Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Adaptibilní systém pro doporučování obsahu

Bc. Jan Bouchner

Vedoucí práce: Ing. Jaroslav Kuchař

8. května 2014

Poděkování

Chci upřímně poděkovat všem, kteří mi věnovali čas, když jsem potřeboval pomoc při psaní této diplomové práce. Především vedoucímu práce Ing. Jaroslavu Kuchaři za správné směrování a cenné rady, které výrazně napomohly ke vzniku této práce. Děkuji také své rodině a všem přátelům za bezvýhradnou podporu během celých mých studií.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 8. května 2014 .....

Odkaz na tuto práci

Bouchner, Jan. *Adaptibilní systém pro doporučování obsahu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords Multi-Armed Bandit, recommendation systems, ensemble, REST, machine learning, Apache Solr, ZeroMQ

Abstrakt

Tato diplomová práce popisuje návrh a vývoj adaptibilního systému schopného vhodně volit a kombinovat algoritmy pro doporučování obsahu. První část práce je zasvěcena zkoumání teorie související s pokročilejšími existujícími systémy, na základě výsledků tohoto zkoumání je provedena volba vhodné učící strategie a stanovení konkrétních požadavků na systém. Druhá část práce je praktická a popisuje návrh systému, obecného rozhraní pro komunikaci, použité technologie a programátorské postupy, jakými bylo dosaženo výsledného systému. Vyvinutými částmi jsou: adaptibilní systém, sada základních algoritmů pro doporučování obsahu, RESTful API a klientská aplikace, jež na závěr umožnila provést se systémem několik experimentů. Všechny části aplikace byly implementovány v programovacím jazyce Java.

Klíčová slova Multi-Armed Bandit, doporučovací systémy, ensemble, REST, strojové učení, Apache Solr, ZeroMQ

Obsah

Seznam obrázků

Seznam tabulek

Úvod

> We are leaving the age of information and entering the age of recommendation.
> —Chris Anderson, 2004 [15]

Když se člověk před nástupem digitálního věku snažil mezi množstvím nabízených informací nalézt takové, které by pro něj byly nějakým způsobem užitečné, musel se často spíše než na vlastní úsudek spoléhat na doporučení svých známých a přátel nebo na obecnou oblibu. Tu ale více než cokoliv jiného určoval aktuální společenský trend. Tato situace úzce souvisela s nedostatečným množstvím dat.

S rozvojem internetu začalo narůstat i množství dostupných informací[1]. Nová data jsou produkována takřka každou sekundu, což vede k opačnému extrému – k jejich setřídění nám nyní nezbývá čas. Ceněným uměním je tak schopnost vytěžit maximum užitečných informací a ty posléze využít k nabytí nových znalostí.

K těmto účelům přispěl velkou měrou rozvoj disciplíny zvané jako informační filtrování, čímž je myšlena selekce a redukce informací. Odtud už je jen krok k personalisovanému doporučování obsahu, které se v posledních několika letech rozmáhá zaváděním *doporučovacích systémů*.

Hned ze začátku práce jsem si dovolil citovat bývalého šéfredaktora časopisu Wired Chrise Andersona (článek *The Long Tail [15]*, který shrnul nastalou situaci týkající se vývoje informací do jedné stručné věty. Je důležité vzít v potaz, že Anderson toto prohlášení učinil v roce 2004.

Vývoj v oblasti informačních technologií je neúprosný a každý další rok je ve znamení výrazných pokroků ve vývoji všech oborů, těch zaměřených na doporučování nevyjímaje. Od vstupu do *doporučovací éry* uplynulo deset let a situace je nyní taková, že s pouhou znalostí ověřených doporučovacích algoritmů lze sice ještě stále dosáhnout měřitelných úspěchů (například nasa-

[1] Organizace IDC došla v článku *Extracting Value from Chaos* k závěru, že objem světových dat se každé dva roky zdvojnásobuje. [27]

zením některého z doporučovacích algoritmů do programové logiky menšího e-shopu), pro globálnější potřeby je to však již poněkud málo.

Tajemstvím úspěšných systémů, jakými disponují například společnosti Google či Amazon, je aplikovaný výzkum a vývoj opírající se nejčastěji o teorii pravděpodobnosti a statistiky, strojového učení a dalších oblastí umělé inteligence.

Budoucnost je tedy v systémech, které dokáží sestavovat svá doporučení na základě pokročilejší analýzy, a které jsou schopny přizpůsobit svá doporučení uživatelům na míru.

Jedním takovým přístupem je kombinování více doporučovacích metod dohromady a sestavování výsledků na základě zaznamenávaného vývoje.

Motivace

Představme si nyní podobný systém jako rádce, který je schopen v každém okamžiku rozhodnout, co je pro uživatele, kteří žádají systém o radu, nejvhodnější. Rádce těmto uživatelům sděluje informace o typu doporučení (doporučovacím algoritmu), jehož použití by v *danou chvíli* mohlo vést k maximalizaci užitku z doporučeného obsahu.

Za *danou chvíli* je považována kombinace několika vstupních proměnných, například denní doba, zařízení, ze kterého je žádáno o obsah, den v týdnu či geografická lokace uživatele.

Pokud se uživatel rozhodne následovat rádce, s největší pravděpodobností obdrží doporučení, se kterým bude spokojen. Svou spokojenost poté vyjádří například tím, že u jedné z doporučených položek zažádá o bližší detaily nebo ji jiným způsobem ohodnotí.

Jinými slovy, uživatel zašle rádci zpětnou vazbu týkající se toho, jak kvalitní bylo dané doporučení.

Rádce sám neprovádí žádné operace (dokud není explicitně vyzván). Ve své paměti si uchovává pouze vnitřní stav, na základě kterého rozhoduje o budoucích doporučeních. Díky průběžné analýze a přepočítávání zpětné vazby je schopen pružně reagovat na situace, kdy dojde k náhlé změně preferencí nebo vyvstanou jiné neočekávané události, jež by měly za následek dlouhodobější doporučování nechtěného obsahu.

Cíle práce

Nejdůležitějším cílem této práce je návrh a implementace rádce popsaného výše, neboli adaptibilního systému, který bude schopen automaticky a vhodně kombinovat metody pro doporučování obsahu, rozdávat rady a přizpůsobovat se podmínkám. Součástí práce je též výběr a implementace vhodné sady doporučovacích algoritmů, které budou použity pro kombinování.

Vzniklý systém bude schopen zpracovávat zpětnou vazbu od uživatelů, která se bude týkat kvality následovaného doporučení. Zpětná vazba bude fungovat na principu odměny za dobré doporučení a trestu za špatné. Takovýto přístup logicky povede k ovlivňování preferencí při kombinování v čase.

Nezbytnou součástí diplomové práce je osvojení základních pojmů a pravidel z oblasti strojového učení, teorie pravděpodobnosti a statistiky. Těchto znalostí bude využíváno zejména v části práce týkající se predikce vhodného algoritmu a dopadu zpětné vazby na vnitřní stav a další vývoj systému.

Vzhledem k povaze projektu bude nutné navrhnout obecné rozhraní jak pro manipulaci s doporučovacími algoritmy, tak pro adaptibilní systém a všechny jeho součásti.

Funkcionalita vyvinutého řešení bude na závěr ověřena implementací modelové úlohy simulující interakci uživatelů s adaptibilním systémem.

K dosažení těchto cílů bude zapotřebí navrhnout a vyvinout následující části:

- **Adaptibilní systém pro doporučování obsahu.**
- **Sada základních algoritmů určených k doporučování.**
- **RESTful API pro manipulaci s doporučovacími algoritmy a systémem.**
- **Klientská aplikace pro simulaci modelové úlohy.**

Struktura práce

Tato práce je strukturována do 3 různých kapitol. Kapitoly jsou řazeny v pořadí, v jakém probíhaly jednotlivé fáze vývoje.

Kapitola Teoretická část Popisuje trendy používané současnými doporučovacími systémy a zkoumá algoritmické a matematické postupy, které by mohly být využity při návrhu strategie adaptibilního systému.

Kapitola Praktická část Shromažďuje veškeré požadavky kladené na systém, dále se zabývá návrhem architektury, detailními návrhy serverové i klientské části, rozhraním zastřešujícím komunikaci těchto částí a výběrem základní sady algoritmů, které budou využity k doporučování obsahu. Popisuje též postupy užité při vývoji aplikace na základě provedené analýzy a návrhu.

Kapitola Experimentální část Shrnuje poznatky nabyté z prováděných experimentů se systémem a stanovuje pravidla pro nejlepší použití.

Kapitola Zhodnocení aplikace Hodnotí vzniklou aplikaci a zamýšlí se nad dalším kroky ve vývoji.

Kapitola 1

Teoretická část

1.1 Doporučování obsahu

Téměř všechna existující doporučení sdílejí stejnou myšlenku – zaujmout či upozornit na konkrétní věc, která by pro nás jako uživatele mohla být nějakým způsobem užitečná či přínosná. Spousta elektronických obchodů, renomovaných aukčních domů, ale též serverů se zábavou na doporučování doslova staví své podnikání.

1.1.1 Analýza vybraných systémů

Na internetu lze narazit na desítky, ba i stovky systémů zapojených do reálného provozu. Záměrně jsem se proto snažil vybral pouze malý zlomek z nich, který by ale zároveň pokrýval většinu typů doporučovaného obsahu (produkty, články, zábava apod.).

U těchto systémů jsem se snažil identifikovat, jaké postupy jsou používány v pozadí jejich doporučování a jaké konkrétní metody využívají. Vzhledem k povaze řešeného problému jsem též zkoumal znaky týkající se kombinování metod, jimiž bych se mohl nechat inspirovat při návrhu svého systému.

1.1.1.1 Amazon.com

Amazon.com, Inc.[2] je jedním z největších a nejstarších internetových prodejců. Společnost začínala jako online knihkupectví, postupem let však zařadila do své prodejní nabídky i hudební a filmové nosiče, software, elektroniku, nábytek a spoustu dalšího zboží včetně vlastní spotřební elektroniky v podobě čtečky elektronických knih a tabletů Kindle.

Firmu lze řadit mezi průkopníky doporučování na internetu. Jako jeden z prvních internetových prodejců začala svým zákazníkům nabízet ke koupi výrobky na základě nákupu jiných uživatelů s podobným osobnostním profilem.

[2] `http://www.amazon.com`

Samotný doporučovací systém společnosti je založen na více zdrojích dat:

- Porovnávání uživatelem prohlížených položek a položek umístěných uvnitř nákupního košíku s položkami, které se společně s těmito prohlíženými položkami v minulosti často prodávaly[3].
- Uchovávání uživatelských hodnocení nabízených položek.
- Zaznamenávání nákupní historie.
- Sledování spousty dalších postupů, jako například vyhodnocování demografických informací (dle doručovací adresy), zaznamenávání pohybu po stránce (jaké všechny položky si uživatel prohlédl, než vložil jednu konkrétní do nákupního košíku) nebo sledování prokliků[4] z marketingových e-mailů obsahující odkazy na zboží [3]. Tyto všechny informace navíc ještě vzájemně kombinují.

Kromě výše uvedeného využívá společnost metod kolaborativního filtrování ve variantě *item-to-item*.

Díky vzájemné kombinaci všech těchto pravidel a metod tak například fanoušek moderních technologií při návštěvě stránek nalezne především odkazy na technologické novinky všeho druhu, zatímco mladá matka bude mít v nabídce té samé stránky ve větší míře zastoupeno dětské zboží.

Výše jsou samozřejmě popsány pouze základní principy. Doporučovací systém společnosti je velmi komplexní a detaily algoritmu jsou drženy jako obchodní tajemství. K nahlédnutí jsou ale patenty, např. *Personalized recommendations of items represented within a database* [25] nebo *Collaborative recommendations using item-to-item similarity mappings* [28].

1.1.1.2 Netflix

Netflix, Inc.[5] je velká mediální společnost. Strategicky významným byl pro ni rok 2007. Tehdy byla nabídka jejích služeb rozšířena o filmy přenášené přes stream[6] [7]. Nyní společnost nabízí obsah v podobě filmů a seriálů pro většinu v dnešní době používaných platforem jako PC, Mac, PlayStation3, Wii, Xbox a také pro mobilní telefony a tablety.

Vzhledem k tomu, že firma staví své podnikání na tom, že její zákazníci si platí za konzumaci zábavy, je v jejím vlastním zájmu, aby těmto uživatelům sledujícím filmy a seriály nabízela automaticky další obsahově či žánrově

[3] affinity analysis – nacházení spojení mezi odlišnými položkami. Základním příkladem budiž vztah mezi šamponem a kondicionérem. Kupující je většinou používá v ten samý čas [12]. Při nákupu jednoho by mohl mít tedy zájem i o druhý.

[4] Jako proklik se označuje takové kliknutí na odkaz, které uživatele dovede na cílovou stránku [9].

[5] `https://www.netflix.com`

[6] Snímek se fyzicky nenachází na koncovém zařízení, ale přehrává se přímo ze serveru poskytovatele .

podobné, zkrátka takové, jenž budou co možná nejvíce lahodit jejich vkusu (dle [6] pochází $\frac{2}{3}$ zapůjčených filmů na Netflix z předchozího doporučení). Úspěšné podnikání společnosti je tak přímo závislé na tom, jak kvalitní má doporučovací systém.

Netflix Prize Za tímto účelem vyvinula společnost vlastní systém *Cinematch*. Postupný vývoj na poli doporučovacích systémů však přivedl společnost na otázku, zda není možné vyvinout systém, jenž by dokázal Cinematch v oblasti předpovídání filmového vkusu porazit. Začátkem října 2006 byla proto vypsána soutěž známá jako *Netflix Prize*. Týmu, kterému by se podařilo zlepšit dosavadní výsledky systému Cinematch alespoň o 10 procent, by byla přidělena odměna ve výši 1 milion amerických dolarů.

Do soutěže byla uvolněna sada testovacích dat obsahující:

- ID uživatele
- ID filmu
- hodnocení na intervalu $\langle 1, 5 \rangle$
- datum uskutečnění hodnocení

Data obsahovala 100 480 507 hodnocení pro 17 770 filmů od 480 189 uživatelů. Uvolněna byla ještě další sada testovacích dat obsahující stejné informace, jen s vynecháním uživatelských hodnocení. Cílem pak bylo predikovat tato chybějící hodnocení opět na intervalu $\langle 1, 5 \rangle$ [31].

Celková výhra byla udělena až v roce 2009 (do té doby sice docházelo k průběžnému zlepšování výsledků, nikdy však nebylo dosaženo požadovaného zlepšení o 10 procent) týmu *BellKor's Pragmatic Chaos*, který vznikl spojením tří do té doby samostatně soutěžících týmů.

Vítězný tým dosáhl cíle pomocí pokročilých technik strojového učení.

Při tom všech bylo zjištěno několik zajímavých poznatků. Například to, že hodnocení každého filmu je silně subjektivní záležitostí, kterou je velmi obtížné programově předpovědět. Ukázalo se také, že ve velké míře záleží na tom, zda uživatel hodnotí právě dosledovaný film nebo film, který zhlédl již před delší dobou. Velkou roli hraje i nálada uživatele v průběhu dne (zda film sleduje například o víkendu, kdy má volno) [1]. Lze tedy opět mluvit o uživatelském kontextu vstupujícího do doporučování.

Výsledný algoritmus, jenž získal cenu, vznikl kombinováním zhruba stovky menších algoritmů. S trochou nadsázky lze tedy prohlásit, že jednou z hlavních taktik pro kvalitní doporučení je *použití tolika algoritmů, na kolik jen máme kapacity.*

1.1.1.3 Mendeley

Mendeley[7] je systém určený k doporučování vědeckých článků využívající jako svou výpočetní vrstvu technologii Apache Mahout[8]. Cílem systému je spojovat dohromady výzkumníky a jejich data. Svým uživatelům napomáhá v organizaci výzkumu, umožňuje jim navázat potenciální spolupráci s dalšími uživateli aplikace a napomáhá též k objevení nových podnětů pro vlastní vědeckou práci. Uživateli této aplikace jsou přední světové university jako University of Cambridge, Stanford University, MIT či University of Michigan. Data pro aplikaci pocházejí z uživatelských importů i z externích importů skrze různé katalogy prací.

Projekt sám se v jedné ze svých prezentací [5] přirovnává k největší hudební databázi na internetu – last.fm[9]. Ta funguje na principu, že potenciální uživatel si na svém počítači nainstaluje desktopovou aplikace, následně s již instalovanou aplikací začne poslouchat ve svém přehrávači hudbu a tím je zahájeno automatické odesílání informace o skladbě (interpret, žánr) na server last.fm. Podle dat odeslaných na server jsou uživateli v budoucnu doporučovány další skladby.

Mendeley tuto analogii vysvětluje tak, že hudební knihovny jsou v jeho případě výzkumné knihovny, role interpretů zastávají jednotliví výzkumníci, hudební skladby jsou pak jimi publikované články a jednotlivé hudební žánry reprezentují vědecké disciplíny.

Doporučení jsou dle dostupných informací [5] generována dvojím způsobem.

Kolaborativní filtrování Používá se pro personalisovaná doporučení. Podporována je jak user-based, tak item-based varianta.

Filtrování založené na obsahu Používá se k nalezení souvisejících výzkumů, například k nalezení článku ze stejné výzkumné kategorie nebo článku majícího podobný název.

Uživatelé vyjadřují svůj zájem či nezájem o každou z doporučených položek zpětnou vazbou. Ta je dvojího typu:

Accept vyjadřuje, že uživatel s daným doporučením souhlasí nebo pro něj bylo nějakým způsobem užitečné.

Remove vyjadřuje nevhodně zvolené doporučení. Uživatel touto volbou dává najevo, že podobné doporučení by příště již raději nedostal.

[7] `http://www.mendeley.com`
[8] https://mahout.apache.org
[9] http://www.last.fm

1.1.1.4 Google News

Vlastní platformu pro doporučování obsahu vytvořila v rámci svého vývoje též společnost **Google, Inc.** [10]. Její výzkumníci se v práci [29] zabývali vývojem prediktivního systému pro personalisované doporučování zpráv na webu Google News [11].

Přihlášeným uživatelům, kteří mají v prohlížeči explicitně povolen záznam historie prohlížení, jsou generována doporučení založená na dřívějším zkoumání různých profilů (sestavených z pozorování chování uživatelů na stránce).

K porozumění, jak se mění zájem uživatelů o zprávy v čase, tak napomohla výzkumníkům analýza logů (záznamy chování anonymních uživatelů na stránce). Na základě této analýzy byl posléze vyvinut bayesovský framework pro předvídání zájmu uživatelů o publikované novinky.

Kombinace tohoto mechanismu filtrace informací s již existujícím mechanismem využívajícího principů kolaborativního filtrování vedla ke vzniku systému generujícího personalisované zprávy. Vzniklá kombinace byla nasazena do reálného provozu a následné experimenty prokázaly, že kombinováním metod došlo ke zlepšení kvality doporučování.

1.1.1.5 Výzkum

Také současný výzkum v oblasti doporučovacích systémů nezahálí. Uvedu zde pro příklad dvě populární akce, jejichž náplní či součástí je problematika doporučování a doporučovacích systémů.

- **ACM RecSys conference.** [12] Konference je předním mezinárodním fórem pro prezentaci nových výsledků výzkumu a postupů na poli doporučovacích systémů. RecSys sdružuje hlavní mezinárodní výzkumné skupiny a též mnoho předních světových společností operujících na trhu e-commerce. Nabízí také doprovodný program v podobě zvaných přednášek, konzultace týkající se problematiky RecSys a sympózium studentů doktorských programů.

  Zajímavé prezentace jsou volně dostupné na internetu, dá se tedy načerpat spousta inspirace k vlastnímu výzkumu. Mě osobně velmi zaujala prezentace [13] z posledního ročníku konference (Hong Kong, 2013), se kterou vystoupil Torben Brodt, jeden z klíčových řečníků. V prezentaci popisuje tzv. *Open Recommendation Platform* 1.1.1.6.

- **ICWSM: Weblog and Social Media.** [14] The International AAAI Conference on Weblogs and Social Media je mezinárodní konference, na

[10] http://www.google.com/about/company
[11] http://news.google.com
[12] http://recsys.acm.org
[13] http://www.slideshare.net/d0nut/open-recommendation-platform
[14] http://www.icwsm.org/2014

které se střetávají výzkumní pracovníci z oblasti počítačových a společenských věd. Konference je pořádaná za účelem sdílení znalostí, diskutování o nápadech a výměny informací. Probíranými body jsou psychologické a sociální teorie, výpočetní algoritmy pro analýzu sociálních médií a jedním z témat jsou i doporučovací systémy.

O tématu doporučování se též píše mnoho článků a každý rok vznikne několik disertačních prací. Dle dostupných informací z ACM RecSys Wiki [15] jich jen za poslední 4 roky bylo přes 50.

1.1.1.6 Open Recommendation Platform

Open recommendation Platorm (zkr. ORP) je projekt společnosti **plista**[16] prezentovaný na posledním ročníku konference ACM RecSys 2013 v Hong Kongu.

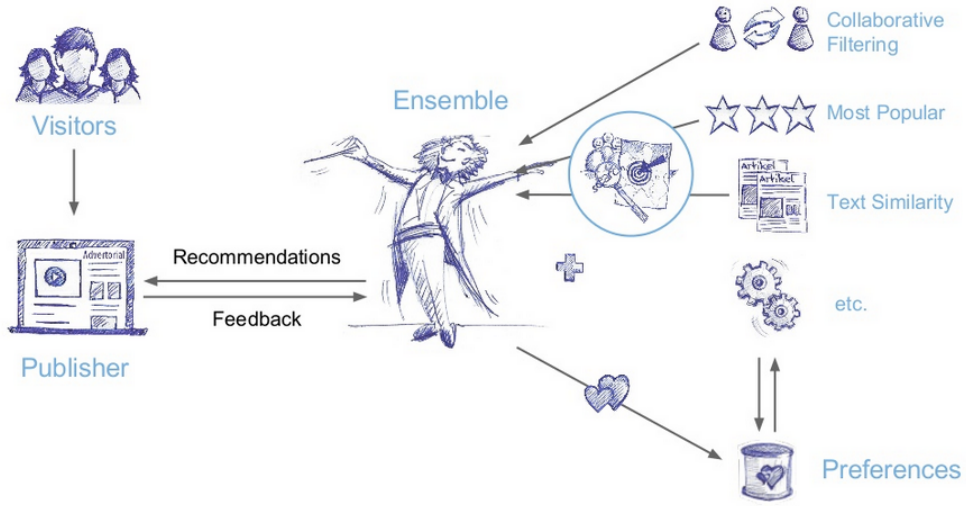
Motivací ORP bylo dosáhnout lepších výsledků doporučování díky přístupu založeném na kombinaci více metod doporučování obsahu společně se zapojením kontextu uživatele (informace čerpané převážně z HTTP hlaviček). Kontextovými atributy jsou například:

- IP adresa, která může prozradit geologickou lokaci uživatele,
- denní doba, kdy uživatel přistupuje ke stránce,
- agent prohlížeče informující o zařízení, ze kterého bylo k obsahu přistupováno (mobilní telefon, PC),
- referer pro zjištění způsobu přístupu (přístup z vyhledávání nebo přímý přístup).

Řešení ORP spočívá v utvoření databáze kontextových atributů (jeden atribut pro každou denní hodinu, pro každou kategorii, stát atd.), kdy je pro každý atribut udržován seřazený seznam doporučovacích metod dle úspěchu, kterého dosáhly v případě použití v daném kontextu. Tyto atributy lze libovolně kombinovat a dosáhnout tak nejlepšího doporučení v daném kontextu.

Abstraktní pohled na ORP znázorňuje obrázek 2.8, na kterém je systém zachycen jako *Ensemble*. Ensemble je schopen pro příchozího uživatele (kombinace kontextových atributů) vybírat z kolekce doporučovacích algoritmů ten nejvhodnější, jenž pak použije k publikování obsahu pro tohoto uživatele (uživatelé jsou značení jako *Visitors*). Po obdrženém doporučení zasílá uživatel systému zpětnou vazbu, díky které dochází k rekalkulaci preferencí (a přepočítání úspěšnosti metod v jednotlivých použitých atributech), což ovlivní výběr nejlepšího algoritmu v dalších doporučeních.

[15] http://www.recsyswiki.com/wiki/List_of_recommender_system_dissertations
[16] https://www.plista.com



Obrázek 1.1: Abstraktní pohled na systém společnosti plista. Zdroj: [18]

V prezentaci bylo zmíněno několik užitečných rad ohledně toho, co všechno by měl systém podobného typu umět. Je zde uvedena potřeba rychlého síťového protokolu, rychlé fronty zpráv a rychlého úložiště pro data.

Důležitá je informace, že k dosažení výběru nejlepšího algoritmu se používá strategie *Multi-Armed Bandit* 1.2.1.15 v bayesovské variantě, což je i jedna ze strategií, kterou mi na úvodní schůzce doporučil vedoucí této práce.

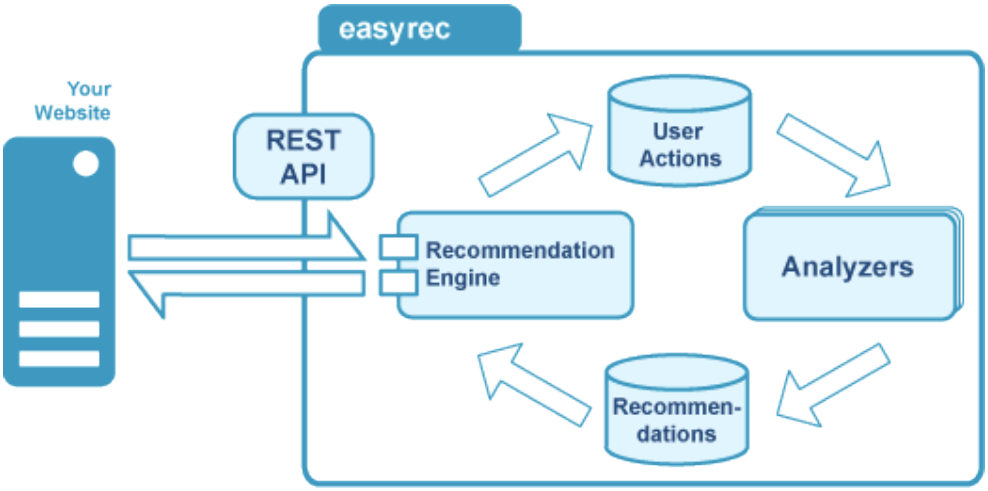
1.1.1.7 easyrec

Technická knihovna společnosti IBM obsahuje přehledný seznam [10] několika doporučovacích systémů, z nichž převážná část vznikla jakou součást univerzitního výzkumu.

Z tohoto seznamu se mi velmi zajímavým jevil systém **easyrec**[17]. Jedná se o open source webovou aplikaci napsanou v programovacím jazyce Java, která nabízí personalisovaná doporučení prostřednictvím RESTful webových služeb. Díky vystavenému REST API[18] je vývojáři umožněno napojit svou aplikaci, kterou píše v libovolném jazyce, na systém easyrec a využívat nabízených funkcionalit. Lze tak například zasílat uživatelské akce jako prohlížení stránky, provedení nákupu či hodnocení položky nebo žádat o doporučení. Všechny provedené uživatelské akce se ukládají do databáze easyrec.

K doporučovacím metodám je možné přistupovat prostřednictvím specifických endpointů, například *zboží související s danou položkou* nebo *ostatní uživatelé prohlíželi též tyto položky* [10].

[17] `http://easyrec.org/recommendation-engine`
[18] REST API systému easyrec: `http://easyrec.org/implement`



Obrázek 1.2: Abstraktní model open source systému easyrec. Zdroj: [2]

Systém na pozadí provádí analytické operace, disponuje databázi asociačních pravidel a nabízí plnou podporu pro online i offline doporučování. Uživatelským aplikacím připojujícím se k systému jsou v odpovědích generována vyžádaná doporučení (viz obrázek 1.2).

1.1.2 Shrnutí poznatků z analýzy

Z výše uvedených příkladů v sekci 1.1.1 lze vypozorovat určité vlastnosti (shrnutí v tabulce 1.1) a zformulovat několik závěrů:

- O uživatelích je vedena spousta záznamů, například historie jejich chování na stránkách.
- Velký důraz je kladen na zpětnou vazbu (vyjádření zájmu uživatele o doporučený obsah), ať už formou přečtení článku, ohodnocení položky apod.
- Stále je využíváno osvědčených metod kolaborativního filtrování a filtrování na základě obsahu.
- Síla není v použití jednoho konkrétního algoritmu, ale v kombinaci více metod a přístupů dohromady.
- Většina systémových výpočtů běží na pozadí v tom samém čase, který uživatel tráví prohlížením stránek.
- Problematika kombinování metod má silnou závislost na strojovém učení a teorii pravděpodobnosti a statistiky (především bayesovské).
- Realizace RESTful API je vhodný způsob komunikace s doporučovacím systémem (ověřeno praxí).

Systém	Vlastnosti
Amazon.com	1. Kolaborativní filtrování užívající podobnostní mapování item-to-item [28]. 2. Personalisovaná doporučování získávaná z databáze informací o uživatelích [25]. Informace jsou mezi sebou kombinovány za účelem vytvoření nejlepšího doporučení.
Netflix	1. Predikce uživatelských hodnocení založená na technikách strojového učení 1.2.2. 2. Pro doporučování je využíváno více jednotlivých algoritmů a jejich kombinování. Velmi záleží na tom, zda uživatel hodnotí film ráno, večer, o víkendu, kdy má volno a podobně.
Mendeley	1. Doporučení založená na kolaborativním filtrování ve variantách user-based i item-based. 2. Doporučení založená na podobnosti obsahu (podobnost článků, nadpisů, klíčových slov apod.) 3. Důraz na pozitivní a negativní zpětnou vazbu. 4. Výpočetní vrstva pro doporučování postavená nad Apache Mahout 2.2.4.1.
Google News	1. Kombinování informací získaných z předchozího chování na stránce s technikami kolaborativního filtrování.
ORP	1. Kombinování více doporučovacích metod společně s kontextem uživatele. 2. Přítomnost mezivrstvy v podobě rádce starajícího se o obsluhu uživatelů a jejich doporučení. 3. Silný důraz na zpětnou vazbu a následné přepočítávání budoucích šancí doporučení. 4. Ke kombinaci je využíváno strategie Multi-Armed Bandit 1.2.3 v bayesovské variantě 1.2.4.
easyrec	1. Veškeré uživatelské akce (provedení akce, zpětná vazba, žádost o doporučení konkrétním algoritmem) s doporučovacím systémem probíhají skrze komunikaci s RESTful API.

Tabulka 1.1: Shrnutí poznatků o jednotlivých systémech vyplývající z provedené analýzy.

Existujících řešení je tedy celá řada. Každé z těchto řešení je přitom využíváno k vlastnímu účelu a není snadno přenositelné.

Potvrdila se domněnka, že době, kdy obchodníkům stačilo na svůj elektronický obchod nasadit jednoduchý algoritmus kolaborativního filtrování, již odzvonilo a budoucnost patří systémům schopným predikce a přizpůsobení vývoje v reálném čase.

1.2 Teorie adaptibilního systému

1.2.1 Minimální teoretický základ

Ještě předtím, než se pustím do popisu navrhované strategie, zopakuji zde některé pojmy týkající se strojového učení, teorie pravděpodobnosti a statistiky. Jedná se o minimální základ potřebný k pochopení postupů popisovaných dále v této sekci.

1.2.1.1 Strojové učení

Strojové učení je vědecká disciplína (jedna z větví oboru umělé inteligence), jež se zabývá tím, jak se má počítač přizpůsobit určité situaci, aniž by byl pro danou situaci explicitně naprogramován. Detailněji v sekci 1.2.2.

1.2.1.2 Agent

Agent je speciální autonomní program, který jedná samostatně a nezávisle bez vedení uživatele. Jeho úkolem je komunikovat s okolím a interagovat v závislosti na okolních podnětech. Dalšími důležitými vlastnostmi jsou schopnost příhodně reagovat na danou situaci a též proaktivně vykonávat činnost a dosahovat cíle prostřednictvím vlastní iniciativy.

1.2.1.3 Zpětná vazba

Zpětnou vazbou je označován proces, kterým na základě obdržených informací (například při komunikaci s nějakým systémem) a reakcí na ně ovlivňujeme budoucí podobu informací.

Rozlišujeme dva typy reakcí, a to pozitivní a negativní zpětnou vazbu. Téma má blízko k psychologickému zkoumání, jakým způsobem lze ovlivnit lidské chování, když do výchovy přidáme princip odměn a trestů.

1.2.1.4 Explorace vs. exploatace

Explorace Nacházení nových neprozkoumaných oblastí. Agent nevyužívá svých dosud načerpaných znalostí a zkouší nové akce.

Exploatace Využívání stávajících znalostí. Agent provádí akce, o kterých ma základě pokusů z minulosti ví, že mu přinášejí užitek. Pokud by dlouhodobě zůstal v tomto procesu, hrozí mu uváznutí v lokálním extrému.

K nalezení optimální strategie je tedy nutné provést nějaký kompromis.

1.2.1.5 Základy teorie pravděpodobnosti

Nejprve uvedu několik pojmů z teorie pravděpodobnosti, se kterými budu v následujícím textu pracovat.

Pravděpodobnostní prostor Pravděpodobnostní prostor prováděného náhodného pokusu je tvořen trojicí $(\Omega, \mathcal{F}, P)$, kde:

- Ω je prostor elementárních jevů (např. čísla od jedné do šesti na šestistranné hrací kostce), kde elementárním jevem nazýváme libovolný možný výsledek $\omega \in \Omega$
- $\mathcal{F}$ je množina náhodných jevů (potenční množina[19] množiny Ω)
- P je přiřazení pravděpodobnosti jednotlivým jevům z Ω ($P(\Omega) = 1$)

Náhodný jev Náhodný jev A výsledek náhodného pokusu. Příkladem jevu může být například hod mincí a pozorování, zda padla panna nebo orel.

Náhodná veličina Jako náhodná veličina X se označuje způsob přiřazení čísla výsledku náhodného pokusu [17].

Náhodná veličina na pravděpodobnostním prostoru $(\Omega, \mathcal{F}, P)$ je tedy funkce $X : \Omega \to R$, která každému $\omega \in \Omega$ přiřadí $X(\omega)$ a pro kterou platí podmínka měřitelnosti:

$$\{X \leq x\} = \{\omega \in \Omega : X(\omega) \leq x\} \in \mathcal{F}, \forall x \in R$$

Pravděpodobnostní rozdělení náhodné veličiny určuje její distribuční funkce.

1.2.1.6 Distribuční funkce

Distribuční funkce náhodné veličiny X je funkce $F : R \to \langle 0, 1\rangle$ definovaná vztahem

$$F(x) = P(X \leq x) = P(\{\omega \in \Omega : X(\omega) \leq x\}).$$

Vyjadřuje pravděpodobnost, že hodnota náhodné veličiny X nabude hodnoty menší nebo rovné zadané hodnotě (libovolnému $x \in R$). Distribuční funkce určuje rozdělení pravděpodobnosti.

[19] Potenční množina množiny X je množina obsahující všechny podmnožiny množiny X

1.2.1.7 Kvantilová funkce

Podobně jako distribuční funkce, i kvantilová funkce se týká rozdělení pravděpodobnosti. Lze ji považovat za funkci inverzní k distribuční funkci, neboť zatímco distribuční funkce $y = F(x)$ vyjadřuje pravděpodobnost, s jakou bude hodnota náhodné veličiny X menší nebo rovna $x \in R$, výsledkem kvantilové funkce $x = F(y)^{-1}$ je hodnota x, pro jakou je výsledek náhodného pokusu se zadanou pravděpodobností y menší nebo roven x. Jinými slovy, hledáme taková x, kterým odpovídá určitá hodnota distribuční funkce $F(x)$. Hodnoty této funkce jsou tedy *kvantily* [4] .

1.2.1.8 Rozdělení pravděpodobnosti

Někdy se též označuje jako distribuce pravděpodobnosti náhodné veličiny X. Rozdělení pravděpodobnosti každému jevu popsanému veličinou X přiřazuje určitou pravděpodobnost. V diskrétním případě přiřazujeme pravděpodobnosti jednotlivým hodnotám, ve spojitém případě pak intervalu hodnot náhodné veličiny.

Rozdělení pravděpodobnosti je celá řada, z diskrétních je známé například *binomické* (n pokusů s rovnocennou pravděpodobností) či *geometrické* rozdělení. Ze spojitých například *normální rozdělení*, *exponenciální rozdělení* nebo *beta rozdělení*.

1.2.1.9 Beta rozdělení

Beta rozdělení $Beta(\alpha, \beta)$ je spojité pravděpodobnostní rozdělení definované na intervalu $\langle 0, 1 \rangle$. Rozdělení má dva vstupní parametry α a β určující tvar. Rozdělení se používá k modelování chování náhodných veličin, které jsou omezené na konečné intervaly.

1.2.1.10 Bayesovská statistika

Jedná se o moderní větev statistiky pracující s podmíněnou pravděpodobností. Základem bayesovské statistiky je známý *Bayesův teorém* (často označovaný jako Bayesova věta) vyjadřující pravděpodobnost hypotézy (H_j) v závislosti na datech (D) a případně modelu (M). Lze pomocí ni stanovit pravděpodobnost, aniž bychom měli k dispozici známá fakta z minulosti. Oproti klasické statistice tedy hypotézy netestujeme, ale provádíme *odhady*.

1.2.1.11 Apriorní pravděpodobnost (prior)

Pravděpodobnost $P(H_j|M)$, označována jako *prior*. Značí to, co víme nebo si myslíme předem, ještě před získáním dat (např. výsledky předchozích experimentů) [38].

1.2.1.12 Aposteriorní pravděpodobnost (posterior)

Pravděpodobnost $P(H_j|D, M)$, označována jako *posterior*. Udává výsledek celého snažení, tedy pravděpodobnost naší hypotézy v závislosti na předchozích znalostech (prior) a současně nových datech [38].

Díky uvedeným pravidlům je možné s každou další objevenou skutečností zpřesňovat pravděpodobnost výchozí hypotézy.

1.2.1.13 Náhodný výběr

Náhodného výběru se využívá k rozpoznání charakteru rozdělení (opakované pokusy dávají za stejných podmínek různé výsledky, které odpovídají hodnotám jednotlivých realizací náhodné veličiny). Jedná se o uspořádanou n-tici $(X_1, X_2, \ldots, X_n)$ náhodných veličin X_i, $1 \leq i \leq n$, které jsou nezávislé a mají stejné rozdělení pravděpodobnosti [8].

Zatímco **náhodným výběrem** označujeme n-prvkovou posloupnost nezávislých náhodných veličin $X_1, X_2, \ldots, X_n$, pojmem **výběr** budeme značit n-prvkovou posloupnost reálných čísel $x_1, x_2, \ldots, x_n$ [34].

1.2.1.14 Statistická inference

Uvažujme pojem *populace*. Populací je myšlena náhodná veličina s rozdělením pravděpodobnosti. Úkolem statistické inference je pak s použitím **výběru** z populace *odhadnout parametr*, neboli číselnou hodnota platící pro celou populaci (tou může být například střední hodnota rozdělení, rozptyl a tak podobně).

Odhadem je myšleno získání číselné hodnoty nebo intervalu hodnot z výběru. Cílem je, aby měl takový odhad blízko skutečné hodnotě parametru.

Rozlišujeme dva typy odhadů, a to bodový, kdy odhadem je jedna hodnota, a intervalový, kdy je odhadem interval hodnot [34].

1.2.1.15 Multi-Armed Bandit

Multi-Armed Bandit je jeden z klasických učících problémů zasahující do teorie pravděpodobnosti. Herní strategie je podobná filosofii tradičního výherního automatu s tím rozdílem, že Multi-Armed varianta má více herních pák. Strategií se detailně zabývá sekce 1.2.3.

S pojmy vysvětlenými výše souvisí strategie tak, že použitím bayesovské varianty provádíme opakovaný výběr z konečného počtu rozdělení a pomocí statistické inference se snažíme maximalizovat průměrnou hodnotu.

Tyto postupy se používají při cílenému doporučování reklamy nebo třeba výběru personalisované úvodní stránky. Což je vlastně analogie k problému, který řeším v této práci.

1.2.2 Význam strojového učení

Jak vyplynulo z rešerše 1.1 existujících řešení, vzhledem k povaze řešeného problému je nutné zaměřit se na metody strojového učení.

Typickým úkolem strojového učení je například vytvořit z dostupných dat model, jenž dokáže:

- predikovat cenu akcií za 6 měsíců (z aktuální výkonnosti společnosti a dostupných ekonomických dat),
- rozpoznat spam od regulérního e-mailu,
- u pacienta hospitalizovaného s infarktem predikovat riziko dalšího infarktu,
- napomoci společnostem zabývajících se internetovou reklamou v rozhodování, kterou reklamní strategii by měly použít k maximalizaci zisku.

Algoritmy strojového učení dělíme do taxonomie (nadtříd a podtříd) založené na požadovaném výsledku nebo typu vstupu, jenž máme k dispozici během učení. Algoritmů je celá řada, zmíním zde alespoň ty nejtypičtější.

Supervised learning [20] je typ učení, které se používá v případě, že máme k našim trénovacím instancím na vstupu korektní výsledky. Pomocí kombinace trénovacích vstupních instancí a jejich požadovaných výsledků lze systém adaptovat na situaci, že dokáže sám předpovídat výsledky pro každé další platné vstupní instance [37].

Využití nachází například v oblastech rozpoznávání řeči či detekci spamu.

Unsupervised learning [21] je již z podstaty absence učitele obtížným problémem. Učení bez učitele se používá k analýzu dat, když nemáme k dispozici trénovací množinu. Pozorovaná data se mají vysvětlit pomocí matematických modelů.

Používá se v oblasti rozpoznávání vzorů [24].

Reinforcement learning [22] je oblast informatiky tykající se chování agentů 1.2.1.2. Jedná se o metodu, při které se agent učí, jakým způsobem má volit akce, aby nalezl optimální strategii pro dané prostředí.

Jedná se o učení bez učitele. Agent sice dostává odezvu, ale přímo z prostředí, takže musí experimentovat a zjišťovat, které stavy jsou nějakým způsobem dobré, a kterým stavům je lepší se vyhnout.

Průzkum probíhá na principu zpětné vazby 1.2.1.3 v podobě odměny za akce, které vedly k dosažení cíle nebo trestu (v opačném případě).

Řeší se zde problém explorace vs. exploatace 1.2.1.4.

[20] učení s učitelem
[21] učení bez učitele
[22] zpětnovazební učení nebo též učení posilováním

Rozlišuje se několik typů zpětnovazebního učení, například tzv. *single-stage* (agent se snaží uplatňovat zpětnou vazbu ihned po každé provedené akci), oproti kterému stojí typ *sekvenční* (agent uplatňuje zpětnou vazbu po obdržení série akcí). Dalšími typy jsou pak například *pasivní* a *aktivní* zpětnovazební učení přizpůsobující svůj vývoj na základě pevně dané strategie, respektive učení se a rozhodování o prováděných akcích za chodu systému [26].

Algoritmus zpětnovazebního učení začíná při svém spuštění ve stavu nevědomí, kdy neví nic o daném prostředí a vědomosti začíná nabývat postupným testováním systému. Postupující dobou běhu (a tím, jak vstřebává data a vyhodnocuje výsledky) se tedy učí rozpoznávat, jaké chování je v daném prostředí nejlepší.

1.2.2.1 Zvolená strategie učení

Za nejvhodnější strategii, jež by byla schopna plnit požadavky kladené na tuto práci, jsem zvolil *algoritmus zpětnovazebního učení*.

O strategii je též možné mluvit jako o *online učení*. Nutno zmínit, že slovem online zde není míněno nic ve smyslu internetu, ale ve smyslu neustále se vyvíjejícího stavu systému. Učící algoritmus v každém kole vykoná nějakou akci, přijme zpětnou vazbu a připíše si daný zisk či ztrátu.

Z matematického hlediska má online učení propojení na klasické online algoritmy, teorii her a teorii pravděpodobnosti.

Díky těmto znalostem tak můžeme navrhovat dynamické pravděpodobnostní systémy, jakým je například již výše zmíněný výherní automat *Multi-Armed Bandit*.

1.2.3 Multi-Armed Bandit algoritmus

1.2.3.1 Princip algoritmu

Základ algoritmu si lze představit tak, že hráč stojí před N výherními automaty (ty jsou podle názvu strategie označeny jako bandité) a v každém kole má možnost vybrat si jeden, na kterém bude hrát.

Strategie je formálně popsána jako skupina výnosových distribučních funkcí 1.2.1.6
$B = \{A_1, A_2, \ldots, A_N\}$, kde N je počet banditů (každý z banditů má tedy přiřazenu právě jednu distribuční funkci vyjadřující pravděpodobnost úspěchu).

Hráč zpočátku nedisponuje žádnými informacemi o průběhu hry, ani o rozložení pravděpodobnosti úspěchu mezi bandity. Maximalizace výhry může dosáhnout pouze tím, že v každém kole vhodně vybere jednoho z banditů.

Kdyby hráč věděl, u kterého z banditů je největší pravděpodobnost výhry, samozřejmě by vždy vybíral právě tohoto. Pravděpodobnosti výher u jednotlivých automatů jsou však neznámé.

Přizpůsobení algoritmu pro potřeby adaptibilního systému Tradiční automatová varianta pracuje s předem definovaným počtem tahů, kterými je

celá hra omezena. Úkolem hráče tedy je nalézt nejlepšího banditu, a to tak rychle, jak jen to je možné, aby jej stihl co nejvíce využít [20].

Vzhledem k povaze mnou řešeného adaptibilního systému je však chování systému zhruba takové:

- Rádce může udělovat rady klidně do nekonečna. Pokud však nebude dostávat pravidelnou zpětnou vazbu o výsledcích svých rad, budou jeho rady čistě náhodné.
- Zpětnou vazbu zasílají uživatelé, kteří jej žádali o radu. Uživatelé mu poskytují informace dvojího typu – zda se řídili při volbě algoritmu jeho radou a zda byli po doporučení obsahu s výsledkem spokojeni či nikoliv.

Popisu toho, jaký vliv má zpětná vazba na vývoj samotného systému, je věnována příslušná podpodkapitola 1.2.4.1.

Návrh strategie učení Návrh učící strategie spočívá v tom, že systém jednotlivé bandity nejdříve testuje 1.2.1.14 za účelem získání znalostí nezbytně nutných pro další vývoj. Jak systém postupně nabývá znalostí, je možné zaměřit se na bandity, kteří poskytují díky zužitkovaným znalostem největší odměnu.

Úkol je komplikován stochastickou povahou banditů vyplývající z odhadů 1.2.1.14. Suboptimální bandita může přinášet spoustu výher, což by nás mohlo přimět uvěřit, že právě tento bandita je tím nejlepším. Stejně tak je ale nutné uvažovat naopak – nejlepší bandita totiž může zpočátku přinášet spoustu proher.

Na místě jsou dvě otázky:

- Měli bychom dávat stále šanci i banditům, u kterých často prohráváme, nebo na ně zanevřít a štěstí zkoušet u jiných?
- Pokud nalezneme banditu, který nám přináší *docela dobré* výsledky, měli bychom se s ním spokojit a nadále maximalizovat svou výhru pouze u něj? Nebo se vyplatí zkoušet i nadále další bandity v naději, že se povede nalézt ještě lepšího?

Opět jsme se dostali k *exploraci a exploataci* 1.2.1.4.

1.2.4 Bayesian Bandits

Nalézt optimální řešení tedy nepatří k triviálním problémům. Systému může trvat i roky, než se k němu dopracuje. Naštěstí existuje spousta *přibližně optimálních* řešení.

Jedním z těchto řešení je algoritmus zvaný *Bayesian Bandits*. Algoritmus přímo souvisí s učením založeným na zpětné vazbě 1.2.1.3.

Bayesovské řešení začíná prior 1.2.1.11 stanovením pravděpodobností výhry pro každého banditu. Hodnoty jsou v rozmezí $\langle 0, 1\rangle$. Jak již bylo řečeno, každého banditu reprezentuje jedna distribuční funkce.

V každém kole, kterých je v případě mnou vytvářeného systému nekonečně (kola jsou závislá na žádostech uživatelů, kteří přistupují k systému – standardní použití Multi-Armed Bandit pracuje s konečným počtem stavů), probíhá následující proces:

1. pro každého z N banditů proveď prior 1.2.1.11 (počty pokusů, výher atd.) výběr náhodné veličiny X_b 1.2.1.5 (vypočti kvantilovou funkci 1.2.1.7).
2. ze získaných dat zvol 1.2.1.13 banditu s největší hodnotou předchozího výběru, například $B = argmax X_b$
3. pozoruj výsledek vrácený banditou B a proveď prior aktualizaci tohoto bandity.
4. vrať se na krok 1

Počáteční prior pravděpodobnost je u každého bandity Beta rozdělení 1.2.1.9 $Beta(\alpha = 1, \beta = 1)$. Toto rozdělení je uniformní.

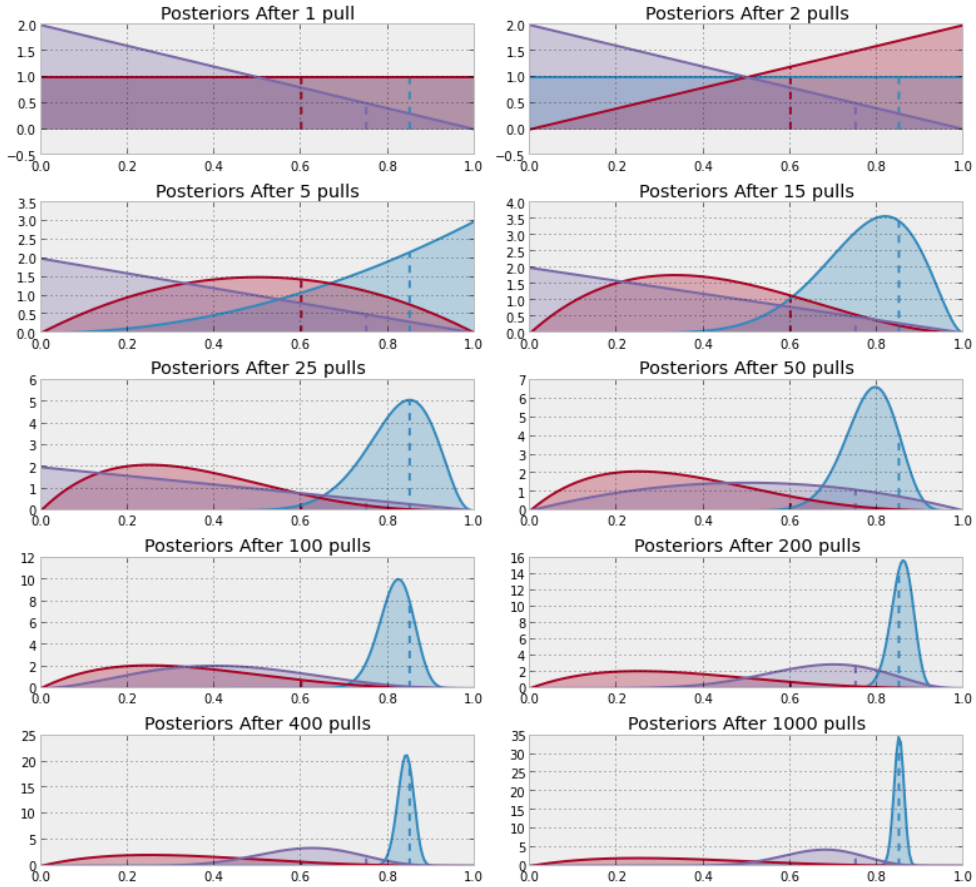
Pozorovaná náhodná veličina X (výhra či prohra, tedy 1 nebo 0) je binomická. Posterior 1.2.1.12 pravděpodobnost se po provedení pokusu přizpůsobuje novému rozdělení:

$$Beta(\alpha = 1 + X, \beta = 1 + 1 - X).$$

V případě jakéhokoliv úspěchu je provedeno navýšení pravděpodobnosti, se kterou bude algoritmus znovu vybrán. V případě neúspěchu se tato pravděpodobnost exponenciálně snižuje. V každé další hře se již systém rozhoduje s touto pravděpodobností (mezí explorací a exploatací).

Pokud tedy chceme odpovědět na dřívější otázku, zda bychom měli dávat stále šanci i banditům, u kterých často prohráváme nebo na ně zanevřít (a zkoušet štěstí u jiných), tak tento algoritmus nám navrhuje, abychom prohrávající bandity přímo nevyřazovali. Stačí, když budou vybíráni s čím dál tím menší šancí, jakmile získáme dostatek jistoty, že existují i lepší bandité.

Obrázek 1.3 ilustruje postup pro problém tří banditů ($N = 3$), jakým se algoritmus učí s rostoucím počtem her. Přerušované čáry pod grafy hustoty každého rozdělení reprezentují skryté reálné pravděpodobnosti (v obrázku mají hodnoty 0.85, 0.60, 0.75). Z uvedeného příkladu vyplývá, že o skryté pravděpodobnosti se až tolik nestaráme. Daleko větší význam pro nás má výběr nejlepšího bandity, což je vidět na distribuci červeného bandity. Ta je velice široká, což představuje skutečnou neznalost o tom, jak velkou skrytou pravděpodobností bandita disponuje. O pravděpodobnosti tedy nemáme nejmenší tušení, jsme si ale docela jistí, že tento bandita není nejlepší. Algoritmus se proto rozhodne ignorovat jej.



Obrázek 1.3: Vizualizace sekvenčního učení řešení od jedné do tisíce her. Zdroj: [20]

1.2.4.1 Zpětná vazba a její vliv na vývoj adaptibilního systému

Pojem zpětné vazby 1.2.1.3 již v předchozím textu zazněl. Pro účely této práce je nutné uzpůsobit její význam tak, aby měla přímý vliv na pružný vývoj adaptibilního systému.

Rádce bude plnit svůj úkol pomocí algoritmu Bayesian Bandits využívajícího *Beta* rozdělení pravděpodobnosti. Zpětná vazba bude mít vliv na oba parametry vstupující do rozdělení, což bude mít ve finále vliv na tvar celé distribuce 1.2.1.8 (k vidění na obrázku 1.3).

Tvar *Beta* rozdělení ovlivňují dva parametry – α a β.

- Informace o zvolení rádcem navrženého algoritmu pro vlastní doporučení (rádci sdělujeme, že jsme se **pokusili** řídit jeho radou) bude vstupovat jako jedna z proměnných do druhého parametru rozdělení.

- Zpětná vazba týkající se spokojenosti doporučení (zda byl tento algoritmus při doporučení **úspěšný** či nikoliv) bude vstupovat jako proměnná do obou parametrů rozdělení.

Při každé žádosti na rádce je pro každého banditu B_i provedeno posterior (značením $B_i(success)$ se snažím naznačit dosavadní míru úspěchů bandity B_i, pomocí $B_i(trials)$ zase dosavadní míru pokusů téhož bandity):

$$Beta(\alpha = 1 + B_i(success), \beta = 1 + B_i(trials) - B_i(success)).$$

Na ukázku jsem připravil dva příklady posterior hustoty rozdělení pravděpodobnosti dle různých hodnot parametrů vstupujících do rozdělení *Beta*. Obrázek B.1 znázorňuje situaci 4 úspěchů ze 4 pokusů, zatímco obrázek B.2 značí 1 úspěch ze 4 pokusů. Na druhém obrázku lze vidět, že distribuce je širší.

Hodnoty zpětné vazby pro adaptibilní systém

- Informace o zvolení algoritmu přičte k míře pokusů tohoto algoritmu (bandity) hodnotu. Ostatním algoritmům se nic přičítat ani odečítat nebude.
- V případě pozitivní zpětné vazby bude zvolenému algoritmu přičtena hodnota k míře jeho úspěchu. Ostatním algoritmům bude od této míry v poměru odečteno.
- V případě negativní zpětné vazby bude zvolenému algoritmu odečtena hodnota z míry jeho úspěchu. Ostatním algoritmům bude hodnota nezměněna.

Stanovení nejvhodnějších hodnot a konkrétního přístupu bude provedeno na základě experimentů.

1.2.4.2 Stanovení míry učení

Vzhledem k tomu, že prostředí se mění velmi rychle v čase, je vhodné stanovit nějakou míru učení.

Stanovením vhodné míry učení lze docílit toho, že algoritmus se bude přizpůsobovat měnícímu se prostředí rychleji a bude fungovat též jako prevence proti vyčerpání hodnot datových typů programu.

Pokud stanovíme míru < 1, algoritmus bude předchozí výsledky zapomínat rychleji a častěji bude zkoumat nové možnosti. Míra > 1 implikuje naopak to, že algoritmus bude sázet na časnější dřívější výhry, čímž ale riskuje to, že se nedokáže rychle adaptovat na náhlé změny (je více konzervativní vůči měnícímu prostředí).

Míra < 1 tak bude pro potřeby adaptibilního systému nejspíš užitečnější (vzhledem k dlouhotrvajícímu běhu systému). Delší uváznutí v lokálním minimu hned ze začátku by bylo nevhodné. Konkrétní hodnotu se opět pokusím stanovit v závislosti na provedených experimentech.

1.2.5 Význam pro kombinování

Význam bayesovské strategie pro kombinování spočívá ve výběru správné metody (tedy bandity), která bude přidána do množiny ostatních vybraných metod. Z této množiny metod je dále nutné vybrat nejvhodnější a tu použít.

Jako nejefektivnější a zároveň nejjednodušší postup se jeví výběr metody s největší četností výskytu.

V případě shody, kdy bude mít více metod stejnou četnost výskytu, bude učiněna volba jedné konkrétní pseudonáhodně.

Kapitola 2

Praktická část

2.1 Analýza a návrh řešení

Náplní této sekce je sepsání požadavků na systém, které vyplynuly z úvodních konzultací s vedoucím práce a též vlastním zkoumáním řešeného problému. Se znalostí těchto požadavků bude možné provést hrubý návrh architektury systému včetně technického řešení komponent, kterými bude systém disponovat.

S ohledem na výstupy získané v této kapitole lze výsledný systém implementovat.

2.1.1 Požadavky

Z důvodu vyšší přehlednosti jsem se rozhodl související požadavky strukturovat do skupin dle částí aplikace, které budou mít jejich obsluhu na starost.

Vyvíjenými částmi jsou:

- **Adaptibilní systém pro doporučování obsahu,**
- **Sada základních algoritmů určených k doporučování,**
- **RESTful API pro manipulaci s doporučovacími algoritmy a systémem.**
- **Klientská aplikace pro simulaci modelové úlohy.**

2.1.1.1 Požadavky na adaptibilní systém

- Systém bude klást důraz na zpětnou vazbu pomocí které bude uzpůsobovat své chování.
- Systém bude přijímat zpětnou vazbu formou informace, že uživatel se rozhodl následovat jeho rad.
- Systém bude přijímat pozitivní (v případě dobrého doporučení) a negativní (v případě nevhodného doporučení) zpětnou vazbu.

- Systém bude v pravidelných intervalech ukládat do databáze svůj aktuální vnitřní stav.
- Systém bude schopen v případě pádu aplikace a po jejím opětovném spuštění načíst naposledy uložený vnitřní stav.
- Systém bude veškeré operace týkající se rad a doporučování provádět a vyhodnocovat v reálném čase.
- Systém bude na každou klientskou žádost vracet příslušnou odpověď (i chybovou) – neexistuje nic jako odpověď *null*.
- Systém bude umožňovat vytváření a uchovávání kontextových kolekcí určených ke kombinování.
- Systém bude umožňovat vytváření a uchovávání kolekcí složených z existujících kontextových kolekcí určených ke kombinování.
- Systém bude schopen odpovídat na žádosti uživatelů poptávajících rady ohledně doporučovacích metod.
- Systém bude schopen odpovídat na další dotazy týkající se kolekcí (jejich existence apod.).
- Systém bude umožňovat asynchronní komunikaci s více klienty.

2.1.1.2 Požadavky na sadu základních algoritmů

- Algoritmus bude komunikovat s daty uložených v Apache Solr a nad těmito daty provádět doporučovací operace.

2.1.1.3 Požadavky na RESTful API

- Skrze rozhraní bude možné vytvářet kontextové kolekce se seznamem algoritmů určených pro kombinování a následnou predikci.
- Skrze rozhraní bude možné vytvářet kolekce složené z existujících kontextových kolekcí určených pro kombinování a následnou predikci.
- Skrze rozhraní bude možné vypsat existující kolekce v systému.
- Skrze rozhraní bude možné zaslat systému žádost o radu a obdržet informaci o nejvhodnější metodě pro doporučení.
- Skrze rozhraní bude možné zaslat systému informaci o zvolené metodě pro doporučení.
- Skrze rozhraní bude možné zaslat systému žádost o doporučení zvolenou metodou.
- Skrze rozhraní bude možné zaslat systému zpětnou vazbu týkající se spokojenosti s metodou, kterou bylo realizováno doporučení obsahu.

- Skrze rozhraní bude možné zaslat systému zpětnou vazbu týkající se hodnocení doporučených položek (informace, že jeden konkrétní uživatel dělá něco s jedním konkrétním dokumentem).
- Skrze rozhraní bude možné mazat v úložišti položky již nerelevantní pro doporučování.

2.1.1.4 Klientská aplikace

- Klientská aplikace se bude připojovat k výše uvedeným funkcím systému a zpracovávat získané výsledky.

Dle disciplín softwarového inženýrství lze výčet uvedený výše označit za **funkční požadavky**.

Definujme nyní i tzv. **nefunkční požadavky**, které specifikují vlastnosti a omezující podmínky kladené na systém:

- Systém bude postaven na platformě Java.
- Pro uchování informací o uživatelích, položkách a jejich vzájemné interakci bude využita platforma pro vyhledávání v textu Apache Solr.
- Systém bude umožňovat přístup aplikacím třetích stran prostřednictvím RESTful webových služeb.
- Systém bude postaven tak, aby se dal snadno parametrizovat.
- Systém bude připraven na situaci, že jej bude využívat více uživatelů současně.
- Systém poběží jako samostatná aplikace na serveru naslouchající na TCP portu. Veškerá komunikace s ní bude probíhat formou zasílání zpráv a volání procedur.
- Pro snadné nasazení a testování systému na libovolné pracovní stanici bude nutné vytvořit Chef cookbook/recipe[23].

2.1.2 Architektura doporučovací platformy

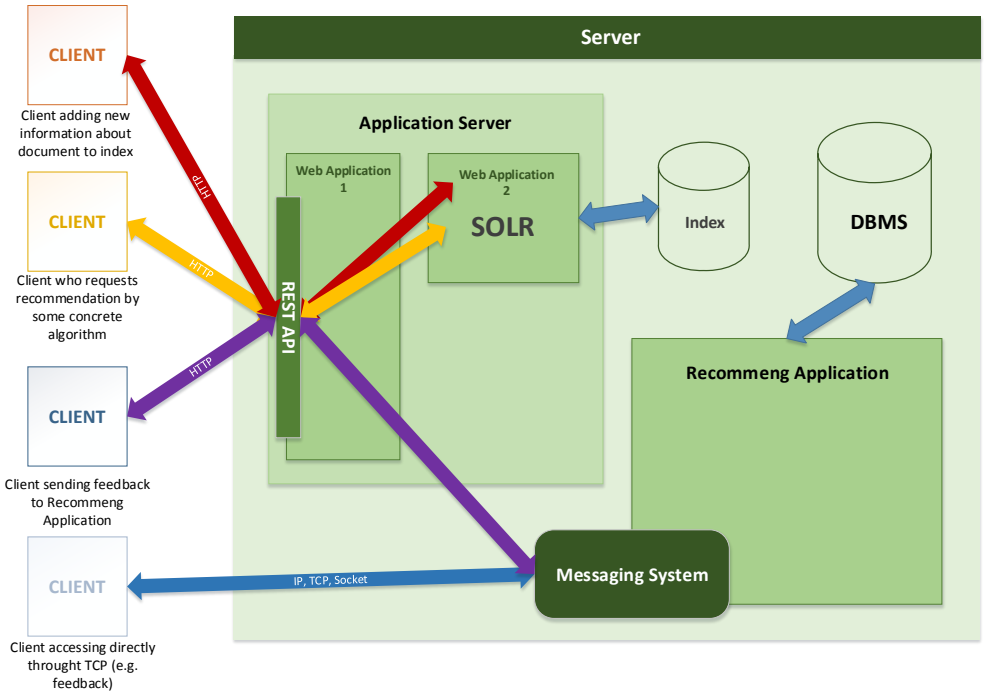
Pro zdárnou implementaci platformy splňující všechny požadavky vzešlé z analýzy výše je třeba důkladně zvážit, jaké komponenty bude obsahovat.

Diagram 2.1 znázorňuje abstraktní návrh architektury takové platformy. Mou snahou bylo zachytit přítomnost jednotlivých komponent v systému, formu jejich vzájemné komunikace a spolupráce a též základní interakci uživatele se systémem.

2.1.2.1 Server

Navrhovanými komponentami pro spuštění a běh na serveru jsou:

[23] `http://community.opscode.com/`



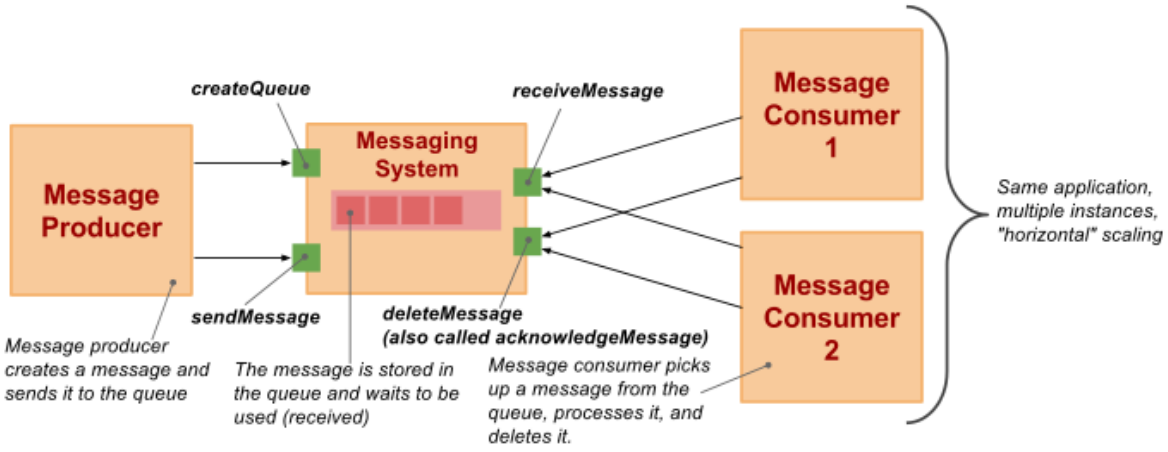
Obrázek 2.1: Abstraktní návrh architektury a komponent adaptibilního systému, ve kterém je též vidět základní interakce uživatele se systémem.

Webová aplikace s rozhraním pro komunikaci s klienty. Viz diagram 2.1, komponenta *Web Application 1*. Aplikace bude mít na starost obsluhu klientských požadavků na systém zasílaných protokolem HTTP. Zároveň bude disponovat sadou algoritmů, které lze použít pro doporučení obsahu. Dále bude aplikace obstarávat komunikační režii s druhou aplikací běžící na aplikačním serveru (*Web Application 2*), kterou je platforma pro vyhledávání v textu – Apache Solr.

Apache Solr Viz diagram 2.1, komponenta *Web Application 2*. Přítomnost této aplikace vychází z nefunkčních požadavků. Apache Solr funguje jako samostatná komponenta umožňující fulltextové vyhledávání (detaily v kapitole 2.3).

Databáze Kvůli potřebě ukládat uživatelem vytvářené kolekce, průběžný stav aplikace (časové snímky), a též možnosti snadného obnovení vnitřního stavu systému v případě přerušení běhu aplikace, je nutné zapojit do návrhu Database Management System (DBMS).

Volba DBMS hraje důležitou roli i při škálování aplikací. V minulosti tolik používané standardní relační DBMS mohou způsobovat zpoždění při provádění čtení/zápisu a v některých případech hrát roli úzkého hrdla aplikace.



Obrázek 2.2: Příklad MQ systému s producentem zpráv a konzumenty. Zdroj: [32]

Kvůli tomuto problému stojí za úvahu prozkoumat možnost použití jedné z NoSQL databází, které jsou již ze svého principu navrženy pro spolupráci s aplikacemi zaměřenými na výkon a škálovatelnost.

Recommeng systém Stěžejní serverovou komponentou je adaptibilní systém 2.1.3. Ten jsem pracovně nazval jako **Recommeng** (zkratka pro recommendation engine), abych o něm nemusel již nadále mluvit jako o *adaptibilním systému* či *systému pro kombinování metod*.

Komunikaci bude umožňovat systém pro zasílání zpráv (Messaging System) s využitím fronty zpráv (Message Queue). Toto řešení je zde nasnadě kvůli očekávání většího množství žádostí směřujících na systém a snadnější škálovatelnosti aplikace. Obecný příklad takového MQ systému znázorňuje obrázek 2.2

Recommeng systém bude taktéž komunikovat s databází kvůli nutnosti průběžného ukládání informací o stavu, respektive kvůli možnosti načíst poslední uložený stav při novém startu aplikace.

Veškerá data pro výpočet a predikci budou jinak udržována přímo ve vnitřní paměti (uvnitř JVM).

2.1.2.2 Klient

Klientská strana není příliš složitá. Potenciální uživatel aplikace má v zásadě dvě možnosti, jak s platformou komunikovat:

- Jen a pouze zasíláním žádostí na REST API.
- Kombinací zasílání žádostí na REST API společně s přímou komunikací s Recommeng systémem prostřednictvím fronty zpráv.

Některé ze systémových funkcionalit nelze bez komunikace s REST API využívat. Do této kategorie spadá například přidání nového vztahu k položce ve fulltextovém indexu nebo zaslání žádosti o doporučení obsahu některým z podporovaných algoritmů.

Přímé spojení s platformou skrze frontu zpráv bez nutnosti zapojení prostředníka v podobě REST API by ale měly umožňovat všechny procedury Recommeng systému. Jmenovitě jde především o vytváření kolekcí se seznamem algoritmů, dále zasílání žádostí o radu pro výběr algoritmu a též metody informující systém o uživatelově chování.

V diagramu 2.1 jsem se snažil zachytit různé formy komunikace klienta se systémem. Pro lepší názornost jsou jednotlivé žádosti barevně odlišeny. Ty lze považovat za modelové případy užití inspirované systémovými požadavky.

Zaslání zpětné vazby k doporučené položce *Pozn.* V diagramu 2.1 je klient odlišen červenou barvou.

Jedná se o klienta přidávajícího nový vztah mezi položkou a jejím uživatelem do úložiště.

- klient zašle žádost na rozhraní
- aplikace (Web Application 1) za rozhraním se spojí s indexem
- v případě zdárného průběhu se přidá do indexu informace o tom, že 1 uživatel ohodnotil 1 dokument
- klient obdrží odpověď s výsledkem žádosti

Žádost o radu při výběru metody pro doporučení obsahu *Pozn.* V diagramu 2.1 je klient odlišen fialovou barvou.

V diagramu je znázorněna situace, kdy klient zasílá platformě zpětnou vazbu ohledně doporučení, které dostal. Tato varianta může ale též představovat situaci, kdy klient žádá systém o radu, kterou metodou si má nechat doporučit obsah ve své budoucí žádosti o doporučení (viz varianta 2.1.2.2).

- klient zašle žádost na rozhraní
- aplikace (Web Application 1) za rozhraním se pokusí přistoupit k frontě zpráv
- pokud fronta existuje, žádost je přidána do fronty pro zpracování konzumentem (Recommeng systém)
- konzument zpracuje žádost a zasílá odpověď, kterou systém zpráv předá zpět do aplikace
- klient obdrží odpověď s výsledkem žádosti

Žádost o doporučení konkrétním algoritmem *Pozn.* V diagramu 2.1 je klient odlišen žlutou barvou.

Situace znázorňuje situaci, kdy klient poptává doporučení obsahu konkrétním algoritmem. Toto doporučení klient poptává na základě odpovědi na žádost, která mu byla udělena systémem (viz varianta 2.1.2.2)).

- klient zašle žádost na rozhraní
- aplikace (Web Application 1) za rozhraním zvolí vybraný algoritmus pro doporučení
- algoritmus přistoupí k datům v indexu, nad kterými se provede doporučení
- klient obdrží odpověď s výsledkem žádosti

Komunikace bez využití RESTful API *Pozn.* V diagramu 2.1 je klient odlišen modrou barvou.

Tento klient se rozhodl nevyužít možnosti komunikovat prostřednictvím REST API. Svou žádost zasílá přímo do fronty zpráv.

- klient zašle žádost do fronty
- pokud fronta existuje, žádost je přidána do fronty pro zpracování konzumentem (aplikace Recommeng)
- konzument zpracuje žádost a zasílá klientovi odpověď

2.1.3 Obecný návrh rozhraní

2.1.3.1 Rozhraní REST

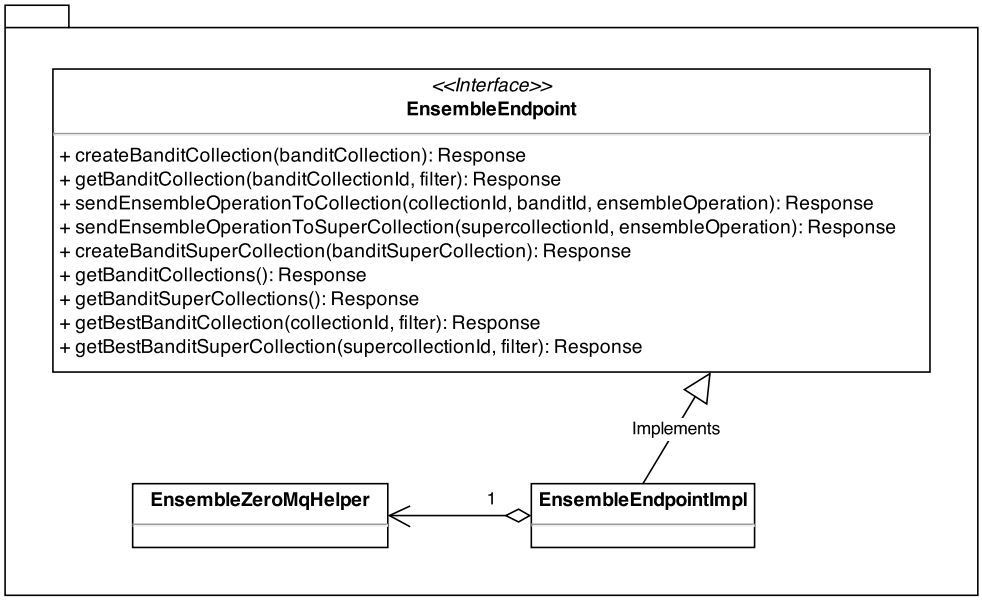
Dle zadání má platforma poskytovat API pro interakci se systémem a daty formou RESTful (HTTP implementace služeb REST 2.2.1). Každý uživatel tak bude moci interagovat s rozhraním prostřednictvím veřejných endpointů.

Z prováděné analýzy vyplynuly požadavky na aplikační rozhraní takové, že je lze rozdělit do čtyř skupin dle podstaty úkolu, který mají plnit.

- Algorithms
- Cores
- Supercollection
- Collection

Návrh tříd na zjednodušených[24] diagramech 2.3 a 2.4 ilustruje metody běžícím za poskytovaným API.

[24]Pro větší srozumitelnost neuvádím v parametrech metod typy, ale spíše sémantické významy parametrů. Není též přítomna spousta dalších, s uvedenými třídami spolupracujících tříd.



Obrázek 2.3: Zjednodušený návrh tříd pro RESTful API endpoint pro komunikaci s Recommeng systémem.

V případě prvního diagramu 2.3 jsou prostřednictvím API metod volány metody třídy **EnsembleZeroMqHelper**, která bude realizovat spojení s frontou zpráv Recommeng systému (třída **AsynchronousServer** na diagramu 2.6).

Po prozkoumání druhého zjednodušeného diagramu 2.4 je možné utvořit si představu o způsobu komunikace metod za RESTful API se serverem Apache Solr (třída **SolrService**) a s jednotlivými doporučovacícmi algoritmy (skrze API **IAlgorithm**).

K dispozici je pohled na část zdrojů (diagram 2.5), který by měl celou věc pomoci osvětlit.

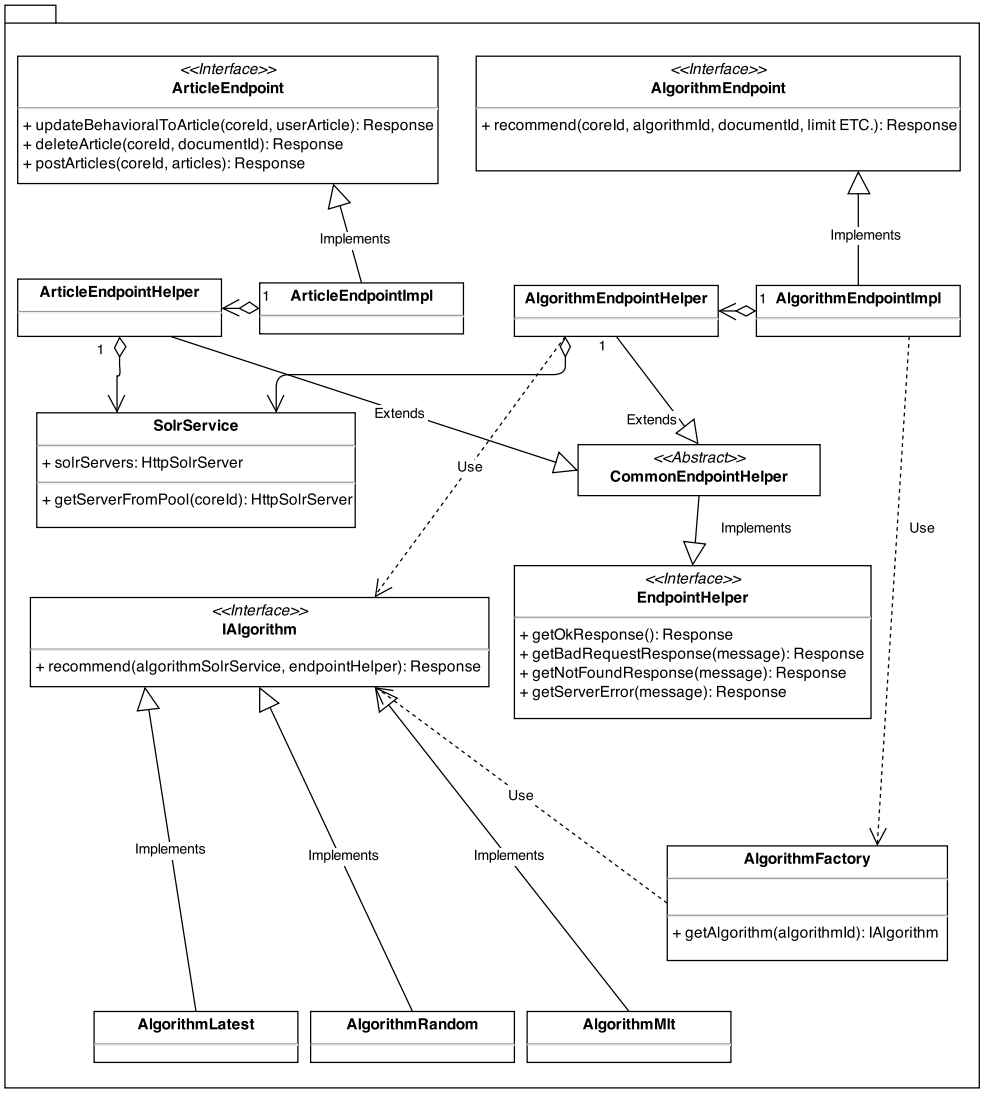
Ensemble Komunikace s rozhraním by měla umožňovat vytvářet v Recommeng aplikaci kolekce s identifikátory algoritmů a kolekce složené z těchto kolekcí (dále je budu označovat jako super kolekce), jenž budou zapojeny do kombinování a predikce.

K tomu účelu budou sloužit endpointy:

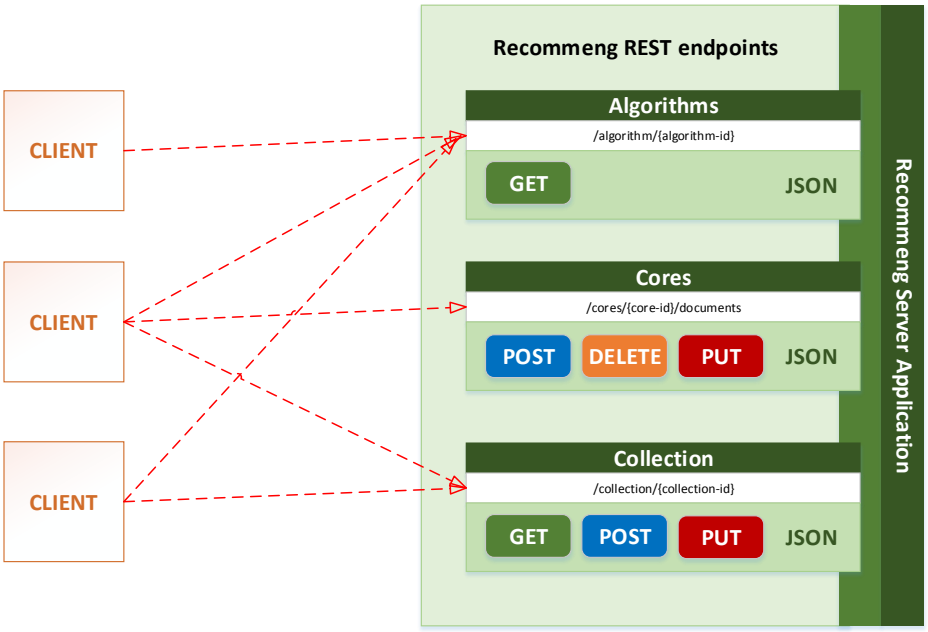
```
/collection
```

```
/supercollection
```

Pokud uživatel zašle metodou POST žádost na tyto endpointy (obsahující jméno kolekce a seznam příslušných položek v těle požadavku), v případě



Obrázek 2.4: Zjednodušený návrh tříd pro RESTful API endpointy pro komunikaci s jádrem Solr a s doporučovacími algoritmy.



Obrázek 2.5: Ukázka některých REST endpointů navrhovaného adaptibilního systému

dosavadní neexistence v systému budou kolekce vytvořeny a připraveny k užití systémem.

Dále by měla existovat možnost zaslat na kolekci či super kolekci prosbu o predikci nejlepšího algoritmu pro doporučení.

```
/collection/{collection-id}

/supercollection/{collection-id}
```

Toho je dosaženo zasláním žádosti s metodou GET, uvedením identifikátoru kolekce a případným specifikováním požadovaného výstupu (zda chceme vrátit pouze nejlepší možnost či více možností seřazených sestupně od nejlepší) v parametru dotazu.

Žádostí metodou GET bez specifikace ID bude navrácen seznam existujících kolekcí a super kolekcí v systému (existují-li v něm nějaké).

Kvůli učení a vývoji znalostí systému je nezbytné mít možnost zaslat systému informaci o výběru konkrétního algoritmu a též zpětnou vazbu vyjadřující, jak byl žádající uživatel s navrhovanou variantou spokojen. Toho bude dosaženo zasláním žádosti metodou PUT s informacemi uvedenými v těle požadavku.

Rozhraní pro sadu základních algoritmů Bude se jednat o jednoduchý endpoint, který si při žádosti vystačí s metodou GET.

```
/algorithms/{algorithm-id}
```

Úkolem je zpracovat uživatelovu žádost o doporučení algoritmem, jehož identifikátor je uveden v cestě.

Doporučení bude provedeno nad dokumenty ze specifikovaného indexu (identifikátor indexu je nutno uvést v parametru dotazu). V parametrech dotazu lze navíc specifikovat další vstupy týkající se doporučení jako limit vrácených výsledků, identifikátor pro doporučení článků z jedné konkrétní skupiny a další.

Rozhraní pro úložiště článků Důležité rozhraní umožňující zaznamenat například chování uživatele vzhledem ke sledovaným položkám.

```
/cores
```

```
/cores/{core-id}/documents
```

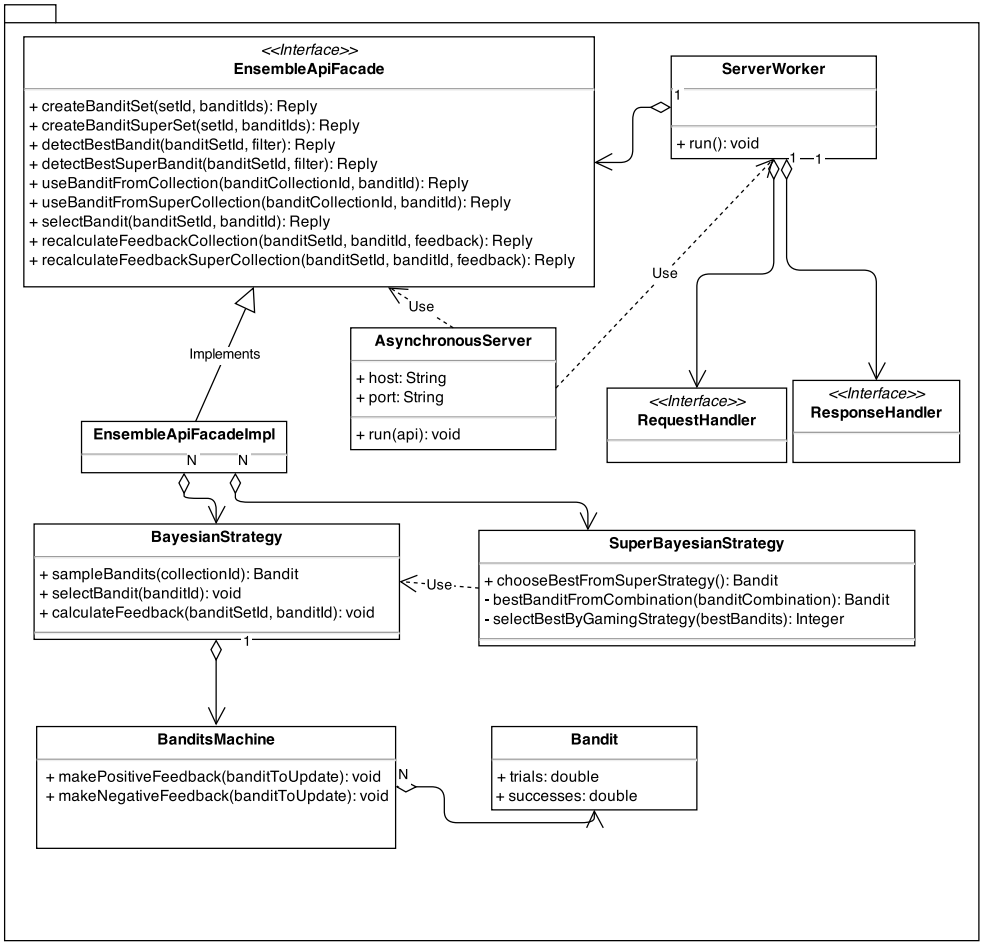
Prostřednictvím metody POST budou zasílány veškeré informace o položkách (u článku např. identifikátor, text, skupina, datum publikace) a jejím uživateli (identifikátor), které mají být uloženy do fulltextového indexu. Druhou podporovanou metodou je metoda DELETE pro vyřazení položky nadobro z doporučování. Metoda je použita v žádosti na týž endpoint jako metoda POST, pouze je nutné v parametru dotazu specifikovat identifikátor článku (z toho důvodu, že identifikátorem článku může být obecně cokoliv, například i URL adresa).

Metody a endpointy uvedené výše jsou hrubým přehledem budoucí funkcionality systému. Kompletní dokumentace navrženého REST API je k nahlédnutí v příloze A na konci práce.

2.1.3.2 Rozhraní Recommeng systému

Znázorňuje jej třídní diagram 2.6. V případě úspěšně navázaného spojení se systémem jsou pomocí pracovních vláken **ServerWorker** volány jednotlivé procedury systému. Tyto žádosti obsluhuje API **EnsembleApiFacade**.

Stejně jako v případě diagramů 2.3 a 2.4 se jedná o zjednodušený návrh, kde chybí spousta dalších tříd zapojených do systému. Diagram slouží pouze k ilustraci metod rozhraní a jejich funkcí pro daný systém.



Obrázek 2.6: Zjednodušený návrh tříd v adaptibilním systému a ukázka API.

2.1.3.3 Komunikace s frontou zpráv Recommeng systému

Dle návrhu bude komunikace uživatele s Recommeng systémem, ať už komunikuje přímo či pomocí RESTful API, řešena prostřednictvím fronty zpráv.

Dalším úkolem tedy je navrhnout formát, jakým si budou vyměňovat producent s konzumentem data prostřednictvím volání vzdálených procedur Recommeng systému.

Vzhledem k obecné známosti protokolu HTTP jsem se rozhodl napodobit jeho chování a zachovat sémantiku:

- metoda (method)
- cesta (path)
- tělo zprávy (body)

Podobně jako v HTTP, i v případě mnou navrhované komunikace bude na každou žádost ve tvaru *method, path* a *body* přicházet odpověď ve tvaru *status* a *body* s využitím různých návratových kódů pro stav (status). Tyto informace budou reprezentovány formátem zpráv JSON 2.2.1.3.

2.1.4 Identifikace sady algoritmů

Uživatel žádající o radu pro výběr nejlepšího algoritmu obdrží od systému identifikátor reprezentující tento algoritmus. Pro účely této práce jsem vybral následující sadu algoritmů:

2.1.4.1 Algoritmus náhodného výběru

Jak je patrné již z názvu, tento algoritmus vybírá položky pro doporučení naprosto náhodným způsobem. Jeho hlavním úkolem je být zde pro srovnání s ostatními algoritmy (náhodný algoritmus by zřejmě neměl dostávat nejlepší zpětnou vazbu).

2.1.4.2 Algoritmus výběru dle nejnovějších položek

Doporučování dle nejnovějších položek je dalším z algoritmů s naivním přístupem k problému. Položky budou v tomto případě doporučovány sestupně dle zveřejněného data.

2.1.4.3 Algoritmus výběru nejlépe hodnocených položek

Jedná se o první algoritmus založený na složitějším výpočtu. Položky vzešlé z doporučení budou dle určitých parametrů nějakým způsobem lepší než ostatní, které se v doporučení neobjevily. Takovým parametrem může být například hodnocení na škále od 1 do 5, počet pozitivních hodnocení, souhrnné číslo udávající počet přečtení článku nebo jiný z mnoha způsobů vyjádření zájmu o položku.

2.1.4.4 Algoritmus výběru dle podobnosti obsahu

Algoritmus se snaží na základě podobnosti obsahu nalézt pro položku několik jí podobných položek z databáze. Podobnost se určuje porovnáním jednotlivých parametrů, například tagů, nadpisů nebo celého textu článku.

2.1.4.5 Algoritmus kolaborativního filtrování

Algoritmus je založen na modelu dřívějšího chování uživatele v systému. Model je většinou konstruován z chování většího množství uživatelů s podobným vkusem. V podstatě lze říci, že doporučení jsou založena na automatické spolupráci více uživatelů a výběru těch, kteří mají co nejpodobnější preference či chování.

Rozlišují se dva hlavní způsoby filtrování.

User-based *"You may like it because your friends liked it."* [22]
Aneb filtrování založené na uživatelích. Jedná se o starší variantu kolaborativního filtrování. Podstatou je vzít na základě určité podobnosti skupinu uživatelů (zdroj [22] udává cca 20 až 50) s podobných vkusem jako má uživatel, pro něhož je doporučení konstruováno, a poté předpovědět, jak moc zajímavá by pro uživatele byla pro něj dosud neznámá položka, se kterou jsou spojení uživatelé se stejným vkusem.

Item-based *"You tend to like that item because you have liked those items."* [22]
Aneb filtrování založené na položkách, které použila v roce 2001 jako první společnost Amazon. Myšlenka je taková, že uživatel, který si v minulosti zakoupil nějakou položku, bude v budoucnu při dalším nákupu vyhledávat položku podobnou. Například předpověď toho, co si uživatel zakoupí v budoucnu, lze uskutečnit analýzou historie nákupů uživatele [14].

2.2 Principy a technologie

2.2.1 RESTful API

2.2.1.1 REST

REpresentational State Transfer (REST) je architektonický styl definující určitá pravidla a vlastnosti návrhu API webových služeb orientovaných na zdroje. REST je silně založen na architektuře Klient-Server (server poskytuje přístup ke zdrojům, klient k nim může přistupovat a modifikovat je) a k jeho realizaci je možné využít protokolu HTTP (takovou realizaci pak nazýváme jako RESTful). Role tohoto protokolu zde není nikterak náhodná, neboť autorem REST není nikdo jiný než Roy Fielding, jenž je u protokolu HTTP podepsaný jako spoluautor [23].

Díky protokolu HTTP lze následovat mnoho pravidel návrhu RESTful API, například přítomnost adresovatelných zdrojů, kdy je každý zdroj adresovatelný pomocí Uniform Resource Identifier (URI). RESTful pro manipulaci se svými zdroji používá též HTTP metody (GET, POST, PUT, DELETE, ale i další, například OPTIONS či PATCH). Další vlastností je užívání standardních stavových kódů[25] HTTP (typicky 2xx, 3xx, 4xx, 5xx) v odpovědi na žádost či bezestavová komunikace.

Data mohou být reprezentována v rozličných formátech jako XML, JSON či YAML.

[25] Definici všech stavových kódů viz `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`.

2.2.1.2 Jersey (JAX-RS)

Java API for RESTful Web Services (JAX-RS)[26] je standard definovaný v Java Specification Request 311[27]. Jedná se o specifikaci pro RESTful webové služby implementované v programovacím jazyce Java.

Pomocí JAX-RS lze s využitím anotací jednoduše a přehledně definovat sémantiku jednotlivých tříd a jejich metod z hlediska architektury REST. Příklady anotací jsou *@Path(relativni_cesta)* pro specifikaci relativní cesty zdroje, *@GET* či *@POST* specifikující typ žádosti nebo třeba *@QueryParam* pro přiřazení parametru HTTP dotazu k hodnotě parametru příslušné metody třídy.

Pro účely implementace rozhraní systému Recommeng jsem zvolil vzhledem k předchozím zkušenostem referenční implementaci tohoto standardu v podobě frameworku **Jersey 2.x**.

2.2.1.3 JSON

JSON [28] je jednoduchý a univerzální formát pro výměnu dat. Zkratka skrývá význam *JavaScript Object Notation*. Jeho MIME (Internet media type) je application/json. V současné době se jedná o jeden z nejpoužívanějších formátů pro výměnu dat se službami REST. Výhodami jsou rychlost zpracování a snadné použití.

Formát je postaven na dvou strukturách.

Array je seřazený seznam hodnot, který je ohraničen závorkami

Object je neseřazená kolekce dvojic klíč:hodnota ohraničený dvěma závorkami

Obě struktury lze do sebe zanořovat a vytvářet tak složitější struktury.

2.2.2 Fronta zpráv a síťová komunikace

2.2.2.1 ØMQ

ØMQ (ZeroMQ)[29] je vysoce výkonná[30] síťová knihovna napsaná v programovacím jazyce C++ vhodná k nasazení v distribuovaných a vícevláknových aplikacích, které vyžadují velkou škálovatelnost. S jejím využitím lze poměrně snadno navrhnout komplexní komunikační systém. Ke komunikaci

[26] https://jax-rs-spec.java.net
[27] https://jcp.org/en/jsr/detail?id=311
[28] `http://www.json.org`
[29] http://zeromq.org
[30] Viz výkonnostní testy na oficiální stránce `http://zeromq.org/results:_start`.

užívá socketů[31]. Duchovním otcem a spoluautorem knihovny je slovenský expert na oblast messaging middleware Martin Sústrik[32].

Hned na úvod je nutné sdělit, že se nejedná o klasický messaging system (message-oriented middleware) takového typu, jakým je například Apache ActiveMQ[33] a jemu podobné systémy. Takové systémy jsou většinou hotová řešení připravená k okamžitému nasazení a integraci s dalšími službami.

Filosofie ZeroMQ je jiná, neboť jde především o multiplatformní knihovnu určenou k programovému využití. Pomocí jednoduchého socketového API umožňuje programátorovi sestavit si vlastní messaging system dle svého nejlepšího uvážení. Programátor využívá ze strany API veškerou podporu usnadňující práci se sítí a s trochou nadsázky lze prohlásit, že se stará pouze o zasílání zpráv.

Knihovna navíc poskytuje kompletní svobodu v tom, jakým způsobem zakódujeme zprávu (JSON, BSON nebo jakýkoliv vlastní navržený formát). Já využil 2.1.3.3.

Podporovány jsou čtyři protokoly pro komunikaci [21]:

tcp jako model síťově založeného přenosu (procesy na jedné síti)

inproc jako model komunikace vláken uvnitř jednoho procesu

ipc jako model komunikace mezi procesy (procesy out-of-box)

multicast komunikující skrze PGM[34].

Knihovna také definuje základní vzory zasílání zpráv, ať už se jedná o doručování zpráv na jednotlivé uzly, mapování uzlů na vlákna, procesy či umisťování zpráv do fronty[35]. Každý vzor zároveň určuje jinou síťovou topologii.

Základními vzory jsou:

- Request-reply
- Pub-sub
- Pipeline
- Exclusive pair

Nejvhodnějším vzorem pro mou práci je vzhledem k povaze vyvíjeného systému (zasílání zpráv a odpovědi na ně) vzor Request-reply, jehož konkrétní použití následuje záhy v sekci Recommeng systém. Ostatní vzory nemá smysl v rámci této práce představovat.

[31] Socket je mechanismus, kterým je možno zprostředkovat lokální čí vzdálenou komunikaci dvou uzlů, která má charakter klient/server [33].

[32] `http://250bpm.com/contact`

[33] `http://activemq.apache.org`

[34] `http://tools.ietf.org/html/rfc3208`

[35] `http://zguide.zeromq.org/page:all#Messaging-Patterns`

Pro účely Recommeng systému jsem se rozhodl využít čistou Java implementaci knihovny ZeroMQ v podobě knihovny **jeroMQ**[36]. Z výkonnostního hlediska za původním řešením zaostává jen nepatrně[37] a navíc je mnohem jednodušší integrovat ji do vyvíjené aplikace prostým přidáním knihovny do projektu.

2.2.3 Úložiště dat

2.2.3.1 Apache Solr

Apache Solr[38] je populární[39] open-source platforma pro vyhledávání textu napsaná v programovacím jazyce Java. Jejími charakteristickými vlastnostmi jsou podpora pro fulltextové vyhledávání, fasetové vyhledávání (analogie ke konstrukci GROUP BY v RDBMS), dobrá škálovatelnost pomocí kešování a distribuovaného vyhledávání, využívání vyhledávací konstrukce *more like this*, o které bude řeč v sekci 2.3.3, a také například tzv. *near real-time indexing*[40] (dokumenty je možné vyhledávat téměř ihned po jejich zaindexování).

Z hlediska architektury programu jde o samostatný server pro fulltextové vyhledávání běžící v servletovém kontejneru (například Apache Tomcat). K indexaci a fulltextovému vyhledávání využívá ve svém jádru knihovnu Apache Lucene.

Apache Lucene[41] je vysoce výkonná knihovna pro účely vyhledávání v textu a indexování.

Vstupem pro indexaci jsou dokumenty. Každý takový dokument obsahuje množinu elementů, kde je tento element nazvaný jako *field*. Každý field má své jméno, datový typ a případně další atributy.

Vstupem pro vyhledávání jsou textové řetězce (viz syntax[42]), případně dotazované objekty.

Index je uložen na disku ve formě souborů ve struktuře invertovaného indexu dokumentů [36].

Ukázka definice několika field dokumentu ve schématu Solr:

```
<field name="userId" type="int" indexed="true" stored="true"
    multiValued="true"/>
<field name="time" type="date" indexed="true" stored="true"/>
```

Ukázka reprezentace dokumentu ve výsledku vyhledávání pomocí Apache Solr ve formátu XML:

[36] https://github.com/zeromq/jeromq

[37] Viz srovnávací testy https://github.com/zeromq/jeromq/wiki/Perfomance.

[38] https://lucene.apache.org/solr

[39] Viz seznam serverů využívajících služeb Solr https://wiki.apache.org/solr/PublicServers

[40] https://cwiki.apache.org/confluence/display/solr/Near+Real+Time+Searching

[41] http://lucene.apache.org/core

[42] http://lucene.apache.org/core/2_9_4/queryparsersyntax.html

```
<doc>
   <int name="id">1</int>
   <str name="articleId">http://somedomain.org/somearticle.html</str
       >
   <str name="articleText">Hello Bob and Alice!</str>
   <int name="group">123</int>
   <long name="_version_">1465487644804775936</long>
</doc>
```

Formu jejich spolupráce lze popsat tak, že Apache Solr poskytuje pro vyhledávání RESTful API, za kterým je skryto a voláno JAVA API knihovny Lucene. Díky tomu je možné pomocí protokolu HTTP komunikovat s Apache Solr z jakékoliv platformy napsané v jakémkoliv programovacím jazyce.

K integraci Apache Solr s dalšími aplikacemi je možné vybírat ze spousty nástrojů a knihoven[43]. Pro účely mé aplikace psané v programovacím jazyce Java jsem zvolil knihovnu **SolrJ**[44] s klientským rozhraním pro vyhledávání, přidávání a aktualizaci indexu.

2.2.3.2 Apache Cassandra 2.0

Apache Cassandra 2.0[45] je open-source distribuovaný DBMS navržený pro obsluhu velkého množství dat. Z hlediska datového modelu je Cassandra jakýmsi hybridem mezi key-value (pod 1 klíčem je uložena 1 hodnota) a column-oriented databázemi. V dokumentaci [19] se lze dočíst, že jde o row-oriented databázi.

Základem modelu je *column family* (analogie tabulky v RDBMS), jež je složena z řádků a sloupců. Každý řádek má unikátní identifikátor ve formě klíče – každý řádek obsahuje více sloupců. Sloupce mají jméno, hodnotu a časovou značku. Výhodou proti RDBMS přístupu je to, že rozdílné řádky ze stejné column family nemusí sdílet stejnou množinu sloupců – do jedné nebo více řádek lze v libovolný čas zapsat jakýkoliv sloupec.

Vzhledem k tomu, že jedním z vyzdvihovaných případů užití databáze je uchovávání časových snímků, rozhodl jsem se ji experimentálně zapojit do vytvářeného Recommeng systému. Ještě předtím jsem však detailněji zkoumal možnost použití jiné NoSQL databáze, a to **Redis**.

Redis je klasickou key-value databází uchovávající data primárně v paměti. Postupným vývojem se jeho funkcionalita propracovala k tomu, že pod jeden klíč je nyní možné uložit několik datových struktur (např. množiny a asociativní pole). Vzhledem k ukládání dat do paměti disponuje značnou rychlostí, navíc jej lze dle konfigurace nastavit tak, aby se obsah paměti průběžně ukládal na disk pro potřeby snadného obnovení dat v případě pádu aplikace.

Jeho zapojení do aplikace jsem zvažoval ve fázi zkoumání, jakým způsobem bude v adaptibilním systému řešen failover dat. Po následném návrhu

[43] http://wiki.apache.org/solr/IntegratingSolr
[44] http://wiki.apache.org/solr/Solrj
[45] http://cassandra.apache.org

datového modelu pro ukládání časových snímků stavu aplikace jsem však sáhl po použití Apache Cassandra jako po lepší z nabízených variant pro případné budoucí potřeby práce (rozsahové dotazy, vizualizace vývoje systému apod.).

Velkým benefitem je podpora Cassandra Query Language (CQL), dotazovacího jazyka umožňujícího vytvářet podobné konstrukty, jaké nabízí jazyk SQL. CQL je nyní k dispozici ve verzi 3.1 a s pomocí **DataStax Java Driver 2.0** mohu snadno manipulovat s databází přímo ze své aplikace.

2.2.4 Ostatní použité technologie

2.2.4.1 Apache Mahout

Apache Mahout[46] je knihovna napsaná v programovacím jazyce Java, jež poskytuje implementaci rozličných technik z oblasti strojového učení:

- **shlukování (clustering)** – položky nacházejících se v určitých třídách (například webové stránky či novinové články) jsou organizovány do skupin tak, že položky nacházející se v těchto skupinách jsou si vzájemně podobné
- **klasifikace (classification)** – učení se ze stávajících kategorizací a zařazování neklasifikovaných položek do nejvhodnější kategorie
- **doporučování (recommendation)**
- **často se vyskytující skupiny položek (frequent itemset mining)** – analýza položek v rámci nějaké skupiny (například nákupní košík) a identifikace, které položky se nejčastěji vyskytují pohromadě

Pro tyto techniky realizuje příslušné algoritmy jako například kolaborativní filtrování, k-means, náhodné lesy, skryté markovské modely a další. Některé algoritmy jsou připraveny pro běh v distribuovaném módu s využitím paradigmatu Map/Reduce, některé pak v lokálním módu (samotný Mahout je založen na Apache Hadoop, ale lze jej pohodlně využívat i bez něj) [30]. Mahout poskytuje též knihovny pro obecné matematické operace (zaměřené hlavně na oblast statistiky) a kolekce[47].

Využít jej pro potřeby své práce jsem se rozhodl poté, co jsem na něj narazil v projektu Mendeley 1.1.1.3 během zkoumání existujících řešení doporučovacích systémů.

2.2.4.2 Spring Framework

Při tvorbě každého nového Java projektu od základu je dobré zamyslet se nad možností využít některý z mnoha frameworků a dalších užitečných nástrojů, s jejichž pomocí si lze do značné míry usnadnit proces vývoje. Díky

[46] `https://mahout.apache.org`
[47] `https://mahout.apache.org/users/basics/mahout-collections.html`

dobrým zkušenostem z dřívějších projektů jsem zvolil open-source framework pro tvorbu moderních enterprise aplikací **Spring**[48].

Jeho výhodami jsou snadná konfigurovatelnost, podpora dependency injection, rozšiřitelnost a také integrace s jinými frameworky. V mém případě to byla integrace s frameworkem Jersey, kterou jsem využil při implementaci RESTful API.

2.2.4.3 Apache Tomcat

Apache Tomcat je známý open-source webový server a servletový kontejner. Jedná se o oficiální referenční implementaci technologií Java Servlet a Java Server Pages (JSP). Na serveru mohou běžet uživatelské servlety (programy napsané v Javě), které umí zpracovávat požadavky zasílané pomocí HTTP protokolu a tímtéž protokolem na ně odpovídat. Apache Tomcat zde slouží jako zásobník servletů starající se o jejich spouštění, běh, ukončování a podobně.

2.2.4.4 Správa závislostí

Často slýchaným pojmem z úst mnoha vývojářů v programovacím jazyce Java je tzv. *classpath hell*. V podstatě jde o problémy spojené s načítáním programových tříd. V dnešní době existuje spousta nástrojů schopných tento problém efektivně řešit používáním správně projektové struktury, sestavovacích nástrojů a nástrojů pro správu závislostí. Jmenujme například Apache Ant společně s Apache Ivy, Gradle nebo třeba Apache Maven.

Apache Maven jsem použil při implementaci všech komponent aplikace.

2.3 Realizace doporučovací platformy

Následující kapitola se zaměřuje na popis programátorských postupů, které jsem použil při implementaci systému.

2.3.1 Recommeng systém

Adaptibilní systém poběží na serveru jako samostatná *Java Application* s názvem **Ensemble**.

2.3.1.1 Zavedení systému

Poté, co je aplikace spuštěna voláním metody *main()* třídy **EnsembleApp**, je v této metodě následně volána metoda *loadEnsembleApplication()* abstraktní rodičovské třídy **EnsembleAppBase**.

[48] `http://projects.spring.io/spring-framework`

Metoda loadConsoleApplication() hraje roli zavaděče aplikace, neboť postupným voláním v sobě obsažených metod zavádí do provozu celý systém (vytvoření instance řídící třídy, napojení na databázi, inicializace API a spuštění asynchronního serveru). Hlavní řídící třídou aplikace je třída **ApplicationBean**. Jejím prostřednictvím je zavedena vrstva pro obsluhu zpráv i komponenty pro komunikaci s datovým úložištěm. Třída je zavedena z aplikačního kontextu frameworku Spring.

Ten je možné velice jednoduše konfigurovat pomocí anotací ve třídě **AppConfig**:

```
@Configuration
@EnableScheduling
@ComponentScan(basePackages = {
     "cz.cvut.bouchja1.ensemble.spring"
})
@PropertySource("classpath:application.properties")
public class AppConfig {
...
```

Anotace *@EnableScheduling* a *@ComponentScan* využijeme pro pravidelné ukládání stavu aplikace 2.3.1.8, anotaci *@PropertySource* zase pro možnosti parametrizace systému 2.3.1.2.

2.3.1.2 Parametrizace

Pro účely experimentování se systémem, volbu databázového úložiště a testování funkčnosti s různě volenou konfigurací je nutné načítat konfiguraci z editovatelného *properties* souboru. K tomu jsem využil Spring bean **PropertySourcesPlaceholderConfigurer**.

Konkrétní načítání v programu je pak velice jednoduché:

```
this.rate = Double.parseDouble(env.getProperty("ensemble.machine.
    rate"));
this.possitiveFeedback = Double.parseDouble(env.getProperty("
    ensemble.feedback.possitive.winner"));
this.possitiveFeedbackOthers = Double.parseDouble(env.getProperty("
    ensemble.feedback.possitive.losers"));
this.negativeFeedback = Double.parseDouble(env.getProperty("ensemble
    .feedback.negative.stupid"));
```

2.3.1.3 Vrstva pro obsluhu zpráv

Následující text popisuje postup, jakým jsem realizoval vrstvu pro obsluhu zpráv pomocí síťové knihovny ZeroMQ.

Většinová funkcionalita je řešena třídou **AsynchronousServer** (obsahující též vnitřní třídu **ServerWorker**) v programovém balíčku:

```
cz.cvut.fit.bouchja1.ensemble.socket
```

Postup tvorby by se dal shrnout do tří kroků:

1. rozhodnutí o komunikačním protokolu
2. definice síťové infrastruktury
3. realizace vzoru pro zasílání asynchronních zpráv REQ/REP

Rozhodnutí o komunikačním protokolu V prvním kroku bylo nutné rozhodnout o tom, jakým způsobem bude probíhat přenos dat směrem k Recommeng systému a naopak. Ze čtyř protokolů, jimiž disponuje ZeroMQ, jsem pro přenos dat zvolil síťový protokol **TCP**, a to z toho důvodu, že dle návrhu architektury by měl adaptibilní systém naslouchat na serveru a uživatelé s ním mít možnost navazovat spojení zvenčí mimo server.

Díky ZeroMQ API stačí připravit kontext, socket a metodě *bind()* nastavit adresu serveru s příslušným portem. Sama metoda se pak postará o zbylou práci (vytvoření endpointu pro příjem jednotlivých spojení a navázání na socket).

```
ZContext ctx = new ZContext();
//  Frontend socket talks to clients over TCP
Socket frontend = ctx.createSocket(ZMQ.ROUTER);
frontend.bind("tcp://" + host + ":" + port);
```

Definice síťové infrastruktury Propojení jednotlivých síťových komponent vychází z nativní povahy architektury Klient-Server. Server zastává roli stabilnější komponenty v síti, bude tedy přijímat spojení (viz ukázka s metodou *bind()* výše). Zároveň bude nutné umístit mezi server a připojující se klienty prostředníka, jehož úkolem obsluha všech žádostí na server a zasílání odpovědí zpět klientům.

V případě více připojených klientů bude ZeroMQ automaticky obstarávat obsluhu všech příchozích žádostí.

Realizace vzoru pro zasílání zpráv REQ/REP Pro zasílání zpráv jsem nejprve zvolil obousměrně komunikující vzor *Request Reply*. Toto paradigma je známé z většiny serverových typů[49]. Klient používá vlastní socket typu **ZMQ.REQ** k inicializaci žádosti, kterou následně odesílá na server. Server též užívá vlastního socketu **ZMQ.REP** ke čtení příchozí žádosti, po které zasílá odpověď.

Problém je, že vzor Request-Reply toho v základní variantě mnoho neumožňuje, proto bylo nutné umožnit asynchronní komunikaci implementací rozšíření v podobě socketů **ROUTER** a **DEALER** (dříve nazývány jako XREP a XREQ).

[49] HTTP, POP či IMAP

Řešení spočívá ve vytvoření více pracovních vláken (*workers*) souběžně řešících příchozí požadavky. Za tímto účelem je nutné vytvořit v serverovém vlákně socket typu DEALER, kterému přiřadíme komunikační protokol **inproc**.

Na serverové straně probíhá inicializace komunikace následovně:

```
//  Frontend socket talks to clients over TCP
Socket frontend = ctx.createSocket(ZMQ.ROUTER);
frontend.bind("tcp://" + host + ":" + port);

//  Backend socket talks to workers over inproc
Socket backend = ctx.createSocket(ZMQ.DEALER);
backend.bind("inproc://backend");

//  Launch pool of worker threads, precise number is not critical
for (int threadNbr = 0; threadNbr < 5; threadNbr++) {
    new Thread(new ServerWorker(threadNbr, ctx, api, new
        RequestHandlerJson(), new ResponseHandlerJson())).start();
}

//  Connect backend to frontend via a proxy
ZMQ.proxy(frontend, backend, null);
```

Kvůli podpoře více TCP spojení utvářených vůči serveru je nutné předřadit před socket typu DEALER ještě další typ socketu, kterým je ROUTER (byl vidět již v ukázce 2.3.1.3) hrající roli frontend socketu.

ROUTER pak přiřazuje vnitřní identifikátor každému k němu se připojujícímu socketu a následně obdrženou zprávu předává dále prostřednictvím socketu typu DEALER (backend socket) na jedno ze svých pracovních vláken.

Pracovním vláknům jsou jako parametry předány instance tříd **RequestHandlerJson** a **ResponseHandlerJson**, takže jsou schopny obsluhovat příchozí žádosti v tomto formátu.

O obsluhu požadavků a správnou komunikaci frontend socketů s backend sockety se stará *ZMQ.proxy* hrající roli prostředníka zmíněného výše 2.3.1.3.

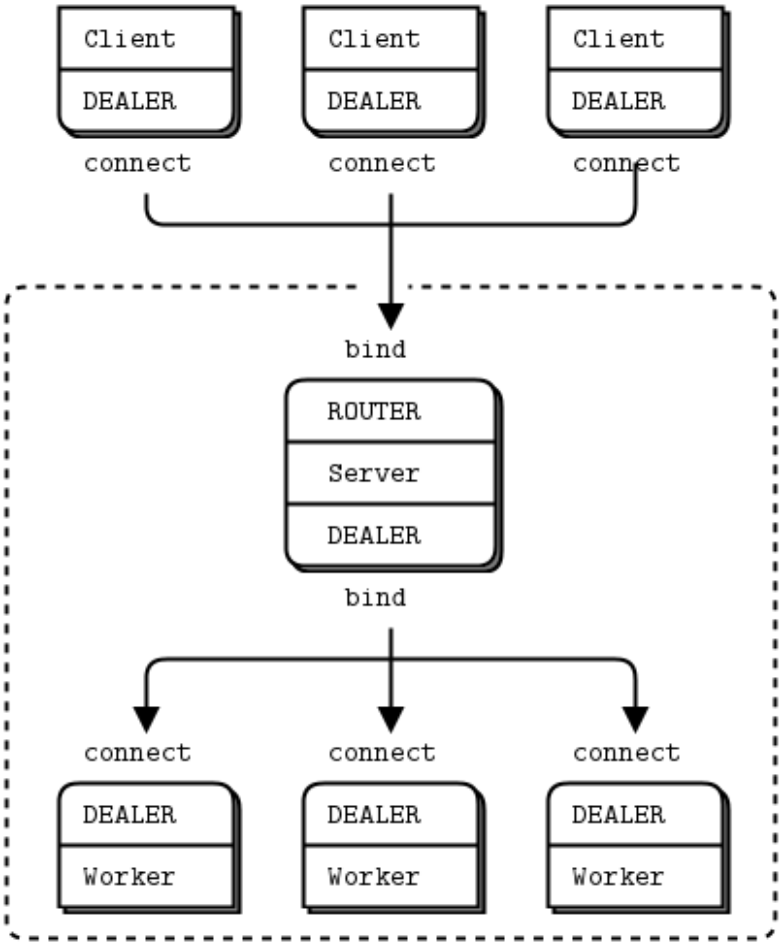
Pracovní vlákna (ServerWorker) používají též komunikační protokol inproc a socket typu DEALER.

```
Socket worker = ctx.createSocket(ZMQ.DEALER);
worker.connect("inproc://backend");
```

Komunikace probíhá tak, že DEALER komunikuje s DEALEREM a oproti serverovému vláknu, který ale používal pro inproc spojení metodu *bind()*, využívají pracovní vlákna metodu *connect()*, čili se k němu připojují, přijímají zprávy a starají se o jejich vyřízení.

Po navázání spojení je v těle metody *run()* pracovních vláken realizována veškerá komunikace s logikou systému. Dochází zde k předávání zpráv jednotlivým metodám systémového API. Bližším popisem se zabývá 2.3.1.3.

Obsloužené zprávy jsou poté zasílány zpět na serverové vlákno, které je přeposílá komunikujícím klientům



Obrázek 2.7: Vícevláknový asynchronní server s využitím ROUTER a DEALER. Zdroj: [13]

Správnou synchronizaci vláken má na starosti již zmíněná ZMQ.proxy. To je jedna z pokročilých vlastností ZeroMQ – pro vývoj vícevláknových aplikací nejsou potřeba žádné mutexy, zámky, ani jakékoliv další formy komunikace [13].

Technika zasílání zpráv a ServerWorker O několik řádku výše bylo vysvětleno, jakým způsobem jsou na serveru obsluhovány příchozí žádosti klientů. Nyní již víme, že tuto funkcionalitu mají na starosti pracovní vlákna typu **ServerWorker** a řekneme si, jakým způsobem jsou žádost zpracovávány.

Vláknová metoda *run()* má na svědomí veškerou obsluhu žádosti. Každý jednotlivý *worker* tak v sobě udržuje kontext komunikace a pokud je na vstupu

validní žádost, pro její zpracování je vyvolána příslušná operace rozhraní systému Recommeng. Sestavená odpověď je poté zaslána zpět klientovi.

```
try {
    Operation op = requestHandler.handleMessage(request);
    if (op.validateOperation()) {
        try {
            Reply reply = op.executeOperation(api);
            responseHandler.setReply(reply);
        } catch (Exception ex) {
            responseHandler.createInternalErrorReply("Internal error
                .");
        }
    } else {
        responseHandler.createErrorReply(op.getErrorMessage());
    }
} catch (MessageFormatException | IOException ex) {
    responseHandler.createErrorReply(ex.getMessage());
} catch (NullPointerException ex) {
    responseHandler.createErrorReply("Unrecognized operation.");
}
```

Výsledkem zpracování zprávy je operace. Spuštění vhodné operace je realizováno návrhovým vzorem Command. **Operation** je rozhraní deklarující metody pro validaci operace (*validateOperation()*) a její vykonání (*executeOperation()*).

Každá třída implementující toto rozhraní je pak konkrétní operací zasílanou na rozhraní systému.

Všechny typy podporovaných operací se nacházejí v aplikaci Ensemble v balíčku:

`cz.cvut.fit.bouchja1.ensemble.operation`

Jedná se o třídy splňující funkcionalitu dle funkčních požadavků 2.1.1. Jsou jimi například:

- `OperationCreateBanditCollection`
- `OperationCreateBanditSuperCollection`
- `OperationDetectBestBandit`
- `OperationFeedbackSupercollectionBandit`
- `OperationUseSupercollectionBandit`

Výsledkem vykonání operace je odpověď (třída **Reply** obsahující stavový kód, tělo zprávy, případně další hodnoty jako seznam algoritmů apod.), která je pomocí třídy **ResponseHandlerDefault** prezentována zpět klientovi.

2.3.1.4 API

API systému Recommeng reprezentuje interface **EnsembleApiFacade**. Podporované metody jsou vidět na obrázku 2.6.

Tyto metody jsou volány prostřednictvím operací z vrstvy pro obsluhu zpráv 2.3.1.3. Metody jsou přímo napojeny na bayesovské strategie, které jsou jedněmi ze stěžejních součástí systému.

Ukázkovou metodou je například metoda zprostředkovávající detekci nejlepšího algoritmu ze zadané kolekce. Nejprve proběhne ověření, zda v systému opravdu taková kolekce, vůči které je žádost směřována, existuje. Následně dojde k provedení žádosti a vytvoření odpovědi.

```
@Override
public Reply detectBestBandit(String banditCollectionId, String
    filter) {
    ResponseHandler responseHandler = new ResponseHandlerJson();
    Reply reply;

    if (existBanditSetId(banditCollectionId)) {
        BayesianStrategy strategyToUse = getStrategyByCollectionId(
            banditCollectionId);
        if ("best".equals(filter)) {
            Bandit banditToChoose = strategyToUse.sampleBandits();
            responseHandler.createSuccessReplyDetection("You␣should␣
                choose␣bandit␣with␣name␣" + banditToChoose.getName()
                 + "␣now.␣He␣is␣the␣best␣for␣this␣context.",
                banditToChoose.getId(), banditToChoose.getName(),
                strategyToUse.getId());
        } else {
            List<Bandit> banditsByOrder = strategyToUse.
                sampleBanditsAll(banditCollectionId);
            responseHandler.createSuccessReply("Bandit␣IDs␣by␣order:
                ␣" + banditsByOrder.toString());
        }
        reply = responseHandler.returnReply();
    } else {
        responseHandler.createNotFoundReply("There␣is␣no␣collection␣
            with␣ID␣" + banditCollectionId + "␣in␣application.");
        reply = responseHandler.returnReply();
    }
    return reply;
}
```

Klient, který žádal systém o takovéto doporučení, obdrží zprávu ve tvaru:

```
{"message":"You should choose bandit with name: latest. He is the
    best for this context.","status":200,"collection":1,"banditId":"
    latest","bestBandit":2}
```

Část tohoto výstupu lze použít jako parametry pro další dotaz (například pro zaslání zpětné vazby).

2.3.1.5 Bayesovská strategie pro kombinování metod

Strategii mají na starosti třídy nacházející se v balíčku:

`cz.cvut.fit.bouchja1.ensemble.bandits`

Bandit Bandit je třída reprezentující jednoho konkrétní banditu. Každý bandita má svůj identifikátor a atributy pro zaznamenávání výher a pokusů během jednotlivých her. Ty reprezentují jeho dosavadní znalosti.

BanditsMachine BanditsMachine je třída reprezentující herní automat. Každá samostatná bayesovská strategie (BayesianStrategy) má právě jeden herní automat. Automat obsahuje seznam banditů a číselné konfigurace pro výpočty zpětné vazby a míry učení. Konfigurace se nastavuje ve vnějším souboru (viz 2.3.1.2).

SuperBayesianStrategy SuperBayesianStrategy je třída reprezentující super kolekci obsahující více bayesovských strategii. V případě přítomnosti dvou bayesovských strategií, například kontextových atributů „ráno“ a „Evropa“, může z těchto strategií vytvořit super strategii „ráno v Evropě“.

BayesianStrategy BayesianStrategy je třída reprezentující online učící strategii k řešení strategie Multi-Armed Bandit 1.2.1.15. Vzhledem k přítomnosti super strategií je již jasné, že v systému může nezávisle na sobě fungovat více bayesovských strategií.

2.3.1.6 Důležité metody bayesovské strategie

Důležitou metodou třídy BayesianStrategy je metoda *sampleBandits()* starající se jednak o výběr z prior pravděpodobností distribucí banditů a následně o volbu nejlepšího z nich. Jedná se o implementaci teorie probírané v 1.2.4.

```
public Bandit sampleBandits() {
    //sample from the bandits's priors, and select the largest
        sample
    for (Integer bandit : banditsMachine.getBanditList().keySet()) {
        BetaDistribution beta = new BetaDistribution(1 +
            banditsMachine.getBanditAtIndex(bandit).getSuccesses(),
            1 + banditsMachine.getBanditAtIndex(bandit).getTrials()
            - banditsMachine.getBanditAtIndex(bandit).getSuccesses()
            );
    double inverseDistribution = beta.inverseCumulativeProbability(
        Math.random());
    roundInverseDistributions.add(inverseDistribution);
    }
    int banditIndexChoice = MathUtil.argmax(
        roundInverseDistributions);
    roundInverseDistributions.clear();
```

```
    return banditsMachine.getBanditAtIndex(banditIndexChoice);
}
```

Do rozdělení vstupují jako parametry dosavadní výsledky pokusů a úspěchů jednotlivých algoritmů z kolekce a dochází k výpočtu kvantilových funkcí 1.2.1.7 pro každý algoritmus, kdy je nakonec vybrán algoritmus s největší hodnotou a navrácen jako nejlepší možnost.

Dalšími metodami jsou pak *selectBandit(String banditId)* zpracovávající zpětnou vazbu o zvolení doporučeného algoritmu uživatelem a *calculateFeedback(String banditCollectionId, String banditId, String feedbackValue)*.

```
public void selectBandit(String banditId) {
    Bandit banditToUpdate = banditsMachine.getBanditAtIndex(Integer.
        parseInt(banditId));
    banditsMachine.updateTrials(banditToUpdate);
}

public void calculateFeedback(String banditCollectionId, String
    banditId, String feedbackValue) {
    Bandit banditToUpdate = banditsMachine.getBanditAtIndex(Integer.
        parseInt(banditId));
    banditsMachine.updateFeedback(banditToUpdate, feedbackValue);
}
```

Tyto metody zajišťují přepočet hodnot uchovávaných jednotlivými algoritmy (trials, successes) jako jejich vnitřní stav (vyjadřující dosud načerpané znalosti). Při každém přepočtu jsou hodnoty ještě násobeny konstantou určující míru učení 1.2.4.2.

V případě super strategie (třída SuperBayesianStrategy) je poměrně důležitou metodou *chooseBestFromSuperStrategy()*. Ze zadané super kolekce je z odhadů vybrán vždy jeden nejlepší algoritmus z jejích kolekcí a pomocnou metodou *bestBanditFromCombination(List<Bandit> banditsCombination)* je spočtena četnost výskytů jednotlivých algoritmů (navrácených jako nejlepší) z kolekcí. V případě shody stejného výskytu dvou a více algoritmů je na náhodným výběrem zvolena jedna, která je doporučena uživateli.

2.3.1.7 Zpětná vazba

Výše byly zmíněny dvě metody (*selectBandit()* a *calculateFeedback()*), které zprostředkovávají zpětnou vazbu pro systém. Nyní si pojďme popsat jejich realizaci.

Tělo metody updateTrials() je jednoduché a zpětná vazba týkající se připočítávání pokusů triviální:

```
double newTrials = rate * trials;
trials = newTrials + 1;
normalizedTrialsFrequencyInTime += totalTrialsCountsToBoost;
```

V případě souhlasu s výsledky doporučenými následovaným algoritmem je k jeho vnitřnímu stavu uchovávající poměr pokusů (kolikrát byl vybrán) přičteno číslo 1.

Zajímavější již je zasílání informace o kvalitě doporučení. Dle typu zpětné vazby (pozitivní či negativní) jsou volány dvě metody – *makePositiveFeedback(banditToUpdate)* či *makeNegativeFeedback(banditToUpdate)* a pomocí parametrů voleny tři hodnoty:

```
ensemble.feedback.possitive.winner=1.0
ensemble.feedback.possitive.losers=0.1
ensemble.feedback.negative.stupid=0.2
```

V případě pozitivní zpětné vazby je danému algoritmu přičítána definovaná odměna (emphresult):

```
double newSuccesses = rate * successes;
successes = newSuccesses + winner;
```

Všem ostatním algoritmům je jistá poměrová hodnota odečtena (za předpokladu, že nemají vnitřní stav menší než odečítaná hodnota):

```
if (successes > (losers*rate)) {
    successes = successes - (losers*rate);
    double newSuccesses = rate * successes;
    successes = newSuccesses;
} else {
    double newSuccesses = rate * successes;
    successes = newSuccesses;
}
```

Těchto počtů je využíváno v případě pozitivní zpětné vazby. Ještě je ale nutné uvažovat případ, kdy dáme na radu systému, zvolíme algoritmus, který nám poradil, ale s výsledky doporučení nejsme spokojeni. V takovém případě je udělována negativní zpětná vazba.

Vazba je udělována pouze algoritmu, se kterým jsme nebyli spokojeni. Vnitřní stav ostatních algoritmů zůstává nezměněn.

Penalizační funkce je realizována následovně:

```
successes = successes - (stupid*rate);
double newSuccesses = rate * successes;
successes = newSuccesses;
```

2.3.1.8 Pravidelné ukládání časových snímků

Pravidelné ukládání časových snímků jsem implementoval Spring démonem (*cron*). Četnost jeho spouštění lze ovlivňovat parametrem 2.3.1.2 v konfiguračním souboru aplikace. K tomuto účelu jsem vytvořil třídu **ScheduledJob** spouštějící pravidelný časový úkol, který zprostředkuje persistenci aktuálního stavu aplikace (tedy všech běžících bayesovských strategií), který je v té době v paměti, do databáze.

Samozřejmě za předpokladu, že je použití databáze v konfiguračním souboru nastaveno. V opačném případě by tovární metoda třídy **StorageFactory** zvolila k použití jinou formu práce s daty.

```
public static IStorage getStorage(Environment env) {
    switch (env.getProperty("storage")) {
        case "cassandra" :
            return new CassandraStorage(env.getProperty("cassandra.
                host"), env.getProperty("cassandra.keyspace"));
        default :
            return new JvmStorage();
    }
}
```

Tento přístup vede k rozšiřitelnosti o další typy databází, které by mohl systém v budoucnu podporovat.

2.3.1.9 Spojení s databází

Úložiště je realizováno třídou **CassandraStorage**. Během jejího zavádění do systému při startu aplikace jsou v konstruktoru volány dvě metody – *connect()* a *createSchema()*. V prvním případě dochází ke spojení s databázovým klastrem, pomocí kterého je následně získána *session*.

Pomocí session a její metody *execute()* lze vykonávat jednotlivé dotazy.

Použita je hned v metodě createSchema(), která je zodpovědná za vytvoření datového modelu navrženého pro systém Recommeng.

Ukázka vytvoření keyspace pro všechny column families aplikace:

```
private void createSchema() {
    session.execute("CREATE␣KEYSPACE␣IF␣NOT␣EXISTS␣" + keyspace + "␣
        WITH␣replication␣"
            + "=␣{'class':'SimpleStrategy',␣'replication_factor':3};
                ");
```

Column families jsem pro účely aplikace vytvořil tři – *collection*, *supercollection* a *algorithm*.

Třída implementuje několik metod svého rozhraní IStorage pro uložení aktuálního stavu aplikace, načtení poslední známé konfigurace a vytvoření kolekcí banditů. Při realizaci funkcionality metod je bohatě využíváno možností, které nabízí DataStax driver pomocí CQL, například tvorba předpřipravených dotazů konstrukcí **PreparedStatement**.

Následuje příklad ukládání aktuálního stavu z paměti systému do databáze.

```
@Override
public void saveCurrentState(LastEnsembleConfiguration strategies) {
    if (strategies != null) {
        for (BayesianStrategy strategy : strategies.getStrategies().
            values()) {
             Map<Integer, Bandit> bandits = strategy.
                 getBanditsMachine().getBanditList();
            if (bandits.size() > 0) {
                PreparedStatement statement = session.prepare(
                        "INSERT␣INTO␣" + keyspace + ".algorithm␣"
```

```
cqlsh> select * from ensemble.algorithm where collection_id=1 and algorithm_id=3;

 collection_id | algorithm_id | event_time               | algorithm_name | successes_rate | trials_rate
---------------+--------------+--------------------------+----------------+----------------+-------------
             1 |            3 | 2014-05-08 04:16:20+0200 |         latest |         4.0951 |      4.0951
             1 |            3 | 2014-05-08 04:16:10+0200 |         latest |          3.439 |       3.439
             1 |            3 | 2014-05-08 04:16:00+0200 |         latest |           2.71 |        2.71
             1 |            3 | 2014-05-08 04:15:50+0200 |         latest |            1.9 |         1.9
             1 |            3 | 2014-05-08 04:15:40+0200 |         latest |              1 |           1
             1 |            3 | 2014-05-08 04:15:30+0200 |         latest |              1 |           1
             1 |            3 | 2014-05-08 04:15:20+0200 |         latest |              0 |           0
             1 |            3 | 2014-05-08 04:15:10+0200 |         latest |              0 |           0
             1 |            3 | 2014-05-08 04:15:00+0200 |         latest |              0 |           0
```

Obrázek 2.8: Ukázka uložení časových snímků vnitřního stavu systému v databázi

```
                        + "(collection_id, algorithm_id,
                            algorithm_name, event_time, trials_rate,
                            successes_rate) "
                        + "VALUES (?, ?, ?, ?, ?, ?);");

                Date actualDate = Calendar.getInstance().getTime();

                for (Map.Entry<Integer, Bandit> entry : bandits.
                    entrySet()) {
                    BoundStatement boundStatement = new
                        BoundStatement(statement);
                    session.execute(boundStatement.bind(
                            strategy.getId(),
                            entry.getValue().getId(),
                            entry.getValue().getName(),
                            actualDate,
                            entry.getValue().getSuccesses()));
                    logger.info("Time: " + actualDate + "Saving
                        bandit with ID " + entry.getValue().getName
                        () + " into collection : " + strategy.
                        getCollectionId());
                }
                        ...
                        ...
}
```

2.3.2 RESTful API

RESTful API je realizováno jako *Java Web Application* s názvem **ensemble-RestApi**.

Důležitou roli v modulu hraje přítomnost a správná konfigurace tzv. *Web Application Deployment Descriptor* (soubor **/WEB-INF/web.xml**). V tomto souboru je definováno vše, co by měl server, na kterém aplikace poběží, o aplikaci vědět (informace o příslušných servletech, filtrech apod.).

Pro potřeby modulu RESTful API jsem v tomto souboru definoval *lis-*

tener[50] *pro Spring*. Dále *servlet pro Jersey*, jemuž jsem parametrem předal třídu **RecommengApplication**, a nastavil příslušné mapování servletu na specifickou URL.

```
<servlet>
    <servlet-name>jersey-serlvet</servlet-name>
    <servlet-class>
        org.glassfish.jersey.servlet.ServletContainer
    </servlet-class>
    <init-param>
        <param-name>javax.ws.rs.Application</param-name>
        <param-value>cz.cvut.fit.bouchja1.mi_dip.rest.client.service
            .RecommengApplication</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>jersey-serlvet</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Pomocí třídy RecommengApplication, rozšiřující třídu ResourceConfig frameworku Jersey, jsem zaregistroval všechny aplikační komponenty, které budou použity JAX-RS aplikací, tedy vytvářeným RESTful API.

```
public RecommengApplication(){
    register(RequestContextFilter.class);
    register(AlgorithmEndpointImpl.class);
    register(ArticleEndpointImpl.class);
    register(EnsembleEndpointImpl.class);
    register(JacksonFeature.class);
}
```

Třída registruje následující komponenty:

- org.glassfish.jersey.server.spring.scope.RequestContextFilter

Jedná se o Spring filter, který poskytuje propojení mezi JAX-RS a Spring žádostmi.

- cz.cvut.fit.bouchja1.mi_dip.rest.client.endpoint.AlgorithmEndpointImpl
- cz.cvut.fit.bouchja1.mi_dip.rest.client.endpoint.ArticleEndpointImpl
- cz.cvut.fit.bouchja1.mi_dip.rest.client.endpoint.EnsembleEndpointImpl

Tyto tři třídy jsou služby REST API.

- org.glassfish.jersey.jackson.JacksonFeature

[50] Listener je aplikace, jež vyčkává na vznik nějaké události. Jakmile událost nastane, listener zareaguje a převezme její řízení.

Registruje Jackson JSON poskytovatele pro zpracování příchozích dat ve formátu JSON.

Nakonec jsem definoval filtr *CharacterEncodingFilter* s UTF-8 kódováním pro všechny URL splňující vzor:

```
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2.3.2.1 Realizace endpointů

Třídy z balíčku:

`cz.cvut.fit.bouchja1.mi_dip.rest.client.endpoint`

jsou služby typu REST obsluhující všechny žádosti směřující na jimi mapované zdroje. Programově má každá tato třída anotaci definující její relativní URI cestu. V případě třídy **ArticleEndpointImpl** vypadá definice následovně:

```
@Component
@Path(ArticleEndpointImpl.ENDPOINT_PATH)
public class ArticleEndpointImpl implements ArticleEndpoint {

    public static final String ENDPOINT_PATH = "/cores";
    public static final String USER_ARTICLE_PATH = "/{coreId}/
        document";

    @Autowired
    private ArticleEndpointHelper articleEndpointHelper;
    ...
```

Anotace *@Path* v tomto případě značí, že třída se bude nacházet na URI */cores*.

Jedna z jejích služeb umožňující vytvářet či aktualizovat informace o zpětné vazbě u dokumentu v indexu (zasíláním žádostí na URI zdroje */cores/{coreId}/document*) je definována takto:

```
@Path(USER_ARTICLE_PATH)
@Consumes({MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@POST
@Override
public Response updateBehavioralToArticle(@PathParam("coreId")
    String coreId, UserArticleDocument userArticle) {
    return articleEndpointHelper.updateBehavioralToArticle(coreId,
        userArticle);
}
```

O samotné reprezentaci zdroje referuje příslušná podpodsekce 2.3.2.2.

Provádění má na starosti v tomto případě **ArticleEndpointHelper**. Ostatní služby mají též své helpery – třídy pomáhající jim v obsluze žádosti zodpovědné za vytváření odpovědí, které rozšiřují rodičovskou třídu **CommonEndpointHelper** implementující rozhraní **EndpointHelper**.

Rozhraní deklaruje více metod souvisejících s odpovědí, například metody pro vytvoření odpovědi dle typu návratového kódu a sestavení odpovědi.

```
public Response getNotFoundResponse(String message);
public Response build(ResponseBuilder builder, String message);
```

Proces tvorby odpovědi pak vypadá tak, že konkrétní helper volá ve své metodě službu zajišťující komunikaci s indexem. V případě metody *updateBehavioralToArticle(String coreId, UserArticleDocument userArticle)* pro vkládání vztahu uživatel-hodnocení-článek do indexu je po provedení příslušných kontrol, kterými jsou validace vstupních dat a podobně, vytvořena odpověď.

```
public Response updateBehavioralToArticle(String coreId,
    UserArticleDocument userArticle) {
    Response resp = null;
    if (solrService.isServerCoreFromPool(coreId)) {
        String message = UserArticleValidator.validateUserArticle(
            userArticle);
        if ("success".equals(message)) {
            try {
                solrService.putUserArticle(coreId, userArticle);
                resp = getOkResponse("User-rating-item␣succesfully␣
                    added␣into␣Solr␣core.");
            } catch (SolrServerException ex) {
                logger.error(ex);
                resp = getServerError(ex.getMessage());
            } catch (IOException ex) {
                logger.error(ex);
                resp = getServerError(ex.getMessage());
            }
                ...
                ...
    return resp;
}
```

Helper tedy volá dle výsledků programu jednu z metod své rodičovské třídy (například *getBadRequestResponse(message)*) s příslušnou zprávou v parametru. Metodou *build(ResponseBuilder builder, String message)* je pak vytvářena samotná odpověď.

```
@Override
public Response getNotFoundResponse(String message) {
    return build(Response.status(Response.Status.NOT_FOUND), message
        );
}

@Override
public Response build(ResponseBuilder builder, String message) {
    return builder.entity(message).build();
```

```
}
```

2.3.2.2 Reprezentace zdrojů

Zdroje jsou jedním ze stěžejních konceptů architektury REST. Kromě toho, že jsou adresovány příslušnými globálními identifikátory (v HTTP realizaci např. pomocí URI), mají též jednu nebo více reprezentací, ve které jsou vystaveny okolnímu světu, a pomocí které je možné s těmito zdroji manipulovat.

V modulu pro RESTful API reprezentuji zdroje jako třídy v Javě. Například při tvorbě zdroje */cores/coreId/document* je vytvářena třída **UserArticleDocument**.

```
@XmlRootElement
public class UserArticleDocument implements Serializable {

    private static final long serialVersionUID =
        -8039686696076337053L;
    private String articleId;
    private int userId;
    private double rating;
    private int groupId;
    ...
```

Reprezentace tohoto zdroje ve formátu JSON by pak mohla vypadat například takto:

```
{
    "articleId":"http://somedomain.org/somearticle.html",
    "group":123,
    "userId":42,
    "userRating":5.0
}
```

2.3.2.3 Komunikace s Recommeng systémem

Vytvořil jsem též službu **EnsembleEndpointImpl** pro komunikaci s Recommeng systémem. Rozdíl oproti zbylým dvou službám (AlgorithmEndpointImpl a ArticlesEndpointImpl) je v rozdílném chování a funkčnosti její pomocné třídy **EnsembleZeroMqHelper**.

Tato služba, ač běžící jako součást serverové aplikace, hraje vůči Recommeng systému roli klientskou. Pro vnější uživatele zastává tradiční roli serveru.

EnsembleZeroMqHelper zpracovává příchozí žádosti od uživatelů prostřednictvím RESTful API. Poté transformuje do JSON formátu žádost, kterou umí Recommeng systém zpracovat.

```
EnsembleRequest req = new EnsembleRequest();
req.setMethod("POST");
req.setPath("/ensemble/services/collection");
req.setBody("collectionId=" + banditCollection.getName() + "&bandits
    =" + formatBanditIds(banditCollection.getBanditIds()));
```

Pro spojení se systémem je vytvořeno vlákno **ClientTask**, jemuž je parametrem předána zpráva ve formátu JSON, a které volá metodu *call()*.

```
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<String> result = executor.submit(new ClientTask(req.toString
    ()));

try {
    resp = buildResponse(result.get());
} catch (Exception ex) {
    resp = getServerError(ex.getMessage());
}
```

V těle metody call se vytvoří klientský socket prostřednictvím tcp a předřazená žádost je odeslána do systému. ClientTask pak vyčkává na odpověď (*resp*). Poté, co ji obdrží a zpracuje do formátu HTTP odpovědi (*buildResponse()*), ji vrací zpět žádajícímu klientovi.

```
ZContext ctx = new ZContext();
Socket client = ctx.createSocket(ZMQ.DEALER);
client.connect("tcp://localhost:5555");
```

2.3.2.4 Komunikace sady algoritmů pro doporučování

Doporučení je vyvoláno klientskou žádostí na rozhraní reprezentované třídou **AlgorithmEndpointImpl** a její zpracování zajišťuje metoda *recommend()*. Metoda má několik vstupních parametrů:

```
@Path(ALGORITHM_PATH)
@Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
@GET
@Override
public Response recommend(@PathParam("coreId") String coreId,
        @PathParam("algorithmId") String algorithmId,
        @QueryParam(value = "coreId") String coreId,
        @QueryParam(value = "groupId") int groupId,
        @QueryParam(value = "userId") int userId,
        @QueryParam(value = "documentId") String documentId,
        @QueryParam(value = "text") String text,
        @QueryParam(value = "limit") int limit) {
```

Jak vidno, metoda umí v žádosti přijmout spoustu parametrů. Při každém vyvolání jsou všechny existující příchozí parametry uloženy do mapy a ta je následně předána tovární metodě, jejímž cílem je vytvořit na základě uživatelského vstupu vhodnou instanci třídy pro doporučení obsahu.

```
Map<String, String> algorithmParams = createAlgorithmParams(coreId,
    algorithmId, groupId, userId, documentId, text, limit);
IAlgorithm algorithm = AlgorithmFactory.getAlgorithm(algorithmId,
    algorithmParams);

if (algorithm == null) {
    return algorithmEndpointHelper.createAlgorithmNotFound();
```

```
}
else return algorithmEndpointHelper.getRecommendation(algorithm);
```

Služby využívají pro zpracování žádostí a vytváření odpovědí pomocnou třídu **AlgorithmEndpointHelper** (podobně jako 2.3.2.3). Tato třída v sobě navíc udržuje odkaz v podobě instance třídy **SolrService**, kterou předává třídám implementující jednotlivé doporučovací algoritmy (viz dále v podsekci 2.3.3).

Třída SolrService Třída implementuje metody pro komunikaci s Apache Solr a funguje též jako pool instancí serverových spojení pro různá jádra Solr. Vytvoření poolu bylo nutností kvůli možnosti znovu použít již vytvořené instance třídy **HttpSolrServer**.

Jejími atributy jsou:

```
private String serverUrl;
private Map<String, HttpSolrServer> validServers = new HashMap<
    String, HttpSolrServer>();
private Set<String> validSolrCores;
```

HttpSolrServer je thread-safe[51] a pokud jej použijeme k vytvoření nové instance s URL některého z jader Solr v parametru, je nutné tuto instanci znovu použít pro všechny žádosti směřující na danou URL [11].

V opačném případě, kdy jsou instance vytvářeny bez jakéhokoliv rozmyslu a strategie, hrozí *leak* připojení [35].

Validní jádra Solr pro připojení do poolu se nastavují v souboru aplikačního kontextu pro Spring:

```
<bean id="solrService" class="cz.cvut.fit.bouchja1.mi_dip.rest.
    client.solr.SolrService">
    <property name="serverUrl" value="http://localhost:8089/solr/"/>
    <property name="validSolrCores">
        <set>
            <value>mi_dip_core1</value>
            <value>mi_dip_core2</value>
        </set>
    </property>
</bean>
```

SolrService obsahuje také metodu *createValidSolrServers()*, jež je anotována jako *@PostConstruct*.

Metoda s anotací @PostConstruct je vyvolána ještě před samotným vytvořením instance třídy. Účelem je naplnění poolu příslušnými validními instancemi spojení.

```
Iterator<String> validCores = validSolrCores.iterator();
while (validCores.hasNext()) {
    String core = validCores.next();
```

[51] Programové operace jsou prováděny správně i tehdy, kdy jsou prováděny více vlákny současně.

```
    validServers.put(core, new HttpSolrServer(serverUrl + core));
}
```

Kdykoliv je pak metodách tříd doporučovacích algoritmů navazováno spojení se serverem, dle zadaného identifikátoru jádra (coreId) se bere instanci z poolu, který má parametry *HashMap<String, HttpSolrServer>*.

```
HttpSolrServer server = solrService.getServerFromPool(coreId);
```

2.3.3 Algoritmy pro doporučování obsahu pomocí Apache Solr

Třídami, které jsou schopné různými způsoby doporučovat obsah, jsou:

- AlgorithmWeightedRating
- AlgorithmLatest
- AlgorithmMlt
- AlgorithmRandom
- AlgorithmUserBasedCf
- AlgorithmItemBasedCf

Všechny tyto třídy implementují rozhraní IAlgorithm a veškerá logika doporučení je vykonávána v implementacích metody recommend() konkrétních doporučovacích tříd a jejich pomocných metod.

2.3.3.1 Reprezentace dokumentů pro účely doporučování

Pro potřeby modelové úlohy bylo nutné zamyslet se nad reprezentací a strukturou dat v Apache Solr. Z požadavků a technických možností Solr vyplynulo přímočaré řešení spočívající ve vytvoření dvou jader - jedno pro vkládání a uchovávání článků (*articleCore*), druhé pro zaznamenávání interakce uživatele s doporučenými články (*behavioralCore*).

Jádro pro články budou při svých doporučeních využívat obsahově založené algoritmy (random, latest, more like this), jádro pro interakce budou využívat algoritmy pracující s uživatelskými hodnoceními (nejlépe hodnocené, kolaborativní filtrování).

Article core Jakýkoliv nově vytvořený článek je zaslán do tohoto jádra a přidán jako dokument s následující strukturou:

```
<doc>
   <int name="id">1</int>
   <str name="articleId">http://pnjj5cr4f9 f500k9vld.org</str>
   <str name="articleText">695t5r2fgy something.</str>
   <int name="group">789</int>
   <date name="time">2012-09-19T09:42:12Z</date>
```

```
    <long name="_version_">1467289878138978304</long>
</doc>
```

Behavioral core Po každé uživatelské interakci se článkem je tato interakce přidána do jádra jako trojice userId, articleId a userId_rating. Pokud dokument s požadovaným articleId není v jádře přítomen, je v něm vytvořen. Pokud přítomen je, proběhne pouze přidání dvojice userId a userId_rating.

```
<doc>
    <int name="id">6</int>
    <str name="articleId">http://pnjj5cr4f9 f500k9vld.org</str>
    <arr name="userId">
      <int>1</int>
      <int>40</int>
      <int>15</int>
    </arr>
    <int name="group">789</int>
    <float name="1_rating">5.0</float>
    <float name="40_rating">5.0</float>
    <float name="15_rating">5.0</float>
    <float name="weightedRating">5.0</float>
    <long name="_version_">1467370760759672832</long>
</doc>
```

2.3.3.2 Algoritmus náhodného výběru

Náhodný výběr by neměl mít již z podstaty věci příliš složitou implementaci. Rozhodl jsem se proto využít možnosti, kterou nabízí Solr. Rou možností je přidání speciálního typu a pole tohoto typu do schématu[52] jádra.

```
<fieldType name="random" class="solr.RandomSortField" indexed="true"
     />
<dynamicField name="random_*" type="random" />
```

Pro dotaz je poté možné využít speciálního vstupu pro řazení výsledků. Solr podporuje parametrem *sort* řazení dokumentů dle specifikovaného pole v indexu. Díky definici náhodného typu lze zaslat jako vstup pro řazení field *random* s náhodným prefixem. Například:

```
q=*:*&sort=random_12939291%20desc
```

Solr tímto způsobem vyhodnotí pořadí dokumentů dle jména náhodného pole a verze indexu. To tedy znamená, že pokaždé, kdy je použito stejného jména náhodného pole a toho samého indexu (který nebyl mezi dotazy změněn), jsou navráceny ty samé výsledky. Proto jsem byl nucen zanést do dotazu

[52]Soubor schema.xml popisující veškeré detaily o tom, které filedy může jádro indexu obsahovat, a jakým způsobem s nimi má být nakládáno při přidávání dokumentů do indexu či dotazování.

ještě další prvek náhody a to tak, že pro názvy pole využívám náhodného čísla v rozsahu 1 až maximální hodnota celočíselného datového typu.

Programová realizace je pak již triviální:

```
int random = generator.nextInt(Integer.MAX\_VALUE) + 1;
String sortOrder = "random\_" + random;

query.setSortField(sortOrder, SolrQuery.ORDER.desc);
```

2.3.3.3 Algoritmus výběru dle nejnovějších položek

Logika řazení funguje stejně jako v případě náhodného výběru výše díky parametru *sort*. S tím rozdílem, že vstupem pro parametr je pole *time* uchovávající datum vytvoření článku.

```
<date name="time">2012-09-19T09:42:12Z</date>
```

Navrácené dokumenty jsou tak řazeny dle této hodnoty.

2.3.3.4 Algoritmus výběru nejlépe hodnocených položek

Pro výběr nejlépe hodnocených položek je již zapotřebí použít sofistikovanějšího mechanismu. Nelze se spoléhat na celkovou sumu či aritmetický průměr. Pomocí takového přístupu by totiž například položka, která byla hodnocena tisíckrát se známkou 1 (z 5 možných) byla považována za lepší než například položka hodnocená stokrát, ale vždy se známkou 5.

Rozhodl jsem se použít podobný přístup, jaký využívá známá filmová databáze IMDB[53].

K výpočtu váženého hodnocení je využito bayesovských odhadů. Výpočet je realizován pomocí následující formule:

$$W = \frac{Rv+Cm}{v+m}$$

W značí výsledné vážené hodnocení

R značí průměrné hodnocení položky

v značí počet hodnotitelů položky

m značí minimální počet hodnocení potřebných k objevení se ve výsledku (IMDB pro potřeby výskytu filmu v prvních 250 užívá konstatnty 250000)

C značí průměr hodnocení všech položek (IMDB používá hodnotu 7.0)

Výpočet váženého hodnocení probíhá po přidání každé další uživatelské interakce do jádra behavioralCore. Přepočet je nutný, neboť do dokumentů přibývají uživatelé s novými hodnoceními, mění se tak počty a průměrné hodnoty vstupující do vzorce. Mechanismu přepočítávání je realizován v metodě třídy **SolrService**.

[53] http://www.imdb.com/chart/top

```
recalculateWeightedRating(SolrInputDocument sid)
```

Pro výpočet jsou důležitá pole userId a userId_rating daného dokumentu.

Navrácení seřazených výsledků pak zajišťuje třída **AlgorithmWeightedRating** a nejedná se o nic jiného, než o seřazení dle vstupního parametru *weightedRating*. Stejně jako v předchozích dvou případech.

2.3.3.5 Algoritmus výběru dle podobnosti obsahu

K doporučování dokumentů obsahově podobných jiným dokumentům v indexu mi byly opět velmi nápomocné nativní mechanismy Apache Solr. Tentokrát je řeč o vyhledávací komponentně známé jako *MoreLikeThis*[54].

Přístup spočívá v přidání request handleru do konfigurace jádra[55].

```
<requestHandler name="/mlt" class="solr.MoreLikeThisHandler">
</requestHandler>
```

Podobnost je pak počítána na základě jednoho nebo více specifikovaných polí v dotazu. Do handleru lze zasílat velký počet parametrů, například:

mlt.fl pro specifikaci pole, které má být použito k výpočtu podobnosti,

mlt.mintf pro minimální frekvenci termů dokumentu,

mlt.minwl pro minimální délku slova uvažovaného pro výpočet podobnosti.

Realizace tohoto mechanismu se nachází ve třídě **AlgorithmMlt**, kde jsem specifikoval parametry.

```
SolrQuery query = new SolrQuery();
query.setRequestHandler("/" + MoreLikeThisParams.MLT);
query.set(MoreLikeThisParams.MATCH_INCLUDE, true);
query.set(MoreLikeThisParams.MIN_DOC_FREQ, 1);
query.set(MoreLikeThisParams.MIN_TERM_FREQ, 1);
query.set(MoreLikeThisParams.MIN_WORD_LEN, 1);
query.set(MoreLikeThisParams.BOOST, false);
query.set(MoreLikeThisParams.SIMILARITY_FIELDS, "articleText");
query.set(MoreLikeThisParams.MAX_QUERY_TERMS, 1000);
query.setRows(limitToQuery);
query.setQuery("articleId:" + document.getFieldValue("articleId"));
query.setFilterQueries("group:" + document.getFieldValue("group"));
```

Jako vstup pro porovnání slouží jeden dokument z jádra.

2.3.3.6 Algoritmus kolaborativního filtrování

Při provádění analýzy stávajících řešení pro doporučování jsem se dozvěděl o knihovně Apache Mahout 2.2.4.1, kterou využívá pro své potřeby systém Mendeley 2.2.4.1, načež jsem se rozhodl pro to ji vyzkoušet.

[54] https://wiki.apache.org/solr/MoreLikeThis

[55] Soubor solrconfig.xml.

Řešení budu demonstrovat na user-based přístupu realizovaného třídou **AlgorithmUserBasedCf**.

Nejprve bylo nutné získat z úložiště seznam uživatelských ID a ID dokumentů, které tito uživatelé hodnotili a tato data uložit do kolekce FastByIDMap.

```
FastByIDMap<FastIDSet> userData = new FastByIDMap<FastIDSet>();
```

Každý uživatel byl poté přidán do kolekce a z této kolekce byl vytvořen datový model.

```
userData.put(userRelatedId, new FastIDSet(itemValues));
DataModel model = new GenericBooleanPrefDataModel(userData);
```

Po vytvoření datového modelu je již možné konstruovat doporučení. Na výběr je několik podobnostních metrik, například euklidovská vzdálenost, pearsonův korelační koeficient či log-likelihood. Vzhledem k použití booleovského modelu preferencí jsem zvolil LogLikelihoodSimilarity. Pro doporučení je ještě stanovena sousedská funkce (doporučované položky pro uživatele jsou počítány na základě podobnosti mezi uživatelem a uživateli nacházejícími se v modelu) a následně provedeno doporučení.

```
UserSimilarity similarity = new LogLikelihoodSimilarity(model);
UserNeighborhood neighborhood = new NearestNUserNeighborhood(2,
    similarity, model);
long[] neighbors = neighborhood.getUserNeighborhood(Long.parseLong(
    userId));
Recommender recommender = new GenericBooleanPrefUserBasedRecommender
    (model, neighborhood, similarity);
List<RecommendedItem> recommendedItems = recommender.recommend(Long.
    parseLong(userId), limit);
```

2.4 Testovací klient

Testovací klient je realizován jako samostatná *Java Application* s názvem **RecommengClient**.

Pro komunikaci s RESTful API je využíváno již jednou použitého frameworku Jersey, tentokrát však v jeho variantě Client. Prvním krokem je inicializace klienta pro komunikaci s rozhraním v metodě *invokeAndRegisterClient()* třídy **EnsembleClient**. Odkaz na vytvořenou klientskou instanci budou pro svou komunikaci užívat všechna vlákna (alias žádající klienti) dále v systému.

```
private void invokeAndRegisterClient() {
    ClientConfig clientConfig = new ClientConfig()
            .register(JsonProcessingFeature.class)
            .property(JsonGenerator.PRETTY_PRINTING, true);
    client = ClientBuilder.newClient(clientConfig);
}
```

Následně jsou vytvářeny a spouštěny instance vláknových tříd, které mají simulovat souběžný přístup více uživatelů k systému. Každému vláknu je kromě identifikátoru značícího ID uživatele předávána vytvořená instance třídy EnsembleClient s inicializovaným klientem.

Vlastní komunikace probíhá tak, že jednotlivá vlákna volají metody třídy *Communication*. Těmto metodám jsou skrze parametry předávány informace týkající se chování klientů.

Například uživatel, který chce v systému vytvořit kontextovou kolekci s názvem *morning* obsahující algoritmy *latest*, *random* a *mlt*, nejprve vytvoří nový JsonObject a ten předá parametrem metodě, jež naváže spojení s rozhraním systému a zašle žádost o vytvoření kolekce.

```
JsonObject collection = Json.createObjectBuilder()
        .add("name", "morning")
        .add("banditIds", Json.createArrayBuilder()
        .add("latest")
        .add("random"))
        .build();

JSONObject createContextCollectionResp = communication.
    createContextCollectionRest(collection);
```

```
public JSONObject createContextCollectionRest(JsonObject collection)
     {
    Client client = clientApi.getClient();

    Response response = client.target(clientApi.
        getRestfulApiLocation())
            .path("/collection")
            .request(MediaType.APPLICATION_JSON_TYPE)
            .post(Entity.entity(collection, MediaType.
                APPLICATION_JSON_TYPE));

    JSONObject jsonObj = new JSONObject(response.readEntity(String.
        class));
    return jsonObj;
}
```

Vzniklá klientská aplikace byla použita k simulaci chování systému.

Kapitola 3

Experimentální část

K ověření funkčnosti systému je nutné provést několik experimentů. Z pozorovaných výsledků experimentů lze často vyvodit jisté obecné vlastnosti a stanovit nejvhodnější konfiguraci pro řešení daného problému.

Vlastnostmi, které mě pro účely experimentů zajímají, jsou:

- četnost doporučení,
- počet kroků (za krok považuji žádost o doporučení + zvolení doporučení + zpětná vazba),
- poměr normalizovaného součtu úspěchů ku normalizovanému poměru pokusů.

Následující experimenty proběhly spouštěním předpřipravených testů v klientské aplikaci 2.4.

3.1 Jeden úspěšný algoritmus

Experiment má za cíl zjistit, jakým způsobem se bude vyvíjet vnitřní stav systému, pokud budou na doporučení jedním konkrétním algoritmem neustále zasílány pozitivní zpětné vazby. Dle předpokladu by měl tento jeden algoritmus po čase zcela převážit.

3.1.1 Pozorované vlastnosti při různě volených parametrech

3.2 Jeden úspěšný algoritmus, který je po několika krocích srážen

Experiment má za cíl zjistit, jakým způsobem se bude vyvíjet vnitřní stav systému, pokud budou na doporučení jedním konkrétním algoritmem neustále zasílány pozitivní zpětné vazby, ale po čase dojde ke změně preferencí a tento

doporučení udělená tímto algoritmem začnou být odmítána. Dle předpokladu by měl tento algoritmus nejprve zcela převážit, po změně preferencí by ale opět měly začít postupně dostávat šanci i jiné algoritmy.

3.2.1 Pozorované vlastnosti při různě volených parametrech

3.3 Všechny algoritmy doporučují rovnoměrně

Experiment má za cíl zjistit, jakým způsobem se bude vyvíjet vnitřní stav systému, pokud bude o všech algoritmech a pozitivních i negativních zpětných vazbách rozhodováno přibližně stejně. Dle předpokládané úvahy by měl systém zůstat po celou dobu běhu nerozhodný a střídavě doporučovat všechny algoritmy.

3.3.1 Pozorované vlastnosti při různě volených parametrech

3.4 Test na odolnost vůči anomáliím

Experiment má za cíl zjistit, jakým způsobem se bude vyvíjet vnitřní stav systému, pokud bude do běhu čas od času zanesena nějaká anomálie. Takovou anomálií může být například omyl při zaslání zpětné vazby (uživatel se splete a místo toho, že je s doporučením spokojený, zašle negativní hodnocení). Dle předpokladů by tyto situace neměly mít na chování systému z dlouhodobého hlediska fatální následky.

3.4.1 Pozorované vlastnosti při různě volených parametrech

3.5 Ignorace rad adaptibilního systému

Experiment má za cíl zjistit, jakým způsobem se bude vyvíjet vnitřní stav systému, pokud bude uživatel zcela ignorovat rad adaptibilního systému. Čas od času tak může vznést žádost o radu, podle které se ale stejně nakonec řídit nebude.

3.5.1 Pozorované vlastnosti při různě volených parametrech

Kapitola 4

Zhodnocení aplikace

Závěrečnou kapitolu věnuji zhodnocení přínosu aplikace. Také zde popisuji další možné kroky ve vývoji práce.

TODO

4.1 Budoucí práce

TODO trochu učesat, přeformulovat...

I přesto, že se podařilo splnit většinu požadavků kladených na systém a bylo vyvinuto funkční řešení, do budoucna je stále prostor pro zlepšování, pořádné zátěžové testování a vývoj nových funkcionalit.

Jako jednou z možných budoucích funkcionalit by mohl být vývoj sofistikovanější strategie volby v případě, že více algoritmů vybraných z jednotlivých kolekcí má stejnou četnost.

Vzhledem k možnostem, jaké nabízejí uživatelská hodnocení u článků, by stálo za to popřemýšlet o dalších doporučovacím algoritmu, který by těchto informací dokázal využít (v této práci s hodnoceními pracoval pouze jeden algoritmus, a to pro výběr nejlépe hodnocených článků). Nabízí se varianta kolaborativního filtrování ve struktuře user-rating-item.

Z časových důvodů nebyl vyvinut ani žádný hybridní doporučovací algoritmus, i když jsem jej měl původně v plánu.

Závěr

TODO dopsat, přepsat... rozvést

Cílem práce bylo navrhnout systém, který by dokázal automaticky volit a dynamicky kombinovat algoritmy a optimalizovat tak jejich cenovou kvalitu. Preference při kombinování měly být ovlivňovány formou pozitivní a negativní zpětné vazby zasílané na systém. Pro účely kombinování bylo nutné vybrat a implementovat sadu základních doporučovacích algoritmů.

Pro účely komunikace se systémem a systémových komponent mezi sebou mělo být navrženo obecné rozhraní. Následně mělo být navrženo a implementováno rozhraní systému v podobě služeb REST.

Funkčnost vyvinutého řešení měla být ověřena experimenty na modelové úloze.

...výhledové použití v systému `http://inbeat.eu/` (Interest Beat is a service for recommendation of content).

Literatura

[1] BellKor, AT&T Labs, Inc. – Research. [online], stav ze dne 21.4.2012. Dostupné z: http://www2.research.att.com/~volinsky/netflix/

[2] easyrec: Recommendation Engine. [online], stav ze dne 24.4.2012. Dostupné z: http://easyrec.org/recommendation-engine

[3] How does the Amazon Recommendation feature work? [online], stav ze dne 21.4.2012. Dostupné z: http://stackoverflow.com/questions/2323768/how-does-the-amazon-recommendation-feature-work

[4] Kvantily. [online], stav ze dne 26.4.2012. Dostupné z: http://www-troja.fjfi.cvut.cz/~limpouch/sigdat/pravdh/node8.html

[5] Mendeley: Recommendation Systems for Academic Literature. [online], stav ze dne 21.4.2012. Dostupné z: http://www.slideshare.net/KrisJack/mendeley-recommendation-systems-for-academic-literature

[6] Netflix Contest: 1 Million Dollars for Better Recommendations. [online], stav ze dne 21.4.2012. Dostupné z: http://www.uie.com/brainsparks/2006/10/02/netflix-contest-1-million-dollars-for-better-recommendations/

[7] Netflix offers streaming movies to subscribers. [online], stav ze dne 21.4.2012. Dostupné z: http://arstechnica.com/uncategorized/2007/01/8627/

[8] Náhodný výběr. [online], stav ze dne 26.4.2012. Dostupné z: ftp://math.feld.cvut.cz/pub/prucha/ubmi/predn/u12.pdf

[9] Proklik. [online], stav ze dne 21.4.2012. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/proklik/

[10] Recommender systems, Part 2: Introducing open source engines. [online], stav ze dne 30.4.2012. Dostupné z: http://www.ibm.com/developerworks/library/os-recommender2/index.html

[11] SorlJ. [online], stav ze dne 29.4.2012. Dostupné z: https://wiki.apache.org/solr/Solrj

[12] What Is Affinity Analysis? [online], stav ze dne 21.4.2012. Dostupné z: http://www.wisegeek.com/what-is-affinity-analysis.htm

[13] ØMQ - The Guide. [online], stav ze dne 26.4.2012. Dostupné z: http://zguide.zeromq.org/page:all

[14] Almazro, D.; Shahatah, G.; Albdulkarim, L.; aj.: A Survey Paper on Recommender Systems. [online], stav ze dne 26.4.2012. Dostupné z: http://arxiv.org/pdf/1006.5278v4.pdf

[15] Anderson, C.: The Long Tail. [online], říjen 2004. Dostupné z: http://archive.wired.com/wired/archive/12.10/tail.html

[16] Anderson, C.: The 80/20 Rule Revisited. 2005, [Online; stav z 30. dubna 2014]. Dostupné z: http://www.longtail.com/the_long_tail/2005/08/the_8020_rule_r.html

[17] Blažek, R. B.; Kotecký, R.; Hrabáková, J.; aj.: BI-PST – Pravděpodobnost a statistika, přednáška 3. [online], stav ze dne 26.4.2012. Dostupné z: https://edux.fit.cvut.cz/courses/BI-PST/_media/lectures/3_handout_pst-v2.pdf

[18] Brodt, T.: Open Recommendation Platform. 2013, [Online; stav z 23. dubna 2014]. Dostupné z: http://www.slideshare.net/d0nut/open-recommendation-platform

[19] DataStax: Architecture in brief | DataStax Cassandra 2.0 Documentation. [online], stav ze dne 27.4.2012. Dostupné z: http://www.datastax.com/documentation/cassandra/2.0/cassandra/architecture/architectureIntro_c.html

[20] Davidson-Pilon, C.: The Multi-Armed Bandit Problem. [online], stav ze dne 27.4.2012. Dostupné z: http://camdp.com/blogs/multi-armed-bandits

[21] Dennis, J.: ZeroMQ: Super Sockets. [online], stav ze dne 27.4.2012. Dostupné z: http://www.slideshare.net/j2d2/zeromq-super-sockets-by-j2-labs

[22] Drachsler, H.: Recommender Systems and Learning Analytics in TEL. University Lecture, 2014, stav ze dne 24.4.2012. Dostupné z: http://www.slideshare.net/Drachsler/rec-sys-mupplelecturekmi

[23] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Dizertační práce, 2000, aAI9980887.

[24] Hlaváč, V.: Učení bez učitele. University Lecture, 2014, stav ze dne 24.4.2012. Dostupné z: http://cmp.felk.cvut.cz/~hlavac/Public/TeachingLectures/UceniBezUcitele.pdf

[25] Jacobi, J.; Benson, E.; Linden, G.: Personalized recommendations of items represented within a database. Září 26 2006, uS Patent 7,113,917. Dostupné z: http://www.google.com/patents/US7113917

[26] Jakob, M.: Reinforcement Learning. University Lecture, 2010, stav ze dne 24.4.2012. Dostupné z: https://cw.felk.cvut.cz/wiki/_media/courses/a3m33ui/prednasky/files/ui-2010-p11-reinforcement_learning.pdf

[27] John Gantz, D. R.: Extracting Value from Chaos. [online], červen 2011. Dostupné z: http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf

[28] Linden, G.; Jacobi, J.; Benson, E.: Collaborative recommendations using item-to-item similarity mappings. Červenec 24 2001, uS Patent 6,266,649. Dostupné z: https://www.google.com/patents/US6266649

[29] Liu, J.; Pedersen, E.; Dolan, P.: Personalized News Recommendation Based on Click Behavior. In *2010 International Conference on Intelligent User Interfaces*, 2010.

[30] Musto, C.: Apache Mahout – Tutorial (2014). [online], stav ze dne 27.4.2012. Dostupné z: http://www.slideshare.net/Cataldo/apache-mahout-tutorial-recommendation-20132014

[31] Prize, N.: The Netflix Prize Rules. [online], stav ze dne 28.4.2012. Dostupné z: http://www.netflixprize.com/rules

[32] Vitvar, T.: Lecture 5: Application Server Services. University Lecture, 2014, stav ze dne 24.4.2012. Dostupné z: http://humla.vitvar.com/slides/mdw/lecture5-1p.pdf

[33] Vychodil, V.: Komunikace pomocí Socketu. [online], stav ze dne 27.4.2012. Dostupné z: http://vychodil.inf.upol.cz/publications/white-papers/socket-referat.pdf

[34] Vychodil, V.: Pravděpodobnost a statistika: Normální rozdělení a centrální limitní věta. [online], stav ze dne 26.4.2012. Dostupné z: http://vychodil.inf.upol.cz/kmi/pras/pr09.pdf

[35] Wiki, M. J.: Connection Leak. [online], stav ze dne 29.4.2012. Dostupné z: http://wiki.metawerx.net/wiki/ConnectionLeak

[36] Wikipedia: Inverted index — Wikipedia, The Free Encyclopedia. 2014, [Online; stav z 27. dubna 2014]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Inverted_index&oldid=591814302

[37] aihorizon: Your Online Artificial Intelligence Resource: Machine Learning, Part I: Supervised and Unsupervised Learning. [online], stav ze dne 24.4.2012. Dostupné z: http://www.aihorizon.com/essays/generalai/supervised_unsupervised_machine_learning.htm

[38] Černý, D.: Bayesovská statistika: klíč k porozumění vesmíru? [online], stav ze dne 26.4.2012. Dostupné z: http://gchd.cz/fygyz/2012_2013/david_cerny-bayesovska_statistika.pdf

Příloha **A**

Dokumentace RESTful API

Pozn. Pro výměnu dat je využíván formát JSON. Návratové kódy mají standardní sémantický význam, tedy **200** (OK), **201** (CREATED), **400** (BAD REQUEST), **404** (NOT FOUND), **500** (SERVER INTERNAL ERROR).

A.0.1 Algorithm

A.0.1.1 Zdroje

/algorithm Kontejner podporovaných algoritmů pro doporučení na serveru.

/algorithm/{algorithm-id}/?coreId={core-id} Doporučení zvoleným algoritmem (filtrace dle jádra v Apache Solr).

A.0.1.2 Operace

/algorithm Vrací seznam podporovaných algoritmů, kterými si lze nechat doporučit články.

Metoda:	GET
Návratové kódy:	200

/algorithm/{algorithm-id}/?coreId={core-id} Vrací seznam doporučených článků daným algoritmem. Pro zadávání vstupů určených konkrétním doporučovacím algoritmům lze ještě využít parametry dotazu: *documentId*, *groupId*, *userId*, *limit*.

Metoda:	GET
Návratové kódy:	200, 400, 404

A.0.2 Cores

A.0.2.1 Zdroje

/cores Kontejner podporovaných jader Apache Solr.

/cores/{core-id}/documents Vkládání dokumentů do zadaného indexu.

A.0.2.2 Operace

/cores Vrací seznam podporovaných jader, se kterými lze komunikovat.

Metoda:	GET
Návratové kódy:	200

/cores/{core-id}/documents Uloží do zadaného jádra nové články.

Metoda:	POST
Návratové kódy:	201, 400, 404, 500

/cores/{core-id}/documents Smaže z daného jádra dokument specifikovaný v parametru dotazu: *documentId* (ID dokumentu může být obecně cokoliv, i URL, proto zasíláno v těle)

Metoda:	DELETE
Návratové kódy:	200, 400, 404, 500

/cores/{core-id}/documents Slouží k přidání informace o uživatelském hodnocení dokumentu (článku v indexu).

Metoda:	PUT
Návratové kódy:	200, 400, 404, 500

A.0.3 Collection

A.0.3.1 Zdroje

/collection Kontejner kontextových kolekcí vytvořených v Recommeng systému.

/collection/{collection-id} Algoritmy v kolekci.

/collection/{collection-id}/?filter=best Filtrování jednoho v danou chvíli toho nejlepšího algoritmu v kolekci.

A.0.3.2 Operace

/collection Slouží k vypsání existujících kolekcí v Recommeng systému.

Metoda:	GET
Návratové kódy:	200

/collection Slouží k vytvoření kontextové kolekce v systému.

Metoda:	POST
Návratové kódy:	201, 400

/collection/{collection-id} Slouží k vypsání algoritmů přítomných v kolekci sestupně od toho nejvhodnějšího pro doporučení.

Metoda:	GET
Návratové kódy:	200, 404

/collection/{collection-id}?filter=best Slouží k vypsání nejlepšího algoritmu pro doporučení z kontextové kolekce.

Metoda:	GET
Návratové kódy:	200, 404

/collection/{collection-id} Slouží k zaslání zpětné vazby do systému.

Metoda:	PUT
Návratové kódy:	200, 404, 400

A.0.4 Supercollection

A.0.4.1 Zdroje

/supercollection Kontejner super kolekcí vytvořených v Recommeng systému.

/supercollection/{supercollection-id} Všechny kontextové kolekce, pomocí kterých byla vytvořena super kolekce.

/supercollection/{supercollection-id}/?filter=best Filtrování jednoho v danou chvíli toho nejlepšího algoritmu kombinovaného ze všech kontextových kolekcí dané superkolekce.

A.0.4.2 Operace

/supercollection Slouží k vypsání existujících super kolekcí v Recommeng systému.

Metoda:	GET
Návratové kódy:	200

/supercollection Slouží k vytvoření super kolekce v systému.

Metoda:	POST
Návratové kódy:	201, 400

/supercollection/{supercollection-id}?filter=best Slouží k vypsání nejlepšího algoritmu pro doporučení z kombinace kontextových kolekcí.

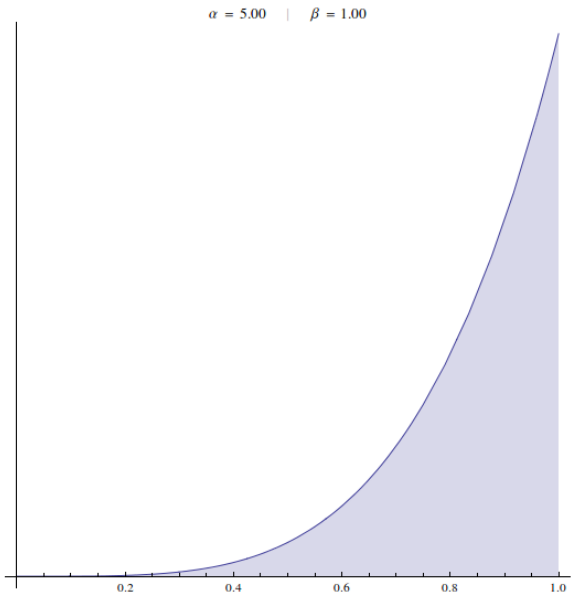
Metoda:	GET
Návratové kódy:	200, 404

/supercollection/{supercollection-id} Slouží k zaslání zpětné vazby do systému.

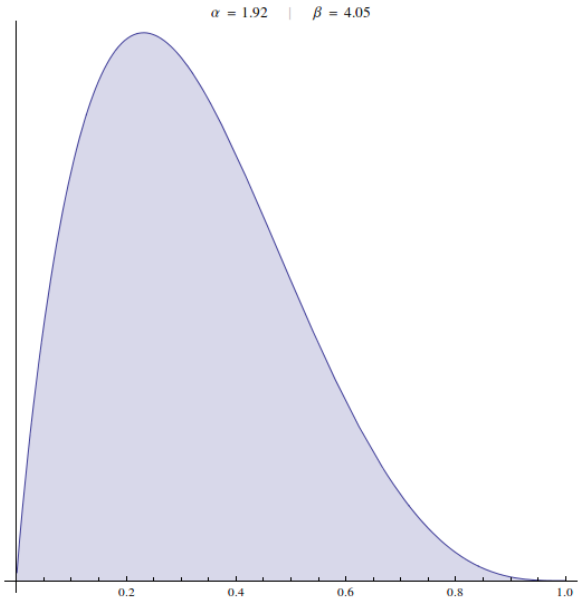
Metoda:	PUT
Návratové kódy:	200, 404, 400

Příloha B

Ukázky posterior rozdělení Beta s různými vstupními parametry



Obrázek B.1: Hustota pravděpodobnosti *Beta* rozdělení pro vizualizaci 4 úspěchů ze 4 pokusů.



Obrázek B.2: Hustota pravděpodobnosti *Beta* rozdělení pro vizualizaci 1 úspěchu ze 4 pokusů.

Příloha C

Seznam použitých zkratek

IDC
CTO
MIT
ACM Association for Computing Machinery
ICWSM
ICML
IBM
REST
API
TCP
DBMS
NoSQL
MQ
JVM
JSON
URL
ORP
HTTP

Příloha D

Obsah přiloženého CD

```
readme.txt ..... stručný popis obsahu CD
exe ..... adresář se spustitelnou formou implementace
src
    impl ..... zdrojové kódy implementace
    thesis ..... zdrojová forma práce ve formátu LaTeX
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
    thesis.ps ..... text práce ve formátu PS
```