Návrh aplikace pro analýzu velkých dat s využitím algoritmů strojového učení

Ing. Martin Burdík

Diplomová práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta managementu a ekonomiky

Univerzita Tomáše Bati ve Zlíně
Fakulta managementu a ekonomiky
Ústav průmyslového inženýrství a informačních systémů
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Ing. Martin Burdík**
Osobní číslo:	**M15498**
Studijní program:	**N6209 Systémové inženýrství a informatika**
Studijní obor:	**Průmyslové inženýrství**
Forma studia:	**kombinovaná**
Téma práce:	**Návrh aplikace pro analýzu velkých dat s využitím algoritmů strojového učení**

Zásady pro vypracování:

Definujte cíle práce a použité metody zpracování práce.

I. Teoretická část

- **V systematickém přehledu prezentujte poznatky z oblasti zpracování velkých dat a strojového učení.**

II. Praktická část

- **Navrhněte architekturu aplikace pro zpracování velkých dat s využitím algoritmů strojového učení.**
- **Navrhněte serverovou infrastrukturu pro provozování aplikace.**
- **Zpracujte podnikatelský plán pro vývoj aplikace.**

Závěr

Rozsah diplomové práce: cca 70 stran

Rozsah příloh:

Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

Data science and big data analytics: discovering, analyzing, visualizing and presenting data. Indianapolis: Wiley, 2015, 410 stran. ISBN 978-1-118-87613-8.
HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. Big Data a NoSQL databáze. Praha: Grada, 2015, 281 stran. Profesional. ISBN 978-80-247-5466-6.
JANÍČEK, Přemysl. Systémové pojetí vybraných oborů pro techniky: hledání souvislostí : učební texty. Brno: Akademické nakladatelství CERM, 2007, 2 sv. ISBN 978-80-7204-554-9.
Mike, Barlow. Real-Time Big Data Analytics: Emerging Architecture. Sebastopol: O'Reilly Media, 2013, 32 stran. ISBN 978-1-449-36421-2.
SUMMERFIELD, Mark. Python 3: výukový kurz. Brno: Computer Press, 2010, 584 stran. ISBN 978-80-251-2737-7.

Vedoucí diplomové práce: **Ing. Martin Kovářík, Ph.D.**
Ústav statistiky a kvantitativních metod

Datum zadání diplomové práce: **15. prosince 2016**

Termín odevzdání diplomové práce: **18. dubna 2017**

Ve Zlíně dne 15. prosince 2016

L.S.

doc. Ing. David Tuček, Ph.D.
děkan

prof. Ing. Felicita Chromjaková, PhD.
ředitel ústavu

PROHLÁŠENÍ AUTORA
BAKALÁŘSKÉ/DIPLOMOVÉ PRÁCE

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen na elektronickém nosiči v příruční knihovně Fakulty managementu a ekonomiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

1. že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
2. že odevzdaná verze diplomové/bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

Jméno a příjmení: Martin Bouda

.....
podpis diplomanta

ABSTRAKT

Velká data jsou data, které není možné analyzovat konvenčním přístupem na jednom zařízení. Jednou z možností je využít distribuovaného zpracování pro rozložení zátěže mezi více zařízení a data zpracovat s využitím strojového učení. Tímto přístupem je možné z velkého množství nestrukturovaných dat získat cenné znalosti. Tyto přístupy jsou popsány v teoretické částí práce. Mimo jiné se ukázalo, že distribuované zpracování má kromě výhod také určité nevýhody. Tyto nevýhody popisuje CAP teorém a je na ně potřeba myslet při návrhu aplikací. V praktické části je navržena aplikace, která na základě analýzy dat doporučuje uživateli produkty. Ukázalo se také, že programovací jazyk Python je díky svým knihovnám kvalitní nástroj, snadno použitelný pro datové analýzy.

Klíčová slova:

Velká data, distribuované zpracování, strojové učení, analýza dat, systém pro doporučování

ABSTRACT

Big data are data, which can´t be analyzed by convention methods on a single device. One of possible solutions is to use distributed processing to divide workload across multiple devices and to process data using machine learning.

With this approach it is possible to gain valuable knowledge from a large number of unstructured data. These approaches are described in the theoretical part of the thesis. Among other things, it was shown that the distributed processing has advantages as well as certain disadvantages. These disadvantages are described by CAP theorem, and it is needed to think about them while designing applications. In the practical part of this thesis, the application is designed for product recommendation based on the data analysis. It was also shown that the programming language Python is a quality tool that showed a good performance and is easy to use for data analysis.

Keywords:

Big data, distributed computing, machine learning, data analysis, recommendation system

Na tomto místě bych rád poděkoval Ing. Martinovi Kovaříkovi, PhD za vedení této práce. Díky jeho znalostem a zkušenostem z oblasti strojového učení a ochotě podělit se o ně mi po odborné stránce poskytl cenné rady. Jeho nadšení pro řešení úloh a přátelský přístup mi v průběhu práce dodávaly motivaci.

Rád bych dále poděkoval svojí rodině. Moji ženě Monice, dceři Valerii a synovi Arturovi děkuji za podporu při studiu, bez které by tato práce nemohla vzniknout a za pozitivní energii, kterou mi společný život s nimi dává.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Režisér James Cameron předpověděl v prvním díle série Terminátor na 29. července roku 2004 den, který ve filmech o terminátorech nazval soudným dnem. V tento den měl armádní počítač jménem Skynet dosáhnout takové úrovně, kdy si začal uvědomovat sám sebe. Bohužel pro filmové postavy umělá inteligence Skynetu vyhodnotila lidstvo jako hrozbu a dva dny poté se Skynet rozhodl lidstvo vyhladit. Naštěstí se scénář úspěšného filmu nenaplnil. Nebo alespoň ne úplně. S umělou inteligencí se totiž běžně setkáváme, často aniž bychom si to uvědomovali. Aplikace nám běžně doporučují produkty v internetových obchodech, hledají přátele na sociálních sítích. Banky pomocí chytrých aplikací ohodnocují bonitu klienta. Také auta, která jsou schopná sama řídit – všechny tyto a mnohé další aplikace mají společného jmenovatele, dochází ke zpracování velkého množství dat s využitím umělé inteligence.

Tato problematika je velmi aktuální. Trendy jako internet věcí (IoT – Internet of Thinks) a velká data (Big Data) skrývají velký potenciál také v podnikové praxi. V této souvislosti se často mluví o Průmyslu 4.0, nebo o čtvrté průmyslové revoluci. Koncepty čtvrté průmyslové revoluce byly představeny na veletrhu v Hannoveru v roce 2013. Základní myšlenkou je využívání kyberneticko – fyzikálních systémů a vytvoření chytré továrny. V chytré továrně jsou všechna zařízení, ale i produkty opatřeny čipy a data jsou shromažďována, chytré aplikace nad těmito daty provádí analýzy a na základě těchto analýz se automaticky realizují opatření. Jedná se o teoretický koncept, který se stává praxí.

Cílem této práce je popsat základní techniky pro ukládání velkých dat a pro distribuované zpracování těchto dat. Aby velká data bylo možné efektivně využívat k rozhodování v reálném čase, není obecně v silách člověka data zpracovat na uspořádané informace a tyto pak efektivně využít jako znalosti. Je k tomu možné a prakticky i nutné použít některou z technik umělé inteligence. V této práci bude dále věnována pozornost základním technikám strojového učení.

V teoretické části budou představeny základní principy zpracování velkých dat, ukládání velkých dat a technik strojového učení. V praktické části bude navržena aplikace využívající tyto principy pro doporučování produktů. V poslední, projektové části budou vytvořeny některé podklady pro sestavení studie proveditelnosti případně podnikatelského plánu. Společnost PRINCIPIA SOLUTIONS s.r.o. bude konzumentem této práce. Práce bude sloužit jako jeden ze základních materiálů pro přípravu nového projektu společnosti.

I. TEORETICKÁ ČÁST

1 VELKÁ DATA

Podle společnosti IBM bude v roce 2020 každý den vznikat 2,8 kvintilionu bajtů dat, na discích bude celkem uloženo 40 zettabajtů dat. Uživatelé serveru YouTube každý měsíc shlédnou 400 hodin videa. Síť Twiter má 200 miliónů uživatelů, síť Facebook 900 miliónů uživatelů aktivních každý den. Společnost Facebook pro provoz jejich sítě v roce 2012 používala 2012 serverů.

1.1 Charakteristiky velkých dat

Společnost Gartner, uznávaná výzkumná společnost z USA, definovala velká data jako: "data, jejichž velikost (**V**olume), rychlost nárůstu (**V**elocity) a různorodost (**V**ariety) neumožňují zpracování pomocí doposud známých a ověřených technologií v rozumném čase" [1]. Tyto tři vlastnosti velkých dat se označují jako **3V**.

- **Volume**: velikost přesahující možnosti jednoho serveru. Používají se desítky, často stovky serverů.
- **Velocity**: data přibývají rychle, často exponenciálně. My je potřebujeme zpracovat velmi rychle, často v reálném čase.
- **Variety**: data nejsou strukturovaná, nelze použít RDBS. Často se jedná o multimediální soubory, dokumenty JSON nebo XML.

1.2 Distribuované zpracování velkých dat

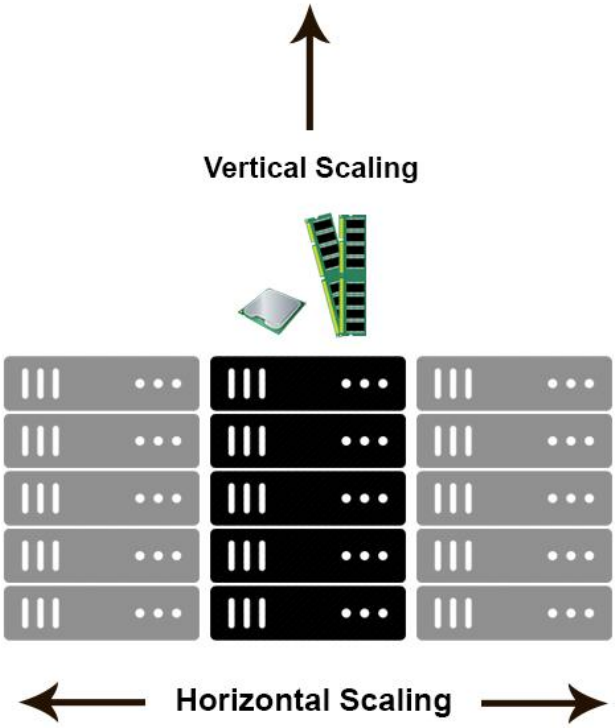
1.2.1 Škálovatelnost

Škálovatelnost je žádoucí vlastnost systému, sítě či procesu – schopnost reagovat na změnu potřeby obsluhy. Je to schopnost dynamicky zvyšovat, respektive snižovat sledované parametry jako reakci na zvýšenou, respektive sníženou poptávku. Škálovatelný systém ustojí náhlou zátěž a zároveň je hospodárný, pokud není vysoký výkon potřeba. Rozlišujeme dva základní přístupy, vertikální a horizontální škálovatelnost.

- **Vertikální škálování** (kvalitativní změna) - změna vlastností stávajícího systému
- **Horizontální škálování** (kvantitativní změna) - přidání nebo ubrání prvku systému, distribuování zátěže.

V souvislostí s fenoménem velkých dat narážíme na potřebu škálovatelnosti aplikací. Vertikální škálování je v tomto kontextu omezené stavem současného hardware. Není totiž

možné nekonečně a zároveň efektivně navyšovat výkon jednoho zařízení. V kontextu velkých dat se jako trend ukazuje horizontální škálování. Přidávání, respektive odebírání běžně dostupných zařízení není spojeno s velkými náklady, může tedy být efektivnější.



Obrázek 1: Vertikální a horizontální škálování (zdroj: [3])

1.2.2 Konzistence

Konzistentní data jsou taková data, která splňují integritní omezení. Integritní omezení jsou podmínky, které na data klademe s pohledu aplikace. Mohou to být pravidla pro hodnoty (věk osoby je kladné číslo), pravidla pro kombinaci hodnot (nástup do zaměstnání musí být dříve, než ukončení pracovního poměru). V systémech postavených na relačních modelech je často vyžadována referenční integrita (k zaměstnanci přiřadíme vedoucího pracovníka, tento vedoucí musí také existovat v tabulce zaměstnanců).

Relační databázové systémy pracují na úrovni transakcí. Transakce je sekvence logicky souvisejících operací, které převádí data z jednoho konzistentního stavu do jiného konzistentního stavu. Až po úspěšném vykonání všech operací se nový stav uloží do systému (Commit). Při neúspěchu se celá transakce odvolá (Rollback). Ukázkovým příkladem transakce je převod částky X z účtu A na účet B [2]:

1) Zkontrolujeme, že na zůstatek na účtu A je větší než X
2) Z účtu A odečteme částku X
3) Na účet B přičteme částku X

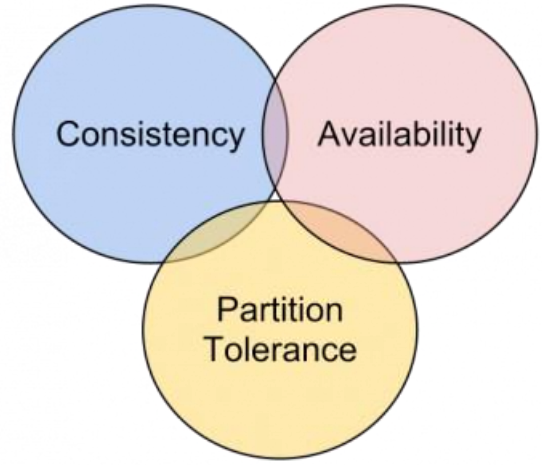
Transakční zpracování nám zajistí, že v případě výpadku při realizaci operace 3 se peníze neztratí, provede se takzvaný rollback a stav účtu A zůstane ve stavu, jako byl před začátkem transakce.

CAP teorém

Výše popsaný přístup je implementovaný prakticky v každém relačním databázovém systému. U systému pro distribuované zpracování dat už tomu tak vždy být nemusí. V ideálním případě od distribuovaného databázového systému očekáváme tři vlastnosti:

- **Konzistence** (**c**onsistency): požadujeme existenci jediné aktuální verze dat.
- **Dostupnost** (**a**vailability): požadujeme, aby všechny požadavky (čtení/zápis) byly systémem obslouženy.
- **Odolnost vůči rozpadu sítě** (**p**artition tolerance): požadujeme, aby systém fungoval i v případě, že se rozpadne na několik izolovaných částí.

Eric Brewer ukázal [3], že v distribuovaném systému jsme schopni dosáhnout vždy maximálně dvou ze tří výše uvedených vlastností současně. Toto tvrzení se nazývá CAP teorém. Z tohoto teorému vyplývá, že v prostředí distribuovaných systémů musíme ze svých požadavků slevit. Můžeme se tedy setkat s termínem občasná konzistence (označované BASE). Je proto vždy na zvážení, zda přínos distribuovaného zpracování vykompenzuje nutnost polevit na požadavcích v souladu s CAP teorémem.



Obrázek 2: CAP teorém (zdroj: [4])

1.2.3 Distribuce

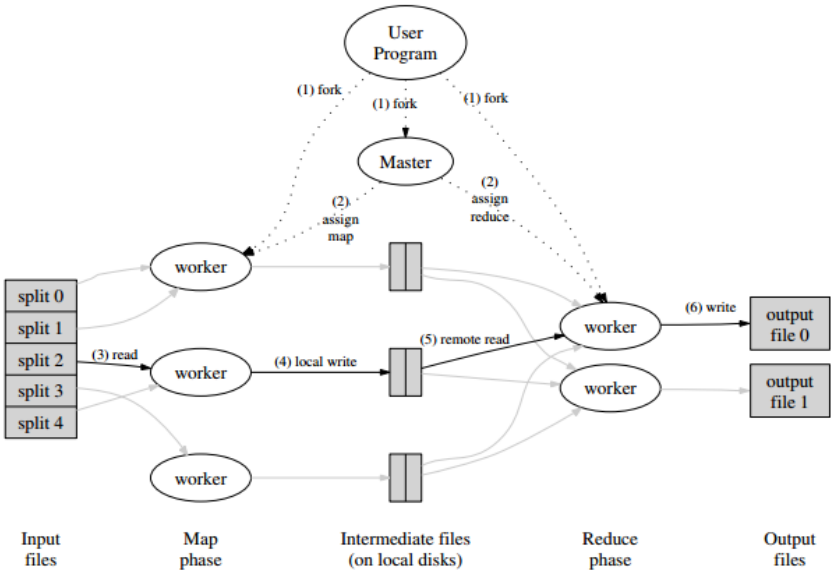
Velká data není možné zpravidla zpracovávat na jednom serveru, zátěž při zpracování je nutné distribuovat na více zařízení. Při distribuovaném zpracování využíváme dvě různé techniky, rozdělení a replikaci.

1. **Rozdělení** (sharding): data jsou rozdělena ne několik částí (shards) a ty jsou potom rozmístěny mezi jednotlivé uzly clusteru.
2. **Replikace**: stejná data jsou umístěna na několik uzlů. Tím se výrazně zvýší dostupnost dat.

1.2.4 Programovací model Map Reduce

MapReduce je programovací model používaný při vývoji aplikací určených k distribuovanému zpracování velkého množství strukturovaných a nestrukturovaných dat. Tento model představila společnost Google [2]. Společnost Google definuje MapReduce jako "jednoduché a výkonné rozhraní, které umožňuje automatickou paralelizaci a distribuci výpočtů nad rozsáhlými daty, a k němu příslušející implementaci tohoto rozhraní, jež umožňuje dosáhnout vysoký výkon s využitím velkého clusteru běžně dostupných počítačů" [4]. Nejznámější implementací tohoto modelu je projekt Hadoop od společnosti Apache. Na tento projekt potom navazuje velké množství dalších, jako Spark, ZooKeeper případně také NoSQL databáze jako například MongoDB.

Model MapReduce je postaven na dvou základních funkcích, Map a Reduce. Funkce Map slouží ke zpracování každého ze vstupních objektů a jejím výstupem je seznam dvojic <klíč, hodnota> pro každý zpracovaný objekt. Funkce Reduce slouží ke sloučení výsledků (hodnot) pro jednotlivé klíče.



Obrázek 3: MapReduce schéma (zdroj: [2])

MapReduce je programovací model, postup řešení problému podle tohoto modelu je možné popsat následovně.

- Knihovna, respektive program implementující MapReduce zajistí rozdělení vstupních dat na M dílů. Typická velikost těchto dílů je 16–64 MB.
- Program spustí své kopie na dostupných serverech. Jedna instance je speciální, jedná se o Master kopii. Ostatní kopie jsou v režimu Worker, dostávají instrukce od Masteru.
- Úloha sestává z M úloh typu Map a R úloh typu R. Master přiděluje tyto úlohy Workerům, které jsou v danou chvíli k dispozici.
- Worker, který řeší jednu z M úloh typu Map zpracuje vždy jeden díl vstupních dat. Worker extrahuje ze vstupních dat dvojce <klíč, hodnota> a ty předává do uživatelem definované funkce Map. Výstup funkce Map je uložen do paměti.
- Výstupy funkce Map uložené v paměti jsou periodicky zpracovávány speciální funkcí a ukládány na lokální disky do R sekcí. Informace o těchto sekcích jsou předány zpět Master instanci.
- Mater instance následně notifikuje volné. Workery. Jejich úkol je načíst zpracovaná data z pevných disků a ty následně zpracovat.
- Reduce Worker data nejprve seřadí a následně seskupí data se stejným klíčem.

- Následně Reduce Worker prochází přes předzpracovaná, seřazená a seskupená data a postupně je předává do uživatelem definované funkce Reduce. Tato funkce už dostává data předzpracovaná, kde klíč je jednoznačný identifikátor.
- Výstupy funkce Reduce je pro každou kopii soubor ve formátu finálního výstupu.
- Ve chvíli, kdy všechny Worker uzly ukončí svou práci Master notifikuje uživatelskou aplikaci, která MapReduce funkci zavolala.

Ukázka práce s implementací MapReduce

Pro vyzkoušení práce s implementací programovacího modelu MapReduce byly vytvořeny dva jednoduché programy v jazyce Python. Oba programy dělají stejnou věc, totiž pro každý prvek – číslo ze seznamu (seřazené množiny) čísel od 1 do 1000 udělají nesmyslný, ale relativně složitý výpočet. Smyslem tohoto výpočtu je pouze zatížit procesor počítače z demonstračních důvodů.

Program mapreduce00.py pro výpočet používá pouze 1cpu, zatímco program mapreduce01.py výpočetní zátěž distribuuje na 4cpu s využitím programovacího modelu MapReduce. Jedná se skutečně o velmi zjednodušenou demonstraci paralelního zpracování dat, nicméně podstata použita v tomto příkladu je podobná pro velké projekty.

Zdrojový kód aplikace mapreduce00.py (bez distribuovaného zpracování):

```
import time

def f(n):
# dummy function only for cpu time usage
    sum=0
    for x in range(100000):
        sum+=x*x
    return n*n*sum

if __name__ == "__main__":

    #emty list for result data
    result = []

    #time of computation begining
    t0=time.time()

    for n in range(1000):
         result.append(f(n))

    # time of computation end
    t1=time.time()

    #total time
    print("Total time:",(t1-t0))
```

Zdrojový kód aplikace mapreduce01.py (distribuované zpracování na 4cpu):

```
from multiprocessing import Pool

import time

def f(n):
    # dummy function only for cpu time usage
    sum=0
    for x in range(100000):
        sum+=x*x
    return n*n*sum

if __name__ == "__main__":

    # emty list for result data
    result = []

    # time of computation begining
    t0=time.time()

    # main run
    pool = Pool(4)
    result = pool.map(f,range(1000))
    pool.close()
    pool.join()

    # time of computation end
    t1=time.time()

    # total time
    print("Total time:",(t1-t0))
```

Výstupy programů

- mapreduce00.py (bez distribuovaného zpracování):

  Total time: 11.948451042175293

- mapreduce01.py (distribuované zpracování na 4cpu):

  Total time: 5.598907947540283

Oba programy udělaly totéž. Druhý program na to potřeboval méně než polovinu času ve srovnání s programem bez distribuovaného zpracování. Jedná se o velmi jednoduchou ukázku. Společnost Google programovací model navrhla pro zpracování skutečně velkých dat na velkém množství zařízení.

Další informace o distribuovaném zpracování dat je možné nalézt například v [5].

2 NERELAČNÍ DATABÁZOVÉ SYSTÉMY

Relační databáze jsou postaveny na principech, pro které je důležitá maximální atomizace záznamů do jednotlivých tabulek. Každá tabulka je sestavena ze sloupců (atributů) a řádků (záznamů). Na data vkládaná do databáze jsou kladeny požadavky, často se mluví o normalizaci dat a o normálních formách. Normální formy jsou sada pravidel pro návrh tabulek relačních databází a vazeb mezi těmito tabulkami. Tradiční relační databáze také implementují transakční zpracování dat (vlastnosti ACID). Díky tomu můžou tyto systémy poskytnout dokonalou konzistenci dat. Ukládání dat v relačních databázích je dále velmi efektivní při častém zápisu. Odpovědí při čtení se v SQL databázovém systému sestavují pomocí spojování tabulek (tzv. JOIN). To může být náročné. V případě složitých dotazů se sestavuje odpověď z desítek, ale klidně i stovek různých tabulek.

Databáze nerelačního typu jdou jinou cestou. Většinou nevyužívají rigidní tabulkové schéma k ukládání dat, často dokonce nevyužívají schéma žádné. V některých případech se dokonce rezignuje na dokonalou konzistenci dat. Odměnou za to je potom rychlost při čtení dat, možnost distribuovat databázi na více uzlů clusteru (vertikální škálovatelnost). O tomto bylo pojednáno v kapitole 1.2.

U relačních databázových systémů používáme k operacím s daty standardizovaný dotazovací jazyk SQL (Structured Query Language). U databázových systémů tento standart k dispozici není. Pro nerelační databázové systémy se tedy zažilo označení NoSQL databáze (No to SQL). Některé nerelační databáze umožňují zpracovávat data také dotazy v jazyce velmi podobném SQL. NoSQL se tedy častěji vykládá jako “Not only SQL”.

Relační databáze	**Nerelační databáze**
Integrita dat je zásadní.	Stačí, pokud je většina dat většinu času v pořádku.
Datový formát je konzistentní a dobře definovaný.	Datový formát nemusí být známý nebo konzistentní.
Předpokládáme dlouhodobé uložení dat.	Vzhledem k velkému množství dat často ukládáme pouze určité časové okno (poslední měsíc, poslední rok).
Aktualizace jsou časté.	Write-once/read-many, tedy vložená data

	už typicky nejsou dále modifikována (nebo alespoň ne příliš často). Obvykle data přibývají, aniž by byla modifikována. Již nepotřebné záznamy jsou pak smazány.
Předvídatelný (lineární) nárůst velikosti dat.	Nepředvídatelný (exponenciální) nárůst velikosti dat.
Nástroje pro dotazování dat umožňují přístup i ne-programátorům.	Typicky pouze programátoři píší (implementují) zpracování dat.
Probíhají pravidelné zálohy dat.	Pro řešení výpadků je využívána replikace dat.
Přístup k datům zajišťuje jediný server.	Data jsou umístěna na více serverech, přistupujeme tedy ke clusteru uzlů.

Tabulka 1: Srovnání relační a nerelační databáze (zdroj: [1])

V literatuře [1] je možné nalézt rozdělení nerelačních NoSQL databází na několik základních typů.

Databáze typu klíč – hodnota: tyto databáze jsou schopny uchovávat prakticky jakákoliv data. Klíč je jednoznačný identifikátor a hodnota je ve formátu blíže nespecifikovaných binárních dat (BLOB, Binary Large OBject). Databázový systém se nesnaží hodnotu pochopit, či s daty manipulovat. Výhodou je tedy absolutní svoboda, co se ukládaných dat týká. Představitelem tohoto typu databázového systému je například Oracle NoSQL.

Dokumentové NoSQL databáze: tyto databáze se používají pro ukládaní strukturovaných dokumentů. Tyto jsou často ve formátu JSON nebo XML, mají tedy samo-popisný charakter. Databáze tohoto typu umožňují přistupovat k dokumentům a vyhledávat v nich podle obsahu. Představitelem dokumentové NoSQL databáze je systém MongoDB.

Sloupcové NoSQL databáze: tyto systémy ukládají data do tabulek. Každý řádek tabulky může mít různé sloupce, bez vlivu na ostatní řádky. Takový řádek ve sloupcové databázi je vlastně kolekce dvojic klíč – hodnota. Klíč je vždy název sloupce. Hodnota může být klidně další tabulka. Tyto databázové systémy bývají masivně distribuované [1]. Představitelem systému tohoto typu je Cassandra od Facebooku nebo Big Tables od společnosti Google.

Grafové NoSQL databáze: tyto systémy jsou vhodné pro reprezentaci dat popisujících sítě, tedy například uzly a hrany. Struktura uložených dat je vhodná pro řešení úloh jako je hledání cesty v grafu, nejbližšího souseda. Představitelem grafové NoSQL databáze je Neo4j. Grafové databáze ve většině případů ze své podstaty nejsou vhodní pro distribuované zpracování.

V dalším textu bude pozornost věnována pouze dokumentovým. Další informace je možné nalézt v literatuře [1]

2.1 Dokumentové databáze

Z celé množiny typů nerelačních databázových systémů bude v této práci věnována pozornost dokumentovým databázím. Tento typ bude také použit v praktické části této práce. Jako konkrétní systém byl vybrán MongoDB. MongoDB je open-source projekt dokumentové databáze, disponuje řadou knihoven pro přístup z aplikací (také z Pythonu), na webu vývojáře je přehledná dokumentace.

2.1.1 Princip dokumentových databází

Klíčovým prvkem konceptu dokumentových databází je dokument. Což je datová struktura se samopopisným charakterem. To znamená, že kromě dat obsahuje i metadata, informace o datech, o významu jejich jednotlivých částí. Typickým příkladem takové struktury je formát JSON nebo XML. JSON je oblíbený formát pro výměnu informací mezi aplikacemi. U aplikací s relačními databázemi se často využívá objektově relačního mapovaní (ORM), kdy se struktury z paměti ukládají do databáze a následně se z databáze načítají zpátky do paměti. Pokud k ukládání struktury využijeme dokumentové databáze, jedná se o objektově dokumentové mapování (ODM). Takto uložené struktury mohou být přímo čteny další aplikací. Výhodou může být, že struktura dat odpovídá struktuře objektu v objektově orientovaném programování.

Dokumentová databáze MongoDB nepoužívá žádné schéma, které by dokumentům předepisovalo strukturu. Jednotlivé dokumenty jsou ukládány do kolekcí. Ukázka dokumentu je na Obrázek 4.

Obrázek 4: Dokument v MongoDB (zdroj: [5])

2.1.2 Ukázka práce s dokumentovou databází

V praktické části bude do databáze MongoDB přistupováno přímo z aplikace vytvořené v jazyce Python. K tomuto je od týmu vývojářů MongoDB dostupná knihovna PyMongo, která přístup zrealizuje. S daty se dále pracuje přímo v kódu použitím dostupných metod PyMongo knihovny. V této kapitole budou pro ukázku použity příkazy, které konzumuje přímo konzole MongoDB při přímém přístupu.

Budou předvedeny základní operace, které se často označují jako CRUD operace (**C**reate, **R**ead, **U**pdate a **D**elete).

Každý z příkazů je sestaven ze tří identifikátorů. První označuje databázi ("db"), druhý označuje kolekci ("users"). Třetí část je název některé z CRUD operací.

Vytvoření dokumentu (create):

Dokument vytvoříme metodou insert(document), která jako parametr příjme konkrétní dokument ve formátu JSON.

Obrázek 5: Vytvoření dokumentu v MongoDB (zdroj: [5])

Metoda má jeden povinný parametr, "document" je dokument ve formátu JSON. Další dva parametry nejsou povinné.

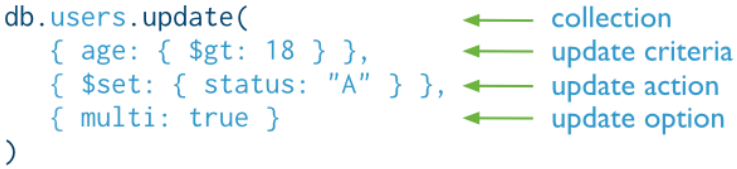
Přečtení dokumentu (read):

Dokument přečteme metodou find(query, projection). Oba parametry metody jsou volitelné, metoda bez parametrů vrátí všechny dokumenty kolekce.

Obrázek 6: Přečtení dokumentu v MongoDB (zdroj: [5])

Úprava dokumentu (update):

Pro úpravu dokumentu, který už je uložen v kolekci použijeme metodu update("query", "document",.,.,.). Tato metoda přijme pět parametrů, z toho dva jsou povinné a tři volitelné. První povinný je "query", což je podmínka pro identifikaci dokumentu. Druhý povinný parametr je "document", což je dokument ve formátu JSON, kterým bude původní nahrazen.



Obrázek 7: Úprava dokumentu v MongoDB (zdroj: [5])

Smazání dokumentu (delete):

Pro smazání dokumentu z kolekce použijeme metodu remove("query"), Jediným povinným parametrem je "query", což je podmínka pro identifikaci dokumentu.

Obrázek 8: Úprava dokumentu v MongoDB (zdroj: [5])

3 STROJOVÉ UČENÍ

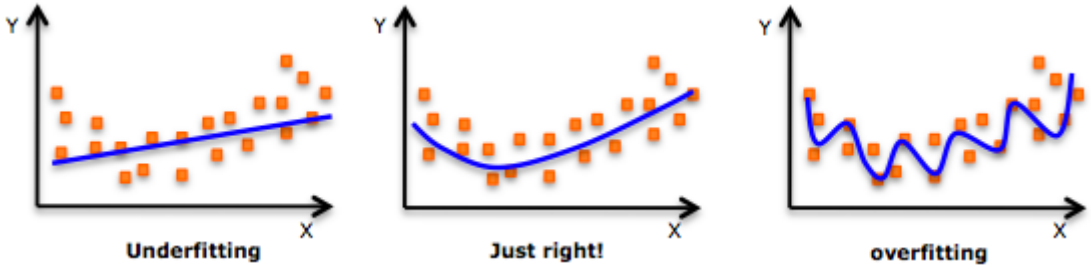
Strojové učení je oblast umělé inteligence, která využívá kombinaci statistiky, pravděpodobnosti a optimalizačních technik a která umožňuje počítačům učit se z dat a identifikovat obtížně odhalitelné vzory ve velkých a nestrukturovaných datech. Se strojovým učením se v současnosti setkáváme na řadě míst, často aniž bychom si to uvědomovali.

Několik příkladů použití strojového učení:

- Těžení znalostí z databází lékařských záznamů.
- Zpracování dat o chování uživatele na internetu (web-click data).
- Programy pro autonomní řízení vozidel.
- Rozpoznávání ručně psaného textu či mluveného slova.
- Systémy pro doporučování (používá například Netflix nebo Spotify).

Strojové učení je možné rozdělit na dvě základní kategorie, na strojové učení s učitelem (Supervised Learning) a strojové učení bez učitele (Unsupervised learning). V některých zdrojích je uvedena ještě třetí kategorie, takzvané strojové učení posilováním (Reinforcement lerning). První dvě kategorie budou představeny v následujících kapitolách. Třetí kategorii v této práci nebude věnována pozornost, informace je možné nalézt například v [7],[9].

Ve všech případech, algoritmus strojového učení se učí z trénovací sady dat. Vedle trénovací sady používáme testovací sadu dat pro validaci modelu. Data z testovací sady musí být odlišná od dat z trénovací sady. Jedině tak můžeme na základě testování hodnotit vlastnosti modelu. Důležité je, aby model dokonale nepopisoval učící sadu dat. V takovém případě by hrozilo, že model je přeučený. Sice by dokonale vystihoval učící data, nicméně při predikci neznámých výstupů pro požadované atributy by výsledky mohly být naprosto scestné. Stejně tak nežádoucím protikladem k tomuto je naopak model podučený, který je natolik zjednodušený, že nedokáže zachytit důležité fenomény. Volba vhodného modelu je stěžejním krokem při strojovém učení.



Obrázek 9: Podučený, optimální a přeučený model (zdroj: [6])

Křížová validace modelu

Pokud nemáme k dispozici sadu dat dostatečně velkou k tomu, aby bylo možné vytvořit sadu dat učících a sadu dat trénovacích, nabízí se technika, jejíž název lze volně přeložit jako křížová validace (cross-validation). Tato technika postupně použije všechny data pro učení i pro testování. Při křížové validaci rozdělíme celou sadu dat na pět stejně velkých sad.

- Sady označíme písmeny A, B, C, D, E.
- V prvním kroku sady B, C, D, E použijeme pro trénování modelu, sadu A použijeme jako testovací pro validaci.
- V další iteraci použijeme A, C, D, E pro trénování a sadu B pro validaci.
- Tímto postupem pokračujeme, dokud nepoužijeme všechny sad pro trénování i pro validaci.

Tímto postupem získáme kvalitnější model, než při trénování na jedné množině dat. **[7]**

	A	B	C	D	E
Iterace 1	Testování	Trénování	Trénování	Trénování	Trénování
Iterace 2	Trénování	Testování	Trénování	Trénování	Trénování
Iterace 3	Trénování	Trénování	Testování	Trénování	Trénování
Iterace 4	Trénování	Trénování	Trénování	Testování	Trénování
Iterace 5	Trénování	Trénování	Trénování	Trénování	Testování

Tabulka 2: křížová validace modelu (zdroj: **[7]**)

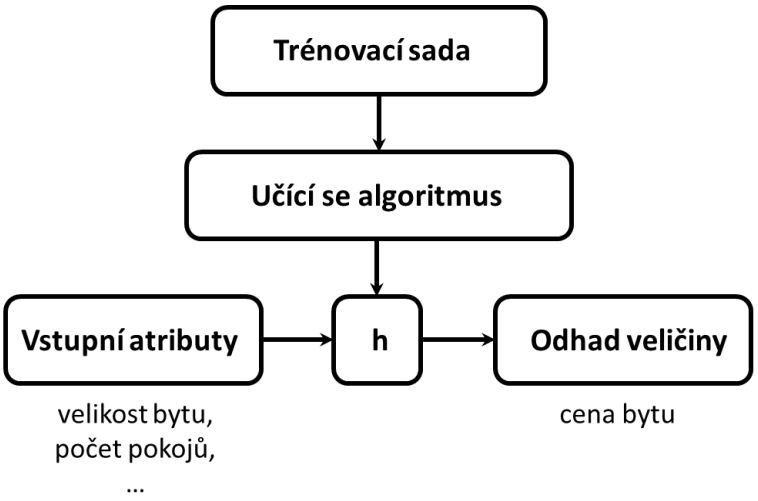
3.1 Strojové učení s učitelem

Základní myšlenkou strojového učení s učitelem je, že disponujeme množinou dat a ke každému vzorku z této množiny můžeme přiřadit správnou odpověď.

3.1.1 Regrese

Typickou úlohou strojového učení s učitelem je předpověď spojité hodnoty sledované veličiny. Pro tento účel se používá regresní analýza. Předpokládá se, že máme k dispozici historické hodnoty atributů a k nim příslušné hodnoty sledované veličiny. Tato množina představuje trénovací sadu, písmenem *m* potom značíme velikost trénovací sady. Písmenem *x* značíme vstupní proměnné, atributy a písmenem *y* hodnoty sledované veličiny. $(x^{(i)},y^{(i)})$ představuje "i-tý" vzorek z trénovací sady.

Typickým příkladem atributů může být plocha bytu nebo počet pokojů a sledovanou veličinou potom cena bytu. Funkce popisující vztah atributů a sledované veličiny je neznámá, úkolem je najít její aproximaci. Především z historických důvodů bývá tato funkce označována písmenem h, podle slova hypotéza. Na Obrázek 10 je znázorněno schéma



Obrázek 10: Schéma učení s učitelem (zdroj: vlastní)

Nejjednodušším tvarem hypotézy h je model ve tvaru lineární funkce závisející pouze na jednom atributu. V takovém případě nabývá hypotéza h tvaru [9]:

$$h_\theta(x) = \theta_0 + \theta_1 x \qquad 3.1$$

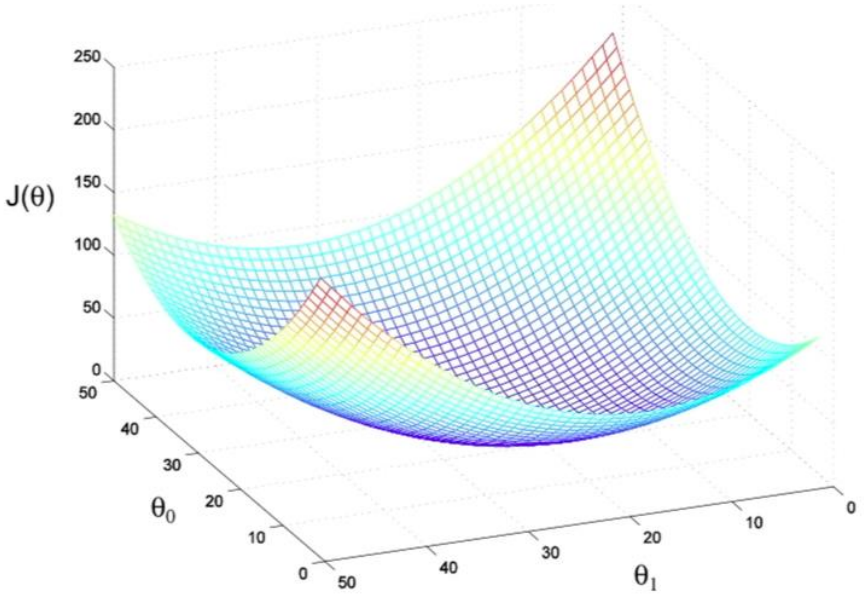
Symboly θ_0 a θ_1 ve vzorci 3.1 označují parametry modelu, *x* je vstupní atribut a $h_\theta(x)$ je odhad sledované veličiny. Hodnoty neznámých parametrů θ_i se snažíme zvolit tak, aby

model co nejlépe vyhovoval vzorkům *(x,y)* z trénovací sady dat. O tom, jak blízko jsou modelem odhadnuté hodnoty $h_\theta(x)$ ke skutečným hodnotám *y* z trénovací sady dat vypovídá ztrátová funkce. Ztrátová funkce je pro výše uvedený lineární model s jedním atributem ve tvaru:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 \qquad 3.2$$

Úkolem je tedy nalézt takovou hodnotu parametrů θ_0 a θ_1, pro kterou ztrátová funkce nabývá minimální hodnoty.

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1) \qquad 3.3$$



Obrázek 11: Graf ztrátové funkce (zdroj: [8])

Graf ztrátové funkce pro jednoduchý model ve tvaru 3.2 je obvykle konvexní [9]. Má jedno globální minimum. Pro nalezení minima ztrátové funkce, a tedy optimální hodnoty parametrů můžeme použít několik přístupů.

- Analytické řešení
- Numerické řešení

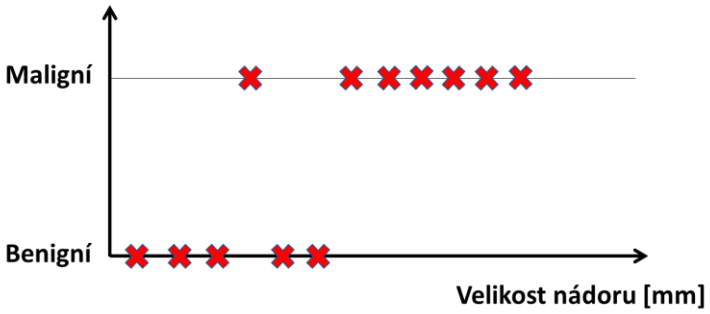
U analytického řešení nemusíme volit počáteční hodnoty parametrů a konstantu učení. Složitost algoritmu je $o(n^3)$. Není tedy vhodný pro sady dat větší než 10^6. Pro větší sady dat je vhodné hledat minimum funkce numerickou metodou, kdy zvolíme počáteční hodnoty parametrů, konstantu učení a postupnou iterací hledáme lepší hodnoty parametrů. Nevýho-

dou iteračních metod je, že v případě složitého tvaru funkce 3.2 teoreticky nemusí nalezené hodnoty být optimální. Příkladem numerické metody je gradientní algoritmus.

Výše popsaný model představuje nejjednodušší případ, kdy sledovanou veličinu aproximujeme přímkou, tedy lineární funkcí závislou pouze na jednom parametru. V praxi často potřebujeme řešit úlohy komplikovanější, kdy sledovaná hodnota závisí současně na více parametrech, případně kdy závislost na nich není lineární. Více informací o této problematice je možné nalézt v [8][9].

3.1.2 Klasifikace

V předchozí kapitole jsme se zabývali regresí, kdy jsme pomocí modelu hodnotám vstupních parametrů přiřadili výstupní hodnotu. Výstupní hodnota u regrese byla spojitou veličinou. Jako příklad byla uvedena predikce ceny bytu na základě jeho velikosti. Jiným typem úlohy je, pokud výstupní hodnota funkce není spojitá, ale nabývá pouze nebo pouze několika možných hodnot. V takovém případě se jedná o diskrétní výstupní veličinu. Úlohu tohoto typu potom nazýváme klasifikací. Pokud data dělíme podle výstupní veličiny na dvě kategorie, potom mluvíme o binární klasifikaci. Takovým příkladem může být rozdělení nádorů na zhoubné (maligní) a nezhoubné (benigní) podle velikosti Obrázek 12. Pokud budeme data třídit do více kategorií, bude se jednat o klasifikaci na více tříd. Takovým případem může být třídění mailů na pracovní, soukromé a nevyžádané.



Obrázek 12: Binární klasifikace (zdroj: vlastní)

Obrázek 10 z kapitoly o regresi platí beze změny také pro úlohy klasifikace. Rozdíly jsou ve tvaru hypotézy. Například pro model logistické regrese použijeme hypotézu (3.4), která je postavena na bázi sigmoidní funkce (3.6) [9].

$$h_\theta(x) = g(\theta^t x) \quad 3.4$$

$$z = \theta^t x \quad 3.5$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad 3.6$$

Pokud je hodnota vstupní proměnné sigmoidní funkce číslo jdoucí ke kladnému nekonečnu, výstupní hodnota funkce se blíží k 1 (3.8). Pokud je naopak vstupní proměnná číslo jdoucí k zápornému nekonečnu, hodnota sigmoidní funkce se limitně blíží nule (3.9). Pokud je hodnota vstupní proměnné 0, výstupní hodnota sigmoidní funkce je 0.5 (3.7). Poslední uvedená hodnota představuje hranici přijetí (decision boundary). Předpokládá se, že všechny hodnoty na jedné straně od hranice přijetí (y = 1) považujeme za kladnou odpověď a všechny hodnoty na druhé straně od hranice přijetí (y = 0) považujeme za zápornou odpověď.

$$z = 0 \Rightarrow g(z) = 0.5 \quad 3.7$$

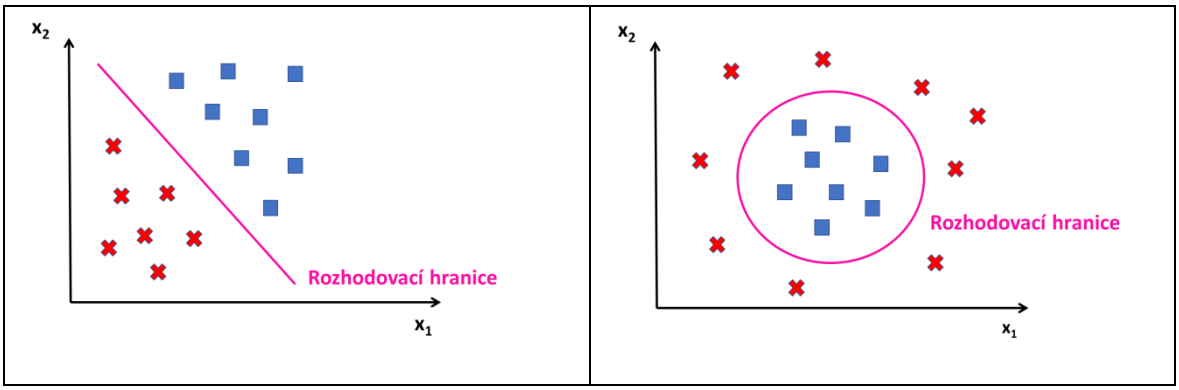
$$z \to \infty \Rightarrow g(z) \to 1 \quad 3.8$$

$$z \to -\infty \Rightarrow g(z) \to 0 \quad 3.9$$

Hranice přijetí je popsána funkcí, která může být lineární (3.10) nebo nelineární (3.11). Nelineární funkce hranice přijetí může být v podstatě libovolně komplikovaného tvaru, nicméně model potom může trpět přeučením Obrázek 9.

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \quad 3.10$$

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2) \quad 3.11$$



Obrázek 13: Lineární a nelineární rozhodovací hranice (zdroj: vlastní)

Pro nalezení optimální hodnoty funkce rozhodovací hranice, tedy nalezení optimální hodnoty parametrů θ použijeme podobně jako v předchozí kapitole ztrátovou funkci. Tato funkce opět popisuje čtverce chyb hodnot trénovacích dat a hodnot určených modelem. V literatuře [8][9] se můžeme setkat s několika způsoby zápisu ztrátové funkce, vztah (3.12) popisuje kompaktní tvar.

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta\left(x^i\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^i\right)\right)\right] \tag{3.12}$$

Pro nalezení optimální hodnoty parametrů θ podobně jako u vztahu potřebujeme minimum funkce (3.12). Komplikací je, že funkce 3.12 není konvexní (jako tomu bylo u lineární regrese), nemá tedy jen jedno minimum. Algoritmus může tedy uvíznout v lokálním minimu, a ztratit možnost nalézt globální minimum.

$$\min_\theta J(\theta) \tag{3.13}$$

Pro nalezení minima (3.13) můžeme použít některou z optimalizačních metod:

- Gradientní metodu
- Metodu konjugovaných gradientů
- BFGS
- L-BFGS

Kromě výše uvedeného rozšíření binární klasifikace na klasifikace do více tříd se běžně setkáme také s klasifikací na základě více parametrů. Úloha klasifikace zhoubnosti nádoru na základě jeho velikosti by se rozšířila na klasifikaci zhoubnosti na základě velikosti, věku pacienta, tloušťce stěny a tak podobně.

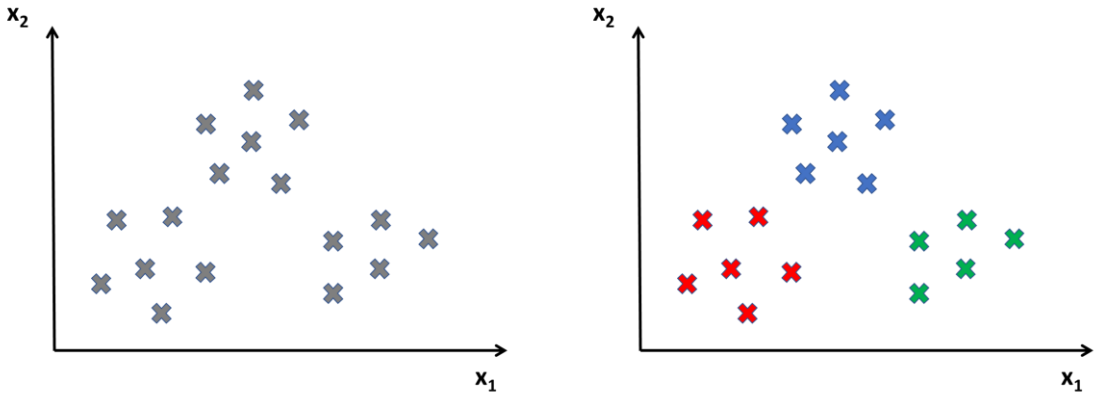
3.2 Strojové učení bez učitele

Učicí a testovací data u regrese a klasifikace měly společnou vlastnost. V obou případech jsme znali vstupní hodnoty i jím odpovídající výstupy. Pro konkrétní velikost bytu jsme znali cenu bytu, pro konkrétní velikost nádoru jsme znali jeho třídu. V takovém případě mluvíme o strojovém učení s učitelem. Strojové učení lze efektivně použít také na datech, kde nemáme dvojice vstupních dat a výstupních hodnot, ale jen vstupní data. V takovém

případě je úkolem najít zajímavé souvislosti mezi daty, data seskupit do klastrů – aniž bychom znali předem jejich třídy. V takovém případě se jedná o strojové učení bez učitele.

3.2.1 Klastrování

Klastrování je v podstatě podobné klasifikaci, s tím, že dopředu neznáme jednotlivé třídy, jejich charakteristiky, ani jejich počet.



Obrázek 14: Klastrování (zdroj: vlastní)

Praktickým příkladem vhodným pro klastrování je například segmentace trhu, analýza sociálních sítí (vytvoření shluků uživatelů se společnými zájmy) nebo seskupování příbuzných příspěvků na internetu.

K-means metoda

Oblíbenou metodou pro klastrování je k-means. Tato metoda je oblíbená díky své jednoduché implementaci a nízkým nárokům na výpočetní techniku. Její nevýhodou je nutnost stanovení počtu klastrů a počátečních hodnot reprezentantů těchto klastrů.

Budeme předpokládat, že máme trénovací sadu dat (3.14). Pro tuto sadu navrhneme počet klastrů = K. Pro každý klastr potom náhodně zvolíme reprezentanta (3.15). Pro přehlednost dále použijeme značení:

$c^{(i)}$ – index klastru, ke kterému patří vzorek *(i)*

μ_K – těžiště klastru k

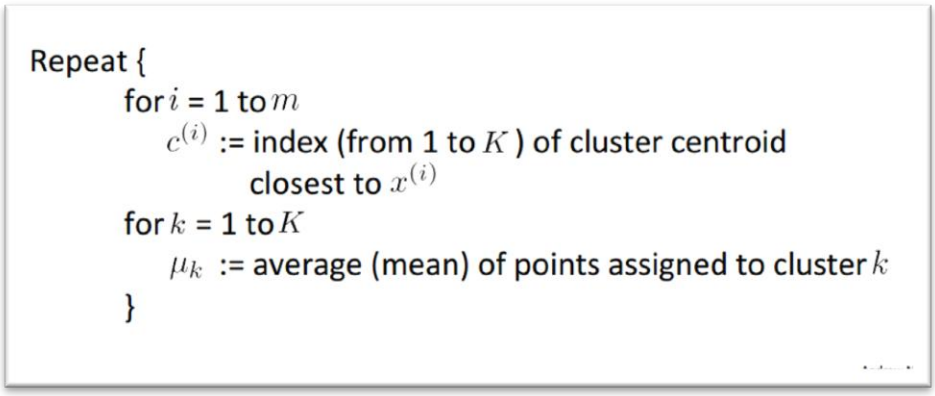
$\mu_{c^{(i)}}$ – těžiště klastru, ke kterému patří vzorek $x^{(i)}$

$$\{x^{(1)}, x^{(2)}, \dots x^{(m)}\} \qquad 3.14$$

$$\{\mu_1, \mu_2, \dots \mu_K\} \qquad 3.15$$

Algoritmus k-means potom hledá takové reprezentanty klastrů (těžiště) a distribuci vzorků v rámci jednotlivých klastrů, aby hodnota ztrátové funkce (3.16) byla minimální.

$$J\left(c^{(1)}, \dots c^{(m)}, \mu_1, \dots \mu_K\right) = \frac{1}{m}\sum_{i=1}^{m} \left|\left|x^{(i)} - \mu_{C^{(i)}}\right|\right|^2 \qquad 3.16$$



Obrázek 15: Algoritmus k-means (zdroj: [9])

Algoritmus k-means pracuje v cyklu (Obrázek 15), kdy v rámci každé iterace proběhnou dva vnořené cykly. V prvním vnořeném cyklu proběhne redistribuce vzorků v rámci klastrů podle metrické vzdálenosti těžiště klastru a vzorku. Vzorek je přiřazen ke klastru, k jehož těžišti má nejblíže. Ve druhém vnořeném cyklu se aktualizují reprezentanti klastrů. Hodnota se spočítá jako průměrná hodnota ze všech vzorků zařazených do klastru (3.17). Celý vnější cyklus se opakuje, dokud nezůstanou stejné hodnoty reprezentantů tříd ve dvou po sobě jdoucích iteracích. Hodnota r ve vzorci (3.17) představuje počet vzorků zařazených do klastru k.

$$\mu_K = \frac{1}{r}(x^{(a)} + x^{(b)} + \dots + x^{(r)}) \qquad 3.17$$

Algoritmus k-means má dvě zásadní nevýhody [9].

- Pří minimalizaci ztrátové funkce (3.16) může algoritmus uvíznout v lokálním minimu.
- Může být obtížné stanovit počet klastrů.

Obecně tyto nevýhody lze eliminovat “hrubou silou“. Optimalizaci opakujeme několikrát pro různé rozmístění reprezentantů klastrů a pro různé klastry. Vybereme pak konfiguraci s absolutně nejnižší hodnotou ztrátové funkce.

II. PRAKTICKÁ ČÁST

4 NÁVRH APLIKACE

V praktické části práce bude navržena aplikace pro doporučování produktů na základě uživatelského hodnocení. Aby bylo možné aplikaci efektivně testovat případně prezentovat, bude její součástí také demo klientské aplikace. Klientská aplikace zjistí od uživatele hodnocení několika produktů a informace předá logické vrstvě, výpočetní komponentě. Výpočetní komponenta na základě hodnocení uživatele a na algoritmu naučeného z trénovací sady dat odhadne hodnocení dalších produktů. Množinu produktů s nejvyšším odhadnutým hodnocením vrátí výpočetní komponenta přes server zpátky klientské aplikaci, která doporučené produkty zobrazí uživateli. Jakkoliv jsou všechny komponenty důležité, předmětem této práce a pozornost bude věnována především výpočetní komponentě. Pro vývojové a prezentační účely bude použito databáze filmů Movielens [9]. Společnost GroupLens uvolnila tyto sady dat právě pro účely vývoje a testování aplikací tohoto typu. Databáze obsahují vždy sadu filmů a k nim příslušné uživatelské hodnocení. Datové sady jsou zformátované do dobře zpracovatelného formátu, data jsou také konzistentní. Tím jsou ideální pro účely využití, jako tomu bude v této práci. Pokud jsou k dispozici konzistentní a čistá trénovací data, může být více pozornosti algoritmu implementaci algoritmu. Některé zdroje s nadsázkou uvádí, že příprava a předzpracování (pre-processing) dat jsou při strojovém učení 90% úspěchu. Zbytek je těch zbylých 90%.

4.1 Použité technologie

Cílem této kapitoly je uvést přehled použitých technologií, jejich stručnou charakteristiku a případně odkazy na vhodné zdroje pro další informace. Společnou vlastností všech použitých technologií je licencování, které umožňuje technologie volně využívat k libovolným účelům (i komerčním).

4.1.1 Programovací jazyk Python

Python je moderní programovací jazyk. Navrhl ho v roce 1991 Guido van Rossum, kdy chtěl navrhnout jednoduchý a výkonný skriptovací jazyk. Jazyk Python a jeho knihovny jsou napsané v jazyce C, umožnuje psát kódy podle různých paradigmat. Jednoduše lze používat objektově orientované programování, vytvářet procedurální kód či používat funkcionální programování. Komunita uživatelů je velká, existuje velké množství zdrojů. Jeden z těch skutečně vydařených je [13].

Knihovna NumPy

Knihovna NumPy rozšiřuje jazyk Python a sadu matematických funkcí a o nový datový typ n-rozměrného pole (nd-array). Pomocí této knihovny je možné velmi efektivně řešit úlohy velkého rozsahu. Výpočty nad n-rozměrným polem jsou rychlé, proto je knihovna NumPy požadována také při používání knihoven pro vědecké výpočty (SciPy a Scikit-Learn). Další informace o knihovně NumPy je možné nalézt v [15].

Knihovna Pandas

Knihovna Pandas rozšiřuje jazyk Python a datové struktury, které umožňují efektivně zpracovávat velká data v jazyce Python. Práce s daty s knihovnou Pandas je v mnohém podobná práci s relačními databázemi a jazykem SQL. Nad daty lze provádět různé dotazy, výsledky filtrovat či spojovat. Další informace je možné nalézt v [15].

Knihovna SciPy

Knihovna SciPy rozšiřuje jazyk Python a velké množství funkcí pro matematické výpočty, vědecké analýzy a inženýrské aplikace. Výpočetní Python aplikace bývají často stavěny právě na knihovně SciPy, Pandas a NumPy. Informace jsou v přehledné formě dostupné na [15].

Knihovna Scikit-learn

Knihovna Scikit-lern využívá knihoven NumPy a SciPy a dodává do jazyka Python funkcionalitu pro strojové učení a těžení dat. V této knihovně jsou implementovány všechny běžně používané techniky strojového učení jako regrese, klasifikace, čí shlukování. Knihovna dále obsahuje řadu pokročilých metod pro sofistikované aplikace. Velmi užitečné informace je možné nalézt v literatuře [7], dále pak na oficiálním webu knihovny [15].

4.1.2 Programovací jazyk Java

Jazyk Java představila společnost Sun Microsystems v roce 1995. Java je objektově orientovaný programovací jazyk. Java byla navržena tak, aby pracovala na zařízeních různého typu. Od mobilů, přes jednoúčelové zařízení a osobní počítače až po velké servery. Toho je dosaženo pomocí virtuálního zařízení (Java Runtime Environment – JRE). V operačním systému je spuštěna tato virtuální platforma, která se stará o spouštění Java aplikací. V současnosti je Java jedním z nejrozšířenějších jazyků. Díky svým vlastnostem je

mmasivně využívána především pro serverové aplikace, kde se využívá rozšíření Enterprise Edition (Java EE).

Spring Framework

Spring framework je velmi populární rozšíření jazyka Java. Do jazyka Java přidává řadu nových funkcionalit. Důležité také je, že mnohé postupy dostupné v JavaEE výrazně zjednodušuje a tím snižuje cenu vývoje. Většina aplikací napsaných v Java EE používá Spring Framework. Kromě jiného, komponenta Spring Boot v sobě integruje webový server Tomcat. Vytvořená aplikace pro svůj provoz tedy nepotřebuje nic víc než operační systém s platformou Java. Společnost Pivotal na oficiálním webu [16] publikuje velmi kvalitní dokumentaci.

Thymeleaf

Thymeleaf je značkovací jazyk pro snadné generování dynamických dokumentů (web, xml) na serveru. Ve spojení z jazykem Java (Springem) umožňuje velmi efektivní práci. Zjednodušeně lze funkcionalitu popsat následovně. Je vytvořena šablona dokumentu v Thymeleaf, kde jsou speciálním výrazem označena data, které se budou generovat dynamicky. Serverová aplikace potom šablonu použije, označená data nahradí dynamickým obsahem a takto vytvořený dokument předá klientské aplikaci. Klientská aplikace dokument pouze vykreslí. Tomuto přístupu se říká generování na straně severu (Server side). Názorné ukázky práce s Thymeleaf je možné nalézt v dokumentaci ke Spring Frameworku [16].

4.1.3 Databáze MongoDB

Databáze MongoDB byla představena v teoretické části této práce. Pro přístup k databázi z programovacího jazyka Python bude použito knihovny PyMongo. Pro přístup z programovacího jazyka Java, respektive Spring Frameworku bude použito Spring Mongo Template.

4.1.4 Server

Celá aplikace poběží na jednom serveru. Společnost PRINCIPIA SOLUTIONS s.r.o. k tomuto účelu zapůjčila jeden virtuální server s operačním systémem Ubuntu server 16.04 lts

Virtuální server VPS

Jedná se o VPS server, což je virtualizovaný server. K serveru je plně administrátorský (root) přístup pomocí protokolu SSH. Parametry serveru jsou následující:

- 40 GB SSD
- 4GB RAM (plně vyhrazená)
- CPU XEON E5/E3 / Opteron 6200/6300 (2x jádro)
- konektivita 100 Mbps

Operační systém Ubuntu Server

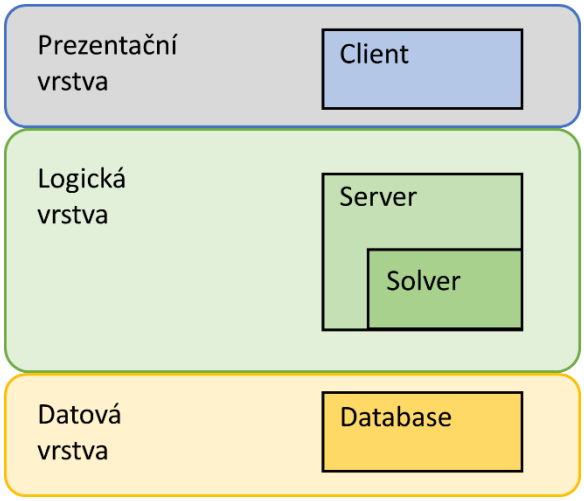
Na serveru je nainstalovaný operační systém Ubuntu Server 16.04 LTS. Je to Linuxová distribuce upravená pro servery. Vychází z distribuce Debian, a je vydávaná společností Canonical. Nové verze vychází relativně často (každých 6 měsíců). Některé verze jsou vydány s garantovanou podporou (aktualizace) na 5 let. Tyto verze jsou označeny LTS (Long Time Suport)

4.2 Architektura aplikace

Aplikace je navržena jako vícevrstvá. V terminologii návrhových vzorů vychází aplikace ze vzoru Model – View – Controller, doplněný o komponentu vycházející z návrhového vzoru Repositary. Návrh aplikací v souladu s návrhovými vzory přináší řadu výhod. Takto navržené aplikace lze dobře integrovat do větších aplikací, aplikace se také dobře zpravují a udržují.

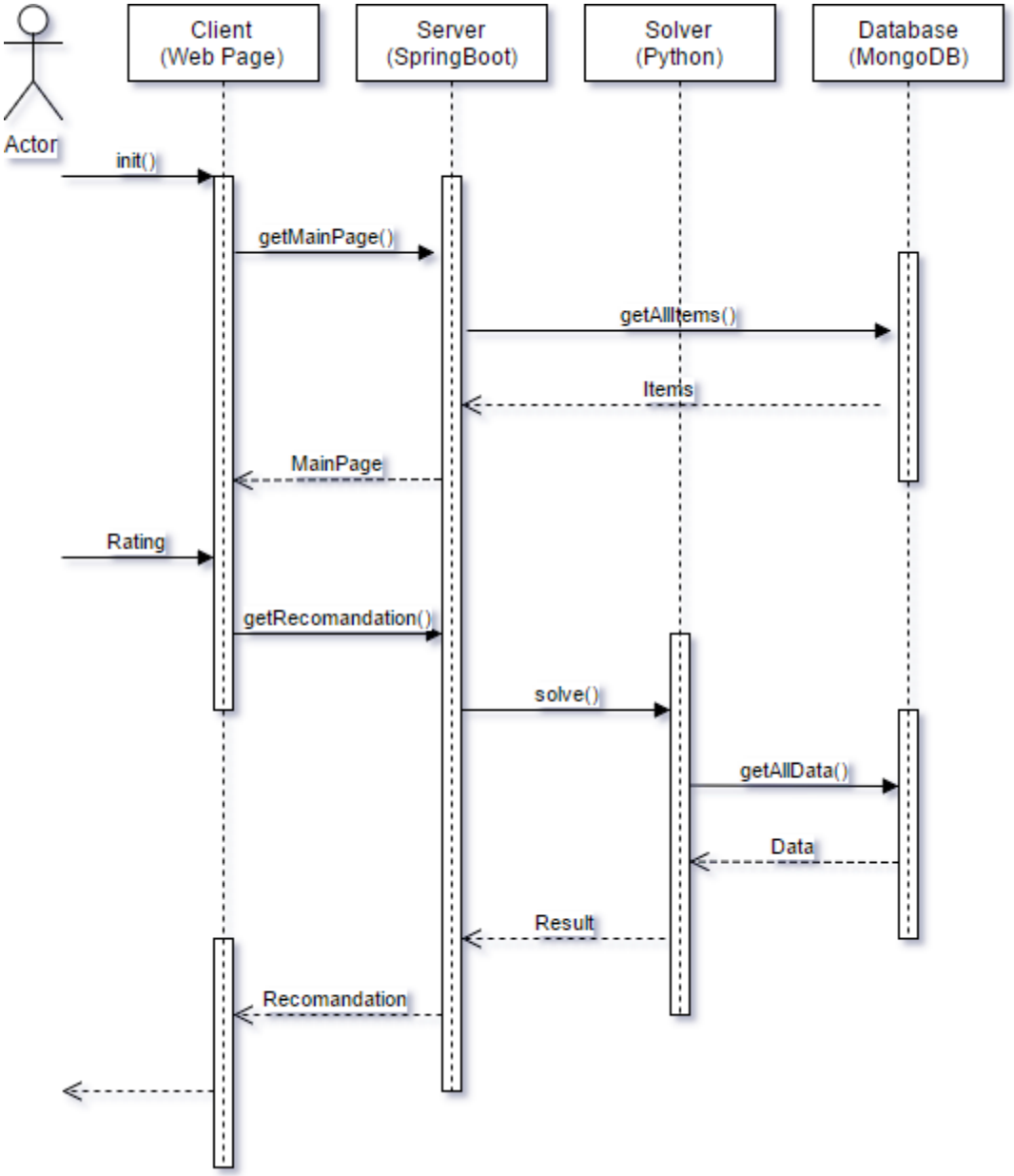
Aplikace bude na základě funkce jednotlivých komponent rozdělená na tři vrstvy:

- Prezentační vrstva je realizována klientskou aplikací – webovou stránkou (Client). V budoucnu bude možné používat jiné klientské aplikace (mobilní aplikace, webové stránky)
- Logická vrstva je zajištěna serverovou aplikací (Server), která zajišťuje komunikaci s datovou vrstvou a generuje data pro prezentační vrstvu. Logická vrstva dále s využitím výpočetní komponenty (Solver) integruje výpočetní logiku.
- Datová vrstva je uložiště dat (Database).



Obrázek 16: Architektura aplikace (zdroj: vlastní)

Z obrázku Obrázek 16 je zřejmé rozdělení aplikace no vrstvy. Realizace jednotlivých vrstev je zajištěna prostřednictvím čtyřech komponent. Jejich funkce a vzájemná interakce je popsána obrázkem Obrázek 17. Podrobnější popis bude předmětem následujících podkapitol.



Obrázek 17: Use case diagram (zdroj: vlastní)

4.2.1 Klientská komponenta (Client)

Pro potřeby této práce a pro následné prezentace aplikace bylo vytvořeno demo klientské aplikace pro práci z internetového prohlížeče. Klient je tedy realizován webovou stránkou, která běží na počítači uživatele. V současnosti má dvě základní funkce:

- Od uživatele zjistí hodnocení produktů
- Uživateli předá doporučené produkty

Klientská aplikace komunikuje se serverovou aplikací prostředním dvou rozhraní. Rozhraní pro inicializaci stránky (“/getMainPage“) je voláno HTTP metodou GET bez parametrů.

Rozhraní vrací HTML stránku s daty a nástroji pro zadání uživatelského hodnocení. Druhé rozhraní (“/getRecomandation“) je pro získání doporučených produktů. Realizuje se voláním HTTP metody GET s parametry. Parametry volení jsou dvojce hodnot id_produktu:hodnoceni. Obě rozhraní vystavuje serverová aplikace. Na takzvaném root kontextu (“/“) demo aplikace vystavuje uvítací stránku.

Webové stránky klientské aplikace jsou generovány dynamicky na serveru. Pro vykreslení jejich komponent je použito jazyka HTML, rozšířeného o technologie JavaScript a Bootstrap.

4.2.2 Serverová aplikace (Server)

Veškerý přístup do logické vrstvy aplikace je realizován přes dvě výše zmíněné rozhraní, které vystavuje serverová komponenta. Tento přístup je vhodný také s ohledem na možnosti budoucího rozšíření aplikace, například zabezpečení systému implementací technologie jako Spring Security. Pokud by se do logické vrstvy přistupovalo jiným klientem, implementace a změny na straně serveru budou relativně nenáročné.

Serverový aplikace tedy vystavuje dvě rozhraní (“/getMainPage“ a “/getRecomandation“):

Název	getMainPage
URL adresa	http://hostname/getMainPage
HTTP metoda	GET
Parametry volaní	Bez parametrů
Výstup metody	Dynamicky generovaná HTML stránka

Tabulka 3: Popis rozhraní “getMainPage“

Název	getRecomandation
URL adresa	/getRecomandation
HTTP metoda	GET
Parametry volaní	n hodnocení (id_product:hodnoceni)
Výstup metody	Dynamicky generovaná HTML stránka

Tabulka 4: Popis rozhraní “getRecomandation“

Rozhraní /getMainPage při volání načte data z databáze a vygeneruje HTML formulář pro klientskou aplikaci. Uživatel je klientskou aplikací vyzván k ohodnocení produktů. Po ohodnocení produktů je voláno druhé rozhraní, /getRecomandation. Uživatelské hodnocení z předchozího kroku je do metody předáno jako parametr metody.

Serverová aplikace konzumuje rozhraní vystavené řešičem a databázovou vrstvou.

Serverová aplikace je vytvořena technologii Java EE. Bylo použito rozšíření Spring Framework a Spring Boot, které umožňuje velmi efektivně vytvářet serverové aplikace. Jednou z výhod rozšíření Spring Boot je možnost vytvářet samostatně spustitelnou serverovou aplikaci, která ke svému provozu nepotřebuje aplikašní server. Rozšíření Spring Boot v sobě obsahuje populární webový kontejner Tomcat, který nahrazuje funkci aplikačního serveru. Pro dynamické generování webových stránek byl použito "template" technologie Thymeleaf. Thymeleaf je technologie pro dynamické generování webových stránek, která velmi dobře spolupracuje právě se Spring Frameworkem.

4.2.3 Výpočetní komponenta (Solver)

Úkolem aplikace je na základě trénovací sady dat a na základě uživatelského hodnocení několika produktů odhadnout hodnocení dalších produktů a z těchto vybrat množinu těch, u kterých je odhadnuto nejvyšší hodnocení. Získání uživatelského hodnoceno je realizováno pomocí klienta a serverové aplikace. Logická část a výpočty jsou implementovány ve výpočetní komponentě Solver.

Výpočetní komponenta je postavena na algoritmu collaborative filtering. Jedná se o strojové učení s učitelem. Je to relativně jednoduše implementovatelný algoritmus, který obecně dokáže kvalitně predikovat uživatelské hodnocení. Ve srovnání s největším konkurentem, content based algoritmem algoritmus collaborative filtering pracuje pouze s informací o uživatelském hodnocení produktů. Vůbec nebere v potaz detailní informace o produktu (například žánry filmu). Na principu informací o produktech pracuje content based algoritmus, kterému v této práci nebude věnována další pozornost. Informace je možné nalézt v [9].

Prakticky všechno v oblasti strojového učení má vždy minimálně dvě varianty. Také algoritmus collaborative filtering má dvě varianty. Jedna je postavena na porovnávání uživatelů (user based collaborative filtering) a druhá je postavena na základě porovnání produktů (item based collaborative filtering). U predikce na základě uživatele se nejprve pomocí

výpočtu určí podobnost uživatele se všemi ostatními (podobnostní matice). Potom se hodnoty z této podobnostní matice použijí jako váhové koeficienty a vypočte se pro každý produkt vážený průměr hodnocení ostatních uživatelů. Podobný princip je použit také u predikce na základě produktů. Podobnostmi matice u predikce na základě produktů představuje podobnost jednotlivých produktů. V následujícím textu bude prezentován a okomentován zdrojový kód výpočetní komponenty. Zdrojový kód je inspirován projekty [11] [12].

Na začátku programu je potřeba načíst potřebné knihovny. Pro efektivní zpracování dat a pro výpočty jsou použity knihovny NumPy, Pandas a Scikit-learn. Pro interakci s operačním systémem je použita knihovna Sys. Pro načítání dat z databáze MongoDB je použita knihovna PyMongo. Všechny uvedené s výjimkou knihovny Sys nejsou součástí standartní instalace Pythonu a je nutné je doinstalovat.

```
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import pairwise_distances
import sys
from pymongo import MongoClient
```

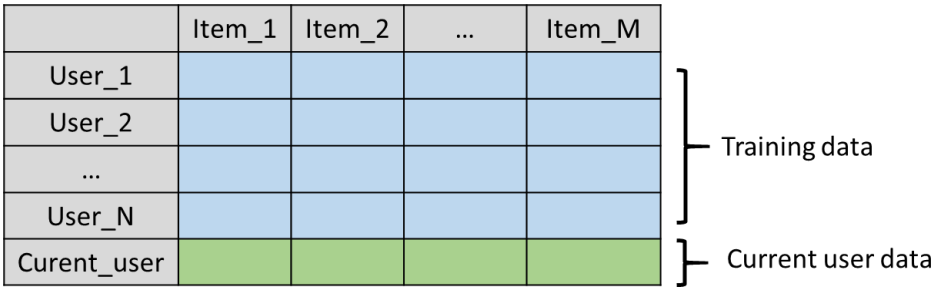
Trénovací data jsou uložena v databázi MongoDB, která je spuštěna na stejném zařízení, jako samotná aplikace. Proto byla při inicializaci databázového klienta použita adresa obsahující localhost. Následně je vybrána konkrétní databáze movielens a z této jsou načteny všechny hodnocení do vícerozměrného pole z knihovny Pandas. Podrobný popis struktury dat bude v následující kapitole, která se bude podrobně věnovat datovému uložišti.

```
#Load data from mongoDB
client = MongoClient("mongodb://localhost:27017")
db = client.movielens
df = pd.DataFrame(list(db.ratings.find()))
```

Výpočetní komponenta příjme hodnocení aktuálního uživatele jako vstupní parametry. Z pohledu operačního systému je to standartní vstup (“Standart IN“). Pokud na vstupu žádné parametry nejsou, nastaví se defaultní hodnoty. Tato možnost slouží čistě pro vývojové a testovací účely. Parametry ze standartního vstupu jsou uloženy do pole typu klíč-hodnota. V Pythonu se pro tyto účely používá objekt typu slovník (“Dictionary“)

```
#Server environment
if (len(sys.argv) > 1):
    new_user_ratings = {}
    for rating in sys.argv[1].split(","):
        new_user_ratings[int(rating.split(":")[0])] = int(rating.split(":")[1])
#Local environment - for testing only
else:
    new_user_ratings = {56:5,96:5,17:4,226:5,222:3,313:2,590:4}
```

Trénovací data jsou pro další zpracování uložena ve dvourozměrném poli (np array) z knihovny NumPy. Jedná se o matici hodnot, kde řádky představují jednotlivé uživatele a sloupce produkty, v tomto případě filmy. Takže například hodnota na pozici (2,3) představuje hodnocení produktu 3 od uživatele 2. Matice je inicializována funkcí np.zeros(), takže všechny hodnoty jsou nastaveny na 0.



Obrázek 18: Matice trénovacích/uživatelských dat (zdroj: vlastní)

Matice obsahující trénovací data je rozšířena o jeden řádek, který obsahuje hodnocení aktuálního uživatele. Počet řádků vytvořené matice je tedy počet uživatelů v trénovací sadě dat +1. Počet sloupců je počet produktů, v tomto případě filmů. Nulu u případů, kde je hodnocení známo jsou nahrazeny skutečnou hodnotou. Tam, kde hodnocení není, zůstávají nuly. To platí i pro poslední řádek, tedy aktuálního uživatele.

```
#Number of users in data set
n_users = df.user_id.unique().shape[0]
#Number of items in data set
n_items = df.item_id.unique().shape[0]

#Matrix with train data
train_data_matrix = np.zeros((n_users+1, n_items))
for row in df.itertuples(index=True, name='Pandas'):
    train_data_matrix[int(getattr(row, "user_id"))-1, int(getattr(row, "item_id"))-1]
= int(getattr(row, "rating"))
#Last row in matrix represents current user
for i in new_user_ratings.keys():
    train_data_matrix[944 - 1, i - 1] = new_user_ratings[i]
```

Pro použití collaborative filtering algoritmu je nutné vypočítat podobnostní matici. Tu lze spočítat funkcí pairwise_distances z knihovny Scikit-learn. Vypočtená matice je čtvercová

matice hodnot od 0 do 1. Hodnoty vyjadřují podobnost uživatelů na základě hodnocení produktů. Hodnota blízká 0 znamená nepodobné uživatele, hodnota blízká 1 naopak podobné uživatele.

	User_1	User_2	...	User_N	Curent_user
User_1	1	s(1,2)	...	s(1,N)	...
User_2	s(2,1)	1	...	...	...
...	...	...	...	...	...
User_N	s(N,1)	...		1	...
Curent_user	...	...	...	...	1

Obrázek 19: Podobnostní matice (zdroj: vlastní)

Funkce použitá pro výpočet podobnostní matice nabízí několik možností výpočtu, například různé typy podobností. V tomto případě byla použita kosinová podobnost (4.1).

$$sim = \cos(\varphi) = \frac{A \cdot B}{\|A\|\|B\|} \qquad 4.1$$

Vektory A a B ve vzorci (4.1) představují v našem případě hodnocení od dvou různých uživatelů. Kosinova podobnost je tedy úhel, které tyto dva vektory svírají.

Také je možné výpočet podobnostní matice realizovat v distribuovaném režimu a pro výpočet použít více procesorů.

```
#Similarity matrix
userSimilarity = pairwise_distances(train_data_matrix, metric='cosine')
```

Matice trénovacích/uživatelských dat obsahuje známé hodnocení a nuly tam, kde hodnocení neznáme. Úkolem algoritmu collaborative filtering je tyto neznámé hodnoty odhadnout. V literatuře [11], [12] so doporučuje nepracovat s absolutní hodnotou hodnocení, ale z rozdílem absolutní hodnoty a průměrného hodnocení daného uživatele. Tímto přístupem dojde k eliminaci efektu, kdy například někteří uživatelé hodnotí pouze pozitivně a někteří uživatelé využívají celou škálu bodovací stupnice. Tato funkcionalita je implementována ve funkci predictUserBased() na prvních dvou řádcích. Na posledním řádku funkce je implementována samotná funkcionalita algoritmu collaborative filtering. Pro nulové hodnoty jsou odhadnuty hodnoty jako vážené průměry od všech ostatních uživatelů. Váho této funkce je hodnota kosinové podobnosti jednotlivých uživatelů. Vše je implementováno s využitím knihovny NumPy, což umožňuje úlohu řešit v relativně krátkém čase.

```
##
## Predict ratings user based colaborative filtering
## Params:
##
def predictUserBased(ratings, similarity):
    """
    :param ratings:
    :param similarity:
    :return: prediction
    """
    #Mean rating for each user
    mean_user_rating = ratings.mean(axis=1)
    # Difference for each rating and user mean rating
    ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
    # Prediction of empty rating
    pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) /
np.array([np.abs(similarity).sum(axis=1)]).T
    return pred
dat
```

Funkce popsaná v předchozím odstavci odhadne hodnocení všech nehodnocených produktů pro všechny uživatele. Pro získání doporučení pro konkrétního uživatele je z těchto hodnot potřeba na základě identifikátoru uživatele vybrat množinu produktů, u kterých je predikováno nejlepší hodnocení.

```
##
##Find the top-k movie based on the ordered ratings
##
def getTopKIdsByUser(prediction, user_id, k=5):
    """
    :param prediction:
    :param user_id:
    :param k:
    :return: top_k_movies
    """
    return [x+1 for x in
np.argsort(prediction[user_id,np.where(train_data_matrix[user_id, :] == 0)[0]])[:-k-
1:-1]]
```

Výpočetní komponenta postupně provede všechny výše uvedené kroky, zavolá obě funkce a výstup druhé, tedy identifikátory k-nejlépe hodnocených produktů pro uživatele zapíše do standartního výstupu.

```
userPrediction = predictUserBased(train_data_matrix, userSimilarity)
sys.stdout.write(str(getTopKIdsByUser(userPrediction,944-1)))
```

4.2.4 Datové uložiště (Database)

Nedílnou komponentou aplikace je datové úložiště. Datové uložiště je realizováno serverem MongoDB spuštěném na stejném zařízení, na kterém je spuštěna serverová aplikace.

Tato konfigurace snižuje riziko bezpečnostních útoků, databázový server může mít v takovémto případě omezenou komunikaci a vůbec nereagovat na požadavky z vnější sítě. Z ohledem na zkušenosti bude jednodušší zabezpečit komunikaci mezi klientskou a serverovou aplikací popsanými v kapitolách 4.2.1 a 4.2.2 než zabezpečovat přístup do databáze z vnějšku.

Databáze MongoDB byla popsána v teoretické části této práce. Bylo uvedeno, že data ukládá ve formátu prakticky stejném s formátem JSON. Sada dat Movieens [9], která je v této práci použita je k dispozici ve formátu textového souboru csv (Comma-separated Values). Pro použití je nutné data načíst z textového souboru, převést do formátu a uložit do dokumentové databáze MongoDB. Pro tyto účely byl vytvořen zaváděcí program. Datový set obsahuje 1000000 hodnoceni na 1682 filmů od 943 různých uživatelů.

Program pro import datové sady je vytvořen podobnou technologií jako výpočetní komponenta. Je naprogramován v jazyce Python a používá knihovny Pandas a PyMongo.

```
from pymongo import MongoClient
import pandas as pd
```

Program se připojí k databázovému serveru, který je spuštěn na stejném zařízení. Z databázového serveru vybere databázi movielens. Pokud databáze tohoto názvu není, bude vytvořena automaticky

```
client = MongoClient("mongodb://localhost:27017")
db = client.movielens
```

Záznamy se budou ukládat do kolekcí movies a ratings. Pokud tyto kolekce už v databázi existují, budou odstraněny.

```
db.movies.delete_many({})
db.ratings.delete_many({})
```

Soubory s daty stažené z webu společnosti GroupLens [12] neobsahují popisky jednotlivých sloupců. Tyto je možné nalézt v souborech s informacemi. Pro pohodlnější práci s daty jsou sloupce v aplikaci pojmenovány. Díky tomu je možné se na data později odkazovat názvem sloupce. Data jsou načtena do datové struktury knihovny Pandas.

```
df_item_header = ['movie id' , 'movie title' , 'release date' , 'video release date',
              'IMDb URL' , 'unknown' , 'Action' , 'Adventure' , 'Animation' ,
              'Childrens' , 'Comedy' , 'Crime' , 'Documentary' , 'Drama' , 'Fantasy'
,
              'Film-Noir' , 'Horror' , 'Musical' , 'Mystery' , 'Romance' , 'Sci-Fi' ,
              'Thriller' , 'War' , 'Western' ]
df_item = pd.read_csv("ml-100k/u.item", sep = '|', dtype=str, encoding = "ISO-8859-
1",names=df_item_header)

df_rating_header = ['user_id' , 'item_id' , 'rating' , 'timestamp' ]
df_rating = pd.read_csv("ml-100k/u.data", delim_whitespace=True, dtype=str, encoding
= "ISO-8859-1",names=df_rating_header)
```

Načtená data je nutné konvertovat do struktury JSON. V Pythonu se často k těmto účelům používá objektu typu slovník (Dictionary). Z každého záznamu je vytvořen jeden dokument typu slovník a ten je vložen do dokumentové databáze.

```
for index, row in df_item.iterrows():
    item = {}
    item['item_id']=row['movie id']
    item['item_name'] = row['movie title']
    item['item_url'] = row['IMDb URL']
    db.movies.insert_one(item)

for index, row in df_rating.iterrows():
    rating = {}
    rating['user_id']=row['user_id']
    rating['item_id'] = row['item_id']
    rating['rating'] = row['rating']
    db.ratings.insert_one(rating)
```

Pro kontrolu, zda import dat proběhl správně, je možné vstoupit do konzole databázového serveru a použít následující příkazy (symbolem "=>" je uveden výstup volání). Očekávaný výstup je takový, že kolekce movies obsahuje 1682 záznamů a kolekce ratings obsahuje 100000 záznamů.

```
db.movies.find().count()

=>1682

db.ratings.find().count()

=>1000000
```

Jednotlivé záznamy jsou v dokumentové databázi uloženy ve formátu podobném JSON. Pro lepší představu jsou uvedeny ukázky vždy jednoho dokumentu z obou vytvořených kolekcí.

```
db.movies.findOne()
=>      {
        "_id" : ObjectId("58c45e81d76f932258a8f780"),
        "item_url" : "http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)",
        "item_id" : "1",
        "item_name" : "Toy Story (1995)"
        }

db.ratings.findOne()
=>      {
        "_id" : ObjectId("58c45e82d76f932258a8fe12"),
        "user_id" : "196",
        "item_id" : "242",
        "rating" : "3"
        }
```

Každý z dokumentů má kromě požadovaných dat k uložení ještě hodnotu označenou "_id". Jedná se o jednoznačný identifikátor dokumentu. MongoDB tuto hodnotu generuje automaticky při vkládání dokumentu.

4.3 Testování aplikace

4.3.1 Test výpočetní komponenty

Pro test výpočetní komponenty je nutné trénovací data rozdělit na dvě datové sady. Poměr rozdělení bude 75% a 25%. Větší sada dat (train_data) bude použita jako trénovací. Menší sada dat (test_data) bude použita pro samotný test. Výsledky záznamů obsažených v testovací sadě budou nejprve odhadnuty algoritmem a odhadnuté hodnoty porovnány se skutečnými.

Pro náhodné rozdělení dat na trénovací a testovací sadu bude použitá funkce z knihovny scikit-learn.

```
from sklearn import model_selection
#Splits dataset into two datasets, one for training, one for testing
train_data, test_data = model_selection.train_test_split(df, test_size=0.25)
```

Ze dvou sad je nutné vytvořit dvě dvourozměrná pole.

```
#Matrix with train data
train_data_matrix = np.zeros((n_users, n_items))
for row in train_data.itertuples(index=True, name='Pandas'):
    train_data_matrix[int(getattr(row, "user_id"))-1, int(getattr(row, "item_id"))-1]
= int(getattr(row, "rating"))
#Matrix with test data
test_data_matrix = np.zeros((n_users, n_items))
for row in test_data.itertuples(index=True, name='Pandas'):
    test_data_matrix[int(getattr(row, "user_id"))-1, int(getattr(row, "item_id"))-1]
= int(getattr(row, "rating"))
```

Chybějící data (nulové hodnoty) v poli obsahujícím trénovací sadu dat odhadne algoritmus, který je použitý ve výpočetní komponentě. Odhadnuté hodnoty spolu s trénovacími daty zapíše do pole userPrediction. Funkce rmse z tohoto pole vezme odhady hodnot, které jsou obsaženy v testovací sadě a tyto porovná. Porovnání hodnot je provedeno pomocí průměrné čtvereční chyby. Funkce rmse vrací odmocninu průměrné čtvereční chyby (4.2.).

$$rmse(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad 4.2$$

Při implementaci funkce rmse je nutné do výpočtu zahrnovat pouze ty odhadnuté hodnoty, které jsou rovněž obsaženy v testovací sadě dat.

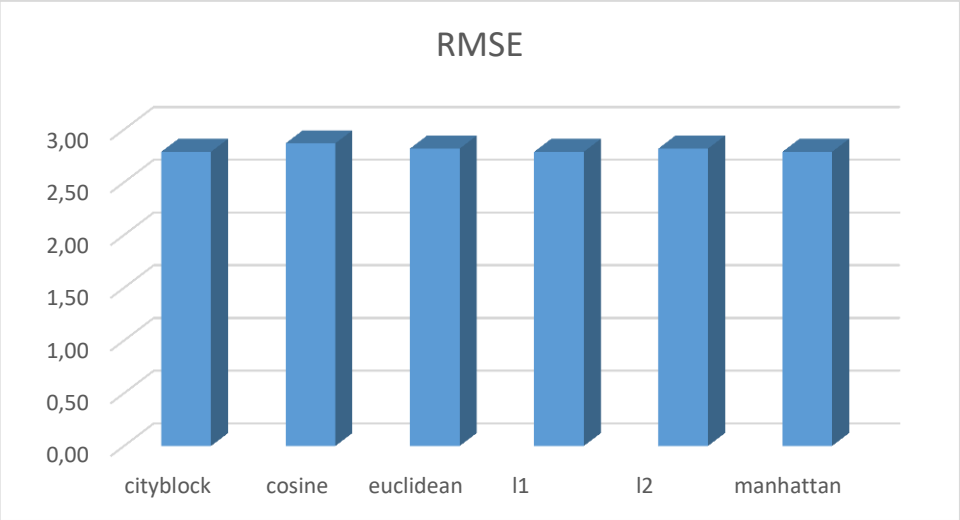
```
##
## Calculate RMSE between two datasets
##
def rmse(prediction, ground_truth):
    '''
    :param prediction:
    :param ground_truth:
    :return:
    '''
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))
```

Pro použitý algoritmus je spočtena hodnota RMSE = 2.8. Takováto hodnota je ve zdrojích [11] a [12] považována za vyhovující. Ve skutečnosti je tato hodnota vysoká. Hodnota blízká nule by znamenala shodu predikovaných hodnot s testovací sadou. V pracích zaměření na collaborative filtering [17], [18] se uvádí, že výpočet střední čtvereční chyby není vhodnou metrikou algoritmu. Obecně u úloh tohoto typu jsou data velmi řídká. Od každého uživatele známe jen několik hodnocení. To znamená, že obsahují velké množství neznámých dat, tedy hodnocení. Datová sada použitá v této práci obsahuje 6.3% ze všech možných hodnocení. Zbytek jsou prázdné místa. Tato skutečnost snižuje vypovídající hodnotu průměrné čtvereční chyby. Průměrná čtvereční chyba bude nadále využita, při vývoji pro vzájemné srovnávání algoritmů. Podobně jako [11],[12] bude hodnota 2.8 považována za dostačující. Podobně jako v [18] budou další testy kvality predikce probíhat na základě subjektivního vyhodnocení člověkem.

Knihovna Scikit-learn nabízí implementace velkého množství algoritmů výpočtu podobnostní matice. Test použitý v této kapitole je možné použít například ke srovnání těchto algoritmů a k výběru toho nejvhodnějšího. Podle dokumentace [15] je možné algoritmy rozdělit na dvě kategorie:

a) Algoritmy použitelné pro řídké matice (cosine, euclidean, l1, l2, manhattan)
b) Algoritmy nevhodné pro řídké matice (cityblock, braycurtis, canberra, chebyshev, correlation, dice, hamming, jaccar, kulsinski, mahalanobis, matching, minkowski, rogerstanimoto, russellrao, seuclidean, sokalmichener, sokalsneath, sqeuclidean, yule)

Protože je matice trénovacích dat řídká (jen cca 6% polí matice obsahuje hodnotu), budou do testu zahrnuty pouze algoritmy použitelné pro řídké matice.



Obrázek 20: Porovnání metod pro výpočet podobnostní matice (zdroj: vlastní)

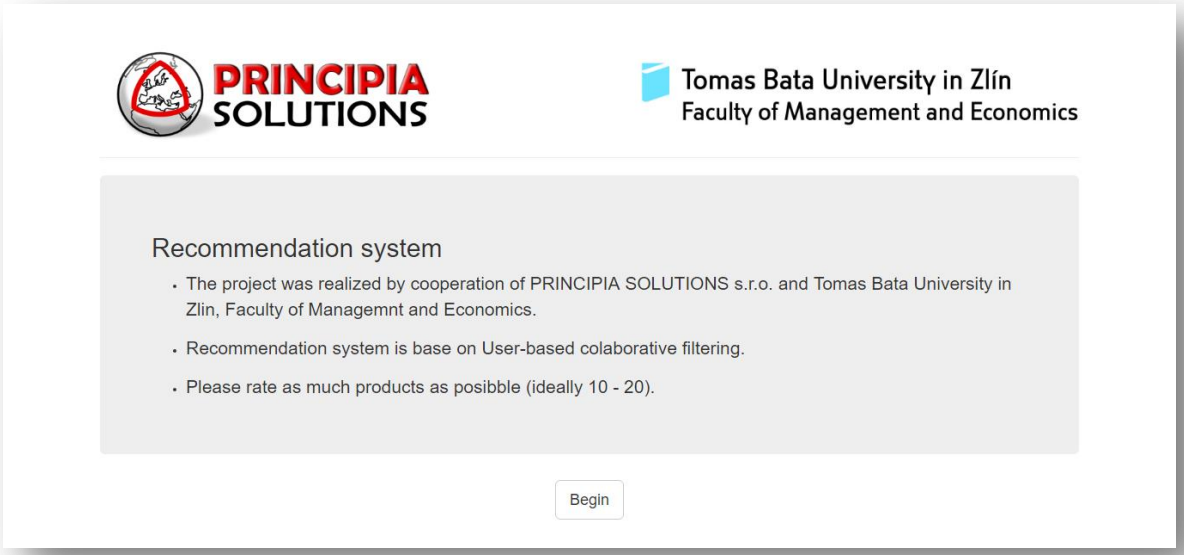
metric	rmse
cityblock	2,79
cosine	2,87
euclidean	2,82
l1	2,79
l2	2,82
manhattan	2,79

Tabulka 5: Porovnání metod pro výpočet podobnostní matice (zdroj: vlastní)

Výsledky pro všechny testované metody ukazují prakticky stejnou hodnotu průměrné čtvereční chyby. S ohledem na doporučení [11],[12] bude tedy nadále použito kosinové podobnosti. Ukazuje se, že tímto testem lze relativně snadno testovat výpočetní komponentu.

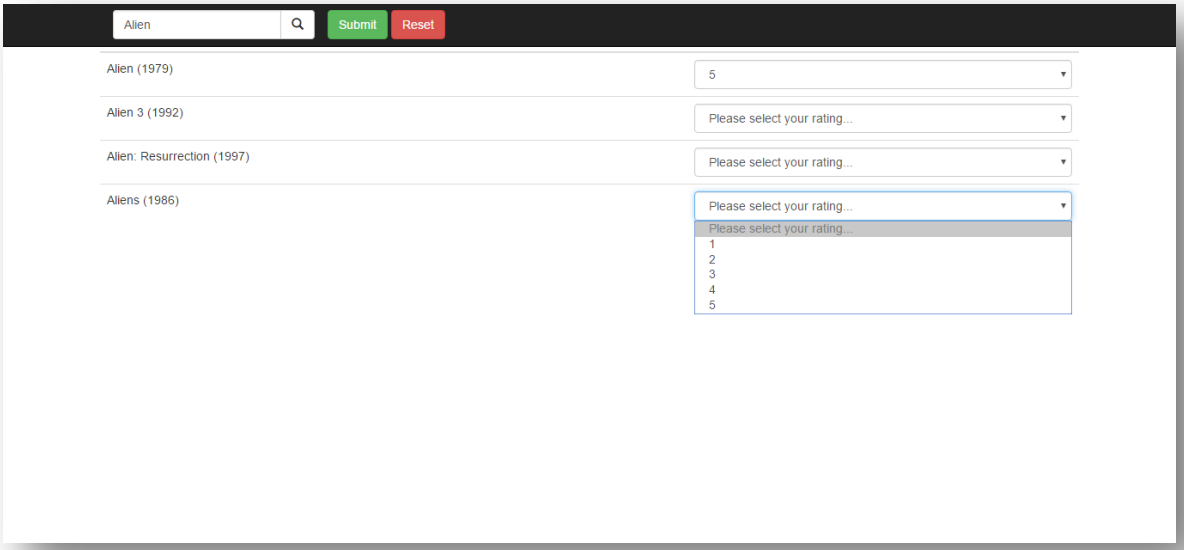
4.3.2 Test podle use-case

Podstatou následujícího testu je projít procesem doporučení v aplikaci, tak, jak je popsán v use-case diagramu. Test je zaměřen především na vzájemnou komunikaci jednotlivých komponent, funkce infrastruktury. Uživatel bude k aplikaci přistupovat přes webový prohlížeč.



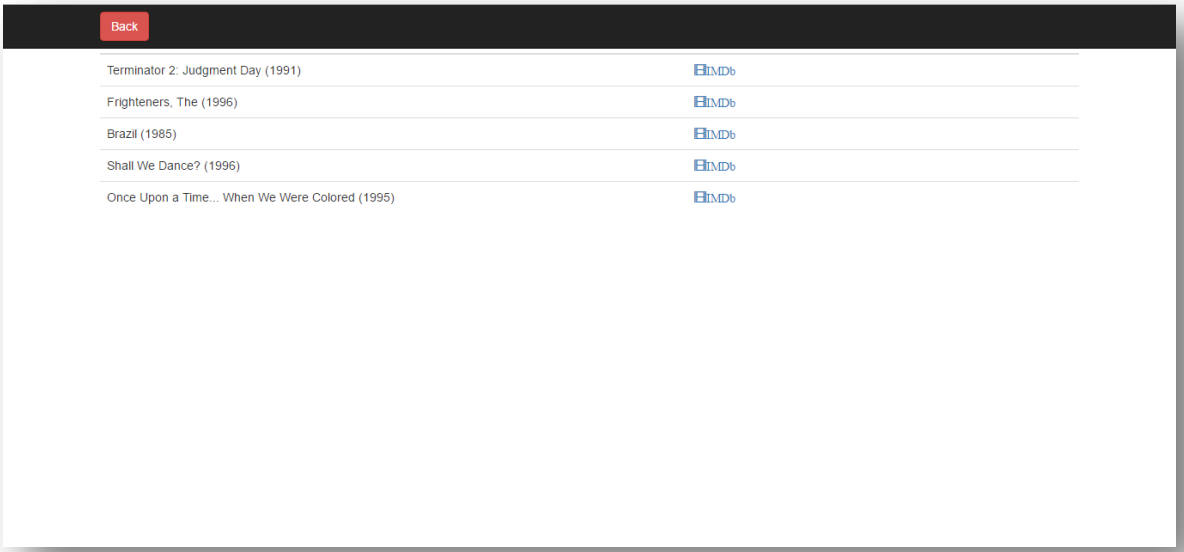
Obrázek 21: Úvodní obrazovka aplikace (zdroj: vlastní)

Na úvodní stránce jsou uživateli předány základní informace. Po stisknutí tlačítka s textem Begin je přesměrován na hlavní stránku, kde je mu umožněno zadat hodnocení jednotlivých filmů. Hodnocení se zadává pomocí rozbalovacího menu u každého filmu. Algoritmy pro doporučovaní produktů mají obecně problém doporučit produkt, ke kterému nemají dost hodnocení nebo doporučit produkt uživateli, od kterého znají jen málo hodnocení. V ideálním případě bude od uživatele zjištěno minimálně 10 hodnocení.



Obrázek 22: Obrazovka pro zjištění uživatelského hodnocení (zdroj: vlastní)

Uživatel po ohodnocení zvoleného množství filmů zvolí možnost Submit. Sytém zpracuje informace od uživatele a doporučí uživateli 5 filmů.



Obrázek 23: Obrazovka s doporučenými filmy (zdroj: vlastní)

Uživatelem zadané hodnocení:

Název filmu	Hodnocení uživatele
Alien (1979)	**4**
Aliens (1986)	**5**
Alien: Resurrection (1997)	**2**
Ace Ventura: Pet Detective (1994)	**5**
Pulp Fiction (1994)	**5**
Star Wars (1977)	**3**
Titanic (1997)	**3**
Desperado (1995)	**4**
Terminator, The (1984)	**4**
Die Hard (1988)	**5**

Tabulka 6: Uživatelský vstup (zdroj: vlastní)

Doporučené filmy:

Doporučené filmy	Hodnocení CSFD
Terminator 2: Judgment Day (1991)	**91%**
Frighteners, The (1996)	**69%**
Brazil (1985)	**81%**
Shall We Dance? (1996)	**73%**
Once Upon a Time... When We Were Colored (1995)	**-**

Tabulka 7: Výstupy aplikace + hodnocení CSFD (zdroj: vlastní)

Doporučené filmy byly doplněny o hodnocení podle serveru www.csfd.cz. Smyslem tohoto testu bylo vyzkoušet funkcionalitu podle use-case. Výsledek testu je pozitivní, aplikace pracuje správně.

5 PODNIKATELSKÝ PLÁN

Tato práce vznikla na základě zadání společnosti PRINCIPIA SOLUTIONS s.r.o. Společnost v období 2017 – 2018 realizuje projekt, jehož výstupem bude řešení pro zpracování velkých dat s využitím umělé inteligence. Cílem této práce bylo vytvořit jeden ze základních vstupů pro úvodní etapu projektu. V kapitole 4 této práce byl proveden návrh aplikace pro zpracování dat s využitím strojového učení. Jednalo se v podstatě o "proof of concept", cílem bylo především otestovat možnosti vzájemné integrace použitých technologií. Cílem této kapitoly bude příprava některých podkladů pro podrobný podnikatelský plán a studii proveditelnosti, které nebudou součástí této práce.

5.1 Představení společnosti

Společnost PRINCIPIA SILUTIONS s.r.o. vznikla zápisem do obchodního rejstříku dne 4.5.2015. Základní poslání/motto firmy je "Umožnit našim zákazníkům profitovat na pozoruhodné schopnosti matematiky řešit problémy reálného světa". Pro potřeby komunikace směrem k zákazníkům, rovněž pro potřeby řízení společnosti jsou nabízené služby rozděleny na tři kategorie:

- Programování a provoz aplikací
- Zpracování dat a matematické analýzy
- Vzdělávací kurzy a odborná školení

Obrázek 24: Logo společnosti (zdroj: [19])

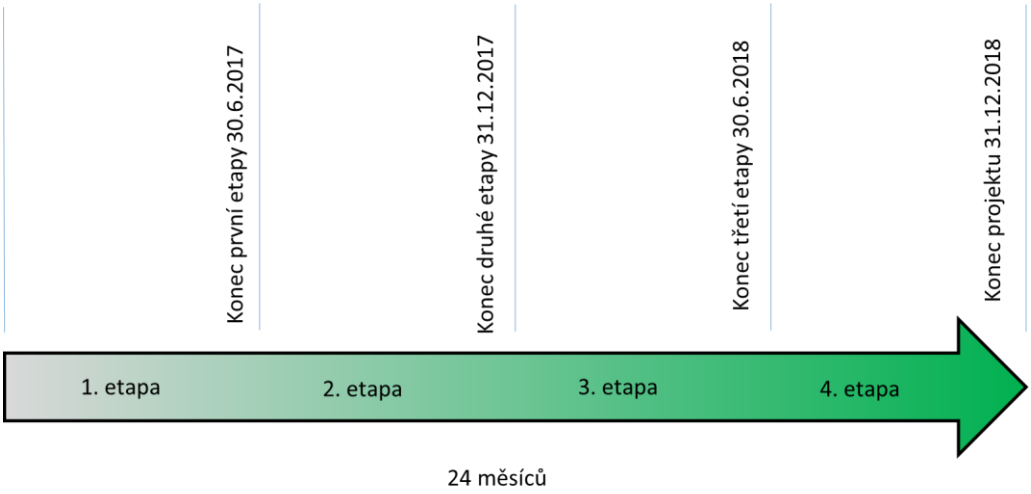
5.1.1 Základní ekonomické ukazatele

Společnost má v době vzniku této práce (jaro 2017) dvě uzavřená účetní období. Výsledek hospodaření prvního účetního období byl ztráta -109 tisíc korun. Výsledek hospodaření druhého účetního období byl zisk 89 tisíc korun. Další údaje o hospodaření si společnost nepřeje zveřejňovat.

5.1.2 Popis připravovaného projektu

Výstupem projektu bude řešení pro různé typy analýz velkých dat. Analýzy budou realizovány aplikacemi s umělou inteligencí, především s algoritmy strojového učení. Struktura řešení a použité technologie budou vycházet z návrhu provedeného v kapitole 4.

Realizace projektu je naplánována na 24 měsíců v období leden 2017 – prosinec 2018, rozdělených na 4 etapy (Obrázek 25). Podrobný popis činností realizovaných v rámci jednotlivých etap je k dispozici v interní dokumentaci společnosti [19].



Obrázek 25: Etapy projektu (zdroj: zdroj: [19])

5.1.3 Realizační tým

Pro realizaci projektu je navržen tříčlenný tým ve složení manažer projektu, analytik/tester a programátor. Předběžné vymezení zodpovědností a kompetencí jednotlivých členů týmu bude následující.

- **Manažer projektu:** manažer projektu zodpovídá za realizaci projektu, vytváří podrobný harmonogram pro jednotlivé etapy a kontroluje plnění tohoto harmonogramu. Manažer projektu bude aktivně vyhledávat možné konzumenty vytvořeného řešení, prezentovat jim řešení a získávat zpětnou vazbu – reálné potřeby a náměty zákazníků. Tyto informace budou podklady pro další vývoj.
- **Tester/analytik:** role testera a analytika bude pro potřeby projektu spojena do jednoho pracovního místa. Analytik bude připravovat analytické podklady pro vývoj na základě potřeb zákazníka. Zároveň bude zodpovědný za implementaci řešení do informačních systémů zákazníka. Pro potřeby testování bude postupně vznikat sada

testovacích scénářů. Role testera bude jednotlivé scénáře testovat, případné neúspěšné testy reportovat. K reportování bude použit vhodný nástroj (například Jira)

- **Programátor:** v prvních etapách projektu bude programátor vyvíjet nový produkt. V další fázi bude programátor zpracovávat podněty od analytika (na základě potřeb zákazníků a neúspěšných testů) a řešení přizpůsobovat.
- Předběžný návrh mzdového ohodnocení a úvazků jednotlivých členů týmů je popsán v Tabulka 8. Jedná se o předběžný návrh, v průběhu projektu bude na základě konkrétní situace upraven.

Superhrubé mzdy	1. etapa	2. etapa	3. etapa	4. etapa
Programátor	20000	40000	20000	0
Projekt manažer	12500	25000	30000	35000
Analytik/tester	0	0	30000	40000

Úvazky	1. etapa	2. etapa	3. etapa	4. etapa
Programátor	50%	100%	50%	0%
Projekt manažer	50%	100%	100%	100%
Analytik/tester	0%	0%	50%	100%

Tabulka 8: Návrh mzdového ohodnocení a úvazků

5.1.4 Vymezení výdajů spojených s realizací projektu

Předpokládá se, že projekt během realizace nebude generovat žádné příjmy. Výdaje spojené s realizací projektu budou tedy financovány ze zdrojů společnosti. Paralelně se vznikem této práce společnost podala žádost o podporu z Operačního Programu Podnikání a Inovace pro Konkurenceschopnost (OPPIK). Projekt odpovídá kritériím a zaměření Výzvy 3 projektu ICT a sdílené služby. Tento program je určen pro začínající podniky. V následujících vymezeních výdajů a především v návrhu jejích krytí bude v této práci předpokládána podpora z operačního programu. Míra podpory uvedeného programu je 60% způsobilých výdajů.

		24 měsíců				
		1. etapa	2. etapa	3. etapa	4. etapa	Projekt
Hardware	NTB grafický	60000				
	Server		60000			
	NTB standartní		20000			
	NTB standartní			20000		
Software	Vývojový sw	15000				
	Kancelářský balík	10000				
	Podpůrný nástroj	10000				
Mzdy	Programátor	120000	240000	120000	0	480000
	Projekt manažer	75000	150000	180000	210000	615000
	Analytik/tester	0	0	180000	240000	420000
	Celkový rozpočet	290000	470000	500000	450000	**1710000**
	Vlastní zdroje	116000	188000	200000	180000	**684000**
	Dotace 60%	174000	282000	300000	270000	**1026000**

Tabulka 9: Vymezení výdajů souvisejících s projektem

5.1.5 Zákazníci a plán odbytu

Společnost si nepřeje zveřejňovat informace o plánu odbytu a případných konzumentech výstupu projektu. Jednání s potenciálními zákazníky proběhla. Základní myšlenky byly postaveny na základě reálné poptávky po produktu tohoto typu.

5.1.6 SWOT analýza projektu

SWOT analýza popisuje mimo jiné slabé stránky projektu. Největší slabinou je personální zajištění – současná nezastupitelnost vývojáře. Protože společnost realizuje více projektů současně, je nutné rozšířit kapacity a v co nejkratším čase do práce na projektu zapracovat nové pracovníky. Tímto se ze slabé stránky stane příležitost ke stabilizaci a dalšímu růstu. V kontextu SWOT analýzy se jedná o W-O strategii. Společnost čelí velké poptávce po nabízených službách (ze stávajících projektů). S tímto je spojena hrozba. S nízkou nezaměstnaností a s velkou poptávkou po odbornících z IT odvětví je komplikované rozšířit kapacity a najímat nové pracovníky. Jako další hrozba je ve SWOT analýze identifikována změna potřeb zákazníka. Tuto hrozbu je možné efektivně eliminovat, silnou stránkou ve formě možnosti dynamicky reagovat na změnu potřeby zákazníka (S-T strategie). Malá firma jakou je PRINCIPIA SOLUTIONS s.r.o. touto možností disponuje.

SWOT analýza

SILNÉ STRÁNKY

1. Technologické know how
2. Dobré vztahy se zákazníky
3. Poptávka po nabízených službách
4. Schopnost dynamicky přizpůsobit produkt a reagovat na příležitosti
5. Vazby na akademickou sféru

SLABÉ STRÁNKY

1. Personální zajištění
2. Současná nezastupitelnost vývojáře
3. neznalosti v oblasti ochrany duševního vlastnictví

PŘÍLEŽITOSTI

1. Nové trendy (Big data)
2. Expanze do zahraničí
3. Růst podniku
4. Spolupráce s VŠ

HROZBY

1. Nabídka na trhu práce (nízká nezaměstnanost)
2. Změna potřeb zákazníka
3. Patenty, licence konkurence
4. Administrativa

6 DOPORUČENÍ PRO DALŠÍ PRÁCI

Předložená práce pojednává o možnostech zpracování velkých dat s využitím strojového učení. V současnosti se velmi často na místo klasického strojového učení používá neuronových sítí. Velmi oblíbenou implementací neuronových sítí je Tensor Flow od společnosti Google. Pro další práce by bylo vhodné prověřit možnosti této knihovny.

V praktické části, byl použit algoritmus Collaborative filtering. Druhou možností by bylo použít Content Based filtering a zajímavé by potom bylo srovnání těchto dvou přístupů. Zatímco použitý algoritmus Collaborative filtering vůbec neřeší obsah doporučovaných produktů, Content based filtering ano.

Komunikace mezi výpočetní komponentou a serverovou aplikací je realizována přes standartní vstup/výstup. Před dalším rozšířením produktu bude vhodné tento způsob nahradit komunikací přes rozhraní, například REST-API.

Společnost PRINCIPIA SOLUTIONS si nepřeje zveřejnit plán odbytu pro výstup projektu. V podnikatelském plánu (kapitola 5) tedy tato informace není. V další práci by nylo vhodné udělat šetření napříč různými odvětvími s cílem zjistit jaké je povědomí o možnostech zpracování velkých dat. Bylo by tak možné kvantifikovat možné přínosy pro případné zákazníky a nastavit vhodný distribuční kanál.

Oblast marketingu bude tématem další práce. Jednou z částí tohoto tématu bude návrh a realizace propagace výstupů projektu. Bude potřeba přepracovat stávající nástroje (web, sociální sítě, katalogy) s ohledem na poslání společnosti a na rozšíření portfolia služeb o zpracování velkých dat s využitím strojového učení.

ZÁVĚR

Cílem předložené práce bylo vytvořit podklad pro projekt, který hodlá společnost PRINCIPIA SOLUTIONS s.r.o. realizovat v letech 2017 – 2018. V teoretické části práce byly proto stručnou formou zdokumentovány výsledky rešerše dostupných literárních pramenů z oblasti zpracování velkých dat s využitím strojového učení.

Při práci s velkými daty se distribuované zpracování jeví na první pohled jako v podstatě nevyhnutelný přístup. Je však potřeba brát v úvahu komplikace s tím spojené. Proto byla v první kapitole věnována pozornost CAP teorému. Ten nás upozorňuje, že relativně snadno získaný výkon při distribuovaném zpracování je vždy na úkor kvality dat. Pokud i přes tento fakt uvažujeme o distribuovaném zpracování, musíme upustit od tradičních relačních databázových systémů. Alternativou v takovém případě jsou nerelační databázové systémy, přezdívané NoSQL databáze popsané ve druhé kapitole této práce. Velmi oblíbenou implementací je MongoDB. Při testech během psaní této práce se ukázala být stabilní a snadno implementovatelná, tedy vhodná pro vývoj aplikací s možností distribuovaného zpracování. Zásadním nástrojem použitelným při zpracování velkých dat je strojové učení. Proto mu byla také věnována pozornost v teoretické části. Poslední kapitola obsahuje v podstatě teoretický úvod do strojového učení. Pro práci s algoritmy je nutné chápat rozdíl mezi strojovým učením s učitelem a bez učitele, dále pak základní rozdělení na regresi, klasifikaci a shlukování. Toto bylo cílem třetí kapitoly.

Teoretické koncepty popsané v prvních třech kapitolách byly aplikovány v praktické části. Ve čtvrté kapitole byla navržena aplikace pro doporučování filmů na základě zpracování dat z veřejně dostupné databáze Movielens. Z pohledu projektu se jedná o proof of concept, jehož cílem bylo především otestovat možnosti použitých technologií. Při vývoji aplikace se osvědčil jazyk Python, který díky knihovnám Scikit-Learn, NumPy, SciPy a Pandas nabízí pod volnou licencí plnohodnotný nástroj pro datové analýzy, který je minimálně srovnatelný s komerčními produkty typu Matlab. Společnost PRINCIPIA SOLUTIONS s.r.o. bude na základě této analýzy rozhodně pro další práci na projektu primárně využívat právě Python a uvedené knihovny.

Společnost si s ohledem na dynamiku trhu nepřeje zveřejňovat informace o plánovaném odbytu, o konzumentech řešení, případně o plánované strategii marketingu. Tyto jsou součástí interní dokumentace podniku. V podnikatelském plánu v šesté kapitole byl navržen realizační tým. Na základě předpokládaných výdajů byly vymezeny způsobilé výdaje pro-

jektu. Společnost se pokusí získat podporu z projektu na podporu inovačního podnikaní (OPPIK). Tato možnost byla při návrhu způsobilých výdajů zohledněna. Vše je tedy navrženo tak, aby v případě získané podpory mohla být míra této podpory maximální. Úplný text žádosti o podporu vznikl během psaní této práce, je součástí interní dokumentace společnosti.

Součástí rešerše literárních zdrojů při psaní této práce bylo nutné projít skutečně zajímavé knihy, přednášky a články. Spousta z těchto zdrojů byla v práci přímo použita, citována. Jeden z těch nejzajímavějších však citován nebyl, autorovi se nepodařilo nalézt vhodnou souvislost. Dojde k tomu až zde. Práce bude zakončena básní ze sbírky Poezie umělého světa [20]. Přestože je jako autor uveden Jiří Materna, všechny básně ve sbírce vygeneroval počítač s umělou inteligencí.

Imaginace

„v pivu je poezie
jako jsou motýli v housenkách
popelník je pro prach
a strach
neboj se vidět a tvořit
spoutané srdce je hrob“

SEZNAM POUŽITÉ LITERATURY

[1] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. Big Data a NoSQL databáze. Praha: Grada, 2015. Profesionál. ISBN 9788024754666.

[2] DEAN, J., GHEMAWAT, S. 2004. Map Reduce: Simplified data processing on large clusters, In OSDI'04 Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, vol. 6, pp 137-149

[3] Difference between scaling horizontally and vertically for databases: questions [online]. 2017 [cit. 2017-02-19]. Dostupné z: http://stackoverflow.com/questions/11707879/difference-between-scaling-horizontally-and-vertically-for-databases

[4] Understanding the CAP theorem: news-and-events [online]. 2013 [cit. 2017-02-19]. Dostupné z: https://www.cybera.ca/news-and-events/tech-radar/understanding-the-cap-theorem/

[5] The MongoDB 3.2 Manual [online]. 2017 [cit. 2017-02-19]. Dostupné z: https://docs.mongodb.com/v3.2/

[6] Model Fit: Underfitting vs. Overfitting: AWS Documentation [online]. 2017 [cit. 2017-02-19]. Dostupné z: http://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

[7] HACKELING, Gavin. Mastering Machine Learning with scikit-learn. 1. Packt Publishing Limited, 2014. ISBN 9781783988365

[8] Standard Linear regression [online]. 2017 [cit. 2017-02-19]. Dostupné z: http://theroadchimp.com/wiki/categories/computing/

[9] Machine Learning | COURSERA: Machine Learning [online]. Stanford University, 2017 [cit. 2017-02-19]. Dostupné z: https://www.coursera.org/learn/machine-learning/home/welcome

[10] MovieLens: GroupLens Datasets [online]. 2017 [cit. 2017-03-10]. Dostupné z: https://grouplens.org/datasets/movielens/

[11] Implementing your own recommender systems in Python: Cambridge coding academy [online]. 2017 [cit. 2017-03-11]. Dostupné z:

http://online.cambridgecoding.com/notebooks/eWReNYcAfB/implementing-your-own-recommender-systems-in-python-2

[12] Data Piques: Intro to Recommender Systems: Collaborative Filtering [online]. 2015 [cit. 2017-03-11]. Dostupné z: http://blog.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering/

[13] SUMMERFIELD, Mark. Python 3: výukový kurz. Brno: Computer Press, 2010. ISBN 978-80-251-2737-7.

[14] Documentation SciPy.org [online]. 2017 [cit. 2017-03-12]. Dostupné z: http://scipy.org/docs.html

[15] Scikit-learn: Documentation of scikit-learn [online]. 2017 [cit. 2017-03-10]. Dostupné z: http://scikit-learn.org/dev/documentation.html

[16] Spring: Spring by Pivotal [online]. 2017 [cit. 2017-03-10]. Dostupné z: https://spring.io/

[17] Stackoverflow: recommendation engine metrics [online]. 2016 [cit. 2017-03-12]. Dostupné z: http://stackoverflow.com/questions/36454082/recommendation-engine-metrics

[18] Quora: How do you measure and evaluate the quality of recommendation engines? [online]. 2015 [cit. 2017-03-12]. Dostupné z: http://stackoverflow.com/questions/36454082/recommendation-engine-metrics

[19] Interní materiály společnosti PRINCIPIA SOLUTIONS s.r.o.

[20] MATERNA, Jíří. Poezie umělého světa [online]. 2016 [cit. 2017-04-03]. Dostupné z: http://www.kosmas.cz/knihy/216522/poezie-umeleho-sveta/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CSV	Comma separated values – datový formát.
IOT	Internet of thinks – internet věcí.
JRE	Java Runtime Environment – prostředí pro Java aplikace.
JSON	JavaScript Object Notation – datový formát.
ODM	Objektově dokumentové mapování.
ORM	Objektově relační mapování.
RDBMS	Relační databázový systém.
SQL	Dotazovací jazyk používaný v RDBMS
XML	Značkovací jazyk.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CLASS DIAGRAM SERVEROVÉ APLIKACE