, přičemž první dva reprezentují ID uživatele a ID předmětu a jsou typu Long. Posledním atributem je hodnocení typu Float.

Zajímavostí je, že souhrn preferencí není reprezentován polem těchto objektů ani není využíváno Java kolekcí. Důvodem je způsob, kterým jsou objekty v Javě alokovány. Prázdný objekt sám o sobě totiž zabírá nějakou paměť (velikost se liší podle architektury). Použité řešení pomocí *PreferenceArray* zabírá pouze čtvrtinu paměti oproti řešení pomocí objektů [1].

Dalším důvodem je, že při ukládání pomocí objektů dochází k ukládání některých informací duplicitně – u každého hodnocení je totiž ukládána dvojice ID. Zmíněné *PreferenceArray* se vždy vztahuje k jednomu konkrétnímu ID. Uchovává všechny preference jednoho uživatele ke všem předmětům, respektive všechny preference uživatelů k jednomu předmětu.

Ušetření paměti lze ocenit zejména u dat s velkým počtem hodnocení. V našem případě daná vstupní data celkem obsahují 1 879 614 hodnocení, která při uložení do *PrefereerenceArray* zabírají zhruba 21,51 MB. Při realizaci pomocí objektů by data zabrala asi 86.04 MB, což je čtyřikrát více.

3.2.2 Model

Jedním ze základních stavebních prvků doporučovacího systému v Apache Mahout je *DataModel*, který uchovává a zprostředkovává informace o hodnocení uživatelů. Při realizaci tohoto modelu je na výběr z několika možností.

Data v paměti

První z nich se jmenuje *GenericDataModel*, který pracuje s doporučeními přímo v paměti. Tento model využívá k uchování jednotlivých *PreferenceArray* takzvaný *FastByIDMap*. Je zde použit místo HashMap, a to opět z důvodu úspory paměti.

Data ze souboru

Dalším modelem je *FileDataModel*, který jako parametr přijímá soubor s preferencemi. Tento soubor má jasně danou strukturu a může být komprimován. Data jsou očekávána ve formátu CSV (Comma-separated values). Jak již z názvu vyplývá, hodnoty jsou od sebe odděleny čárkami v rámci řádky. Každý údaj o hodnocení zastává jednu řádku.

```
0,25,4.0
0,168,5.0
0,999,2.0

1,3158,3.0
1,21899,4.0
```

První hodnota reprezentuje ID uživatele, druhá pak ID předmětu, poslední hodnotou je samotné hodnocení předmětu, které je zapisováno s přesností na jedno desetinné místo. Jednotliví uživatelé jsou od sebe odděleni prázdným řádkem.

Tento model nahraje data z příslušného souboru a pak je uloží do zmíněného *GenericDataModelu*. Změny provedené ve vstupním souboru za běhu systému jsou modelem pozorovány a předávány systému. Připisování na konec velkého souboru může být časově náročné, proto je k jeho aktualizaci možné využít aktualizačních souborů. Tyto soubory se musí nacházet ve stejném adresáři jako vstupní soubor a jejich název se musí shodovat až k první tečce, pak následuje číslovka určující pořadí souboru a až pak přípona.

Například vstupní soubor s názvem *data.csv* může mít aktualizační soubory *data.1.csv* a *data.2.csv*. Stávající preference je možné zrušit pomocí nové preference s prázdnou hodnotou. Například následující řádka smaže první hodnocení v předchozí ukázce.

```
0,25,
```

Data z databáze

Je podporován i přístup k datům v databázi a to pomocí *JDBCDataModelu*. Ten podporuje několik typů databází jako například MySQL. Tento přístup může být použit zejména, když je soubor s daty příliš velký (v tom smyslu, že se nevejde do operační paměti). Nevýhodou však je, že práce s daty v databázi je pomalejší, než práce s daty v paměti.

Použité řešení

Právě kvůli rychlosti přístupu k datům byl pro realizaci systému vybrán *FileDataModel*. Také díky tomu, že soubor se vstupními daty není příliš velký

(zhruba 20 MB).

3.2.3 Vytvoření doporučovacího systému

Při tvorbě systému je na výběr z několika přístupů. Vzhledem k informacím ve vstupních datech (viz kapitola 3.3) byl využit přístup Collaborative-filtering. Vstupní data totiž neobsahují žádné informace o filmech, které by mohly být využity při doporučování pomocí Content-based filtering. Pro využití tohoto přístupu by bylo nejdříve nutné zjistit nějakou informaci o všech filmech, například žánr. V datech se však vyskytuje celkem 77 447 filmů a doplňovat tuto informaci ručně by zabralo mnoho času.

Vytvoření doporučovacího systému lze v Apache Mahout napsat na pouhých pár řádkách (Kód 3.1).

Kód 3.1: Ukázka kódu doporučovacího systému

```
1 DataModel model = new FileDataModel(new File("data.csv"));
2 UserSimilarity similarity = new EuclideanDistanceSimilarity(model);
3 UserNeighborhood nbh = new ThresholdUserNeighborhood(0.5,
    similarity, model);
4 Recommender recommender = new GenericUserBasedRecommender(model,
    nbh, similarity);
```

Jedná se o Collaborative filtering, konkrétně user-based přístup. Objekt *Recommender*, reprezentující doporučovací systém, lze vytvořit pomocí několika atributů (záleží na typu *Recommenderu*). Prvním atributem je již dříve zmíněný model, který je v tomto případě vytvořen pomocí souboru. Druhý atribut určuje sousedstvo uživatele – to lze určit buď počtem, nebo prahem. Poslední atribut definuje způsob počítání podobnosti (viz kapitola 2.4).

Porovnáním možných přístupů a výběrem vhodných parametrů pro konkrétní vstupní data se zabývá kapitola 4.

3.2.4 Návrh doporučení

Po vytvoření systému je možné získat navržená doporučení pomocí metody *recommend()* volané nad objektem *Recommender*. Tato metoda má dva vstupní parametry – první z nich určuje ID uživatele, druhý pak počet doporučení,

která mají být vypočtena, návratová hodnota je List<RecommendedItem>.

3.3 Vstupní data

K implementaci doporučovacího systému je zapotřebí vstupních dat, jelikož nepřítomnost nebo nedostatek vstupních dat způsobuje již zmíněný cold-start problem.

Pro tuto práci byla zadána data, která jsou uvedena v příloze. Tato data slouží jako zdroj informací potřebných k vypočtení doporučení.

3.3.1 Formát

Vstupní data jsou rozdělena a uchována v několika souborech s příponou `.xml`. Důvodem rozdělení je rozsáhlost těchto dat. V každém souboru je uveden seznam filmů a u každého filmu se vyskytují příspěvky. Součástí každého příspěvku je komentář spolu s dalšími informacemi o uživateli a času vložení. Hlavní a jedinou důležitou informací pro tuto práci je hodnocení filmu, které se však nevyskytuje u všech příspěvků.

3.3.2 Příprava dat

Z uvedených dat je relevantní pouze jejich část, proto je nutné vstupní data projít a vybrat z nich informace o jednotlivých hodnoceních. Ve vstupních souborech se vyskytují některé neviditelné znaky, které komplikují parsování souboru pomocí SAX (Simple API for XML) Parseru. Před parsováním je tedy potřebné se těchto znaků zbavit. Jakmile jsou vstupní soubory zbaveny neviditelných znaků (ne však všech, zůstává znak konce řádku, tabulátor atp.), je možné projít všechny příspěvky s hodnocením a příslušnou informaci si uchovat. Data je nutné uložit tak, aby s nimi mohl Apache Mahout pracovat, viz kapitola 3.2.2.

Pro zlepšení výsledků byli z dat odstraněni uživatelé, kteří ohodnotili malé množství filmů. Mohlo by se stát, že z důvodu nedostatku informací by doporučení nešlo vypočítat. Není zřejmé, kde přesně udělat hranici a jaké

uživatele do výpočtu zahrnout. Bylo proto vytvořeno několik souborů (Tabulka 3.1) s uživateli, kteří ohodnotili alespoň 20, 50, 100 a 200 filmů a jejich výsledky pak byly porovnány. Z dat byli také odstraněni ti uživatelé, kteří přiřadili stejnou hodnotu všem filmům. Je pravděpodobné, že tito uživatelé si nedali práci s hodnocením filmů a zahrnutí jejich informací do výpočtu by bylo spíše negativní.

Název	Min. počet filmů jednoho uživatele	Počet uživatelů	Počet doporučení	Velikost
20.dat	20	10 734	1 794 383	26,1 MB
50.dat	50	6 603	1 663 465	24,1 MB
100.dat	100	4 102	1 486 484	21,4 MB
200.dat	200	2 238	1 223 623	17,5 MB

Tabulka 3.1: Přehled souborů

Dále byly vytvořeny soubory *movies.dat* a *users.dat*, které obsahují seznam názvů filmů respektive přezdívek uživatelů, přičemž každý údaj je na samostatné řádce. Celkový počet filmů je 77 447 a uživatelů je 23 609.

3.4 Webová aplikace

V rámci této práce byla vytvořena webová aplikace, která usnadňuje vyzkoušení doporučovacího systému. Také byl vytvořen servlet, jehož běh je řízen pomocí servletového kontejneru Apache Tomcat[1]. Tento servlet umožňuje interakci mezi doporučovacím systémem a klientskou částí aplikace. Komunikace je dosaženo pomocí HTTP požadavků.

3.4.1 Servlet

Třída MovieRecommender rozšiřuje *javax.servlet.http.HttpServlet* a dědí z ní metody *init()*, *doGet()* a *doPost()*. Tyto metody jsou v MovieRecommender překryty vlastní implementací. Metoda *init()* je volána při spuštění servletu, má za úkol inicializovat recommender a načíst data o jménech uživatelů a filmů ze souboru. Po dobu běhu servletu mohou být volány metody *doGet()* a *doPost()*, které zpracují požadavek a případně poskytnou žádané informace.

[1] http://tomcat.apache.org/

HTTP požadavky

Aby mohl servlet na požadavky reagovat, musí být zadány ve specifickém tvaru. Přehled možných HTTP požadavků je možné vidět v tabulce 3.2. Všechny vrácené odpovědi ze servletu jsou uloženy ve formátu JSON[2].

3.4.2 Klientská část

HTTP dotazy je sice možné pokládat i v adresovém řádku prohlížeče, ale pro snadnější obsluhu byla naprogramována klientská část aplikace. Tato část se skládá z několika souborů. Hlavním z nich je *index.jsp*, který pomocí jazyka HTML definuje strukturu stránky a využívá ostatních `.css` a `.js` souborů. CSS soubory definují vzhled stránky pomocí kaskádových stylů. Soubor *datahandler.js* se stará o načtení dat ze serveru pomocí asynchronních HTTP GET dotazů. Tento způsob umožňuje plynulé zpracování požadavků bez zamrzání grafického rozhraní.

3.4.3 Soubor WAR

Celá webová aplikace je obsažena v souboru WAR (Web application archive), který umožňuje snadné zavedení aplikace pomocí servletového kontejneru. Tento soubor je generován nástrojem Maven na základě parametrů uvedených v souboru *pom.xml* po zadání příkazu `mvn package`. V kořenovém adresáři archivu se vyskytují soubory pro klientskou část a složka WEB-INF. Uvnitř této složky je uložen *web.xml*, který určuje hlavní třídu servletu a také URL cestu aplikace. Ve stejné složce se nachází zdrojové soubory aplikace a všechny potřebné knihovny.

3.4.4 Aktualizace systému

Pro aktualizaci doporučovacího systému lze využít již dříve zmíněné aktualizační soubory. Problémem však je, že soubory uvnitř již vytvořeného WAR archivu běžícího v servletovém kontejneru nelze upravovat ani vytvářet.

[2]http://www.json.org/json-cz.html

Tento problém byl vyřešen tím, že při inicializaci servletu (v metodě *init()*) je přečten zdrojový soubor s daty, který se nachází uvnitř archivu. Následně je mimo archiv vytvořen nový soubor se stejnými daty. Pak je možné vytvářet aktualizační soubory ve stejné složce. Všechny tyto soubory mají nastavený parametr *deleteOnClose*, takže při skončení JVM (Java Virtual Machine) dojde k jejich smazání. Při vytváření souborů by mohlo dojít k problémům s oprávněním k zápisu, proto je vyžádána systémová složka s dočasnými soubory, kde by měl být přístup většinou povolen.

Frekvence obnovení datového modelu podle změn v aktualizačních souborech je řízena samotným modelem. Aby bylo možné pohotově pozorovat změny v systému, je možné toto obnovení modelu explicitně vynutit. Obnovení však může trvat několik desítek vteřin a volat jej po přidání každého hodnocení by mohlo systém značně zpomalit. Aby uživatel nemusel po každém zadání nového hodnocení čekat na obnovení modelu, je jeho obnovení spuštěno až po zadání dvaceti nových hodnocení (od libovolných uživatelů).

3.4.5 Funkce aplikace

Účelem této aplikace je pouze demonstrace funkčnosti doporučovacího systému. Je umožněno zobrazit hodnocení libovolného uživatele a volně je přidávat, není požadováno žádné heslo ani jiné přihlašovací údaje. Byl vytvořen testovací uživatel, který je ve výchozím stavu vybrán. Tento uživatel nemá zadané žádné hodnocení a slouží k vyzkoušení systému.

Dotaz	Příklad	Funkce	Příklad odpovědi
getmovies	getmovies=true	Vrátí pole filmů, přičemž každý film je dán polem o dvou prvcích – ID a název. Filmy jsou seřazeny podle názvu.	[[”167”,”Název1”], [”2”,”Název2”]]
getusers	getusers=true	Vrátí pole uživatelů, přičemž každý uživatel je dán polem o dvou prvcích – ID a název. Uživatelé jsou seřazeny podle názvu.	[[”7445”,”Adam”], [”25”,”Karel”]]
refresh	refresh=true	Vynutí aktualizaci data modelu podle změn v aktualizačních souborech.	
userID	userID=123	Vrátí seznam filmů, které uživatel viděl. U každého filmu je uvedeno hodnocení a ID.	{”items”:[{”value”: ”4.0”,”id”:”14”}, {”value”:”5.0”, ”id”:”2452”}]}
userID& howMany	userID=1& howMany=100	Vrátí seznam doporučených filmů.	{”items”:[{”value”: ”5.0”,”id”:”73”}, {”value”:”5.0”, ”id”:”2”}]}
userID& itemID& rating	userID=1& itemID=10& rating=3.0	Zapíše nové hodnocení do aktualizačního souboru a vrátí zprávu o průběhu.	ok nebo fail

Tabulka 3.2: Seznam možných HTTP požadavků

4 Experimenty

Pro implementaci doporučovacího systému je nutné vybrat jednu ze dříve zmíněných metod pro počítání podobnosti (viz kapitola 2.4). Je zapotřebí spustit každou z těchto metod s různými parametry prahu podobnosti nebo velikosti sousedstva a určit hodnoty jejich výsledků. Za vhodnou metodu je považována ta, která dosahuje co nejlepšího skóre, ale zároveň má co nejkratší dobu výpočtu pro daná vstupní data.

K otestování těchto metod byla vytvořena aplikace, kterou lze spustit příkazem `java -jar evaluator-1-jar-with-dependencies.jar`. Po spuštění aplikace bez parametrů se zobrazí nápověda s instrukcemi, jak určit metodu testování a jiné parametry.

Výsledky spuštěných testů jsou uloženy ve složce *res* do příslušné podsložky podle množství hodnocení uživatele (20, 50, 100 nebo 200). Soubor je pojmenován podle názvu metody a na každé jeho řádce je uveden jeden výsledek měření. V každém výsledku je zobrazena velikost prahu podobnosti, nebo v případě sousedstva počet sousedů, dále je uvedeno dosažené skóre a doba výpočtu.

Právě doba výpočtu poněkud komplikuje vyhodnocení optimální metody. Vzhledem k počtu metod a možných parametrů je potřeba otestovat zhruba 60 různých kombinací pro první soubor s uživateli, kteří ohodnotili alespoň 20 filmů. Z těchto výsledků lze vybrat menší množství parametrů, které dosahují dobrých hodnot a spustit je na další souborech s jiným vzorkem uživatelů.

Na počítači se čtyřjádrovým procesorem o taktu 2.4 GHz a operační pamětí o velikosti 4 GB byla doba nejméně náročného testu necelé tři hodiny. Spuštění šedesáti testů na tomto počítači by tedy zabralo zhruba týden. Výpočet pro některé metody však trval několikanásobně déle než tři hodiny, takže celkový čas výpočtu by byl podstatně delší.

Velkou výhodou byla možnost využít výpočetního gridu MetaCentrum[1], který umožňuje akademickým pracovníkům, zaměstnancům a studentům vědeckovýzkumných institucí v České republice využívat vysoký výpočetní výkon. Stejný test spuštěný na jednom z počítačů MetaCentra zabral pouze 10 minut. Tímto bylo umožněno pohodlnější ozkoušení metod a parametrů.

[1] http://metavo.metacentrum.cz/

Nicméně spouštění úloh na strojích MetaCentra přineslo i nějaké komplikace. Nejdříve bylo zapotřebí nastudovat pravidla o používání strojů a spouštění úloh. Hlavním problémem byl limit procesorů přidělených pro jednu úlohu.

Díky tomu, že Apache Mahout vyhodnotí počet procesorů a jejich jader na konkrétním počítači, a pak rozdělí výpočet do vláken úměrných tomuto počtu, docházelo k tomu, že byl překračován počet přidělených procesorů. Kvůli překročení tohoto limitu byla pak úloha plánovačem MetaCentra ukončena. Protože je výpočet spuštěn pokaždé na jiném stroji (vyhodnoceno plánovačem, pokud není explicitně určeno), pak se stává, že počet procesorů je jiný než limit, který byl definován při spuštění úlohy, a tím může dojít k jeho překročení.

Tento problém lze vyřešit vyžádáním celého jednoho stroje nehledě na to, kolik má procesorů. Další možností je vyhlédnout si jeden konkrétní stroj a při spuštění úlohy explicitně zadat požadavek pro zabrání tohoto stroje. Pak je zřejmé, jak velký limit pro počet procesorů dopředu nastavit.

4.1 Dosažené výsledky

První dosažené výsledky evaluací metod jsou získány na základě vstupního souboru s uživateli, kteří ohodnotili alespoň 20 filmů. Pro hodnocení systému byla vybrána metoda průměrné odchylky (viz kapitola 2.2.2). Je dobré si připomenout, že čím menší je vypočítaná hodnota, tím lepší je výsledek.

Ve výsledcích jsou zahrnuty oba způsoby výběru sousedstva, a to jak přímým počtem sousedů, tak i velikostí prahu podobnosti (viz kapitola 2.3). První ze zmíněných je možné vidět v tabulce 4.1. Proměnná n určuje počet sousedů, kteří jsou zahrnuti ve výpočtu.

	$n = 2$	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$	$n = 128$
Pearson	0.496	0.556	0.556	0.556	0.562	0.568	0.578
Euclidean	0.000	0.000	0.000	0.000	0.001	0.004	0.019
Tanimoto	0.827	0.830	0.828	0.824	0.822	0.819	0.816
Loglikelihood	0.876	0.876	0.866	0.854	0.844	0.837	0.832

Tabulka 4.1: Sousedstvo dané počtem

Nevýhodou určování přímého počtu sousedů je velké množství možností.

Není zřejmé, zda použité hodnoty dostatečně pokrývají interval možných hodnot. Nejmenší možná hodnota je 1, výpočet je totiž nutné založit alespoň na jednom uživateli. Horní mez je teoreticky dána celkovým počtem uživatelů. Je třeba dbát na to, že při rozdělení dat na testovací a trénovací je část dat odstraněna z výpočtu, a tím je změněn i celkový počet uživatelů zahrnutých ve výpočtu.

Využitím prahu podobnosti je možné tento interval snadněji pokrýt. Práh může nabývat hodnot od -1 do 1 včetně. Při prahu s hodnotou 1 bude sousedstvo tvořeno pouze těmi uživateli, kteří mají naprosto shodné hodnocení. Naopak při hodnotě -1 budou do sousedstva zahrnuti i ti nejméně podobní uživatelé. Pokrytím tohoto intervalu lze tedy ošetřit všechny možnosti a vyhodnotit, které parametry jsou nejvhodnější. V tabulce 4.2 jsou uvedeny výsledky evaluace pro několik hodnot prahu a metod podobnosti.

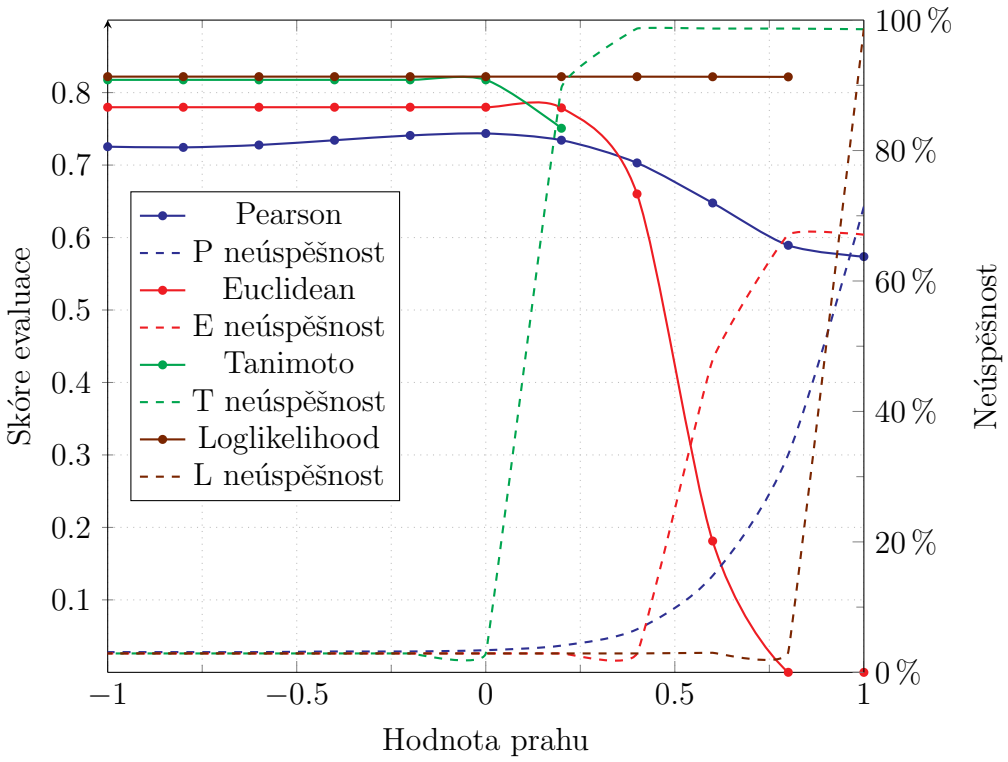
Práh	Pearson	Euclidean	Tanimoto	Loglikelihood
1.0	0.574	0.000	NaN	NaN
0.8	0.589	0.000	NaN	0.822
0.6	0.648	0.181	NaN	0.822
0.4	0.703	0.660	NaN	0.822
0.2	0.734	0.779	0.751	0.822
0.0	0.744	0.780	0.818	0.822
−0.2	0.741	0.780	0.818	0.822
−0.4	0.734	0.780	0.818	0.822
−0.6	0.728	0.780	0.818	0.822
−0.8	0.724	0.780	0.818	0.822
−1.0	0.725	0.780	0.818	0.822

Tabulka 4.2: Sousedstvo dané prahem

Při pohledu na tabulku se zdá, že počítání podobnosti pomocí Euklidovské vzdálenosti dosahuje perfektních výsledků. Pro hodnotu prahu 1.0 a 0.8 se navržené doporučení vůbec neliší od doporučení v testovacích datech.

Bohužel je třeba zahrnout do výsledků ještě jeden fakt, a to ten, že v některých případech nelze doporučení vypočítat. V tabulce se dokonce vyskytují hodnoty NaN (v Javě definováno jako Not a Number), a to v případech, kdy systém nedokázal výslednou hodnotu vůbec vypočítat. Může se také stát, že podobnost není v určitém případě definována – například Pearsonova kore-

lace pro uživatele s naprosto stejným hodnocením, nebo když je k výpočtu využito příliš malé množství dat.



Obrázek 4.1: Porovnání úspěšnosti výsledků evaluace

V grafu 4.1 je možné pozorovat procenta neúspěšnosti jednotlivých metod, která jsou značena přerušovanou čarou a vztahuje se k nim popisek uvedený u pravé osy. Naopak popisky na levé ose reprezentují naměřené hodnoty a vzahují se k plným čarám.

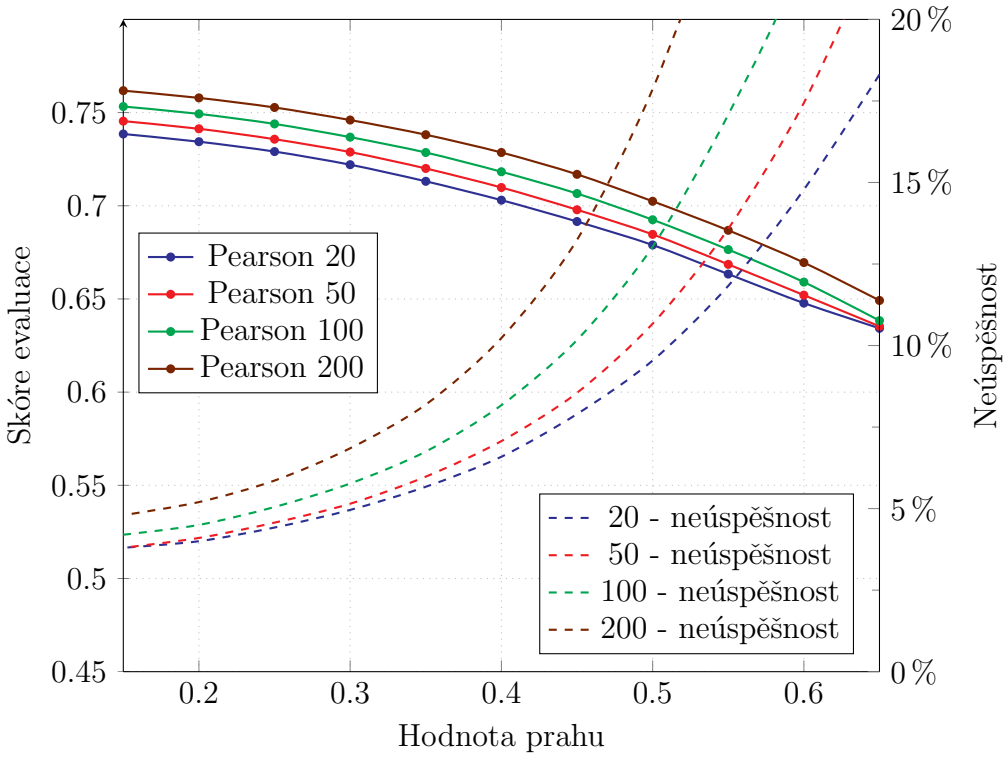
Doba výpočtu jedné hodnoty pomocí Pearsonovy korelace a Euklidovské vzdálenosti byla zhruba 10 minut (spuštěno na MetaCentru na 64 procesorových jádrech se 100 GB operační paměti). Oproti tomu doba výpočtu pomocí zbylých dvou metod se za stejných podmínek pohybovala okolo dvou a půl hodiny. Z důvodu zachování přehlednosti grafu, nebyly tyto údaje znázorněny.

U metody počítání pomocí Tanimoto koeficientu je dobře vidět, že tam, kde se vyskytovaly hodnoty NaN, se neúspěšnost blíží ke 100 %. U hodnot prahu, kde Euklidovská vzdálenost dosahuje nejlepších výsledků, nebylo možné vypočítat doporučení v 67 % případů. Pro nalezení vhodné hodnoty

prahu je nutné se soustředit na co nejnižší výsledné hodnoty, ale tak, aby neúspěšnost doporučení byla stále přijatelná (například pod 20 %).

Ze stejného obrázku je také možné vypozorovat, že u dvou metod se výsledky evaluace i přes měnící se hodnotu prahu téměř nemění. Počítání podobnosti pomocí Tanimoto koeficientu a Loglikelihood testu se pro zvolená vstupní data příliš nehodí. Při všech hodnotách prahu dosahují horších výsledků, než zbylé dvě metody. Zelená křivka v grafu sice naznačuje mírný pokles skóre evaluace, ale zároveň s ním se rapidně zvyšuje neúspěšnost doporučení. Zbylé experimenty se tedy soustředí pouze na Euklidovskou vzdálenost a Pearsonovu korelaci.

Následující graf (obrázek 4.2) zobrazuje výsledky Pearsonovy korelace pro hodnoty prahu, které dosahují nejlepších výsledků (tj. interval od 0.15 do 0.75) s krokem o velikosti 0.5. Jsou zde uvedeny výsledky pro všechny čtyři vzorové soubory.



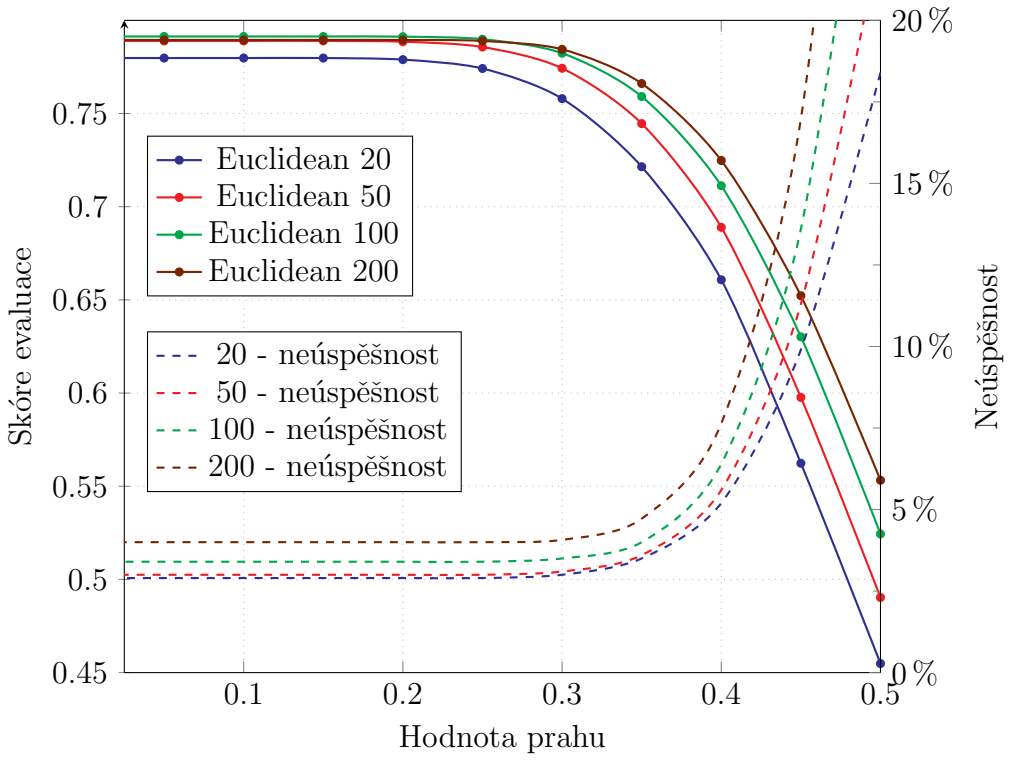
Obrázek 4.2: Pearsonova korelace

Je zajímavé, že když jsou do výpočtu zahrnuti jen uživatelé, kteří ohodnotili větší množství filmů, jsou produkovány horší výsledky. Dokonce čím

je nárok na počet hodnocení vyšší, tím horší jsou výsledky. Nejenže dosahují vyššího skóre, ale i procenta jejich neúspěšnosti jsou podstatně vyšší.

Podobný průběh lze pozorovat i v grafu 4.3, ve kterém jsou zobrazeny výsledky vypočtené pomocí Euklidovské vzdálenosti. Tento graf zobrazuje jiné hodnoty prahu, a to proto, že oproti Pearsonově korelaci se neúspěšnost výpočtů pohybuje okolo dvaceti procent již u nižších hodnot prahu.

Výsledky dosažené pomocí této metody jsou o něco lepší než ty, které byly vypočtnené předchozí metodou. Nejlepších výsledků je opět dosaženo u souboru s nejmenším počtem hodnocení na uživatele.



Obrázek 4.3: Euklidovská vzdálenost

4.2 Zhodnocení výsledků

Bylo prokázáno, že čím větší je nárok na počet hodnocení filmů jednoho uživatele, tím horších výsledků je dosahováno. Toto by mohlo být způsobeno

například tím, že uživatelé kteří ohodnotili větší množství filmů, ohodnotili i filmy, které se jim nelíbili, nebo byly ohodnoceny malým počtem uživatelů. Pokud byly právě tyto filmy vybrány do testovacích dat, je méně pravděpodobné, že budou doporučeny. Oproti tomu je možné předpokládat, že uživatelé, kteří ohodnotili menší množství filmů, hodnotili převážně známé snímky, které je snadnější doporučit (byly ohodnoceny mnoha uživateli).

Další roli by také mohl hrát celkový počet uživatelů v daném vzorkovém souboru. V souboru *20.dat* se nachází 10 734 uživatelů, narozdíl od souboru *200.dat*, ve kterém je pouze 2 238 uživatelů (viz tabulka 3.1). Čím více uživatelů v databázi existuje, tím větší je pravděpodobnost, že bude nalazen podobný uživatel.

Nejlepších výsledků bylo dosaženo pomocí Euklidovské vzdálenosti při hodnotě prahu 0.5. Bylo sice dosaženo i menších odchylek, ale s tím zároveň stoupala četnost neúspěchů při výpočtu doporučení. Tato metoda a hodnota prahu pro soubor *20.dat* byla z předchozích důvodů vybrána pro realizaci systému.

5 Závěr

V této práci byly prozkoumány technologie užívané v doporučovacích systémech. Na základě těchto technologií byl navržen doporučovací systém, který byl následně implementován. Pro implementaci byl ze zmíněných knihoven vybrán Apache Mahout, zejména pro možnost distribuce výpočtů a možnost evaluace výsledného systému.

Na výsledném systému byly provedeny experimenty s různými metodami a parametry a jejich výsledky byly analyzovány. Na základě těchto experimentů bylo vybráno počítání podobnosti pomocí Euklidovské vzdálenosti se sousedstvem daným prahem o velikosti 0.5. Pro ulehčení testování parametrů byla vytvořena jednoduchá aplikace, která byla z důvodů časové a výpočetní náročnosti spuštěna na strojích virtuální organizace MetaCentrum.

Aby bylo možné doporučovací systém snadno vyzkoušet, byla vytvořena webová aplikace. Tato aplikace umožňuje uživateli přidat nová hodnocení a na jejich základě doporučí filmy, které by se mohly uživateli líbit. V těchto návrzích se občas vyskytují doporučení filmů, které ohodnotilo pouze pár uživatelů a mohlo by tedy být zavádějící.

Lepších výsledků by mohlo být dosaženo využitím rozsáhlejších dat. Založením výpočtů na větším množství hodnocení by byla zvýšena pravděpodobnost nalezení podobného uživatele. Také by bylo možné získat více informací o jednotlivých filmech a tyto údaje zahrnout do výpočtu doporučení.

Pro snadnější ověření výsledků dosažených v této práci byl sepsán postup pro sestavení projektu, instalaci nástroje Apache Tomcat, který umožňuje zprovoznění webové aplikace, a také byla napsána uživatelská dokumentace. Všechny tyto postupy jsou uvedeny v přílohách.

Veškerá zdrojová data, soubory a výsledky experimentů se nachází na přiloženém DVD. Aplikace umožňující testování parametrů se nachází ve složce *evaluator*, ve stejné složce jsou také vstupní data. Webová aplikace je uložena ve složce *recommender*.

Literatura

[1] Ted Dunning Sean Owen, Robin Anil and Ellen Friedman. *Mahout in Action*. Manning Publications Co., Shelter Island, NY, 2012.

[2] Schafer J.B. Frankowski D. Herlocker J. Sen S. *Collaborative filtering recommender systems. In: The Adaptive Web*. Springer, Berlin Heidelberg, 2007.

[3] Joseph Konstan Badrul Sarwar, George Karypis and John Riedl. *Item-Based Collaborative Filtering Recommendation Algorithms. In: Proceedings of the tenth international conference on World Wide Web - WWW '01*. ACM, New York, 2001.

[4] Bracha Saphira Paul B. Kantor Francesco Ricci, Lior Rokach. *Recommender systems handbook*. Springer, New York, 2011.

[5] Rashmi R. Sinha and Kirsten Swearingen. *Comparing recommendations made by online systems and friends. In: DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*. Springer-Verlag, New York, 2001.

[6] Michael D.John T Michael D. Ekstrand. *Collaborative filtering recommender systems*. now Publishers, Hanover Mass, 2011.

Přílohy

A Sestavení projektu

Všechny testy a jiné postupy uvedené v této kapitole je možné zrekonstruovat. Veškerá zdrojová data jsou přístupná na přiloženém DVD, případně je možné zdrojový kód stáhnout pomocí nástroje git[1]. Po instalaci tohoto nástroje lze zadáním příkazu `git clone https://github.com/lochman/movie-recommender.git` stáhnout všechny zdrojové soubory vytvořené v průběhu této práce. Pokud není příkaz git v příkazové řádce rozeznán, je nutné nejprve upravit systémovou proměnnou *PATH*. Tento proces se na různých operačních systémech liší.

V systému Windows je nutné kliknout pravým tlačítkem na *Tento počítač* a v nabídce možností vybrat *Vlastnosti*. V následujícím okně se po sledu těchto příkazů *Upřesnit nastavení systému*→*Upřesnit*→*Proměnné prostředí* zobrazí požadovaná nabídka. Proměnnou *PATH* je třeba upravit tak, aby obsahovala cestu do složky *bin* a *cmd* v adresáři, kde je git nainstalován. Například pokud je git nainstalován v umístění `C:\Program Files (x86)\Git`, je nutné na konec proměnné přidat `;C:\Program Files (x86)\Git\bin;C:\Program Files (x86)\Git\cmd`.

V Linuxu lze proměnnou nastavit příkazem `export PATH=/usr/local/maven/bin:$PATH` (záleží na tom, kde je git nainstalován). Toto řešení je však jen dočasné a po zavření konzole přestane účinkovat. Pro trvalé nastavení je potřeba upravit soubor `~/.profile` a přidat do něj výše uvedený příkaz.

Dalším potřebným nástrojem pro sestavení tohoto projektu je Apache Maven[2]. Kromě usnadnění samotného sestavení projektu je jeho velkou výhodou snadné přiložení potřebných knihoven. Po instalaci může být nutné opět upravit proměnnou *PATH* tak, aby obsahovala odkaz na binární soubory Mavenu. Příkazem `mvn --version` lze ověřit, zda je vše dobře nastaveno.

Projekt je možné sestavit zadáním příkazu `mvn package` ve složce *recommender* se zdrojovými soubory. V této složce se nachází soubor `pom.xml` a na základě jeho parametrů je vygenerována složka *target* s přeloženými `.class` soubory a spustitelnými `.jar` soubory.

Vygenerovány jsou celkem tři `.jar` soubory. Kromě základního souboru bez přiložených knihoven se zde vyskytuje `jar-with-dependencies`, který

[1] http://git-scm.com/downloads
[2] https://maven.apache.org/

obsahuje všechny potřebné knihovny a je doporučen pro spuštění tohoto projektu. Poslední soubor obsahující název `job` umožňuje spuštění na clusteru Apache Hadoop. Soubor `pom.xml` také umožňuje snadné importování projektu do vývojového prostředí Eclipse, kde je pak možné provést úpravy programu.

B Sestavení Mahoutu

Pro spuštění tohoto projektu sice není nutné Apache Mahout instalovat, ale mohl by být využit při jeho úpravě či rozšíření. Dalším důvodem, proč je tento postup uveden, jsou komplikace, ke kterým může během instalace dojít.

Apache Mahout je knihovnou v Javě, proto je nezbytné nejprve Javu nainstalovat. Nejnižší podporovaná verze je Java 6. Napsáním příkazu `java -version` do příkazové řádky lze zjistit aktuální nainstalovanou verzi. Samotnou instalací Javy se tato práce nebude zabývat. Dále je potřebný již zmíněný Apache Maven.

Při stahování zdrojových souborů[1] Mahoutu je nutné vybrat jeden z balíčků s názvem *src*. Po rozbalení lze Mahout sestavit napsáním příkazu `mvn install` ve složce se zdrojovými soubory. Tento proces může zabrat až několik desítek minut, dochází totiž k různému testování. Zde také nejčastěji dochází k chybám. Je možné se tomuto testování vyhnout pomocí příkazu `mvn -DskipTests install`. Při první instalaci je však doporučena instalace s testováním. Sestavený Mahout lze pak importovat do vývojového prostředí Eclipse. Pro zakomponování do příkazové řádky, je zapotřebí opět upravit proměnnou *PATH* jako v předchozí kapitole.

[1] http://www.apache.org/dyn/closer.cgi/mahout/

C Instalace Apache Tomcat

Pro vyzkoušení webové aplikace je zapotřebí nástroje Apache Tomcat[1]. V systému Windows je instalace tohoto nástroje usnadněna pomocí instalačního souboru. Podrobný návod pro konfiguraci tohoto nástroje v sytému Windows se nachází zde[2].

V systému Linux lze k instalaci využít nástroje pro správu balíčků. Nejjednodušší zavedení WAR souboru je s pomocí GUI manageru, je však nejprve zapotřebí ho nainstalovat a nastavit. Stažení tohoto manageru lze na Debianu či Ubuntu dosáhnout příkazem `sudo apt-get install tomcat7-admin`. Po instalaci je ještě zapotřebí upravit soubor */etc/tomcat7/tomcat-users.xml* tak, aby vypadal následovně:

```
<tomcat-users>
        <user username="admin"password="password"roles="manager-
gui,admin-gui"/>
</tomcat-users>.
```

Přičemž username a password slouží jako přihlašovací údaje do aplikace a mohou být upraveny dle libosti. Aby byly tyto změny rozpoznány, je nutné Tomcat restartovat, toho lze dosáhnout příkazem `sudo service tomcat7 restart`. Po obnovení Tomcatu je možné se do manageru přihlásit zadáním adresy *http://localhost:8080/manager* internetového prohlížeče. Po zadání přihlašovacích údajů se zobrazí tabulka s běžícími aplikacemi a také možnost zavedení nových balíčků.

Po stisku tlačítka *Choose File* a vybrání balíčku WAR, který má cestu *recommender/target/recommender.war* (pokud složka *target* neexistuje, je zapotřebí nejdříve spustit příkaz `mvn package`), je možné ho zavést pomocí tlačítka *Deploy*. Pokud vše proběhne v pořádku, zobrazí se nad tabulkou s aplikacemi zpráva OK. Poté lze aplikaci spusit zadáním adresy *http://localhost:8080/recommender/* do prohlížeče.

[1] http://tomcat.apache.org/

[2] http://tomcat.apache.org/tomcat-7.0-doc/windows-service-howto.html

D Uživatelská příručka

Jak již bylo zmíněno, aplikaci lze po zavedení balíčku WAR spustit zadáním adresy *http://localhost:8080/recommender/* do prohlížeče (viz obrázek D.1). Pro správné zobrazení stránky je doporučeno využít prohlížeče Google Chrome nebo Mozilla Firefox.

Obrázek D.1: Webová aplikace

Při zobrazení stránky se pošle požadavek pro získání informací o uživatelích a filmech. Po chvíli jsou tato data zpracována a zobrazí se uvnitř rolovací nabídky. Na horní části stránky se nachází panel, který umožňuje výběr uživatele a vypočtení návržených doporučení pomocí tlačítka a posuvné lišty.

Při výběru uživatele v levé části horního panelu, se zobrazí filmy, které uživatel ohodnotil. Pod horní lištou se vyskytuje další rolovací nabídka, která zobrazuje abecední seznam všech filmů a umožňuje přidání nových hodnocení. Po vybrání jednoho z filmů, ohodnocení určitým počtem hvězdiček a stisknutí tlačítka *OHODNOŤ* je přidáno hodnocení filmu. K jednomu filmu lze vyjádřit několik hodnocení, nicméně počítáno bude pouze to poslední.

V horním panelu lze pomocí posuvné lišty vybrat kolik doporučení má být navrženo a po stisku tlačítka *DOPORUČ* proběhne předání požadavku doporučovacímu systému, který se pokusí navrhnout doporučení. Tato akce může trvat několik desítek vteřin. Důvodem může být aktualizace datového

modelu na základě nových dat. Může se stát, že systém nedokáže navrhnout žádný film, a to v případě, že uživatel ohodnotil malé množství filmů. Také se může stát, že se uživatel zatím vůbec nenachází v databázi. Nová doporučení jsou totiž zahrnována do výpočtů až po zadání dvaceti hodnocení, tím dojde k obnovení datového modelu.

Přehled zkratek

CSS	Cascading Style Sheets, kaskádové styly
CSV	Comma-separated values, hodnoty oddělené čárkami – souborový formát
HTML	HyperText Markup Language, značkovací jazyk
HTTP	Hypertext Transfer Protocol, internetový protokol
JSON	JavaScript Object Notation, JavaScriptový objektový zápis – datový formát
SAX	Simple API for XML, parsovací nástroj pro soubory XML
WAR	Web application Archive, souborový formát pro webové aplikace
XML	Extensible Markup Language, rozšiřitelný značkovací jazyk