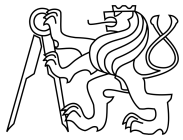
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Analýza chování modelů strojového učení
Student:	Bc. Jan Veselý
Vedoucí:	Ing. Jan Černý
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Nastudujte postupy a metody pro analýzu chování modelů strojového učení (dále jen modelů). Navrhněte metriky pro měření:
1. přesnosti výstupu modelů,
2. rozdílnosti výstupu modelů při různých konfiguracích.
Naimplementujte systém pro vyhodnocování těchto metrik na běžících modelech nebo historických datech v programovacím jazyce Java.
Otestujte a vyhodnoťte přínos jednotlivých metrik. Na základě výsledků práce popište možné využití těchto metrik, jako jsou například kontrola funkčnosti modelu při nasazování nové verze nebo automatická optimalizace parametrů modelu.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

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

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 13. října 2016

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

Diplomová práce

Analýza chování modelů strojového učení

Bc. Jan Veselý

Vedoucí práce: Ing. Jan Černý

8. května 2017

Poděkování

Rád bych poděkoval vedoucímu práce, Ing. Janu Černému, za jeho ochotu a pomoc při tvorbě této práce. Dále bych chtěl poděkovat své přítelkyni Martině nejen za pomoc při psaní této diplomové práce, ale i za podporu ve všech oblastech mého života. Za neutuchající podporu patří poděkování i mé rodině.

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 2017 .....

Odkaz na tuto práci

Veselý, Jan. *Analýza chování modelů strojového učení.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Předmětem této práce je návrh a implementace systému pro analýzu chování modelů strojového učení. V textu je popsána architektura systému splňující možnost škálování na neomezený počet zákazníků a napojení na libovolný model strojového učení. Další část je zaměřena na návrh a otestování metrik pro doporučovací systém. Jsou probrány základní metriky pro měření přesnosti výstupu doporučovacího modelu a pro měření změn výstupu při různých konfiguracích. Výsledky jsou ověřeny na datech reálných zákazníků.

Klíčová slova strojové učení, model, data, analýza chování, doporučovací systém, metrika

Abstract

The goal of this thesis is to design and implement a system for analyzing the behaviour of machine learning models. We present a system architecture that is scalable to an unlimited number of customers and can connect to any type of machine learning models. The system is then tested on metrics for a recommender system. We propose basic metrics for measuring precision of the recommendation model and for measuring output differences in distinct configurations. The results of the metrics are validated on real customer data.

Keywords machine learning, model, data, behaviour analysis, recommender system, metric

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Strojové učení je v dnešní době přítomno ve všech oblastech našeho života. Díky němu dnes dostáváme kvalitní personalizovaná doporučení při nákupu v internetových obchodech, auta řídí bez našeho přičinění a algoritmy překládají text z původního jazyka do jiného. To je jen pár z mnoha působivých a užitečných aplikací strojového učení.

Modely strojového učení jsou však ve většině případů hodně komplexní systémy, u kterých se jen těžko odhaduje, jak se zachovají. Je tedy potřeba je měřit a hlídat kvalitu jejich výstupu. A k tomu slouží systém pro analýzu chování modelů strojového učení vytvořený v této práci.

Jedním z typů modelů strojového učení jsou doporučovací systémy. Dnes se na internetu nachází nepřeberné množství internetových obchodů a každý z nich nabízí mnoho produktů. Není v silách běžného uživatele, aby se v tomto množství informací dokázal vyznat. S tím mu pomohou doporučovací systémy: analyzují produkty a chování všech uživatelů, a následně dokáží uživateli doporučit na míru produkty, o které by mohl mít zájem.

Kvalitu doporučování je však také potřeba hlídat. K tomu se dá využít systém pro analýzu chování modelů strojového učení. Naiplementujeme do něj napojení na doporučovací model, a metriky, které budou charakterizovat relevanci vrácených doporučení.

Cíle práce

Hlavním cílem této práce je navrhnout a naimplementovat funkční systém pro analýzu chování modelů strojového učení. Je potřeba, aby systém bylo možné škálovat na libovolný počet zákazníků a aby byl systém schopný se připojit na jakýkoliv model strojového učení. Od systému je také očekávána spolehlivost a jednoduché používání.

Druhým cílem práce je integrovat do systému napojení na doporučovací systém. Dále je potřeba navrhnout metriky, které budou měřit přesnost vý-

stupu doporučovacího modelu a rozdílnost výstupů při různých konfiguracích. Každou metriku je potřeba proměřit na datech reálných zákazníků a diskutovat možnost jejího přínosu při vylepšování a údržbě doporučovacího systému.

Struktura práce

V kapitole 1 jsou vysvětleny základní pojmy týkající se strojového učení. Dále se věnujeme vysvětlení doporučovacího systému a jeho základním typům.

Následující kapitola se věnuje návrhu, implementaci a využití systému pro analýzu chování modelů strojového učení. Jedna z částí kapitoly diskutuje již existující řešení.

Poslední kapitola se zaobírá doporučovacím systémem. Je popsán konkrétní doporučovací systém, na který se systém napojuje, a specifická rozšíření napsaná na míru pro doporučovácí model. Následně jsou navrženy metriky pro měření přesnosti výstupu doporučovacího modelu a metriky pro měření rozdílnosti výstupů při různých konfiguracích. Výsledky metrik jsou testovány na reálných datech zákazníků doporučovacího systému a je diskutován jejich přínos.

Kapitola 1

Teorie

V této kapitole jsou definovány základní pojmy používané v rámci této práce. Je zde vysvětleno, co je to strojové učení, a s jakými typy úloh se v této podoblasti umělé inteligence můžeme setkat dle [1] [2] [3] [4]. Dále je popsáno, co je to doporučovací systém a základní přístupy k doporučování obsahu dle [5] [6] [7] [8] [9].

1.1 Strojové učení

Klasické počítačové algoritmy fungují na základě deterministických rozhodnutí. Tím je myšleno, že pro stejný vstup vrátí algoritmus vždy stejný odpovídající výstup. Toto chování je však ne vždy žádoucí. Například pokud jsme majitelem internetového obchodu a chceme našim zákazníkům doporučit při nákupu další zboží, které by mohli také chtít. Každý jsme ale trochu jiný. Každý máme jiné preference, a proto bychom měli dostat doporučení ušitá na míru přímo nám. Klasické algoritmy selhávají, neboť nedokáží brát v potaz uživatelovy preference, jeho nákupní historii a další data specifická pro každého z nás. A právě v tomto případě je vhodné použít některý z algoritmů strojového učení.

Principem strojového učení je možnost upravit nebo adaptovat akce vykonávané algoritmem tak, aby byly přesnější. Přesností je v tomto případě myšleno, jak moc vykonané akce odpovídají těm, které by měly být provedeny v ideálním případě. Algoritmus je schopen se učit, a tím se stává v řešení určitého problému lepším a lepším.

Princip strojového učení je hezky vidět třeba na hraní hry *Scrabble*. Představte si, že hrajete proti počítači, který ji nikdy dříve nehrál. Pokud byste hráli proti klasickému algoritmu, vždy při začátku hry by začínal takzvaně od nuly, to znamená, že by neměl předchozí znalost hry. Nejspíše byste ho pokaždé porazili. Pokud byste hráli proti programu, který by byl založen na strojovém učení, ze začátku by to vypadalo stejně – nejspíše byste ho poráželi. S každou další hrou by se tento algoritmus však zlepšoval. Zjišťoval by, jaká taktika

na vás platí, a postupně by vás začal porážet. Po čase by se algoritmus zlepšil natolik, že už byste nebyli schopni ho porazit.

Aby bylo možné počítač přimět se učit, spojuje strojové učení řadu nápadů a konceptů z neurovědy, biologie, matematiky nebo fyziky. Například umělé neuronové sítě, jeden z často používaných algoritmů strojového učení, se inspiroval tím, jak funguje lidský mozek. Umělá neuronová síť se skládá z neuronů, které jsou vzájemně propojeny a předávají si signály transformované pomocí určitých přenosových funkcí.

1.1.1 Typy

Algoritmy strojového učení mohou být rozděleny do čtyř základních skupin podle toho, zdali ví, jestli se zlepšují nebo ne, a jak se zlepšit:

- učení s učitelem, tzv. *supervised learning*
- učení bez učitele, tzv. *unsupervised learning*
- zpětnovazebné učení, tzv. *reinforcement learning*
- evoluční algoritmy, tzv. *evolutionary learning*

1.1.1.1 Učení s učitelem

Algoritmu je poskytnuta trénovací množina, kde každý vzor má přiřazenou správnou odpověď. Algoritmus se na základě této množiny zkusí naučit odpovědět správně na libovolný možný vstup.

Do této skupiny úloh patří například klasifikace. Vstupní vzory jsou v tomto případě rozděleny do N tříd, a úkolem algoritmu je přiřadit každému neznámému vstupnímu vzoru třídu, do které patří. Jedním z nejznámějších klasifikačních problémů je hledání nevyžádaných e-mailových zpráv v příchozí poště.

Dalším zástupcem učení s učitelem je regrese. Ta, na rozdíl od klasifikace, přiřazuje vstupním vzorům spojitou hodnotu. Algoritmus se snaží odhadnout závislost výstupní veličiny na jedné nebo více vstupních veličinách. Jedna z úloh, kterou lze řešit regresí, je například odhad teploty na základě ostatních známých parametrů o počasí v daném dni.

1.1.1.2 Učení bez učitele

V tomto případě nejsou algoritmu poskytnuty žádné správné výstupy. Algoritmus se tedy v datech pokusí najít společné vzory a přiřadit k sobě vstupy na základě podobnosti.

Zástupcem úloh učení bez učitele je shluková analýza. Jejím úkolem je rozdělit vstupní vzory do skupin. Na rozdíl od klasifikace nejsou skupiny dopředu známé. Algoritmus se snaží vytvořit takové přiřazení, aby si vzory jedné

skupiny byly více podobnější s ostatními ze stejné skupiny než se vzory z jiných skupin. Mezi úlohy shlukové analýzy patří například rozdělení lidí podle odpovědí na otázky v průzkumu.

1.1.1.3 Zpětnovazebné učení

Nachází se na pomezí mezi učením s učitelem a bez učitele. Algoritmus dostane na výstupu zpětnou vazbu, zdali je odpověď správná nebo ne, ale v případě chyby se správnou odpověď nedozví. Musí tedy prozkoumat různé možnosti a sám na ni postupně přijít.

1.1.1.4 Evoluční algoritmy

Skupina algoritmů založená na principech přírody: organismus se v průběhu času zlepšuje a přizpůsobuje svému prostředí tak, aby zvýšil své šance na přežití a možnost mít potomky. Algoritmy využívají myšlenky `fitness`, což je skóre vyjadřující kvalitu daného řešení.

1.2 Doporučovací systém

Doporučovací systémy jsou souborem nástrojů a technik poskytující návrhy produktů, které by mohly zajímat uživatele. Patří do skupiny systémů filtrujících informace. Odhalují kvalitu a relevanci jednotlivých produktů vzhledem k preferencím konkrétního uživatele, a snaží se mu doporučit jen ty produkty, které ho zajímají nejvíce.

Doporučovací systémy změnily způsob, jakým webové stránky a počítačové aplikace nabízejí uživatelům produkty. Algoritmy, probírající se velkou hromadou dat za účelem objevování preferencí jednotlivých uživatelů, změnily statické zobrazení na dynamické. Dříve uživatelé museli požadované produkty vyhledávat, nyní jim jsou zobrazovány na míru dle preferencí.

Ke vzniku doporučovacích systémů vedlo jednoduché pozorování: většina z nás se při denních rutinních rozhodnutích řídí doporučením ostatních.

Doporučovací systémy lze využít v mnoha oblastech, například zobrazování produktů v internetových obchodech, řazení výsledků ve vyhledávačích, doporučování dalších článků k přečtení, či vybírání příspěvků na sociálních sítích.

Klasické doporučovací systémy zaznamenávají data o třech základních entitách. Tou první jsou produkty – objekty doporučované uživatelům. Mohou být charakterizovány atributy, jako je například název, cena, kategorie apod. Při doporučování je produktům přiřazeno určité skóre. Například kladné skóre může znamenat, že by se produkt mohl uživateli líbit, kdežto záporné může říkat, že tento produkt není pro uživatele vhodný. Doporučovací systém pak doporučí uživateli produkty s nejvyšším skóre.

Druhou entitou jsou uživatelé. Doporučovací systém zpravidla o uživatelích ukládá a používá různé informace. Některé metody doporučování mohou pracovat s atributy uživatelů, jako například jméno, věk, pohlaví nebo bydliště. Stejně tak může systém využívat historii chování uživatelů, tím je myšleno s jakými produkty uživatel interagoval v minulosti.

Poslední entitiou jsou transakce – zaznamenané interakce mezi určitým uživatelem a produktem. Typy používaných interakcí se vztahují k doméně zákazníka doporučovacího systému. Například internetový obchod pracuje s interakcemi zobrazení produktu, přidání produktu do košíku nebo nákupu, kdežto hudební aplikace ukládá interakce přehrání, přeskočení písně či hodnocení. U interakce mohou být uloženy i další atributy, například u hodnocení písně bude zaznamenána jeho výška, k interakci nákupu lze ukládat počet kusů koupeného produktu apod.

Dobře naprogramovaný doporučovací systém může být velmi přínosný. Nejlépe funguje pro zvýšení obratu, prodej rozmanitějších produktů, zlepšení zákaznické spokojenosti, zvýšení věrnosti zákazníků nebo přesnější zjištění jejich preferencí.

Podle způsobu doporučování existují tři základní typy doporučovacích systémů:

- kolaborativní filtrování, tzv. *collaborative filtering*
- filtrování založené na obsahu, tzv. *content-based filtering*
- hybridní přístup, tzv. *hybrids*

1.2.1 Kolaborativní filtrování

Algoritmy patřící do skupiny kolaborativního filtrování fungují na základě předchozího chování uživatele. Model bere v potaz nejen historii konkrétního uživatele, kterému je doporučováno, ale i historii ostatních s podobnými preferencemi. Doporučovací systém automaticky udržuje historii chování jednotlivých uživatelů a při doporučování využívá znalosti získáné z chování uživatelů s podobnou historií (kolaborace). Myšlenka, na níž stojí tato technika, je, že pokud Martina má stejný názor na určitou věc jako Jan, Martina bude mít pravděpodobně stejný názor na jinou věc jako Jan než jako jiný náhodně vybraný uživatel.

Předpokládejme, že vytváříme stránku s informacemi o filmech, kde je mohou uživatelé hodnotit podle toho, jak se jim líbily. Tyto uživatele můžeme rozdělit tak, aby v každé skupině byly lidé s podobnými preferencemi (v jedné skupině mohou být například uživatelé, kteří mají rádi akční filmy, v jiné zas ti preferující komedie). V těchto skupinách můžeme identifikovat nejoblíbenější filmy. Když pak generujeme doporučení pro určitého člověka, podíváme se, které oblíbené filmy z dané skupiny ještě neviděl ani nehodnotil, a ty mu doporučíme seřazené například podle oblíbenosti.

Nevýhoda tohoto přístupu je, že musíme znát a udržovat historii chování uživatelů. Z toho vyplývá další problém, tzv. studený start: u uživatele, u něhož nemáme žádnou nebo jen krátkou historii, nelze efektivně použít tento postup. Tato metoda je tedy učinnější s rostoucí historií uživatele.

Tato technika je při implementaci doporučovacích systémů momentálně používaná nejčastěji.

1.2.2 Filtrování založené na obsahu

Algoritmy patřící do skupiny filtrování založeném na obsahu berou, stejně jako předchozí skupina, v potaz uživatelovu historii, avšak na rozdíl od nich ji nesrovnávají s historií ostatních, ale doporučují na základě podobnosti produktů. Výstupem filtrovacího algoritmu tak bývají produkty, které jsou kategoricky či atributově podobné s těmi, se kterými v minulosti uživatel interagoval.

Základní myšlenkou tohoto přístupu je, že pokud se uživateli líbí produkty z kategorie A, je dost pravděpodobné, že se mu budou líbit i jiné produkty ze stejné kategorie, které ještě neviděl. Například pokud uživatel na zpravodajském webu čte často články o hokeji a doporučíme mu ty, které ještě z této kategorie nečetl, je dost pravděpodobné, že se mu budou taktéž líbit.

Nevýhodou této metody je, že doporučení postrádají rozmanitost.

1.2.3 Hybridní přístup

Jednou z možností, jak zvýšit efektivitu doporučovacího systému, je tzv. hybridní přístup, kdy jsou zkombinovány výsledky metody kolaborativního filtrování a filtrování založeném na obsahu. Každé z metod přiřadíme určitou váhu a doporučíme ty produkty, které mají nejvyšší součet výsledných skóre.

Tento přístup lze využít i k odstranění problému studeného startu u kolaborativního filtrování. Pokud ještě nemáme o uživateli žádné informace, doporučíme mu především podle výsledků filtrování založeném na obsahu. Čím bohatší historii o chování uživatele získáváme, tím zvyšujeme důležitost výsledků kolaborativního filtrování oproti druhé metodě.

Kapitola 2

Systém pro analýzu chování modelů strojového účení

V druhé kapitole je popsána základní myšlenka, proč vznikl systém pro analýzu chování modelů strojového učení, a požadavky na něj. Dále se budeme věnovat realizaci navrženého systému a nakonec jeho využití.

2.1 Návrh

Hlavním úkolem této práce je navrhnout systém, který se dokáže připojit na jakýkoliv model strojového učení, a vypočítat na základě jeho výstupu určenou metriku. Je potřeba, aby byl spolehlivý, intuitivně použitelný a jednoduše škálovatelný na mnoho zákazníků. Požadavky na funkce, jaké má systém mít a jak se má chovat, jsou sepsány do funkčních[1] a nefunkčních[2] požadavků.

2.1.1 Funkční požadavky

1. Možnost napojení systému na libovolný model strojového učení
2. Možnost přidání libovolné metriky do systému
3. Flexibilní formát výstupu metriky
4. Vytvoření testu s určenými metrikami
5. Získání výsledků jakéhokoliv testu vytvořeného dříve
6. Vytvoření testu, vypočtení a vrácení výsledků v jednom požadavku
7. Zkopírování testu

[1] Funkční požadavky určují, co by systém měl umět a dělat [10]
[2] Nefunkční požadavky určují nebo omezují, jak by se měl systém chovat [10]

8. Opakovaný výpočet testu
9. Monitoring čekajících, probíhajících a skončených testů
10. Vytvoření, úprava a mazání předpřipravených napojení na model
11. Získání seznamu všech uložených napojení na model

2.1.2 Nefunkční požadavky

1. Zopakování testu, pokud během výpočtu dojde k chybě
2. Škálovatelnost na libovolný počet zákazníků a velikost provozu
3. Fungování aplikace v cloudovém prostředí
4. Dostupnost přes REST API

2.1.3 Upřesnění požadavků

V této části jsou podrobněji popsány jednotlivé požadavky na systém.

2.1.3.1 Možnost napojení systému na libovolný model strojového učení

Systém musí být od počátku navržen tak, aby byl schopný vypočítat libovolnou metriku z výsledků jakéhokoliv modelu. Nezáleží tedy na typu modelu (doporučování, predikce, ...), v každém případě musí být možné se na něj připojit, získat jeho výstup a zpracovat ho v rámci metriky.

2.1.3.2 Možnost přidání libovolné metriky do systému

Systém nesmí mít žádná omezení, jaké metriky lze vypočítat nebo co by daná metrika měla dělat. Musí umožňovat integrovat jakoukoliv metriku, ať už bude provádět prostý výpočet, nebo se bude napojovat i na externí služby.

2.1.3.3 Flexibilní formát výstupu metriky

Systém nesmí nijak omezovat formát výstupu metriky, například by neměl požadovat, aby každá metrika vrátila výslednou hodnotu výpočtu v atributu `result`. Metrika má sama určovat, jaké hodnoty a v jakém formátu vrátí jako výsledek.

Metrika také může vrátit podrobnější informace o výpočtu, což může být v určitých případech hodně dat, a tak systém nesmí omezovat ani velikost výstupu.

2.1.3.4 Vytvoření testu s určenými metrikami

Systém musí umožňovat sdružení výpočtu více metrik do jednoho testu a vrátit výsledky najednou. Každý test musí mít svůj unikátní identifikátor, aby se uživatel, který test spustí, mohl později dotázat na výsledky.

2.1.3.5 Získání výsledků jakéhokoliv testu vytvořeného dříve

Jak již bylo zmíněno v předchozím funkčním požadavku, systém musí umožňovat získání výsledků vypočítaného testu, a to i zpětně z historie. To znamená, že si systém musí udržovat výsledky uložené někde v databázi nebo souboru.

2.1.3.6 Vytvoření testu, vypočtení a vrácení výsledků v jednom požadavku

Pro napojení jiných programů musí systém poskytovat metodu, která vytvoří test s předanou konfigurací, spustí ho, a počká na výsledek. Může nastat situace, kdy externí program nebude moci volat více metod, a tak je potřeba vše vyřešit najednou. Je to vlastně blokující varianta předchozích dvou funkčních požadavků.

2.1.3.7 Zkopírování testu

Systém musí poskytovat metodu ke zkopírování testu. Měla by fungovat následovně: program vezme zdrojový test, vytvoří nový a spustí ho se stejným nastavením jako měl ten původní.

Tuto metodu použijeme v případě, kdy je potřeba spustit test znovu se stejným nastavením a porovnat jeho výsledky s předchozím testem.

2.1.3.8 Opakovaný výpočet testu

Systém musí poskytovat možnost zopakování výpočtu testu. Tento funkční požadavek je vhodný především k opakovanému spuštění testu, který nedoběhl v pořádku a z nějakého důvodu se sám znovu nespustil.

2.1.3.9 Monitoring čekajících, probíhajících a skončených testů

Systém musí monitorovat, které testy čekají na zpracování, které jsou zrovna zpracovávány a které již byly dokončeny. Součástí systému by měl být dashboard, kde bude vidět, co se v systému děje, a ze kterého půjde zjistit, že něco nefunguje správně.

2.1.3.10 Vytvoření, úprava a mazání předpřipravených napojení na model

Systém by měl umožňovat vytvářet, upravovat nebo mazat běžně používaná napojení na modely strojového učení. K již existujícím se bude přistupovat přes jejich unikátní identifikátor.

2.1.3.11 Získání seznamu všech uložených napojení na model

Systém musí poskytovat metodu, která vrátí všechna uložená napojení na modely strojového učení včetně jejich unikátních identifikátorů a konfigurací.

2.1.3.12 Zopakování testu, pokud během výpočtu dojde k chybě

První z nefunkčních požadavků na systém se zaměřuje na jeho spolehlivost. Systém monitoruje průběh výpočtu metriky a testu. Pokud z nějakého důvodu není výpočet dokončen v pořádku a skončí s neznámou chybou, systém by ho měl po nějaké době spustit znovu. Jinak řečeno, musí být garantováno, že jednou vytvořený test bude za každých podmínek dokončen.

2.1.3.13 Škálovatelnost na libovolný počet zákazníků a velikost provozu

Druhý nefunkční požadavek se věnuje škálovatelnosti systému. Systém by měl být schopný zvládat v budoucnu mnoho zákazníků, a řešit více testů najednou. Zátěž se s přibývajícím časem zvyšuje, a proto by mělo být možné jednoduché rozšiřování na více strojů a do více datacenter.

2.1.3.14 Fungování aplikace v cloudovém prostředí

Je nutné, aby systém běžel vzdáleně na serveru v datacentru v cloudovém prostředí. Z předchozího bodu vyplývá požadavek na libovolnou škálovatelnost takového řešení.

2.1.3.15 Dostupnost přes REST API

Z důvodu, že se na systém budou napojovat jiné externí aplikace, je potřeba, aby s ním bylo možné komunikovat pomocí REST API.

2.2 Analýza existujících řešení

Během tvorby práce se nepodařilo nalézt žádnou již existující aplikaci, která by splňovala všechny funkční a nefunkční požadavky na systém pro analýzu chování modelů strojového učení.

Existuje řada článků věnující se měření přesnosti konkrétního typu modelu strojového učení. Například v [11] je představen *REFEREE*, framework

pro testování přesnosti doporučovacích systémů nad databází vědeckých článků *ResearchIndex*, nebo v [12] diskutují generování testovacích dat a postup testování MartiRank algoritmu (algoritmus na hodnocení dat).

Žádný z nalezených článků však nepopisuje systém nebo jeho architekturu, která by byla schopná se napojit na libovolný model strojového učení a která by byla škálovatelná na neomezený počet zákazníků.

2.3 Realizace

V této části kapitoly je popsán systém navržený a zrealizovaný v rámci této diplomové práce na základě uvedených funkčních a nefunkčních požadavků.

2.3.1 Architektura systému

Celý systém je rozdělen do dvou hlavních částí: rozhraní systému (tzv. API, přijímá požadavky na vytvoření testů a vrací výsledky) a výpočetní část (tzv. Evaluator, komunikuje s modely strojového učení a provádí vyhodnocení určených metrik). Části mezi sebou komunikují pomocí datové vrstvy, jejíž součástí je fronta RabbitMQ[3] a relační databáze PostgreSQL[4]. K logování testů a metrik určenému k monitoringu o dění v systému se používá Elasticsearch[5]. Pro vizualizaci uložených logů je na něj napojena Kibana[6]. Architekturu celého systému znázorňuje obrázek 2.1.

Nyní něco k průběhu výpočtu testu. Uživatel, používající systém pro analýzu chování modelů strojového učení, zavolá jeho rozhraní, kterému předá konfiguraci testu ke spuštění. Rozhraní požadavek zpracuje, uloží kompletní informace o testu do relační databáze, založí nový logovací záznam v monitorovací části datové vrstvy a přidá unikátní identifikátor testu do fronty.

Na druhé straně fronty jsou připojeni dostupní Evaluatoři (kvůli jednoduché škálovatelnosti je systém navržen tak, aby jich mohl být libovolný počet). Jeden z nich, který zrovna nic nepočítá, přečte z fronty záznam o nově přidaném testu, stáhne si o něm informace z relační databáze, aktualizuje jeho stav v relační databázi a v logovacím záznamu, a začne postupně vypočítávat jednotlivé metriky. Po dokončení výpočtu každé metriky uloží její výsledky do relační databáze.

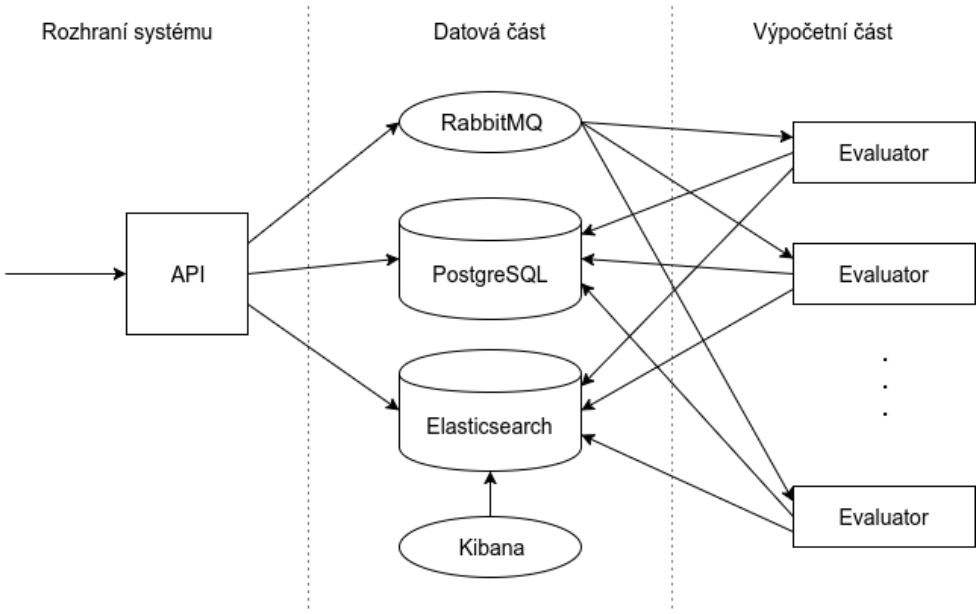
Mezi tím se uživatel, jenž celý test inicioval, dotazuje rozhraní systému, v jakém stavu je výpočet, a zdali už jsou dostupné výsledky. Hned, jak daný Evaluator dokončí výpočet, rozhraní systému je uživateli vrátí.

[3]RabbitMQ je jedna z nejpoužívanějších front na posílání zpráv. Více na `https://www.rabbitmq.com/`.

[4]PostgreSQL je oblíbená relační databáze s volně dostupným zdrojovým kódem. Více na `https://www.postgresql.org/`.

[5]Elasticsearch je distribuovaný vyhledávací a analytický nástroj přístupný přes REST rozhraní. Více na `https://www.elastic.co/products/elasticsearch`.

[6]Kibana je nástroj pro vizualizaci dat uložených v nástroji Elasticsearch. Více na `https://www.elastic.co/products/kibana`.



Obrázek 2.1: Architektura systému pro analýzu chování modelů strojového učení

V dalších kapitolách jsou popsány tři základní koncepty, se kterými se v systému pracuje. Poté následuje podrobnější popis jednotlivých částí systému.

2.3.1.1 Konektor

Konektor je napojení na nějaký model strojového učení. Ke každému typu modelu je dostupný alespoň jeden konektor. Každý typ konektoru může mít libovolně strukturovanou konfiguraci, kterou uživatel nastavuje při vytváření testu. Příklad konfigurace je vidět v ukázce 2.1. Nastavení běžně používaných konektorů lze v systému ukládat a při použití jen uvést unikátní identifikátor uloženého záznamu.

Příkladem konektoru může být napojení na model pro doporučování. Konektor dostane vybranou historii požadavků a nastavení, která má změnit. Konektor postupně projde danou historii požadavků, načte jejich nastavení a změní v nich to, co uživatel uvedl v konfiguraci. Následně požadavky postupně pošle na doporučovací model, získá doporučení a vrátí je metrice, která konektor zavolala.

2.3.1.2 Metrika

Metrika je jednotka, která je vyhodnocována Evaluatory. Každá metrika má název určující způsob vyhodnocení, konfiguraci a výsledky. Součástí konfi-

Zdrojový kód 2.1: Příklad konfigurace konektoru

```
{
  "batchSize": 1,
  "template": {
    "templateId": "homepage",
    "algorithms": {
      "attribute": 0,
      "user": 1.4
    },
    "userWeight": 3.4,
    "itemWeight": 1
  }
}
```

gurace je seznam a nastavení konektorů (atribut `connectors`), které budou během výpočtu volány. Metrika nemusí mít žádný konektor, maximální počet není omezen. Struktura konfigurace může být pro každý typ metriky odlišná, ale atribut s konektory musí být vždy uveden. Příklad konfigurace metriky je vidět v ukázce 2.2.

Stav výpočtu metriky může být popsán následujícími pojmy:

- *waiting* – metrika byla v pořádku přidána do systému, ale ještě není zpracovávána žádným z Evaluatorů
- *processing* – metrika je právě zpracovávána jedním z Evaluatorů
- *done* – metrika byla v pořádku vypočtena a jsou dostupné její výsledky
- *error* – výpočet metriky neproběhl v pořádku, během zpracovávání se vyskytla chyba

2.3.1.3 Test

Test je celek, který k sobě sdružuje více metrik vyhodnocujících související výpočty. Test může mít libovolný počet metrik. Vezme si ho ke zpracování jeden z dostupných Evaluatorů, který postupně provádí výpočet metrik uvedených v konfiguraci testu.

Každý test má svůj stav určený na základě výpočtu metrik, jenž jsou součástí. Stav testu může být popsán následujícími pojmy:

- *waiting* – test byl v pořádku přidán do systému, ale ještě si ho nevzal ke zpracování žádný z Evaluatorů
- *processing* – test je právě zpracováván jedním z dostupných Evaluatorů a brzy budou k dispozici výsledky všech metrik

Zdrojový kód 2.2: Příklad konfigurace metriky

```
{
  "requestsCount": 2000,
  "timeframe": 120,
  "customer": "cvut",
  "template": "homepage",
  "connectors": [
    {
      "name": "Recommender.RepeatedRecommendations",
      "configuration": {
        ...
      }
    },
    {
      "name": "Recommender.RepeatedRecommendations",
      "configuration": {
        ...
      }
    }
  ]
  "interactions": [
    "Detail view",
    "Purchase"
  ]
}
```

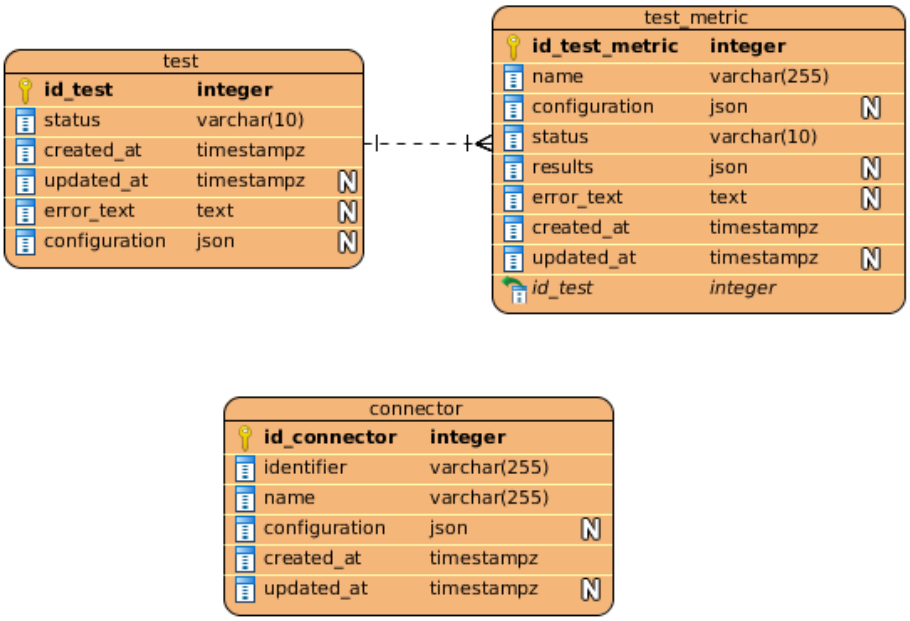
- *done* – test byl zpracován bez chyby a jsou dostupné výsledky všech metrik
- *error* – během zpracovávání testu nebo jedné z metrik se vyskytla chyba

Test může mít také tzv. sdílené konektory. Sdílený konektor je napojení na model strojového učení, které může využívat více metrik. Hodí se to například v situacích, kdy potřebujeme v rámci jednoho testu vypočítat více metrik na výsledcích modelu, jehož učení na testovacích datech trvá dlouhou dobu. Bez sdílených konektorů bychom pro každou metriku museli model naučit znovu, díky nim však model naučíme jednou a pak už jen počítáme metriky na základě jeho výstupu.

2.3.2 Datová část systému

Data, která systém ukládá nebo používá, se dají rozdělit do tří skupin podle účelu:

- relační databáze – slouží k uložení konfigurací a výsledků testů



Obrázek 2.2: Schéma relační databáze systému

- fronta zpráv – slouží ke komunikaci mezi rozhraním systému a Evaluatory
- monitorovací část – slouží k uchovávání logů (co se v systému vypočítávalo)

2.3.2.1 Relační databáze

Účelem relační databáze je ukládání veškerých informací o uložených konektorech a o testech, které probíhají nebo již proběhly (v databázi je uložena kompletní historie testů). To zajišťují tři tabulky, jejichž schéma je na obrázku 2.2. Pro účely systému byla jako relační databázový systém zvolena oblíbená a široce používaná databáze PostgreSQL.

V následujících částech této kapitoly jsou podrobněji popsány tři základní tabulky systému.

Tabulka connector Tabulka `connector` slouží k ukládání informací o předpřipravených konektorech napojujících se na modely strojového učení. Obsahuje tyto sloupce:

- `id_connector` – unikátní číselný identifikátor uloženého konektoru
- `identifier` – unikátní identifikátor, kterým se může uživatel vytvářející test odkazovat na uložený konektor

- `name` – název typu konektoru
- `configuration` – konfigurace konektoru ve formátu JSON
- `created_at` – datum, kdy byl konektor přidán do databáze
- `updated_at` – datum, kdy byl konektor naposledy upraven

Tabulka test Účelem tabulky `test` je ukládat základní informace o testech. Obsahuje tyto sloupce:

- `id_test` – unikátní identifikátor, kterým se uživatel může na test odkazovat
- `status` – stav, ve kterém se aktuálně nachází výpočet
- `created_at` – datum, kdy byl v databázi vytvořen záznam testu
- `updated_at` – datum, kdy byl záznam testu naposledy upraven
- `error_text` – chybový text, který je vyplněn v případě, že se nepodařilo výpočet dokončit v pořádku
- `configuration` – konfigurace testu ve formátu JSON; aktuálně slouží především pro uložení konfigurace sdílených konektorů v atributu `connectors`

Tabulka test_metric V tabulce `test_metric` jsou informace o konfiguraci, stavu a výsledcích metrik. Nalezneme v ní tyto sloupce:

- `id_test_metric` – unikátní identifikátor metriky
- `name` – název metriky, která má být použita pro výpočet
- `configuration` – konfigurace metriky ve formátu JSON
- `status` – stav, ve kterém se aktuálně nachází výpočet
- `results` – výsledky ve formátu JSON
- `created_at` – datum, kdy byl v databázi vytvořen záznam metriky
- `updated_at` – datum, kdy byl záznam metriky naposledy upraven
- `error_text` – chybový text, který je vyplněn v případě, že se nepodařilo výpočet dokončit v pořádku
- `id_test` – cizí klíč určující, ke kterému testu metrika patří

Zdrojový kód 2.3: Příklad zprávy posílané do fronty

```
1 {
2   "testId": 10
3 }
```

2.3.2.2 Fronta zpráv

Účel fronty zpráv v systému je zprostředkovat komunikaci mezi rozhraním systému a jednotlivými Evaluatory. Když uživatel vytváří nový test, je do fronty zaznamenán unikátní identifikátor nového testu, a hned, jakmile je některý z Evaluatorů volný, vyzvedne si z fronty následující nevypočítaný test a začne ho řešit.

Jak již bylo zmíněno výše, jako fronta zpráv je v systému používán jeden z nejoblíbenějších nástrojů tohoto typu, a to konkrétně RabbitMQ.

V systému je používána jedna ze základních variant fronty. Rozhraní systému zastává pozici producenta, kdy do fronty přidává záznamy o nově vytvořených testech. Na druhé straně jsou na frontu napojeni Evaluatory, kteří zastávají pozici konzumentů. Ti postupně vybírají záznamy s unikátními identifikátory nově přidaných testů a zpracovávají je.

Do fronty je jako zpráva přidáván serializovaný JSON obsahující unikátní identifikátor testu, který se má zpracovat. Příklad takovéto zprávy lze vidět v ukázce 2.3. Zbytek informací o testu lze získat z relační databáze.

Fronta je nakonfigurovaná tak, aby vynucovala takzvané potvrzování zpráv. To funguje tak, že aby zpráva byla považována za úspěšně zpracovanou, nestačí, aby ji některý z konzumentů vybral z fronty, ale po dokončení zpracování úlohy musí poslat takzvané ACK potvrzující, že zpráva byla úspěšně zpracována. Pokud fronta neobdrží ACK od konzumenta a konzument skončí (odpojí se od fronty), zpráva je považována za nezpracovanou a obdrží ji jiný konzument.

Mechanizmus potvrzování zpráv v systému zajišťuje to, že každý test bude vždy za všech okolností vypočítán. Během výpočtu testu může daný Evaluator z nějakého důvodu neočekávaně skončit. Může mu dojít paměť, může dojít k neošetřené výjimce apod. Tím, že Evaluator musí po skončení výpočtu testu ještě poslat ACK zajišťuje, že pokud ho fronta neobdrží a Evaluator se z ní odpojí, test bude znovu zařazen do fronty a vypočítán jiným Evaluatorem.

2.3.2.3 Monitoring systému

Monitorování stavu a dohled nad tím, co se v systému děje, je důležitá součást každého složitějšího softwarového projektu. Stejně tak tomu je v systému pro analýzu chování modelů strojového učení, kde je potřeba kontrolovat mnoho věcí. Některé z nich uvádím pro představu v následujícím seznamu:

- *Přeplněnost fronty.* Pokud žádá mnoho uživatelů o výpočet testů najednou, systém se může při nedostatečném počtu instancí Evaluatorů rychle zaplnit a na výpočet testů se může dlouho čekat. Administrátor systému by se to měl dozvědět a zasáhnout například tím, že zvýší počet instancí Evaluatorů.

- *Chybně vypočítané testy.* Pokud skončí výpočet některé z metrik chybou, je potřeba zkontrolovat, co se stalo za chybu a proč vznikla. Nejprve je ale potřeba identifikovat testy, kde došlo k chybám. A právě chybně vypočítané testy by měly být v monitoringu nějakým způsobem vidět.

- *Jaké testy se počítají.* Z monitoringu by mělo být mimo jiné jasné, jaký typ testů se v systému aktuálně počítá a pro které zákazníky jsou výpočty prováděny.

- *Anomálie.* Nic není dokonalé, a proto by z monitoringu mělo být patrné, když se v systému děje nějaká nestandardní událost, jako například nějaký test zůstane nevypočítaný ve frontě a žádný z Evaluatorů se ho nemůže ujmout.

Na monitoring systému pro analýzu chování modelů strojového učení jsem zvolil kombinaci vyhledávacího a analytického nástroje Elasticsearch, který je používán pro ukládání logovacích záznámů ve formátu JSON, a vizualizačního nástroje Kibana, jenž slouží k vytváření dashboardů nad uloženými logovacími záznamy.

Logování metrik, které se v systému počítají, začíná již v části rozhraní systémů. Při vytváření nového testu je pro každou metriku testu založen nový logovací záznam s následujícími základními informacemi o počítané metrice:

- `testId` – unikátní identifikátor testu, jehož je metrika součástí

- `timestamp` – čas a datum odpovídající okamžiku, kdy byl vytvořen logovací záznam; používá se pro řazení záznamů v nástroji Elasticsearch

- `createdAt` – čas a datum vytvoření metriky

- `metric` – název metriky, která má být použita pro výpočet výsledků

- `configuration` – konfigurace metriky ve formátu JSON serializovaná do textové podoby

- `status` – stav, ve kterém se aktuálně nachází výpočet metriky

- `testType` – typ testu, díky čemuž se dají odlišit například výpočty metrik na doporučovacích modelech od výpočtů metrik na predikčních modelech

Zdrojový kód 2.4: Příklad logovacího záznamu úspěšně vypočtené metriky (konfigurace je pro stručnost zkrácena)

```
{
  "template": "homepage",
  "doneAt": "2017-04-19T17:31:01+0000",
  "configuration": "{\"requestsCount\":2000, ...}",
  "requestsCount": 2000,
  "testType": "Recommender fitness",
  "interactions": [
    "Purchase"
  ],
  "processingStartAt": "2017-04-19T17:17:18+0000",
  "createdAt": "2017-04-19T19:17:15+02:00",
  "timeframe": 120,
  "metric": "Recommender.Interacted",
  "testId": 12345,
  "timestamp": "2017-04-19T19:17:15+02:00",
  "status": "done",
  "customer": "cvut"
}
```

- `customer` – jméno zákazníka, na jehož datech je test prováděn; tento atribut se bere z konfigurace metriky, pokud je v ní dostupný

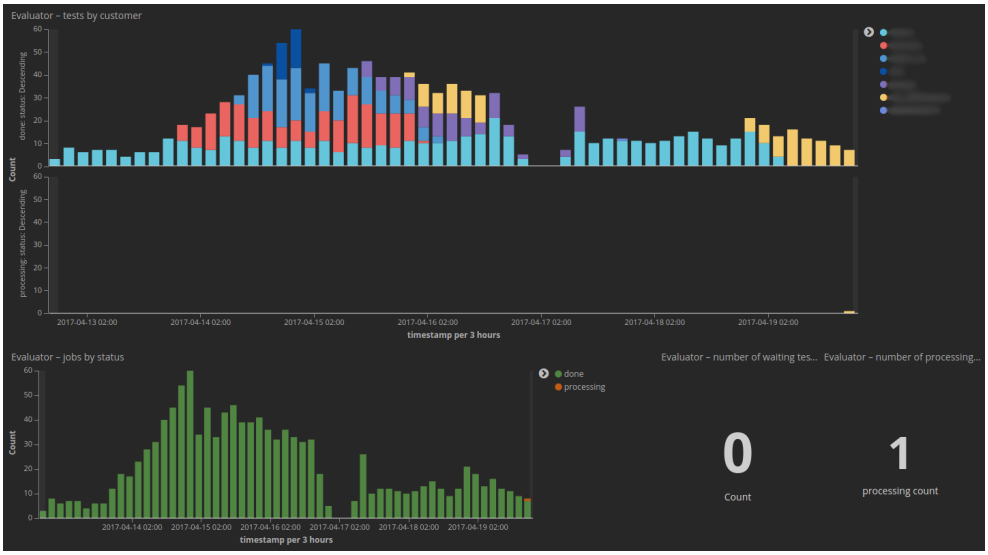
Když některý z dostupných Evaluatorů dostane k výpočtu danou metriku, získá si její záznam z Elasticsearch pomocí unikátní kombinace identifikátoru testu a názvu metriky (v jednom testu se každá metrika může vyskytovat maximálně jednou). Během výpočtu metriky je aktualizován její stav (atribut `status`). Evaluator během aktualizace logovacího záznamu přidává také atributy `processingStartAt` a `doneAt`, což jsou časové údaje začátku a konce výpočtu. V případě, že se při výpočtu vyskytne chyba, přidá se ještě atribut `failedAt`, jenž určuje její čas.

Jelikož rozhraní systému neví nic o struktuře konfigurace jednotlivých metrik, každá metrika před začátkem výpočtu může vrátit mapu atributů, které se mají přidat do logovacího záznamu. Lze tak logovat například specifické nastavení metriky. Stejně tak může metrika vrátit mapu atributů k zalogování po dokončení výpočtu, to slouží především k uložení informací o průběhu.

Příklad logovacího záznamu je vidět v ukázce 2.4.

V ukázce si lze všimnout atributů `requestsCount` a `timeframe`. Ty jsou specifické pro metriku `Recommender.Interacted` a jsou dodány do logovacího záznamu až před začátkem výpočtu.

Logovací záznamy jsou následně v nástroji Kibana agregovány dle jednotlivých atributů do různých vizualizací, z nichž jsou skládány dashboardy.



Obrázek 2.3: Příklad monitorovacího dashboardu

Díky tomu lze jednoduše sledovat průběh testů v systému a snadno objevit případnou chybu.

Příklad jednoho z monitorovacích dashboardů je vidět na obrázku 2.3. Nahoře je graf, který rozděluje počítané metriky dle stavu, ve kterém se aktuálně nacházejí, jednotlivé sloupce jsou pak děleny podle zákazníků, na jejichž datech se metriky počítají. Graf vlevo dole ukazuje, kolik metrik bylo v daný čas úspěšně dopočítáno (zelená barva) a kolik se jich ještě počítá (oranžová barva). Pokud by se objevila nějaká chyba, byla by vidět červenou barvou. Vpravo dole je pak vidět počet metrik, které čekají na výpočet, a počet aktuálně počítaných.

2.3.3 Výpočetní část systému

Výpočetní část systému se sestává z tzv. Evaluatorů. Těch může být libovolný počet. Kromě toho je každá instance Evaluatoru od ostatních zcela oddělena, díky čemuž je systém škálovatelný na neomezený počet zákazníků.

Evaluator je napojen na frontu zpráv a poslouchá, zdali se v ní neobjeví nový test. V případě, že dostane od fronty přidělen nový test, stáhne si o něm informace z relační databáze. Poté postupně vypočítá jednotlivé metriky a uloží výsledky zpět do databáze.

Díky tomu, že jsou Evaluatoři odstíněni od uživatelů systému rozhraním, vstupní formát do výpočetní části je jednotný pro všechny typy testů. Pokud nějaký externí systém vyžaduje speciální formát napojení, vše je vyřešeno v API části systému a Evaluator už jen dostane test ve standardním formátu. Tento návrh značně zjednodušil programovou podobu Evaluatoru. Ten se může

soustředit pouze na samotný výpočet, příjem požadavků je vyřešen v druhé části systému.

2.3.3.1 Použité technologie

Výpočetní část systému je naimplementovaná v programovacím jazyce Java[7]. Pro správu používaných knihoven byl využit nástroj Maven[8].

Pro zpřehlednění kódu a zlepšení testovatelnosti jednotlivých částí je použit Guice[9], framework pro vkládání závislostí. Jednodušší komunikaci s REST API ostatních služeb zajišťuje knihovna Unirest[10].

Pro logování chyb a zpráv na kontrolu běhu výpočtu je využita populární knihovna log4j[11]. Při ukládání záznamů do nástroje Elasticsearch byla komunikace s jeho REST rozhraním zjednodušena díky knihovně Jest[12].

2.3.3.2 Struktura

Základní struktura projektu výpočetní části systému vypadá takto:

- `/`
  - `logs` ..... Soubory s logy
  - `out` ..... Složka pro přeloženou verzi aplikace
  - `src`
    - `main`
      - `java` ..... Zdrojové kódy aplikace
        - `model` .... Objekty pro komunikaci s externími datovými službami
        - `recommender` .... Metriky a konektory určené pro měření doporučovacího systému
        - `Evaluator.java` ..... Třída s metodou *main*
      - `resources` ..... Konfigurační soubory
  - `pom.xml` ..... Konfigurace nástroje Maven

Pro přehlednost struktury byly vynechány podsložky složky *java*, ve kterých se nacházejí zdrojové kódy v programovacím jazyce Java.

2.3.3.3 Průběh výpočtu testu

Evaluator se nejprve připojí na frontu zpráv a čeká, dokud mu nebude přidělen nový test. Po obdržení testu k vyřešení si o něm stáhne podrobná data z relační databáze (od fronty dostane pouze unikátní identifikátor) a započne výpočet.

[7] https://java.com/
[8] https://maven.apache.org/
[9] https://github.com/google/guice
[10] http://unirest.io/
[11] https://logging.apache.org/log4j/2.x/
[12] https://github.com/searchbox-io/Jest

Evaluator postupně prochází jednotlivé metriky testu a řeší je. Před započetím výpočtu jsou vytvořeny konektory, se kterými bude metrika pracovat. Typ se pro jednotlivé konektory určí podle názvu předaném v konfiguraci metriky. Framework Guice pro vkládání závislostí ví o všech existujících typech. Je mu předán název konektoru a on vrátí tovární objekt, pomocí kterého je vytvořena nová instance odpovídajícího typu. Pokud daný typ neexistuje, je vyhozena výjimka přerušující výpočet metriky.

V konfiguraci testu lze mimo jiné uvést i sdílené konektory. Ty jsou vytvořeny před započetím výpočtu první metriky. Při vytváření metriky jsou už jen předány instance použitých sdílených konektorů.

Po dokončení přípravy konektorů je vytvořena nová instance metriky. Vytvoření probíhá podobně jako u konektoru: frameworku Guice je předán název metriky a ten vrátí tovární objekt. Tomu je následně předána konfigurace metriky a instance konektorů. Objekt si zvaliduje, zda dostal správný počet konektorů a zda jsou správného typu. Pokud ne, výpočet metriky je přerušen vyhozením výjimky. V případě, že vše proběhne v pořádku, je vrácena nová instance metriky připravená k výpočtu.

Na metrice je dále zavolána metoda určená k samotnému výpočtu a vrácené výsledky jsou uloženy do odpovídajícícho záznamu v relační databázi. Průběh zpracování je zcela v režii daného typu metriky, předané konektory mohou být zavolány v libovolné části výpočtu.

Po dokončení zpracování a uložení výsledků jsou na použitých konektorech zavolány destruktory sloužící k uvolnění využitých prostředků.

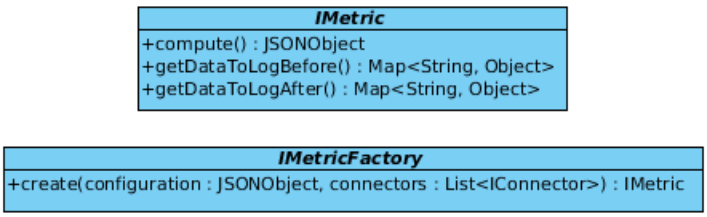
Před začátkem výpočtu metriky a po jeho skončení je měněn její stav jak v relační databázi, tak v nástroji Elasticsearch, který slouží pro monitoring celého systému.

Kromě toho může metrika před započetím výpočtu i po jeho skončení vrátit mapu atributů, jež budou uloženy k logovacímu záznamu do Elasticsearch. Mezi ně může například patřit jméno zákazníka, nad jehož daty je výpočet prováděn, nebo doba trvání výpočtu.

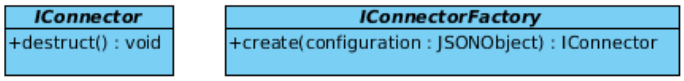
2.3.3.4 Základní třídy metriky

Každá metrika se skládá ze dvou základních tříd: třída pro výpočet výsledků metriky, ta musí implementovat rozhraní `IMetric`, a tovární třída implementující rozhraní `IMetricFactory`. Signatura obou rozhraní je vidět na obrázku 2.4.

Tovární třída musí poskytovat metodu `create`, která dostane jako první parametr konfiguraci metriky a jako druhý napojení na modely strojového učení. Výstupem této metody je instance metriky připravená k výpočtu výsledků. Metoda na vytvoření má za úkol sestavit novou instanci metriky, součástí čehož může být i kontrola typu a počtu předaných konektorů, sestavení konfiguračního objektu apod.



Obrázek 2.4: Základní rozhraní třídy pro metriku a její továrny



Obrázek 2.5: Základní rozhraní třídy pro konektor a jeho továrny

Třída každé metriky pak musí nabízet metodu `compute`, jejíž účelem je spočítat výsledky metriky. Tato metoda nepřijímá žádné parametry, kompletní nastavení metriky musí být provedno přes konstruktor nebo speciální metody před zahájením výpočtu, to znamená v továrním objektu v metodě `create`. Dále metrika poskytuje metody `getDataToLogBefore` a `getDataToLogAfter` určené k získání dat pro logovací účely.

2.3.3.5 Základní třídy konektoru

Každý konektor se podobně jako metrika skládá ze dvou základních tříd: třída pro napojování na model strojového učení, ta musí implementovat rozhraní `IConnector`, a tovární třída implementující rozhraní `IConnectorFactory`. Signatura obou rozhraní je vidět na obrázku 2.5.

Tovární třída musí poskytovat metodu `create`, která z předané konfigurace ve formátu JSON vytvoří novou instanci konektoru. Součástí vytváření může být například sestavení konfiguračního objektu, validace hodnot apod.

Třída konektoru pak nabízí pouze metodu `destruct`, která je volána po dokončení výpočtu metriky a má za úkol uvolnit všechny prostředky používané konektorem. Pro sdílený konektor je tato metoda volána až po dopočítání všech metrik. Kromě této metody může konektor poskytovat libovolné rozhraní.

Při vytváření metriky lze zvalidovat typ konektoru a převést všechny konektory z obecného rozhraní `IConnector` na dané konkrétní. Díky tomu může například metrika doporučovacího systému vyžadovat konektory určené k napojení na doporučovací model a pracovat s konkrétním rozhraním poskytující metodu, která vrací doporučené produkty.

2.3.4 Rozhraní systému

Rozhraní systému slouží k odstínění uživatelů od výpočetní části systému (tzv. Evaluatorů). Díky němu se komunikace zjednouší a zpřehlední. Navíc je systém schopen poskytnout formát komunikace na míru jiným externím systémům, například komponentě pro optimalizaci modelů. Přizpůsobení komunikace je díky tomuto návrhu jen otázka úpravy API rozhraní. Specifický formát je v této části přetransformován do objektů, se kterými pracuje výpočetní část a tu tedy není potřeba vůbec upravovat.

2.3.4.1 Použité technologie

Rozhraní systému je napsané v programovacím jazyce PHP[13]. Jako základ pro aplikaci byl použit framework Silex[14], který umožnil jednoduše rozdělit projekt dle architektury MVC. Příchozí požadavky jsou díky zabudovanému směrovači (v angličtině *router*) rozděleny do funkcí, které je vyřizují. Tyto funkce jsou slučovány podle významu do tzv. kontrolerů (v angličtině *controller*), což celkově zpřehledňuje jednotlivé části aplikace. V projektu jsou použity konfigurační soubory ve formátu YAML[15].

Na komunikaci s databází je použita knihovna Nextras\Dbal[16]. V rozhraní systému jsou mimo jiné umístěny i migrace na změny ve struktuře relační databáze. Na ty je použita knihovna Phinx[17]. Na komunikaci s frontou zpráv RabbitMQ je určena knihovna BunnyPHP[18], pro zjednodušení posílání dokumentů do nástroje Elasticsearch slouží PHP knihovna Elastica[19].

Rozhraní komunikuje s uživateli ve formátu JSON. Na usnadnění komunikace s jinými službami využívajícími HTTP komunikaci byla v projektu použita knihovna GuzzlePHP[20].

K logování chyb a událostí v rozhraní je určena knihovna Monolog[21]. Je nastavena tak, aby do souboru `all.log` ukládala všechny události v systému a do souboru `errors.log` pouze ty s úrovní závažnosti `error` a vyšší.

Na sestavení projektu je používán nástroj Phing[22]. Ten sestavuje konfigurační soubory a také spouští migrace databáze při nasazování projektu do testovacího nebo produkčního prostředí.

Všechny výše zmíněné knihovny jsou spravovány přes nástroj Composer[23], který je dnes již standardem pro správu závislostí v PHP aplikacích.

[13] https://php.net/
[14] https://silex.sensiolabs.org/
[15] http://yaml.org/
[16] https://nextras.org/dbal/docs/2.1/
[17] https://phinx.org/
[18] https://github.com/jakubkulhan/bunny
[19] http://elastica.io/
[20] http://docs.guzzlephp.org/en/latest/
[21] https://seldaek.github.io/monolog/
[22] https://www.phing.info/
[23] https://getcomposer.org/

2.3.4.2 Struktura

Základní struktura projektu rozhraní systému vypadá takto:

```
/
├── application
│   ├── app
│   │   ├── Controllers ..... Kontrolery vyřizující požadavky
│   │   ├── Library ..... Konfigurace směrování a pomocné objekty
│   │   ├── Model ..... Business logika aplikace
│   │   ├── Views ..... Třídy na formátování výstupu
│   │   ├── config ..... Konfigurace aplikace
│   │   └── bootstrap.php ... Soubor spojující jednotlivé části aplikace
│   ├── log ..... Soubory s logy
│   ├── vendor ..... Závislosti aplikace
│   ├── www
│   │   └── index.php ..... Vstupní soubor aplikace
│   └── composer.json ..... Konfigurace nástroje Composer
└── build ..... Skripty a konfigurace sestavení projektu
    └── migrations ..... Migrační soubory pro relační databázi
```

V následujících sekcích, které odpovídají dostupným kontrolerům, je seznam všech základních API volání včetně jejich funkcí. API volání jsou uvedena ve formátu začínajícím HTTP metodou, pod kterou fungují, a následně URL adresou, na níž jsou dostupné. Některá API volání obsahují v URL adrese proměnnou, ta bude uvedena vždy ve složených závorkách a v popisu je vysvětlena.

2.3.4.3 Kontroler HomepageController

Základní kontroler poskytující uvítací hlášku.

GET / Vrátí v odpovědi uvítací hlášku. Toto API volání existuje z důvodu testování dostupnosti rozhraní systému.

2.3.4.4 Kontroler ConnectorsController

Kontroler, ve kterém se nacházejí všechna API volání týkající se správy uložených konektorů.

GET /connectors Vrátí pole všech dostupných konektorů a jejich konfigurací.

POST /connectors/{connectorId} Uloží nový konektor do databáze. V URL adrese přijímá parametr `connectorId`, který určuje, pod jakým unikátním identifikátorem bude dostupný. Pokud již nějaký konektor pod daným

Zdrojový kód 2.5: Příklad těla požadavku na vytvoření nového konektoru

```
{
  "name": "Recommender.RepeatedRecommendations",
  "configuration": {
    "batchSize": 10,
    "template": {
      "algorithms": {
        "attribute": 1,
        "user": 0
      }
    }
  }
}
```

identifikátorem existuje, vrátí HTTP stavový kód 400. Ke stejné chybě dojde i v případě, že v těle požadavku chybí název konektoru.

Konfiguraci a jméno konektoru přijímá ve formátu JSON v těle požadavku. Jeho příklad je vidět v ukázce 2.5.

GET /connectors/{connectorId} Vrátí konfiguraci uloženého konektoru. V URL adrese přijímá parametr `connectorId` určující unikátní identifikátor konektoru, jehož konfigurace by měla být vrácena. Pokud pod zadaným identifikátorem není uložen žádný konektor, vrátí stavový kód 404.

PUT /connectors/{connectorId} Upraví konfiguraci uloženého konektoru. V URL adrese přijímá parametr `connectorId` určující unikátní identifikátor konektoru, jehož konfigurace má být upravena. V těle požadavku očekává konfiguraci a jméno konektoru, podobně jako volání na vytvoření nového konektoru. Toto API volání upravuje pouze ty atributy, které jsou uvedeny v těle požadavku. Pokud pod zadaným identifikátorem není uložen žádný konektor, vrátí kód 404.

DELETE /connectors/{connectorId} Smaže z databáze uložený konektor. V URL adrese přijímá parametr `connectorId` určující unikátní identifikátor konektoru, který by měl být smazán. Pokud pod zadaným identifikátorem není v databázi uložen žádný konektor, vrátí kód 404.

2.3.4.5 Kontroler TestsController

Tento kontroler zajišťuje vytváření a správu testů. Lze zde najít také volání na vrácení výsledků.

Zdrojový kód 2.6: Příklad těla požadavku na vytvoření nového testu (konfigurace konektorů byla kvůli stručnosti vynechána)

```
{
  "Recommender.FMeasure": {
    "connectors": [
      ...
    ],
    "customer": "cvut",
    "template": "homepage",
    "requestsCount": 2000,
    "timeframe": 3600,
    "interactions": [
      "Purchase"
    ]
  }
}
```

Zdrojový kód 2.7: Příklad použití uloženého konektoru (pro stručnost ukázky vynechána konfigurace metriky)

```
{
  "Recommender.FMeasure": {
    "connectors": [
      {
        "connectorId": "MyConnector"
      }
    ],
    ...
  }
}
```

POST /tests Vytvoří nový test s předanou konfigurací. Jeho nastavení se předává v těle požadavku ve formátu JSON, příklad je vidět v ukázce 2.6.

Klíčem je v nastavení název metriky, hodnotou její konfigurace. Jako odpověď dostane volající uživatel v případě úspěchu HTTP stavový kód 200. URL adresa pro získání podrobných informací o testu je k dispozici v hodnotě hlavičky `Location`. Rozhraní navíc v těle odpovědi vrátí podrobné informace ve formátu, ve kterém je dává API volání `GET /tests/{testId}`. Příklad vráceného JSON objektu je uveden v ukázce 2.9.

V tomto API volání lze použít místo kompletní konfigurace konektoru odkaz na uložený. Ten lze získat pomocí identifikátoru zvoleného při jeho vytváření (parametr `connectorId`). Použití uloženého konektoru je vidět v ukázce 2.7. Pokud uvedený identifikátor v systému neexistuje, v odpovědi je vrácen HTTP stavový kód 400.

Zdrojový kód 2.8: Příklad použití sdílených konektorů (pro stručnost ukázky byly vynechány některé části konfigurace testu)

```
{
  "connectors": [
    {
      "name": "Recommender.RepeatedRecommendations",
      "sharedId": "sharedConncetorId",
      "configuration": { ... }
    }
  ],
  "Recommender.FMeasure": {
    "connectors": [
      {
        "name": "Recommender.RepeatedRecommendations",
        "configuration": { ... }
      },
      {
        "sharedId": "sharedConncetorId"
      }
    ],
    ...
  },
  "Recommender.Precision": {
    "connectors": [
      {
        "sharedId": "sharedConncetorId"
      }
    ],
    ...
  }
}
```

Při vytváření testu je možné využít také sdílených konektorů. Ty jsou definovány v atributu `connectors`. Každý z nich je označený unikátním identifikátorem `sharedId`, kterým se lze na konektor odkázat při definování metrik. Příklad použití sdílených konektorů je vidět v ukázce 2.8.

GET /tests/{testId} Vrátí podrobné informace o testu. Ty dostaneme v JSON objektu s následujícím obsahem: unikátní identifikátor testu, datum vytvoření a poslední změny, stav testu a seznam metrik, které jsou součástí. U metrik je uvedena jejich konfigurace, stav, a případně výsledky výpočtu (pokud jsou již dostupné) v atributu `results`.

API volání přijímá v URL adrese parametr `testId` určující unikátní identifikátor testu, jehož podrobnosti mají být vráceny. Pokud není v systému dostupný, API volání vrátí stavový kód 404.

Zdrojový kód 2.9: Příklad podrobných informací o testu (pro stručnost byla vynechána konfigurace metriky)

```
{
  "testId": 35462,
  "status": "done",
  "createdAt": 1492365437,
  "updatedAt": 1492378791,
  "Recommender.FMeasure": {
    "status": "done",
    "configuration": {
      ...
    },
    "results": {
      "fitness": [
        0.040091260368998,
        0.039985757052237,
        0.039801126247906
      ],
      "support": 2000
    }
  }
}
```

POST /tests/{testId}/recompute Znovu spočítá předaný test. API volání přijímá parametr `testId` určující unikátní identifikátor testu, který má být znovu spočten. V případě, že v systému neexistuje, vrátí stavový kód 404.

Toto volání najde využití především v případech, kdy není test z nějakého důvodu dopočítán v pořádku. Lze ho díky němu znovu zařadit do fronty. Následně ho dostane jeden z dostupných Evaluátorů a zpracuje ho znovu.

POST /tests/{testId}/copy Načte předaný test a vytvoří nový s identickou konfigurací. API volání přijímá parametr `testId` určující unikátní identifikátor testu, který má být znovu spočten. Pokud neexistuje, vrátí HTTP stavový kód 404.

Volání najde využití především v případech, kdy chceme některý z existujících testů spočítat znovu, ale je důležité zachovat původní výsledky.

2.4 Využití

Systém byl již od počátku navrhován tak, aby byl co nejobecnější, dal se využít v co nejvíce situacích, a mohl na něj být napojen libovolný typ modelu strojového učení. Jelikož ho může používat mnoho jiných systémů, je postaven jako oddělená komponenta s vlastním API.

Díky této flexibilitě má systém mnoho různých využití. V následujících podkapitolách jsou popsaná čtyři nejtypičtější.

2.4.1 Chování systému

Každý model strojového učení má parametry, kterými lze ovlivňovat jeho chování. Prostým sledováním výstupu modelu lze většinou jen těžko zjistit, jak se chová při různém nastavení. Je to potřeba nějakým způsobem zautomatizovat, a to právě umožňuje systém navržený v této práci.

Lze do něj zavést metriky, které poskytují informaci o výstupu modelu. Systém může změřit hodnotu této metriky při určitém nastavení, uložit si výsledek, změnit hodnoty parametrů a znovu provést výpočet. Takto lze měřit třeba i několik desítek nastavení a sledovat, jak se hodnoty mění.

Příkladem může být zjištění rozmanitosti vrácených produktů doporučovacím systémem při různém nastavení modelu. Čím vyšší hodnota rozmanitosti je nastavena, tím rozmanitější typy produktů by měly být doporučovány. Můžeme tedy postupně zvyšovat rozmanitost doporučení a vždy spočítat danou metriku. Pokud se rozmanitost nezvyšuje, je potřeba zjistit důvod a případně najít chybu.

2.4.2 Monitoring správné funkčnosti modelu

Model běží nasazený v produkčním prostředí, funguje a vrací uživateli výstup. Ale jak zjistit, zdali se výsledky nezhoršují, například kvůli chybě v modelu, nebo kvůli snížení kvality dat, na kterých se model postupně učí?

Systém umožňuje průběžnou kontrolu funkčnosti modelů. Stačí nastavit opakující se volání, které změří vybranou metriku nebo více metrik na daném modelu a kontrolovat jejich výsledky.

Tímto způsobem lze kontrolovat, zdali model vůbec funguje. Pokud ne, systém může vrátit předem určenou hodnotu značící chybu nebo vyvolat výjimku. Jiný systém může chybovou hodnotu zpracovat a upozornit administrátora.

Lze s tím měřit i správnost fungování modelu strojového učení. Stačí do systému zavést metriku, která bude vybraným způsobem reprezentovat kvalitu výstupu modelu. Pokud klesne pod určenou mez, je potřeba zasáhnout a zjistit důvod. Stejně jako v předchozím případě lze upozornění zautomatizovat, nebo výsledky metriky vizualizovat.

2.4.3 Kontrola při nasazování nové verze modelu

Stačí drobná chyba ve zdrojovém kódu modelu a kvalita výstupu se může rapidně zhoršit. To se může stát i vinou programátora. Jak toto při nasazování nové verze modelu poznat? Programátor může zkontrolovat pár ukázkových výstupů modelu, ale to nemusí být dostatečné. Je potřeba kontrolu zautomatizovat.

K tomu lze také využít systém pro analýzu chování modelů strojového učení. Při každém nasazování lze nad novým modelem spočítat určené metriky a zjistit, zdali mají hodnoty v předem stanové normě. To vše se může dít automaticky. Hned jak budou známy výsledky metrik, může je nějaký další systém zpracovat, a buď model postupně nasadit do produkčního prostředí, nebo administrátora informovat o chybě.

Stejně tak si může programátor při nasazování nové verze modelu nechat spočítat vybrané metriky a podívat se na jejich výsledky. Díky tomu může zjistit, jestli svojí změnou ovlivnil výstup a jak. Díky těmto informacím se pak může rozhodnout, zdali nový model nasadí do produkčního prostředí.

2.4.4 Optimalizace modelů

Při automatické (nebo i manuální) optimalizaci modelů je potřeba zjišťovat, jak úspěšné budou jednotlivé konfigurace parametrů. Opět to lze zjistit pohledem na výstup, ale to ve většině případů bývá neefektivní a nepřesné.

Do optimalizace lze jednoduše zapojit systém pro analýzu chování modelů. Stačí přidat metriky reprezentující kvalitu výstupu modelu, tzv. *fitness*. Systém provádějící automatickou optimalizaci se bude postupně dotazovat na úspěšnost prohledávaných konfigurací, a bude zjišťovat, které jsou úspěšnější.

Úspěšnost konfigurací může být měřena i pomocí AB testů, kdy se zkoušejí různá nastavení na reálném provozu. Testování pomocí systému má však značnou výhodu v rychlosti. AB test musí být pro získání věrohodných výsledků puštěn třeba i několik dní, kdežto vyhodnocení metriky systémem typicky trvá maximálně pár minut.

Kapitola 3

Měření doporučovacího systému

V této kapitole je podrobně popsán doporučovací systém, na který byl napojen systém pro analýzu chování modelů strojového učení. Dále jsou zde uvedeny metriky pro měření doporučovacího modelu a postup jejich měření na reálných datech. Na konci kapitoly je shrnutí měření a jsou zde popsány výsledky, jichž bylo dosaženo.

3.1 Architektura doporučovacího systému

Doporučovací systém běží jako webová služba v cloudu přístupná přes standardní protokol HTTPS. API systému je navržené podle architektury REST, a je zabezpečené HMAC autentizací. Systém komunikuje ve formátu JSON.

3.1.1 Produkty

První entitou doporučovacího systému jsou produkty. To jsou objekty systémem doporučované uživatelům. Mohou reprezentovat libovolné objekty z domény zákazníka. Například pro elektronický obchod je to zboží, které zákazník prodává, pro hudební službu písně a interpreti, nebo pro zpravodajský server články.

Produkty mají definovány systémové atributy, jako například interní unikátní identifikátor, datumy vytvoření a poslední úpravy nebo příznak, zdali jsou smazané nebo ne. Systém rozlišuje dva druhy smazání: buď úplné smazání, nebo vyřazení z doporučování a ponechání pro učení modelu. Kromě systémových atributů může mít každý produkt nastavený libovolný počet vlastních atributů, které určuje zákazník. Pro internetový obchod to může být třeba atribut název zboží, rozměry, váha, popis nebo cena.

Systém umožňuje definovat atributy 5 typů: text, celé číslo, desetinné číslo, boolean nebo JSON. Ke každému textovému atributu může zákazník nastavit jazyk, ve kterém je hodnota zadána, což je pak použito při extrakci informací

z textu. U každého atributu lze nastavit i metatyp – příznak určující speciální význam atributu. K dispozici jsou celkem tři metatypy:

- *categories* – určuje, do které kategorie produkt patří; kategorizace produktů může významně pomoci zkvalitnit doporučení pro uživatele, u něhož ještě neznáme jeho historii interakcí nebo je zatím krátká
- *price* – určuje cenu produktu (musí to být celé nebo desetinné číslo); ta je následně použita k počítání obratu a jiných souvisejících metrik
- *title* – určuje název produktu; titulek je použit v administrátorském rozhraní místo unikátního identifikátoru při zobrazování produktu

3.1.2 Uživatelé

Další entitou datového modelu doporučovacího systému jsou uživatelé. Stejně jako produkty mají systémové atributy. Mohou mít také libovolný počet vlastních atributů, které plně určuje zákazník. Ty nejsou potřeba přímo pro účely doporučovacího systému, ale mohou být využity v jiných modelech. Povolené typy atributů, nastavení jazyka a metatypy jsou shodné s atributy produktů.

Uživatele lze zkopírovat, což způsobí zduplikování všech interakcí zdrojových uživatelů do toho cílového. Několik jich lze spojit do jednoho nového uživatele, který disponuje interakcemi všech původních.

3.1.3 Interakce

Interakce vzniká mezi uživatelem a produktem. Každá z nich má, stejně jako produkty a uživatelé, pevně dané systémové atributy. V tomto případě je to pouze čas provedení. Kromě toho si zákazník může opět určit libovolný počet vlastních atributů se stejnými pravidly, které se vztahovaly na ostatní entity.

Typ interakce určuje takzvaný interakční typ. Interakční typy říkají, které všechny druhy interakcí jsou dostupné v doméně zákazníka. Dostupné typy si určuje sám zákazník. Díky tomu je doporučovací systém velice flexibilní a dokáže se přizpůsobit potřebám zákazníka: typický internetový obchod pracuje s úplně rozdílnými interakčními typy než například hudební služba.

Stejně jako ostatní entity, i interakční typ může mít vlastní atributy. Kromě nich má speciální systémové atributy, které určují jeho vlastnosti:

- `interactionMetaType` – atribut určující speciální význam interakčního typu; může mít dvě hodnoty: akce a konverze; například u internetového obchodu je akcí prohlédnutí produktu a konverzí nákup; díky těmto speciálním typům je pak možné počítat metriky, jako například míru prokliku nebo konverzní poměr
- `interactionWeight` – atribut určující váhu interakčního typu používaný při generování doporučení; lze ho využít na rozlišení polarity: v případě

hudební služby by byla pozitivní hodnota přiřazena interakčnímu typu *líbí se* a negativní hodnota interakčnímu typu *nelíbí se*

- `interactionLearnWeight` – atribut určující váhu interakčního typu používaný při učení doporučovacího modelu; určuje polaritu interakce, stejně jako předchozí atribut

Na základě poskytnutých interakcí jsou následně počítány statistiky. Systém nabízí široké možnosti, jaké statistiky lze počítat. Dá se zjistit základní počet výskytu určité interakce, nebo komplexní vlastní statistiky s možností součtu, rozdílu, násobku nebo podílu. Dále třeba také statistiky, které hledají přesně dané souslednosti interakcí za sebou.

3.1.4 Konfigurace

Pro každého zákazníka existuje v systému separátní konfigurační objekt. V něm se konfiguruje vše, co se týká doporučování, statistik a optimalizace modelů. Lze zde tedy například najít nastavení hardwarových prostředků pro doporučovací modely nebo nastavení výpočtu statistik, které se mají pro daného zákazníka vyhodnocovat.

Část, která nás z konfiguračního objektu zajímá nejvíce, je určena k nastavení automatické optimalizace modelů. Nachází se pod klíčem `experimentConfig` a jsou zde k dispozici tyto atributy:

- `conversionInteractions` – seznam názvů interakcí, které jsou při počítání úspěšnosti konfigurace brány jako konverze
- `reactionTimeSeconds` – velikost atribučního okna, to znamená jak maximálně velké časové okno může následovat mezi doporučením nějakého produktu a jeho odpovídající konverzí tak, aby se konverze započítala jako vzniklá na základě doporučení
- `support` – počet požadavků z historie doporučování, který se má využít při počítání úspěšnosti konfigurace
- `batchSize` – počet požadavků na doporučení, která mají být najednou dávkově poslána na doporučovací model

Tato část se používá pro vytváření testů pro optimalizační komponentu doporučovacího systému 3.2.1 a měření přesnosti metrik vůči AB testům 3.2.2.

3.1.5 Doporučení

Doporučení, které systém vrátí, se výrazně liší podle nastavení zvoleného zákazníkem. Lze nastavit tyto základní parametry:

- `userId` – uživatel, pro kterého je doporučováno

- `itemId` – produkt, ke kterému je doporučováno; pokud je nastaven uživatel i produkt, lze určit, který z nich má na výsledná doporučení větší vliv (a jak velký)
- `count` – množství doporučených produktů, které se má vygenerovat
- `template` – určuje pokročilé nastavení doporučovacího modelu; může být určen buď unikátním identifikátorem, který odkazuje na uložené nastavení, nebo výčtem jednotlivých parametrů; pokud je použit jak unikátní identifikátor, tak výčet parametrů, uložené parametry budou přepsány předaným výčtem
- `filter` – výraz určující, které produkty se mohou doporučit
- `booster` – výraz, který může některé produkty zvýhodnit, případně znevýhodnit
- `diversity` – hodnota určující rozmanitost doporučených produktů; čím vyšší je, tím více se systém bude snažit doporučit relevantní produkty, které si jsou vzájemně co nejméně podobné
- `userInteractionTime` – určuje čas, ke kterému budou doporučení vygenerována; jinak řečeno, umožňuje vrátit se v čase a získat doporučení tak, jak by byla vygenerována ve zvolený čas

Výstupem požadavku pro získání doporučení je pole produktů. U každého produktu je uvedeno skóre, které systém vypočítal při generování doporučení.

Lze poslat i více požadavků najednou, v tomto případě mluvíme o tzv. dávkovém doporučení. V odpovědi je pak pole doporučení pro jednotlivé požadavky. V dávce lze nastavit, zda se mají jednotlivé požadavky zpracovávat sekvenčně či paralelně. Dále lze zvolit, zda mohou doporučení obsahovat stejné produkty, čímž lze z jednotlivých požadavků odfiltrovat duplicity. To se může hodit v případě doporučování pro více pozic na jedné stránce, uživatel díky tomu uvidí každý produkt maximálně jednou.

3.1.6 AB testování

Jak již název napovídá, AB testování [13] [14] je postup, kdy jsou uživatelé rozděleni do dvou nebo více skupin, a každé skupině jsou prezentovány odlišné varianty předmětu testování, například obsahu nebo designu. Uživatelé jsou rozděleni do skupin zcela náhodně, ale je důležité, aby jeden uživatel po celou dobu testu spadal do té samé skupiny. Testování by ztratilo význam, pokud bychom jednomu uživateli poprvé zobrazili variantu A a podruhé variantu B. Úspěšnost variant je pak vyhodnocena podle určité cílové metriky. U internetového obchodu to bývá nákup, u zpravodajského serveru přečtení článku nebo označení za oblíbený.

Platforma testovaného doporučovacího systému umožňuje transparentní a efektivní AB testování a jeho vyhodnocení podle zvolené metriky. AB testy je možné řadit hierarchicky, to znamená, že AB testy mohou dělit uživatele na skupiny a podskupiny.

V doporučovacím systému se typicky testuje úspěšnost skupiny se zapnutým doporučováním proti skupině, u které je doporučování vypnuto. AB testováním lze také porovnat dvě (nebo více) konfigurací a zjistit, která funguje na datech daného zákazníka lépe. Ta nejlepší konfigurace se většinou pro různé zákazníky a pozice liší.

3.1.7 Schéma databáze

V metrikách implementovaných do systému pro analýzu chování modelů strojového učení budu používat i data přímo z databáze doporučovacího systému. V následující kapitole proto ve stručnosti popíšu strukturu nejdůležitých tabulek:

- tabulka `items` – sem se ukládají informace o produktech včetně systémových atributů i atributů definovaných zákazníkem
- tabulka `users` – tabulka podobná té předchozí, akorát sdružující informace o uživatelích
- tabulka `interaction_definitions` – zde jsou uloženy názvy typů interakcí a jejich atributy
- tabulka `interactions` – seznam interakcí, které byly uskutečněny; základem každého řádku je čtveřice produkt, uživatel, interakce a čas, která unikátně identifikuje každý řádek; jinak řečeno, základem je informace, který uživatel s kterým produktem a kdy interagoval a jaký typ interakce to byl; dále můžou být v každém řádku přítomny hodnoty atributů interakce definované zákazníkem
- tabulka `recommended_requests` – úložiště požadavků na doporučení a jejich výsledků; v této tabulce lze dohledat, kdy a jakému uživateli byly jaké produkty doporučeny; lze odtud získat i kompletní konfiguraci požadavku na doporučení a informace o tom, jakého skóre dosáhly doporučené produkty při generování doporučení modelem; tato tabulka slouží především k dohledávání chyb a nesrovnalostí v doporučování a k optimalizaci modelů

3.1.8 Optimalizace modelů doporučovacího systému

Jednou z komponent doporučovacího systému je automatický optimalizátor modelů. Typický zákazník systému má několik pozic, na které chce doporučovat. U elektronického obchodu to je například přehled nejoblíbenějších

produktů na domovské stránce nebo podobné produkty na stránce detailu produktu. Ke každé z těchto pozic se uživatel dostane při trochu odlišné příležitosti, liší se pro ně tedy i nejvhodnější nastavení modelu pro doporučování. U jedné pozice je vhodné nastavit větší rozmanitost doporučení, jinde je zas lepší použít větší váhu doporučování podle podobnosti produktů a snížit váhu doporučování na základě historie chování uživatele.

Pro jednotky klientů by se optimalizace dala teoreticky zvládat ručně, pokud chceme ale škálovat doporučovací systém na stovky, tisíce nebo dokonce desetitíce zákazníků, je potřeba začlenit do systému komponentu, která bude tuto optimalizaci dělat automaticky a ideálně daleko přesněji a rychleji, než by ji dělal člověk.

Tato komponenta je začleněna i do testovaného doporučovacího systému. Optimalizační program postupně prochází stavový prostor možných konfigurací a testuje ty, o kterých si myslí, že by mohly být úpěšné. Testování funguje ve dvou fázích:

1. offline testování konfigurací

2. testování konfigurací pomocí AB testu

3.1.8.1 Offline testování konfigurací

Komponenta pro automatickou optimalizaci doporučovacích modelů nejprve testuje prohledávané konfigurace pomocí offline testování. Vyhodnocení zdatnosti konfigurace je svěřeno jiné části systému, optimalizátor zná pouze URL adresu, které má předat konfigurace k otestovaní. Optimalizátor pak už jen vyhodnotí výsledky.

Offline testování probíhá na historických datech, do testování tedy není vůbec zapojen reálný provoz doporučovacího systému. Může probíhat například tak, že systém vezme nějaký požadavek na doporučení z historie, a s konfigurací změněnou na testovanou tento požadavek znovu pošle na doporučovací model. Systém následně projde doporučené produkty a podívá se, zdali uživatel s některým z doporučených produktů interagoval nebo ne. Tímto způsobem systém vyhodnotí zadaný počet doporučovacích požadavků a vrátí vypočítanou úspěšnost.

Tato forma testování má výhodu v rychlosti. Znovu zopakovat a vyhodnotit výsledky několika stovek nebo tisíc doporučení je řádově rychlejší než čekat několik dní na výsledek AB testu.

Aby bylo offline testování co nejefektivnější a nejpřínosnější, je potřeba zvolit správnou metriku pro vyhodnocení úspěšnosti jednotlivých konfigurací doporučovacího modelu.

3.1.8.2 Testování konfigurací pomocí AB testu

Po tom, co je prohledávaný prostor možných konfigurací zúžen pomocí offline testování, nastává fáze, kdy jsou nejúspěšnější konfigurace testovány pomocí AB testů. Stejně jako v předchozím případě, komponenta pro automatickou optimalizaci neví o AB testech nic, pouze zná URL adresu, kterou může zavolat, předá ji konfigurace k otestování a počká na výsledky.

Testování konfigurací pomocí AB testu probíhá tak, že jsou uživatelé rozděleni do tolika skupin, kolik je potřeba otestovat konfigurací. Každé skupině je přidělena jedna z konfigurací a testuje se na reálném provozu doporučovacího systému, která z konfigurací je úspěšnější.

Oproti offline testování má tento postup výhodu v přesnosti. Předchozí způsob se jen snaží pomocí co nejvhodnějších metrik odhadnout, jak by jednotlivé konfigurace dopadly v reálném provozu, kdežto AB testy testují konfigurace na reálných uživatelích a jejich chování. Nevýhodou je, že pro získání věrohodných výsledků je potřeba nechat test běžet několik dní.

3.2 Rozšíření systému pro analýzu chování modelů strojového učení pro doporučovací systém

V této části kapitoly popíšu rozšíření zavedená do rozhraní systému pro analýzu chování modelů strojového učení speciálně určené pro účely měření doporučovacích modelů.

Do PHP aplikace tvořící rozhraní systému byly přidány dva kontrolery určené pro obsluhu specifických požadavků od externích systémů doporučování. Aby byly od zbylých kontrolerů odděleny, zařadil jsem je do samostatné složky `Recommender`.

3.2.1 Měření zdatnosti konfigurace

Na rozhraní systému se napojuje speciální komponenta určená pro optimalizaci konfigurace modelů doporučování. Aby se tato komponenta mohla napojit na nějaký systém, je potřeba pro ni připravit určité rozhraní. Z toho důvodu byl do systému přidán nový kontroler `FitnessController` obsluhující požadavky optimalizační komponenty.

POST /recommender/{customer}/fitness Vytvoří nový test zdatnosti doporučovacího modelu s předanými konfiguracemi. Toto API volání přijímá parametr `customer` určující unikátní identifikátor zákazníka, pro nějž má být zdatnost spočítána.

V těle požadavku přijímá ve formátu JSON pozici, pro kterou se optimalizují parametry (atribut `originalConfigurationId`), a konfigurace, pro než má systém vrátit zdatnost (ty jsou uvedeny v atributu `configurations` indexované svými identifikátory). Příklad těla požadavku je vidět v ukázce 3.1.

Zdrojový kód 3.1: Příklad těla požadavku na výpočet zdatnosti konfigurací doporučovacího modelu

```
{
  "originalConfigurationId": "homepage",
  "configurations": {
    "configId1": {
      "algorithms": {
        "attribute": 1
      }
    },
    "configId2": {
      "algorithms": {
        "attribute": 0.5
      }
    }
  }
}
```

Rozhraní si po přijmutí požadavku načte konfiguraci daného zákazníka a vytvoří podle ní nový test s jednou metrikou, v níž bude pro každou konfiguraci k ohodnocení nastaven jeden konektor. Dále už je test systémem zpracován standardním postupem.

V odpovědi rozhraní na požadavek je uvedena URL adresa, na které budou po dokončení zpracování dostupné výsledky testu. Tato URL adresa vede na API volání popsané v následující části této podkapitoly.

GET /recommender/fitness/{testId} Vrátí zdatnost konfigurací doporučovacího modelu ve formátu, který vyžaduje optimalizační komponenta. V URL adrese vyžaduje parametr `testId` určující unikátní identifikátor testu, z kterého se načtou výsledky. Pokud test s předaným identifikátorem neexistuje, volání vrátí chybový stavový kód 404. Pokud pod daným identifikátorem není uložen test vytvořený předchozím voláním, vrátí chybový kód 400.

Toto volání vezme výsledky načteného testu a transformuje je do podoby, kterou vyžaduje optimalizační komponenta. Příklad tohoto formátu je v ukázce 3.2.

Volání vrátí pro každou testovanou konfiguraci dosaženou zdatnost a základní informace o výpočtu, mezi než patří datum a čas začátku a konce výpočtu, jaká je podpora dosažené zdatnosti (tzv. *support*) a zdali je výpočet deterministický.

POST /recommender/{customer}/fitness/sync Blokující varianta předchozích dvou volání. Přijme požadavek ve stejném formátu jako volání na vytvoření nového testu zdatnosti, vytvoří test, počká na dokončení zpra-

Zdrojový kód 3.2: Příklad odpovědi se zdatností různých konfigurací doporučovacího modelu

```
{
  "fitness": {
    "configId1": {
      "fitness": 0.89,
      "support": 1000,
      "startTime": 1475876232,
      "endTime": 1475876242,
      "type": "deterministic"
    },
    "configId2": {
      "fitness": 0.15,
      "support": 1500,
      "startTime": 1475876232,
      "endTime": 1475876242,
      "type": "nondeterministic"
    }
  }
}
```

cování a poté vrátí výslednou zdatnost ve stejném formátu jako předchozí volání.

Toto volání bylo do systému zavedeno z důvodu kompatibility se starší verzí optimalizační komponenty.

3.2.2 Měření přesnosti metrik vůči AB testům

Při měření úspěšnosti metrik přidaných do systému pro měření doporučovacího modelu je potřeba zjišťovat, jak přesně odrážejí skutečné chování uživatelů, kterým bylo doporučeno. Pro zjišťování, zdali metriky dobře rozlišují úspěšnost doporučovacího modelu při různých konfiguracích, se nabízí srovnání zdatností vrácených metrikou s výsledky AB testů. Je důležité podotknout, že v tomto srovnání nejde o výslednou hodnotu zdatnosti, ale spíše o to, zdali metriky dokáží správně určit, která z konfigurací je lepší, a o kolik (důležitý je procentuální rozdíl, nikoli absolutní hodnota).

Aby se daly jednoduše pozorovat a vizualizovat výsledky metrik vůči AB testům, bylo do systému zavedeno několik nových API volání. Ta jsou sdružena do kontroleru nazveného `AbTestController`.

Pro každý AB test je v systému vytvořen nový test, ve kterém jsou spočteny zdatnosti vybraných metrik. Každé skupině AB testu odpovídá v konfiguraci metriky jeden konektor s odpovídajícím nastavením doporučovacího modelu. Tím se ve výsledcích objeví stejný počet zdatností jako je skupin.

Výsledky z metrik bude díky tomu možné jednoduše porovnat s výsledky AB testu.

Aby bylo srovnání vždy aktuální, jsou výpočty starší než je určitá mez označeny systémem za zastaralé a budou na vyžádání spočteny znovu. Tato doba je aktuálně nastavena na 1 den.

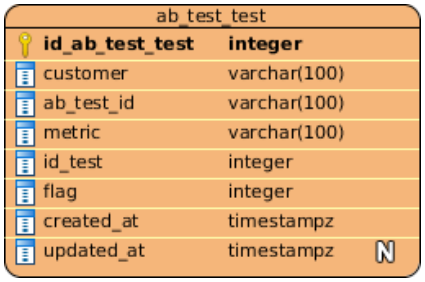
3.2.2.1 Rozšíření datového modelu

Aby systém dokázal jednoduše a rychle zjistit, ve kterém testu se nacházejí výsledky pro daný AB test a metriku, do relační databáze byla přidána tabulka `ab_test_test`. Její schéma je na obrázku 3.1. Tabulka obsahuje tyto sloupce:

- `id_ab_test_test` – unikátní identifikátor záznamu v relační databázi
- `customer` – jméno zákazníka, pro nějž byla daná metrika pro daný AB test spočtena
- `ab_test_id` – unikátní identifikátor AB testu, pro nějž byla metrika spočtena
- `metric` – název spočítané metriky
- `id_test` – unikátní identifikátor testu, ve kterém se nacházejí výsledky
- `flag` – příznak, zdali jsou data z tohoto testu aktuální. Může nabývat tří celočíselných hodnot:
  - 0 (*actual*) – říká, že data z tohoto testu jsou aktuální a neexistuje žádný nový výpočet dané metriky pro daný AB test
  - 1 (*outdated*) – říká, že existuje jeden novější test pro danou metriku a daný AB test; pokud novější test ještě není dokončen, použijí se výsledky z tohoto testu, protože jsou poslední platné
  - 2 (*invalid*) – říká, že existují minimálně dva čerstvější testy, z nichž určitě jeden má novější výsledky; tento test již není platný, v systému zůstává pro zachování historie výpočtů
- `created_at` – datum a čas vytvoření záznamu v databázi
- `updated_at` – datum a čas poslední úpravy záznamu v databázi

3.2.2.2 Rozšíření rozhraní systému

GET /recommender/ab-tests/metrics Vrátí názvy všech dostupných metrik, které lze pro AB testy doporučovacího systému vypočítat.



Obrázek 3.1: Schéma tabulky určené k ukládání vypočítaných metrik pro AB testy

GET /recommender/{customer}/ab-tests/{abTestId} Vrátí poslední vypočítané výsledky pro předaného zákazníka a předaný AB test. API volání přebírá celkem dva povinné a jeden nepovinný parametr:

- `customer` – unikátní identifikátor zákazníka, pro nějž mají být vráceny výsledky; pokud zákazník neexistuje, API vrátí chybový stavový kód 404
- `abTestId` – unikátní identifikátor AB testu, pro nějž mají být vráceny výsledky; pokud AB test u předaného zákazníka neexistuje, vrátí rovněž stavový kód 404
- `metrics` – nepovinný seznam názvů metrik, jejichž výsledky se mají vrátit; více metrik lze oddělit čárkou; pokud není uveden, vrátí výsledky všech dostupných metrik

V odpovědi rozhraní jsou pro každou metriku uvedeny výsledky spočtené Evaluatory (atribut *evaluator*), výsledky AB testu (atribut *abTest*), a stav říkající, zdali jsou výsledky platné (atribut *status*). Výpočet pro danou metriku může být celkem ve 4 stavech:

- *unknown* – tento AB test ještě nebyl pro danou metriku spočítán a žádný test pro spočtení aktuálně neprobíhá
- *processing* – tento AB test ještě nebyl pro danou metriku spočítán, ale aktuálně probíhá test, který ji spočte
- *fresh* – tento AB test již byl pro danou metriku spočítán a výsledky jsou aktuální
- *outdated* – tento AB test již byl pro danou metriku spočítán, ale výsledky nejsou aktuální, jsou starší než 1 den

Zdrojový kód 3.3: Příklad odpovědi s posledními vypočítanými výsledky AB testu (pro stručnost je část výstupu zkrácena)

```
{
  "Interacted": {
    "status": "fresh",
    "evaluator": {
      "group0": {
        "fitness": 0.34431630971993,
        "support": 607,
        "startTime": 1484225114,
        "formattedStartTime": "2017-01-12 12:45:14",
        "endTime": 1484225195,
        "formattedEndTime": "2017-01-12 12:46:35"
      },
      "group1": {
        ...
      }
    },
    "abTest": {
      "group0": {
        "fitness": 9.6303280661409,
        "support": 1642,
        "startTime": 1484225536,
        "formattedStartTime": "2017-01-12 12:52:16",
        "endTime": 1484225536,
        "formattedEndTime": "2017-01-12 12:52:16"
      },
      "group1": {
        ...
      }
    }
  }
}
```

Ve výsledcích z Evaluatoru i AB testu je uvedena dosažná zdatnost (tzv. *fitness*, u AB testů tuto zdatnost určuje doporučovací systém), podpora (tzv. *support*) a čas a datum začátku a konce výpočtu. Příklad těla odpovědi systému se všemi uvedenými atributy je v ukázce 3.3.

POST /recommender/{customer}/ab-tests/{abTestId} Obnoví výsledky předaných metrik pro daného zákazníka a daný AB test. Pokud test pro uvedené metriky ještě nebyl nikdy spočten nebo jsou výsledky zastaralé (starší než 1 den), spustí nový výpočet, jinak neudělá nic.

API volání přijímá stejné parametry jako předchozí, navíc však nabízí jeden nepovinný:

Zdrojový kód 3.4: Příklad odpovědi rozhraní na obnovu výsledků metrik

```
{
  "Interacted": {
    "status": "outdated => processing"
  }
}
```

- `force` – pokud je tento parametr v požadavku uveden, spustí nový výpočet i v případě, že jsou poslední dostupné výsledky aktuální

Rozhraní vrátí v těle odpovědi pro každou uvedenou metriku původní a nový stav. Příklad odpovědi je vidět v ukázce 3.4, kdy pro daný AB test a metriku `Interacted` byly výsledky zastaralé, a tak byl vytvořen nový test, ve kterém budou spočteny znovu (stav metriky se mění z *outdated* na *processing*).

GET /recommender/{customer}/ab-tests Funguje podobně jako `GET /recommender/{customer}/ab-tests/{abTestId}` jen s tím rozdílem, že vrátí výsledky výpočtů pro všechny dostupné AB testy předaného zákazníka.

POST /recommender/{customer}/ab-tests Funguje podobně jako `POST /recommender/{customer}/ab-tests/{abTestId}` jen s tím rozdílem, že obnoví výsledky výpočtů všech dostupných AB testů předaného zákazníka.

GET /recommender/ab-tests Funguje podobně jako `GET /recommender/{customer}/ab-tests/{abTestId}` jen s tím rozdílem, že vrátí výsledky výpočtů pro všechny dostupné AB testy všech zákazníků uvedených v parametru `customers` (více zákazníků lze oddělit čárkou).

POST /recommender/ab-tests Funguje podobně jako `POST /recommender/{customer}/ab-tests/{abTestId}` jen s tím rozdílem, že obnoví výsledky výpočtů všech dostupných AB testů všech zákazníků uvedených v parametru `customers` (více zákazníků lze oddělit čárkou).

3.3 Konektory

V této části kapitoly jsou popsány konektory, které byly do systému zavedeny pro doporučovací systém.

3.3.1 Konektor OriginalRecommendations

Účelem tohoto konektoru je vrátit originální doporučení, která byla vygenerována, když byly dané požadavky zpracovávány doporučovacím systémem.

Konektor dostane v metodě `getRecommendations` pole požadavků na doporučení z minulosti. Následně pro každý požadavek získá z databáze doporučené produkty a vrátí je v poli volajícímu.

3.3.2 Konektor RepeatedRecommendations

Tento konektor slouží k zopakování požadavků na doporučovací model z minulosti. Lze je zopakovat se stejným nebo odlišným nastavením modelu.

Konektor dostane v metodě `getRecommendations` pole doporučovacích požadavků. U těch je přepsáno nastavení parametry předanými v konfiguraci konektoru (kromě parametru `batchSize`, viz. dále). Parametrům původního požadavku, které nejsou přepsány v konfiguraci konektoru, zůstává originální hodnota.

Následně jsou požadavky postupně posílány na doporučovací model. Konektor doporučené produkty sloučí seskupené po jednotlivých požadavcích do pole a vrátí je volajícímu.

3.3.2.1 Nastavení konektoru

Konektor dostane při inicializaci JSON objekt s konfigurací. Pokud je v něm nastaven parametr `batchSize`, posílají se na doporučovací model dávková doporučení o velikosti odpovídající hodnotě parametru. Pokud nastavený není, posílá se najednou pouze jedno doporučení.

Zbylé parametry z konfigurace konektoru jsou použity na změnu nastavení doporučovacích požadavků.

3.4 Měření přesnosti výstupu doporučovacího modelu

V této části kapitoly budou popsány metriky, které byly do systému zavedeny pro měření přesnosti výstupu doporučovacího modelu. Při vymýšlení metrik byla brána inspirace v [15] [16].

3.4.1 Kostra metrik

Všechny metriky pro měření přesnosti výstupu doporučovacího modelu mají stejnou kostru výpočtu, liší se pouze tím, jakým způsobem se na konci vyhodnotí výsledná zdatnost konfigurace, tzv. *fitness*. Metriky přijímají pouze konektory typu `RepeatedRecommendations`.

Výpočet začíná načtením doporučovacích požadavků z historie uložené v databázi. Ne všechny požadavky splňují konfiguraci metriky, a tak je postupně procházena historie a jsou vybírány pouze ty požadavky, které jsou určeny na testovanou pozici a které byly doporučeny nějakému konkrétnímu uživateli. Metrika se kvůli co nejvěrohodnějším výsledkům snaží od každého uživatele použít maximálně jeden požadavek. Pokud však dojde na konec historie a nemá dostatečný počet požadavků (je menší než parametr `requestsCount`), berou se v potaz i další požadavky od již zahrnutých uživatelů.

Následující kroky se provádějí postupně pro všechny předané konektory: načtené doporučovací požadavky z historie jsou předány konektoru, který získá doporučení s nastavením modelu uvedeným v jeho konfiguraci. Ta jsou následně předána zpět do metriky, která vypočte výslednou zdatnost. Výpočet se liší podle typu metriky a bude u každé uveden zvlášť. Zdatnost je nakonec uložena do pole výsledků metriky.

Příklad formátu výstupu je v ukázce 3.5.

3.4.2 Nastavení metrik

Ve všech metrikách pro měření přesnosti výstupu doporučovacího modelu jsou dostupné tyto konfigurační parametry:

- `customer` – zákazník, na jehož datech se má metrika spočítat
- `template` – název uloženého nastavení doporučovacího modelu, pro nějž se mají brát požadavky z historie
- `requestsCount` – počet doporučovacích požadavků z historie, které se mají použít pro výpočet metriky
- `timeframe` – velikost časového okna, po které se interakce uživatele s doporučeným produktem bere jako interakce vzniklá z doporučení (produkt je pro uživatele relevantní)
- `interactions` – pole názvů konverzních interakcí (pokud uživatel provede jednu z těchto konverzí s jedním z doporučených produktů v daném časovém okně, je to bráno jako relevantní doporučení); pokud není vyplněno, konverzní jsou všechny interakce

3.4.3 Metrika Precision

Metrika *Precision* patří mezi nejpoužívanější metriky pro ohodnocení úspěšnosti systémů pro vyhledávání informací, tzv. *information retrieval systems*. V oblasti doporučovacích systémů vyjadřuje pravděpodobnost, že doporučený produkt je relevantní, a lze ji vypočítat následovně:

$$\text{Precision} = \frac{\text{počet relevantně doporučených produktů}}{\text{počet doporučených produktů}}$$

Zdrojový kód 3.5: Příklad výsledků metriky na měření přesnosti výstupu doporučovacího systému (pro stručnost zkráceno)

```
{
  "testId": 1234,
  "status": "done",
  "createdAt": 1484444388,
  "updatedAt": 1484447634,
  "Recommender.Interacted": {
    "status": "done",
    "configuration": {
      "connectors": [
        {
          "name": "Recommender.RepeatedRecommendations",
          "configuration": {
            ...
          }
        },
        ...
      ],
      ...
    },
    "results": {
      "fitness": [
        0.18031968031968,
        0.22077922077922,
        0.23726273726274
      ],
      "support": 2000
    }
  }
}
```

3.4.3.1 Metrika TurnoverPrecision

U většiny zákazníků dává smysl optimalizovat doporučovací systém na maximální obrat získáný z doporučení. Upravíme tedy metriku *Precision* tak, aby nepočítala s počtem relevantně doporučených produktů, ale obratem získaným z nich. Upravenou metriku *TurnoverPrecision* lze vyhodnotit podle následujícícho vzorce:

$$\text{TurnoverPrecision} = \frac{\text{součet cen relevantně doporučených produktů}}{\text{počet doporučených produktů}}$$

3.4.4 Metrika Recall

Metrika *Recall* je další široce používanou metrikou pro hodnocení úspěšnosti systému pro vyhledávání informací. V oblasti doporučování vyjadřuje pravděpodobnost, že relevantní produkt bude vybrán k doporučení. Hodnotu metriky lze získat podle následujícího vzorce:

$$\text{Recall} = \frac{\text{počet relevantně doporučených produktů}}{\text{počet relevantních produktů}}$$

3.4.4.1 Metrika TurnoverRecall

Stejně jako u metriky *Precision* dává smysl upravit metriku *Recall* tak, aby počítala s obratem z doporučených relevantních produktů. Tato metrika byla nazvána *TurnoverRecall* a lze ji vypočítat následujícím poměrem:

$$\text{TurnoverRecall} = \frac{\text{součet cen relevantně doporučených produktů}}{\text{počet relevantních produktů}}$$

3.4.5 Metrika FMeasure

Metrika *FMeasure* je kombinací dvou předešlích metrik. Lze ji interpretovat jako vážený průměr mezi metrikami *Precision* a *Recall*, a počítá se podle následujícího vzorce:

$$\text{FMeasure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.4.5.1 Metrika TurnoverFMeasure

Stejně jako předchozí metriky i tato může být počítána s obratem z doporučených produktů. Metrika je pojmenována *TurnoverFMeasure* a lze ji vyhodnotit tímto způsobem:

$$\text{TurnoverFMeasure} = \frac{2 \cdot \text{TurnoverPrecision} \cdot \text{TurnoverRecall}}{\text{TurnoverPrecision} + \text{TurnoverRecall}}$$

3.4.6 Metrika Interacted

Metrika *Interacted* je jedna z alternativních metrik pro hodnocení přesnosti doporučovacích systémů. Vyjadřuje poměr doporučení, ve kterých byl alespoň jeden relevantní produkt pro uživatele, k počtu všech zkoumaných doporučení. Jinak řečeno metrika vyjadřuje poměr doporučení, která byla relevantní pro uživatele. Lze ji vypočítat následovně:

$$\text{Interacted} = \frac{\text{počet relevantních doporučení}}{\text{počet doporučení}}$$

Tabulka 3.1: Procento správně odhadnutých AB testů

	Zákazník A	Zákazník B	Zákazník C
Precision	48,28 %	59,26 %	55,88 %
TurnoverPrecision	37,93 %	62,96 %	60,61 %
Recall	48,28 %	59,26 %	55,88 %
TurnoverRecall	37,93 %	62,96 %	57,58 %
FMeasure	48,28 %	59,26 %	55,88 %
TurnoverFMeasure	37,93 %	62,96 %	58,82 %
Interacted	48,28 %	51,85 %	50 %

3.4.7 Měření metrik

Pro změření úspěšnosti metrik na reálných zákaznících byl zvolen experiment srovnání výsledků AB testů s výsledky metriky. V tomto experimentu není důležité, zdali velikost vrácené hodnoty ze systému se shoduje s velikostí zdatnosti vrácené samotnými AB testy, ale je potřeba zkoumat, zdali metriky dokáží správně rozlišit, která z testovaných konfigurací je zdatnější, a zdali dokáží odhadnout trend rozdílu zdatností. Jinak řečeno není důležité, zdali metrika odhadne správně absolutní rozdíl zdatností mezi konfiguracemi, ale je důležité, že správně odhadne procentuální rozdíl.

Všechny metriky byly proměřeny na celkem třech zákaznících z oboru internetového obchodu.

Tabulka 3.1 ukazuje, v kolika procentech případů dokázaly metriky u jednotlivých zákazníků odhadnout, která z konfigurací v AB testu je zdatnější.

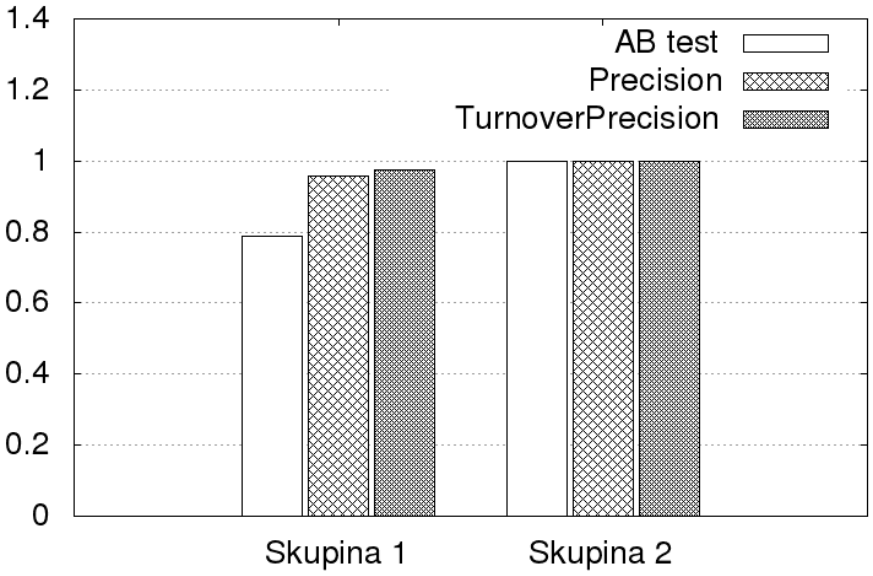
Tabulka 3.2 pak ukazuje míru odlišnosti výsledků metrik a AB testů, jinak řečeno o kolik se liší výsledky ze systému oproti zdatnostím vráceným AB testy. Toto číslo bylo vypočítáno následovně: pro každý AB test byly zdatnosti vrácené systémem znormalizovány na 1 (nejvyšší zdatnost je po přeškálování rovna 1, menší zdatnosti jsou přeškálovány na odpovídající hodnotu mezi 0 a 1). Stejným způsobem jsou znormalizovány i zdatnosti vrácené AB testy. Od znormalizované hodnoty ze systému je následně odečtena znormalizovaná hodnota odpovídající skupiny AB testu a na toto číslo je aplikována absolutní hodnota. Výsledná míra odlišnosti je pak průměrem těchto čísel přes všechny skupiny všech AB testů.

Při pohledu na tabulku je vidět, že všechny testované metriky se chovají podobně. Patrné je to především z druhé tabulky, kde pro jednotlivé zákazníky jsou míry odlišnosti téměř stejné.

Během detailního zkoumání výsledků bylo zjištěno, že ani jedna z metrik nedokáže pořádně rozlišit rozdíl mezi zdatnostmi konfigurací doporučovacího systému. V případech, kdy AB test má jednu z konfigurací výrazně úspěšnější než druhou, metrika v některých případech dokáže rozlišit, která z konfigurací je lepší, ale míra odlišnosti zdatnosti jednotlivých konfigurací není zdaleka tak velká jako u daného AB testu. Příklad jednoho z takových případů je vidět

Tabulka 3.2: Míra odlišnosti výsledků systému a AB testu

	Zákazník A	Zákazník B	Zákazník C
Precision	0,251	0,136	0,0776
TurnoverPrecision	0,252	0,136	0,0606
Recall	0,25	0,136	0,0798
TurnoverRecall	0,252	0,136	0,0597
FMeasure	0,251	0,136	0,0777
TurnoverFMeasure	0,251	0,136	0,0759
Interacted	0,25	0,136	0,0828



Obrázek 3.2: Příklad výsledků metrik *Precision* a *TurnoverPrecision* pro konkrétní AB test

na obrázku 3.2, kdy obě metriky *Precision* i *TurnoverPrecision* správně určily zdatnější skupinu, ale míra odlišnosti zdatností mezi skupinami je výrazně menší než v daném AB testu.

3.4.8 Zhodnocení

Ani jedna z testovaných metrik nedokáže u vybraných zákazníků spolehlivě odhadovat zdatnosti konfigurací doporučovacího systému. V některých případech metriky odhadnou správně zdatnější z konfigurací, ale už nedokáží postihnout správnou míru odlišnosti zdatností.

Všechny testované metriky jsou postavené na podobné myšlence a jejích variantách: zdatnost konfigurace se vypočte jako počet relevantních doporu-

čení k celkovému počtu doporučení. Z naměřených dat je vidět, že tento typ metrik nedokáže v některých případech dobře odhadovat zdatnosti konfigurací testovaného doporučovacího systému. Metriky lze k tomuto účelu použít, ale pro získání spolehlivějších výsledků je potřeba otestovat další metriky, ideálně i nějaké zcela odlišně koncipované.

Výsledky by také mohlo zlepšit detailnější zkoumání interakcí předcházejících konverzi. Zmíněnými metrikami je jako úspěch započítán i tento případ: uživatel si přidá produkt do košíku, pak si prohlíží další stránky obchodu, na nichž je mu produkt přidaný do košíku doporučen. Uživatel následně nákup dokončí a daný produkt si koupí. To by metriky neměly brát jako úspěšné doporučení, protože nákup by byl proveden i bez něj. Konverze tak nevznikla na základě doporučení, ale doporučení vzniklo na základě přidání produktu do košíku.

3.5 Měření rozdílnosti výstupu doporučovacího modelu pro různé konfigurace

V této části kapitoly bude popsána metrika, která byla do systému zavedena pro měření rozdílnosti výstupu doporučovacího modelu pro různé konfigurace.

3.5.1 Metrika RecommendationsEquality

Metrika `RecommendationsEquality` slouží k měření rozdílnosti doporučených produktů mezi dvěma různými nastaveními doporučovacího modelu. V metrice lze použít jak konektor `OriginalRecommendations`, který vrátí doporučení z reálné odpovědi na požadavek, tak konektor `RepeatedRecommendations`, jenž požadavky zopakuje se změněným nastavením. Díky tomu lze v této metrice porovnat výsledky doporučení mezi dvěma různě nastavenými modely, nebo třeba produkty reálně doporučené uživateli s doporučeními získanými odlišně nastaveným modelem.

Výsledky metriky obsahují poměr mezi počtem stejných produktů a celkovým počtem doporučených produktů. Výsledky jsou rozděleny na dvě části. V části `withoutOrder` jsou výsledky, kdy se nebere zřetel na seřazení doporučených produktů (tzn. je důleižté, zdali je produkt obsažen v obou doporučeních), a v části `withOrder` jsou výsledky, kdy je důležité i seřazení doporučení (tzn. je důležité, aby produkty byly v obou doporučení na stejné pozici). Příklad výsledků metriky `RecommendationsEquality` je vidět v ukázce 3.6. Vyšší je vždy hodnota, u které se nebere v potaz seřazení produktů.

3.5.1.1 Nastavení metriky

Metrika `RecommendationsEquality` pracuje s těmito parametry:

Zdrojový kód 3.6: Příklad výsledků metriky RecommendationsEquality

```
"ratios": [
  {
    "withoutOrder": {
      "overallRecommendations": 95932,
      "sameRecommendations": 43419,
      "ratio": 0.45260184297211
    },
    "withOrder": {
      "overallRecommendations": 95932,
      "sameRecommendations": 36968,
      "ratio": 0.38535629404161
    }
  }
]
```

- `customer` – identifikátor zákazníka, na jehož datech má být metrika spočtena
- `requestsCount` – počet požadavků na doporučovací model, které mají být použity k výpočtu
- `timestampFrom` – časové razítko omezující maximální stáří požadavku na doporučení
- `timestampTo` – časové razítko omezující minimální stáří požadavku na doporučení

3.5.1.2 Měření metriky

Doporučovací systém má předpočítané kandidáty, které uvažuje při doporučování. To umožňuje odpovědět na daný požadavek daleko rychleji. Jak přicházejí nové interakce a data o produktech a uživatelích, je potřeba tyto kandidáty v průběhu času přepočítávat.

Při každém přepočítání se kandidáti mohou změnit. Například u vydavatele obsahu je změna kandidátů častá: na stránkách se objevují neustále nové články a je potřeba doporučovat především novinky, protože to je to, co uživatelé chtějí číst. Na druhou stranu například u internetového obchodu se zboží moc nemění a i chování uživatelů je v průběhu času poměrně stálé. Kandidáti se tak v tomto případě nebudou měnit tak často. A právě tato vlastnost doporučovacího systému byla experimentálně ověřena.

Experiment probíhal tak, že pro každého zákazníka byla změřena metrika `RecommendationsEquality` postupně pro posledních 5 dní. Při každém měření došlo k omezení pouze na doporučení z daného dne (pomocí parametrů `timestampFrom` a `timestampTo`). Metrika využívala dvou konektorů: prvním

Obrázek 3.3: Výsledky metriky RecommendationsEquality pro vydavatele obsahu

byl konektor `OriginalRecommendations`, jenž vrátil produkty doporučené v reálné odpovědi na daný požadavek, druhým byl konektor `RepeatedRecommendations`, který zopakoval požadavek na aktuálním doporučovacím modelu beze změny nastavení.

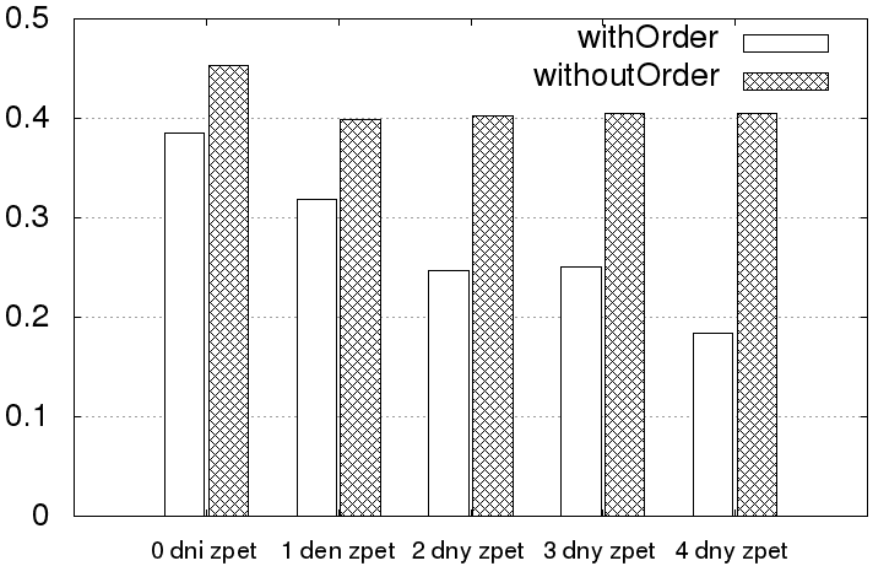
Výsledky pro vydavatele obsahu jsou na obrázku 3.3. Je vidět, že kandidáti v doporučovacím systému se v tomto případě mění rychle. I pro aktuální den je shoda doporučení neberoucí v potaz seřazení pouze na hodnotě 0.25. V dalších dnech tato hodnota dále rychle klesá.

Výsledky měření pro internetový obchod jsou na obrázku 3.4. Je vidět, že kandidáti na doporučení se v tomto případě mění mnohem méně častěji. Hodnota neberoucí v potaz seřazení je pro aktuální den téměr dvakrát vyšší než u vydavatele obsahu. Větší rozdíl je však vidět v následujících dnech, kdy hodnoty na rozdíl od prvního měřeného zákazníka klesají jen mírně.

3.5.1.3 Zhodnocení

Díky metrice `RecommendationsEquality` jsme experimentálně ověřili jednu z vlastností testovaného doporučovacího systému, konkrétně jak často jsou u různých typů zákazníků obměňováni kandidáti pro doporučení.

Tato metrika je vhodná nejen k měření rozdílnosti výstupu při různých konfiguracích doporučovacího modelu, ale osvědčila se i na měření rozdílnosti výstupu doporučovacího modelu v čase.



Obrázek 3.4: Výsledky metriky RecommendationsEquality pro internetový obchod

Závěr

Tato práce měla dva hlavní cíle. Prvním bylo navržení a naimplementování systému pro analýzu chování modelů strojového učení a jeho spuštění v cloudovém prostředí. Ten se povedlo zcela naplnit. Systém splňuje všechny definované funkční a nefunkční požadavky, včetně škálovatelnosti na neomezený počet zákazníků a možnosti připojit se na libovolný model strojového učení.

Druhým cílem bylo naprogramování metriky pro měření přesnosti doporučovacího systému a pro měření změn v doporučeních při různých konfiguracích modelu. Systém byl rozšířen i o několik API volání, která ulehčují napojení doporučovacího systému na systém pro analýzu chování modelů strojového učení, a která umožňují také napojení komponenty zajišťující automatickou optimalizaci modelů.

V práci bylo navrženo sedm metrik pro měření přesnosti výstupu doporučovacího modelu a všechny byly otestovány na datech třech internetových obchodů. Byly proměřeny i často využívané metriky *Precision*, *Recall* a *F-Measure*. Navržené metriky však nedokázaly v některých případech spolehlivě odhadovat míru odlišnosti zdatnosti různých konfigurací doporučovacího modelu.

Dále byla prezentována metrika `RecommendationsEquality`. Ta měří shodnost doporučení získaných z dvou různě nastavených modelů a byl s ní úspěšně proveden experiment na změření četnosti změny kandidátů na doporučení.

Přestože se do systému povedlo naimplementovat všechny potřebné funkcionality (a i některé navíc), stále zůstává spousta užitečných vlastností, které se do něj mohou v budoucnu doplnit. Například možnost pustit instanci Evaluatoru jen pro určitý typ testů by se mohla hodit především při vyvažování zátěže na jednotlivé typy modelů strojového učení.

Díky této práci je systém napojen na doporučovací model. Kromě toho již funguje i napojení na predikční model. Do budoucna však zůstává prostor doplnit do systému konektory na řadu dalších modelů.

Co se týče metrik na měření přesnosti doporučovacího modelu, žádná z navržených nefungovala na testovacích datech spolehlivě. Do budoucna se tedy

nabízí možnost vyzkoušet nové metriky, které by mohly odhadovat zdatnost doporučovacích modelů úspěšněji. Šlo by například otestovat metriky, které zvýhodňují nejen konverzní interakce, ale i tzv. akce uživatele (zhlédnutí detailu produktu, interakce typu *líbí se* apod.).

Literatura

[1] Stephen, M.: *Machine Learning: An Algorithmic Perspective*. Taylor & Francis, 2009, ISBN 9781420067194.

[2] Dutton, D. M.; Conroy, G. V.: A review of machine learning. *The Knowledge Engineering Review*, ročník 12, 1997: s. 341–367.

[3] Kotsiantis, S. B.: Supervised machine learning: A review of classification techniques. *Frontiers in Artificial Intelligence and Applications*, ročník 160, 2007: s. 3–24.

[4] Dayan, P.: Unsupervised Learning. Citováno 2017-04-03. Dostupné z: `http://www.gatsby.ucl.ac.uk/~dayan/papers/dun99b.pdf`

[5] Ricci, F.; Rokach, L.; Shapira, B.: Introduction to Recommender Systems Handbook. kapitola Chapter 1, 2011, ISBN 978-0-387-85820-3. Dostupné z: `http://www.inf.unibz.it/~ricci/papers/intro-rec-sys-handbook.pdf`

[6] Jones, M. T.: Recommender systems, Part 1: Introduction to approaches and algorithms. 2013, citováno 2017-04-03. Dostupné z: `https://www.ibm.com/developerworks/library/os-recommender1/os-recommender1-pdf.pdf`

[7] Adomavicius, G.; Tuzhilin, A.: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, ročník 17, 2005: str. 734–749.

[8] Balabanovic, M.; Shoham, Y.: Fab: Content-Based, Collaborative Recommendation. *ACM*, ročník 40, 1997: s. 66–72.

[9] Schafer, J. B.; Konstan, J. A.; Riedl, J.: E-Commerce Recommendation Applications. *Data Mining and Knowledge Discovery*, ročník 5, 2001: s. 115–153.

[10] Ulf, E.: Functional requirements vs non functional requirements. 2012, citováno 2017-04-15. Dostupné z: http://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

[11] Cosley, D.; Lawrence, S.; Pennock, M. D.: REFEREE: an open framework for practical testing of recommender systems using ResearchIndex. *VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases*, 8 2002: s. 35–46.

[12] Murphy, C.; Kaiser, G. E.; Arias, M.: A Framework for Quality Assurance of Machine Learning Applications. *Columbia University Computer Science Technical Reports*, 2006.

[13] Paras, C.: The Ultimate Guide To A/B Testing. 2010, citováno 2017-04-08. Dostupné z: https://www.smashingmagazine.com/2010/06/the-ultimate-guide-to-a-b-testing/

[14] Chapman, C.: A Beginner's Guide To A/B Testing: An Introduction. 2011, citováno 2017-04-08. Dostupné z: https://blog.kissmetrics.com/ab-testing-introduction/

[15] Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; aj.: Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, ročník 22, 1 2004: s. 5–53.

[16] Olmo, F. H.; Gaudioso, E.: Evaluation of recommender systems: A new approach. *Expert Systems with Applications*, ročník 35, 2008: str. 790–804.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

REST Representational state transfer

JSON JavaScript Object Notation

HMAC Keyed-hash Message Authentication Code

HTTPS Hypertext Transfer Protocol Secure

URL Uniform Resource Locator

ACK Acknowledgement

PHP PHP: Hypertext Preprocessor

MVC Model-view-controller

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS