VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2018 Peter Horečný

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

VÝPOČETNÍ ÚLOHY PRO PŘEDMĚT PARALELNÍ ZPRACOVÁNÍ DAT
COMPUTATIONAL TASKS FOR PARALLEL DATA PROCESSING COURSE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Peter Horečný

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Jan Mašek, Ph.D.

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Peter Horečný ***ID:*** 186083
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Výpočetní úlohy pro předmět Paralelní zpracování dat

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte technologie Apache Spark a Hadoop a vytvořte výpočetní úlohy vhodné pro výuku k předmětu Paralelní zpracování dat. Úlohy budou řešit základní operace s daty (načítání, transformace, agregace) a dále pak pokročilejší operace s daty (klasifikace, regrese, shlukování, četné vzory). Součástí práce bude vypracovaný návod k těmto úlohám. Výpočetní úlohy otestujte a výsledky zobrazte v grafech.

DOPORUČENÁ LITERATURA:

[1] ESTRADA, Raul. Big data smack: a guide to Apache Spark, Mesos, Akka, Cassandra, and Kafka. ISBN 9781484221747.

[2] HAGER, Georg a Gerhard WELLEIN. Introduction to high performance computing for scientists and engineers. Boca Raton: CRC Press, c2011. Chapman & Hall/CRC computational science series, 7. ISBN 978-1-4398-1-92-4.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 29.5.2018

Vedoucí práce: Ing. Jan Mašek, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

UPOZORNĚNÍ:
Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Cieľom tejto práce bolo vytvoriť laboratórne úlohy pre predmet „Paralelní zpracování dat“, pomocou ktorých sa študenti zoznámia s prácou a možnosťami technológie Apache Spark. Úlohy sa venujú práci zo základnými operáciami a predspracovaniu dát, práci s konceptami a algoritmami strojového učenia. Využitím algoritmov pre lineárnu regresiu, klasifikáciu, zhlukovanie dát a početné vzory študenti podľa vypracovaných návodov vytvoria programy, ktoré riešia vopred zadané problémy z reálneho sveta. Týmto sa zoznámia s reálnym využitím a výhodami Sparku. Ako vstupné dáta budú poskytnuté pripravené databázy českých a slovenských firiem s mnohými údajmi, ktoré musia byť v rámci prvej úlohy upravené, filtrované a usporiadané pre ďalšie spracovanie. Ďalšia vec, s ktorou sa naučia pracovať v rámci úloh je funkcionálne programovanie, keďže v návodoch nie sú programy vypracované kompletne, ale iba s postupnými nápovedami, ktoré sa v následných úlohách už neopakujú. Po absolvovaní všetkých úloh získajú ucelený prehľad o možnostiach tejto technológie.

KĽÚČOVÉ SLOVÁ

Apache Hadoop, Apache Spark, klasifikácia, lineárna regresia, paralelné spracovanie dát, početné vzory, strojové učenie, veľké objemy dát, zhlukovanie dát

ABSTRACT

The goal of this thesis was to create laboratory excercises for subject „Parallel data processing“, which will introduce options and capabilities of Apache Spark technology to the students. The excercises focus on work with basic operations and data preprocessing, work with concepts and algorithms of machine learning. By following the instructions, the students will solve real world situations problems by using algorithms for linear regression, classification, clustering and frequent patterns. This will show them the real usage and advantages of Spark. As an input data, there will be databases of czech and slovak companies with a lot of information provided, which need to be prepared, filtered and sorted for next processing in the first excercise. The students will also get known with functional programming, because the are not whole programs in excercises, but just the pieces of instructions, which are not repeated in the following excercises. They will get a comprehensive overview about possibilities of Spark by getting over all the excercices.

KEYWORDS

Apache Hadoop, Apache Spark, classification, linear regression, parallel data processing, frequent patterns, machine learning, big data, clustering

HOREČNÝ, Peter. *Výpočetní úlohy pro předmět paralelní zpracování dat*. Brno, 2018, 61 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Jan Mašek, Ph.D.

Vysázeno pomocí balíčku `thesis` verze 2.63; `http://latex.feec.vutbr.cz`

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Výpočetní úlohy pro předmět paralelní zpracování dat“ vypracoval(a) samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor(ka) uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil(a) autorské práva tretích osôb, najmä som nezasiahol(-la) nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý(-á) následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce pánovi Ing. Janu Maškovi, Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popsaný v tejto bakalárskej práci bol realizovaný v laboratóriách podporených projektom SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

Úvod **10**

1 Veľké objemy dát **11**
1.1 História 11
1.2 Definícia 12
1.3 Oblasti využitia 13
1.4 Problémy využitia 14

2 Paralelné spracovanie dát **16**
2.1 Princíp 16
2.2 MapReduce 17
2.3 Základné operácie 18
2.4 Strojové učenie 20

3 Prostredie Apache Spark **25**
3.1 Apache Hadoop a Apache Spark 25
3.1.1 Spôsob spracovania dát 25
3.1.2 Komponenty 26
3.2 Resilient Distributed Dataset 28
3.3 Súčasné využitia 29

4 Návrh laboratórnych úloh **31**
4.1 Základní operace v Sparku 31
4.2 Lineární regrese 38
4.3 Klasifikace 44
4.4 Shlukování dat, četné vzory 49

5 Záver **54**

Literatúra **55**

Zoznam symbolov, veličín a skratiek **59**

ZOZNAM OBRÁZKOV

2.1 Sériové spracovanie dát 16
2.2 Paralelné spracovanie dát 17
2.3 Proces strojového učenia 21
2.4 Presnosť zhody 22
2.5 KMeans zhlukovanie 23
3.1 Komponenty Sparku 28
4.1 Lineární rovnice úsečky 39
4.2 Rozhodovací strom 45

ZOZNAM VÝPISOV

4.1 Úvodní program 32
4.2 Filtrování 35
4.3 Vytvoření PairRDD 36

ÚVOD

V súčasnej dobe technologických inovácií vzniká čoraz viac zariadení, ktoré musia navzájom pre správne fungovanie komunikovať. Či už sa jedná o rôzne senzory, domáce spotrebiče, alebo automobily, všetky využívajú pre komunikáciu internet. Takisto počet pripojených užívateľov je čoraz väčší a výsledkom všetkej tejto komunikácie sú veľké objemy dát, ktoré pre ich efektívne využitie potrebujeme určitým spôsobom spracovávať.

Pre spracovanie dát sa dajú používať pomerne jednoduché sériove programy, avšak pri veľkých objemoch je takýto postup časovo veľmi náročný. Preto bol navrhnutý, pôvodne Google proprietárny, programovací princíp MapReduce, ktorý spracúva dáta paralelne, teda rozdelí úlohu medzi viaceré výpočtové stroje, ktoré pracujú zároveň a k výsledku dospejú oveľa rýchlejšie.

Táto práca sa venuje aktuálne používaným technológiám Apache Hadoop a Apache Spark, ktoré fungujú na princípe MapReduce. Cieľom je zoznámiť sa s ich fungovaním a možnosťami, a následne vhodne navrhnúť laboratórne úlohy pre novo pripravovaný predmet Paralelné spracovanie dát na Fakulte elektrotechniky a komunikačných technológií, VUT v Brne.

Hlavným prínosom práce je návrh štyroch laboratórnych úloh, ktoré praktickým a zrozumiteľným spôsobom predstavujú výhody a použitia Sparku na situáciach z reálneho sveta. Každá úloha rieši konkrétny problém a týmto umožňuje pochopiť význam tejto technológie a takisto inšpirovať študenta k novým nápadom využitia.

V prvej kapitole je popísané, čo sa skrýva za pojmom Veľké objemy dát, čím je spôsobený ich náhly nárast, a aké problémy a následné riešenia z toho vznikajú. V druhej časti je vysvetlený princíp sériového, distribuovaného, paralelného spracovania dát a jednotlivé algoritmy. Ďalšia kapitola je zameraná na samotné prostredie Apache Spark, jeho možnosti, prvky a súčastné využitia. Posledná časť sa venuje návrhu laboratórnych úloh, ktoré budú zamerané na výpočtové úlohy v prostredí Apache Spark v programovacom jazyku Java.

1 VEĽKÉ OBJEMY DÁT

Denne sa na celom svete vygeneruje 2,5 exabajtov dát, teda $2,5 \times 10^{18}$ B. Toto číslo neustále rastie a ak chceme z tejto záplavy informácií využiť maximum pre zlepšenie našej životnej úrovne, potrebujeme k tomu nové nástroje a postupy. Pojem veľké objemy dát predstavuje akýsi koncept všetkých problematík spojených s týmto nárastom dát a jeho spracovaním.

Táto kapitola sa venuje práve týmto problematikám, vhodným riešeniam a aktuálnym využitiam v rôznych oblastiach.

1.1 História

Prvé zmienky o veľkom náraste množstva informácií a dát sa objavovali dávno predtým, než sa tento fenomén začal označovať známou frázou „veľké objemy dát“, alebo „Big Data“.

Fremont Rider [1], knihovník na univerzite Wesley, už v roku 1944 odhadoval vo svojej knihe The Scholar and the Future of the Research Library, že knižnice na amerických školách sa zdvojnásobujú každých šestnásť rokov, a že týmto tempom bude v roku 2040 obsahovať knižnica na univerzite Yale 200 miliónov výtlačkov, čo by zaberalo 6 tisíc míľ poličiek.

V šesťdesiatych a sedemdesiatych rokoch vychádzalo množstvo publikácií, ktoré sa venovali štatistickým prognózam a možným riešeniam. Vedecké poznatky narastali exponenciálne, čo dokazoval aj stále väčší počet vedeckých článkov, časopisov a kníh. Pri tejto informačnej explózii pokladalo veľa odborníkov nájdenie dostačujúceho úložiska za kľúčový problém.

Prvý článok v digitálnej knižnici ACM, v ktorom sa objavil pojem veľké objemy dát, vyšiel v októbri 1997, teda rok po tom, ako sa stali digitálne úložiská cenovo výhodnejšie ako papier [2]. Takto nazvali autori problém, keď sú súbory dát príliš veľké pre uloženie na lokálnej, poprípade externej pamäti.

Na prelome tisícročia sa vyprodukovalo približne 5 exabajtov digitálnych dát za rok. Mnohí odborníci a celosvetové spoločnosti ako Cisco následne odhadovali ďalší nárast, no nikto neočakával, že sa dostaneme na číslo 1227 exabajtov v roku 2010 a 2837 exabajtov o dva roky neskôr. Tieto čísla hovoria iba o vytvorených dátach. Podľa štúdie vedca Chattanya Baru z roku 2011, svetové servery spracovali v roku 2008 9,57 zettabajtov informácií [1]. Očakáva sa, že týmto tempom bude v roku 2020 vyprodukované približne 44 zettabajtov dát.

Ľudia a spoločnosti si uvedomujú, aké nespočetné možnosti sa v týchto dátach skrývajú a preto sa vyvíjajú stále efektívnejšie nástroje pre ich spracovanie a využitie. Je to jedna z najväčších inovácií v informatike, ktorá ma tendenciu stále rásť.

1.2 Definícia

V publikácii [3] Kate Crawfordová poukazuje na veľké objemy dát ako na kultúrny, technologický a vedecký fenomén, ktorý pozostáva zo súhry technológie, analýzy a mytológie. Technológiou je myslené stále narastajúci výkon výpočtových strojov a presnosť algoritmov pre prácu s nimi, analýza predstavuje hľadanie určitých vzorov a ich efektívne využitie a mytológia všeobecne rozšírenú dôveru, že veľké súbory dát nám môžu poskytnúť nové poznatky a postrehy s presnosťou a objektivitou aké by nebolo v minulosti možné dosiahnuť.

Na konferencii organizácie IEEE v roku 2015 [4] boli veľké objemy dát definované jednoducho ako súbory dát príliš rozsiahle a zložité na to, aby boli spracovávané tradičnými algoritmami, či už sa jedná o analýzu, zachytávanie, filtrovanie, vyhľadávanie, bezpečnosť a mnoho ďalších problematík.

Existuje množstvo ďalších definícií, ale všetky sa vo výsledku zhodujú v tom, že koncept veľké objemy dát označuje informácie, ktoré nemožno spracovať a analyzovať obvyklými spôsobmi a nástrojmi.

Formát

Formát dát možno všeobecne rozdeliť do troch skupín. Prvou sú dáta štrukturované. Je to jednoduchá forma slov, čísel, dátumov, ktoré sa efektívne dajú spracovávať pomocou relačných databáz. V minulosti neexistovala iná forma ako štrukturovaná.

Ďalšie sú semi-štrukturované dáta. Môžu byť vo forme štrukturovaných dát, ale narozdiel od nich nie sú organizované do klasických modelov relačných databáz. Sú to napríklad tabuľky. Na ich spracovanie sú potrebne komplexnejšie nástroje.

A najrozšírenejšími sú dáta neštrukturované. Tie tvoria dnes značnú väčšinu všetkých vygenerovaných dát. Ich analýza a spracovanie je veľmi komplikované, nedajú sa využívať tradičné relačné systémy. Patria sem textové správy, lokačné informácie, videá, dáta vygenerované na sociálnych sietiach a všetky, ktoré nemajú žiadnu konkrétnu formu.

Zdroje

Jedným z najväčších zdrojov dát je v posledných rokoch nepochybne internet vecí (IoT). Internet vecí pripája fyzický svet do internetu tak, aby všetko navzájom komunikovalo prostredníctvom senzorov a iných technickým noviniek. Podľa odhadov bude do roku 2020 do internetu pripojených 50 miliard inteligentných objektov.

Ďalším významným zdrojom sú sociálne siete. Každý užívateľ produkuje veľké množstvo neštrukturovaných dát, či už pridávaním a zdieľaním informácií, alebo jednoducho pohybom po webe.

Takisto sú dáta generované strojmi a inteligentnými zariadeniami automaticky, senzormi monitorujúcimi fyzikálne veličiny, pri vedeckých testoch, finančných transakciách a mnoho ďalších.

1.3 Oblasti využitia

Veľké objemy dát sa týkajú v súčasnosti najmä veľkých firiem, pre ktorých efektivitu sú množstvá informácií kľúčové. Avšak čím ďalej viac zasahujú aj do spôsobu každodenného života každého z nás a začína sa o tomto koncepte hovoriť ako o akejsi revolúcii, ktorá má zmeniť fungovanie sveta v mnohých oblastiach našej činnosti.

Počet zdrojov dát a užívateľov, ktorý ich produkujú, neustále narastá a tým vznikajú problémy pri ich spracovaní. Avšak, či už ide o odvetvie ekonomiky, vedy, informatiky, marketingu a mnoho ďalších, v tomto obrovskom množstve dát sa ukrývajú veľmi užitočné informácie [5]. Ich správne využitie nepochybne prináša zvýšenie produktivity alebo nové vedecké objavy.

Biológia

Spočiatku boli veľké objemy dát doménou prevažne astronómov a fyzikov. Dnes však nezaostávajú ani iné vedné odbory. Biológovia sa ku veľkým dátam obracajú stále častejšie, najmä pri testoch ako regulácia génov, vývoj genómu, alebo sledovaní umiestnenia mikróbov v tele [6].

Veda a výskum

V oblasti vedy a výskumu je jasným príkladom CERN, Európska organizácia pre jadrový výskum. Tá vyprodukuje ročne 30 pettabajtov dát. Inžinieri z CERNu uvádzajú že Hadoop a Spark zohrávajú kľúčovú rolu pri analýze dátových prúdov[13]. Všetci analytici môžu pristupovať k výsledkom takmer v reálnom čase.

Priemysel

V priemysle sú veľké objemy dát spájané najmä s ďalšou úrovňou digitalizácie. Inteligentné továrne budú na základe vygenerovaných a zozbieraných dát z okolia viac zautomatizované a výrobné stroje budú samočinne vyhodnocovať situácie [15]. Spojenie konceptu veľkých objemov dát a súčasne ďalších technológií v priemysle sa označuje ako „Industry 4.0“, alebo štvrtá priemyselná revolúcia.

Zdravotníctvo

Zdravotnícke organizácie obsahujú množstvo záznamov, ktoré nie sú žiadnym spôsobom zdielané. Ukrývajú sa v nich ale informácie, ktoré pri správnom spracovaní za použitia konceptu veľkých objemov dát môžu výrazne zlepšiť zdravotnú starostlivosť a predikciu ochorení. Služba Google Flu Trends napríklad na základe pohybu obyvateľstva ponúka krízové riešenia pri rozšírení epidémií [14]. Ďalším využitím môžu byť prenosné zariadenia sledujúce stav pacienta, ktoré varujú v prípade rizika jeho alebo príslušného zdravotníka.

Finančné služby

Článok z časopisu Forbes [16] uvádza niekoľko najčastejších využití analýzy veľkých dát v oblasti bankovníctva. Jedným z nich je detekcia podvodov na základe strojového učenia. Analýzou sa určí bežná aktivita užívateľov a každá podozrivá je následne odhalená. Veľký dôraz sa kladie aj na segmentáciu užívateľov z pôvodnej segmentácie podľa produktov, čo koncept veľkých dát výrazne zjednodušuje.

1.4 Problémy využitia

Možnosť pracovať s veľkými dátami a novými analytickými nástrojmi neznamená automaticky, že z nich spoločnosť bude mať úžitok v podobe hodnotných informácií. Pri práci s nimi sa vyskytuje niekoľko problémov.

Aby bolo možné dáta využiť, musia sa preniesť zo zdroja do koncových staníc a po ceste vykonať niekoľko operácií a integráciu dát z iných zdrojov. Pre transformačné úlohy sa využívajú najmä hadoopové riešenia, napr. procesy Map a Reduce.

Značným problémom sú aj úložiská. Aberdeen Group uvádza, že uložené dáta rastú ročne si o tretinu, čiže musia zdvojnásobovať svoju úložnú kapacitu každých 24 - 30 mesiacov [8]. Aj keď sa náklady na dátové disky znižujú, ich zväčšovanie nedokáže pokryť dlhodobú tendenciu rastu objemov dát. Veľké spoločnosti sa začínajú uberať smerom neinteligentných zariadení pre ukladanie dát a serverom, ktoré zabezpečujú ich správu. Tieto uzly využívajú často práve Hadoop a je vhodné aby boli umiestnené v tesnej blízkosti k dátam pre ich rýchle spracovanie. Rýchlosť možno navyšovať aj diskami SSD, ktoré už sú cenovo dostupné aj pre menšie firmy.

Riešenie Cloudom

Významnú úlohu v problematike veľkých objemov dát zohráva cloud computing, teda využitie výpočtovej sily a úložného priestoru vzdialených dátových centier.

Mnohí poskytovatelia cloudových služieb začínajú integrovať frameworky pre paralelné spracovanie dát do svojich služieb [9]. Toto riešenie je výhodné pre mnohé firmy, najmä ak zbierajú dáta z veľa zdrojov, ktoré môžu byť rozmiestnené po celom svete a ich prenos do vlastného dátového centra by bol náročný. Takisto sú tieto služby cenovo výhodnejšie ako kupovanie vlastných zariadení.

Aktuálne sa v rámci cloud computingu venuje zvýšená pozornosť tzv. systémom pre spracovanie znalostí (KPS - Knowledge Processing Systems). Tieto systémy spracovávajú už existujúce znalosti alebo informácie na základe určitých pravidiel a generujú z nich nové znalosti, alebo nám poskytnú lepšie postrehy [10]. Kvôli veľkým objemom existujúcich znalostí sú potrebné techniky ako MapReduce a distribuované dátové úložiská s príslušnými nástrojmi pre ich správu.

Vzhľadom na masívny nárast výpočtového výkonu a úložného priestoru poskytovaného poskytovateľmi cloudových služieb sa stále viac kladie dôraz na bezpečnosť informácií firiem a užívateľov, ktori ich využívajú. Jedným z využívaných prostriedok na jej zaistenie je multidimensionálna anonymizácia, avšak pri veľkých objemoch dát sa stáva cenovo neefektívna [11]. Využitím princípu MapReduce sa táto neefektivita značne znižuje. Hoci teda cloud computing prináša organizáciám mnohé benefity, bezpečnostné riziko tvorí stále veľkú bariéru pred jeho úplným nasadením [12]. Paralelné spracovanie a analýza dát však umožňuje vyvíjať stále dokonalejšie nástroje pre odstránenie tohto rizika.

2 PARALELNÉ SPRACOVANIE DÁT

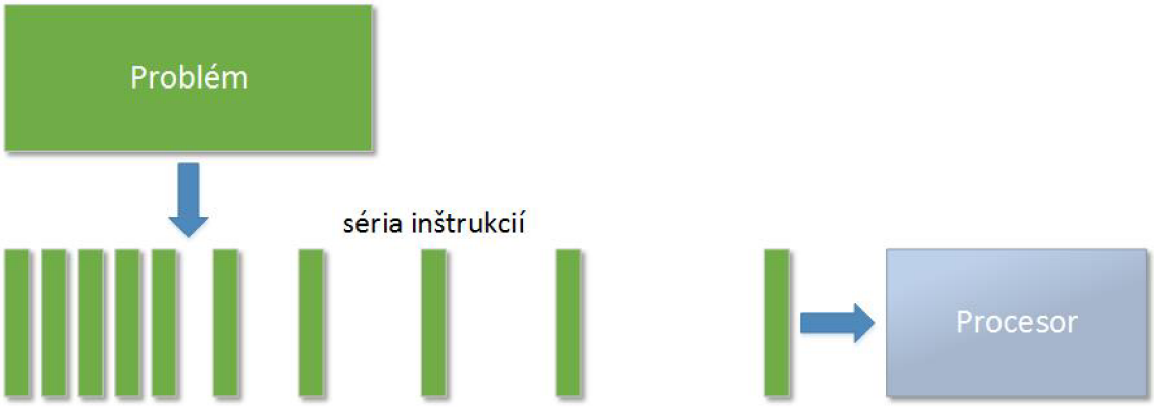
Spracovanie veľkých objemov dát, často nazývané „Data mining“, predstavuje proces získavania doposiaľ neznámych a potenciálne užitočných informácií z dátových databáz. Data mining sa často považuje za jeden z krokov získavania nových znalostí, alebo „Knowledge discovery“, avšak tieto pojmy predstavujú ten istý proces [17]. Ten sa skladá z postupných krokov ako združovanie, selekcia, transformácia dát, vyhodnotenie vzorov až k výslednej informácii.

2.1 Princíp

Pre pochopenie významu paralelného spracovania dát je dôležité porovnanie so sériovým prístupom a objasnenie potrieb spoločností s distribuovanou infraštruktúrou.

Sériový

Pri tradičnom sériovom prístupe sa určitý problém rozdelí na diskrétnu sériu inštrukcií, ktoré sa postupne vykonávajú jedna po druhej v jedinom procesore. Takže v jednom momente môže byť spracovávaná len jedna inštrukcia. Výpočtový výkon a pamäť jediného procesora sú značne obmedzené, čo spôsobuje problém najmä pri spracovaní veľkých objemov.



Obr. 2.1: Sériové spracovanie dát

Distribuovaný

Celosvetové organizácie potrebujú získavať informácie z viacero geograficky vzdialených zdrojov dát. Takisto užívatelia môžu byť vzdialený od zdrojov a zaistenie

služieb klasickými systémami vedie v takýchto prípadoch k veľkému nárastu komunikačných nákladov. Práve pre potreby takéhoto distribuovaného prostredia spoločností vznikli systémy pre distribuované spracovanie dát.

Paralelný

S príchodom veľkých objemov dát hrá však hlavnú úlohu paralelné spracovanie. Ide o súčasné využitie viacerých výpočtových strojov pre riešenie jedného problému. Problém je rozdelený na diskrétne časti, ktoré môžu byť riešené zároveň a tieto časti rozdelené na série inštrukcií sú spracované rôznymi procesormi. Celý proces riadi jeden kontrolný mechanizmus. Takto sa dajú efektívne využiť počítače s viacerými procesormi (jadrami), alebo väčší počet takýchto sieťovo prepojených počítačov (cluster).

Obr. 2.2: Paralelné spracovanie dát

2.2 MapReduce

Technológia MapReduce, zverejnená spoločnosťou Google, bola jedna z prvých pre prácu s veľkými objemami dát. Na jej základoch je postavená väčšina ďalších technológií, ktoré sa používajú a inovujú dodnes [18].

MapReduce popisuje dve nezávislé funkcie. Počítač prijme vstupné dáta a rozdelí ich ostatným počítačom v clustri. Funkcia Map vytvorí zo vstupných dát dvojice kľúč a hodnota. Výstup funkcie Map je poslaný funkcii Reduce ako vstupné dáta, ktoré sú následne spojené podľa kľúča. Takto zredukovaný výsledok je vrátený späť užívateľovi.

2.3 Základné operácie

Pri práci v jadre Sparku bez využitia ďalších komponentov, prebiehajú všetky operácie na objektoch alebo dátových množinách RDD (Resilient Distributed Dataset), alebo na PairRDD, čo je typ, v ktorom sú dáta uložené v pároch kľúč-atribúty. Obe sú bližšie popísané v kap. 3.2. Používané operácie sa delia na transformácie a akcie.

Transformácie

Používajú sa na existujúce RDD a po prevedení operácie na každý záznam jednotlivo vzniká nové RDD [26]. Všetky transformácie sa môžu spúšťať paralelne, každá výpočtová jednotka spracováva časť transformácie nezávisle na ostatných, čo značne urýchľuje výpočty.

- map: `noveRdd=Rdd.map(function)`
  Základná transformácia, ktorá prijíma funkciu ako argument. Pracuje takmer rovnako ako Map funkcia pôvodného MapReduce modelu. Prejde každý záznam a prevedie na ňom operáciu danú funkciou. Výsledné RDD má rovnaký počet záznamov (ale môžu byť iného typu). Používa sa napr. na štandardizáciu dát, konvertovanie dátových typov, výpočty na úrovni záznamov, pridanie nových atribútov alebo kontrolu a čistenie dát.
- flatMap: `noveRdd=Rdd.flatMap(function)`
  Pracuje rovnako ako Map transformácia, ale výstupom môže byť viac záznamov ako bolo v pôvodnom RDD. Používa sa pre všetky rozdeľovacie operácie, keď chcete vstupný záznam rozdeliť na viacero výstupných.
- filter: `noveRdd=Rdd.filter(function)`
  Prijíma filtrovaciu funkciu ako argument. Funkcia vracia hodnotu true/false, podľa toho či záznam spĺňuje zvolenú podmienku. Záznamy, ktoré podmienku spĺňujú sú vrátené na výstup v novom RDD, ktoré bude obsahovať menej záznamov ako pôvodné. Záznamy sami o sebe ale zmenené nie sú.
- union: `unionRdd=prveRdd.union(druheRdd)`
  Spája 2 RDD do jedného, ktoré obsahuje všetky záznamy z oboch pôvodných. Rovnaké záznamy sa duplikujú. Používa sa najmä pri spájaní databáz alebo záznamov, ktoré sú získavané z viacerých zdrojov.
- intersection: `intersectionRdd=prveRdd.intersect(druheRdd)`
  Vracia iba spoločné rovnaké záznamy dvoch RDD.
- mapValues: Map transformácia pre PairRDD, ktorá transformuje hodnoty, ale nezasahuje žiadnym spôsobom do kľúča.
- flatMapValues: FlatMap transformácia pre PairRDD, ktorou možno vytvoriť viaceré hodnoty s rovnakým kľúčom.

Akcie

Sú to operácie, ktoré pracujú na celom RDD, nie iba na jednotlivých záznamoch ako v prípade transformácii. Výsledkom akcie nie je nové RDD, ale výsledná hodnota ako napr. súčet alebo priemer.

- Reduce – najdôležitejšia akcia, ktorá pracuje veľmi podobne ako Reduce funkcia pôvodného MapReduce modelu. Reduce prijíma funkciu s dvomi vstupnými hodnotami. Vykonáva operáciu cez všetky elementy RDD, a to takým spôsobom, že ju vykoná na prvých dvoch elementoch, výslednú hodnotu použije ako vstupnú hodnotu pre operáciu s ďalším elementom a takto pokračuje až do posledného elementu. Na jednoduchom príklade je tento princíp vysvetlený v kap. 4.1.
- Collect – vráti všetky elementy jedného RDD. RDD môže byť pri výpočtoch rozdelené na časti (partitions), ktoré sú paralelne spracovávané naprieč clustrom, a pri zavolaní akcie collect sú všetky tieto časti zozbierané do jedného počítača. To je pri veľkom počte výpočtových strojov a veľkých objemoch dát náročná operácia a treba si vždy premyslieť jej použitie.
- count – vráti počet elementov v RDD
- first – vráti prvý element v RDD
- take(n) – vráti n elementov z RDD
- countByKey – spočíta počet záznamov s rovnakým kľúčom
- groupByKey – vykoná agregáciu podľa kľúča, napr. súčet, priemer
- reduceByKey – reduce operácia pre každý kľúč
- aggregateByKey – agregácia záznamov podľa kľúča

2.4 Strojové učenie

Strojové učenie je obor, ktorý nám sprístupňuje skutočný potenciál veľkých objemov dát. Jeho algoritmy spracovávajú známe dáta a učia sa na nich. Získané „poznatky“ potom aplikujú na nové dáta a tým nám odhaľujú výsledky, ku ktorým by ľudský mozog nedokázal dospeť. Je to teda určitá forma umelej inteligencie. Takýto program môže napr. predpovedať cenu projektov na základe predošlých, predvídať šírenie chorôb, ceny akcií na trhu a mnoho ďalších.

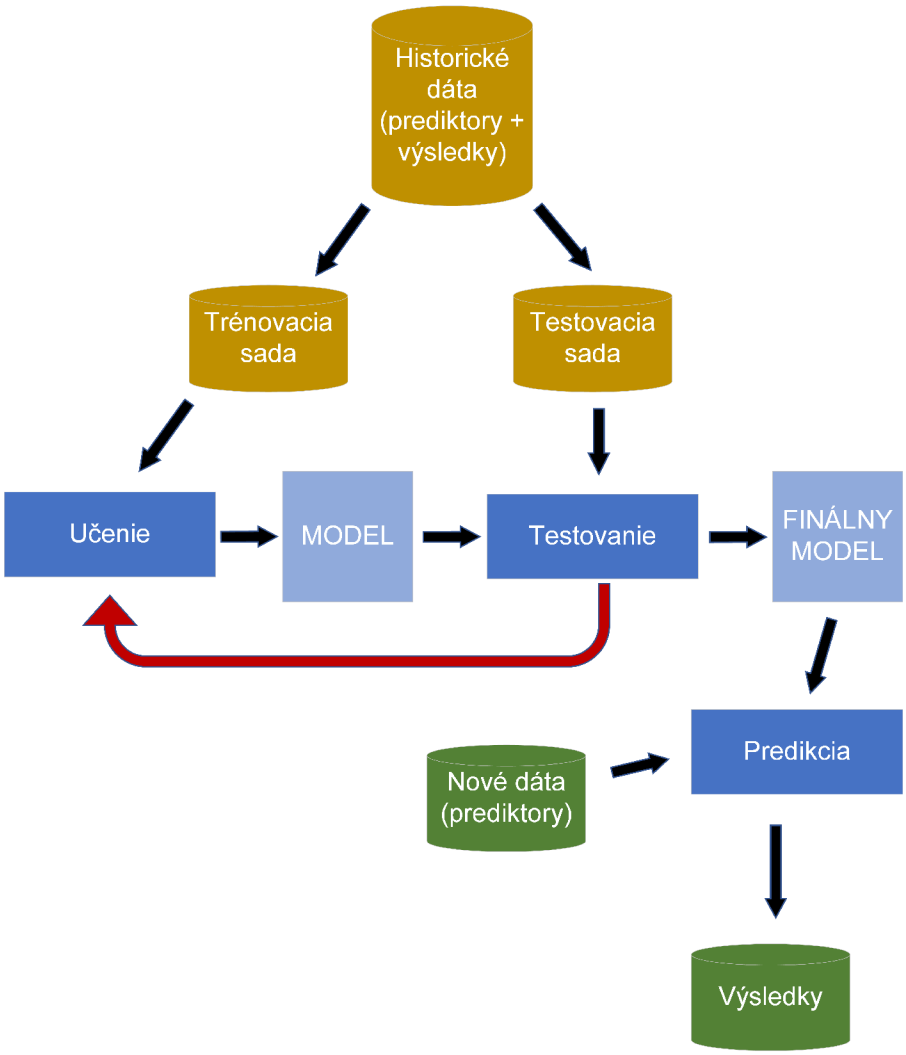
Všetky dáta obsahujú určité atribúty medzi ktorými existujú vzťahy. To, do akej miery sa tieto atribúty navzájom ovplyvňujú nazývame úroveň korelácie. Učiť sa v tomto prípade znamená porozumieť týmto vzťahom medzi atribútmi a strojové učenie je teda obor, v ktorom využívame počítač pre analýzu dát, odhalenie korelácií a ich následné využitie. Korelácie môžu vznikať medzi veličinami, u ktorých to očakáva ľudský mozog na základe naučených asociácií, ale v ére veľkých objemov dát dokáže počítač často odhaliť úplne nečakané vzťahy. V takom prípade už nemá význam snažiť sa odhaliť dôvod, ale jednoducho využiť získané poznatky. Práve tento prístup prináša nové revolučné využitie veľkých objemov dát [27].

Výsledkom algoritmov strojového učenia je model. Je to definícia alebo vysvetlenie vzťahov medzi atribútmi. Konkrétne sa môže jednať o rovnicu, ktorá definuje ako sa dá získať jedna hodnota z druhej, rozhodovací strom, ktorý určí výsledný atribút na základe viacerých prediktorov, model pre zoskupenie podobných hodnôt alebo ich predikciu.

Typický proces strojového učenia je zobrazený na obr. 2.3. Na začiatku sú vždy potrebné historické dáta, ktoré obsahujú prediktory aj výsledné hodnoty, ktoré chceme následne predikovať. Tieto dáta sa náhodne rozdelia s určitým pomerom na trénovaciu a testovaciu sadu. Na trénovaciu sadu sa aplikuje algoritmus strojového učenia, ktorého výsledkom je model. Ten je otestovaný na testovacej sade, ktorá obsahuje skutočné výsledky a tie sú porovnané s predikovanými. Na základe získanej presnosti možno využiť iteračný proces a vrátiť sa späť k učeniu a vylepšiť tento proces. Po získaní modelu s dostačujúcou presnosťou sú naň aplikované nové dáta a tým sú získané predikované výsledky.

Lineárna regresia

Lineárna regresia je jeden z algoritmov strojového učenia. Cieľom tejto metódy je získať lineárnu rovnicu (model), ktorá reprezentuje vzťah premenných, a na základe ktorej možno predikovať ďalšie hodnoty. Rovnica reprezentuje úsečku, ktorej súčet štvorcov vertikálnych vzdialeností medzi každým bodom a úsečkou je minimálny [28]. Používa sa v prípadoch, keď chceme predikovať číselné hodnoty. Pri výpočtoch sa sleduje parameter „Goodness of Fit“(zn. R^2), na ktorom sa dá overiť, ako presne

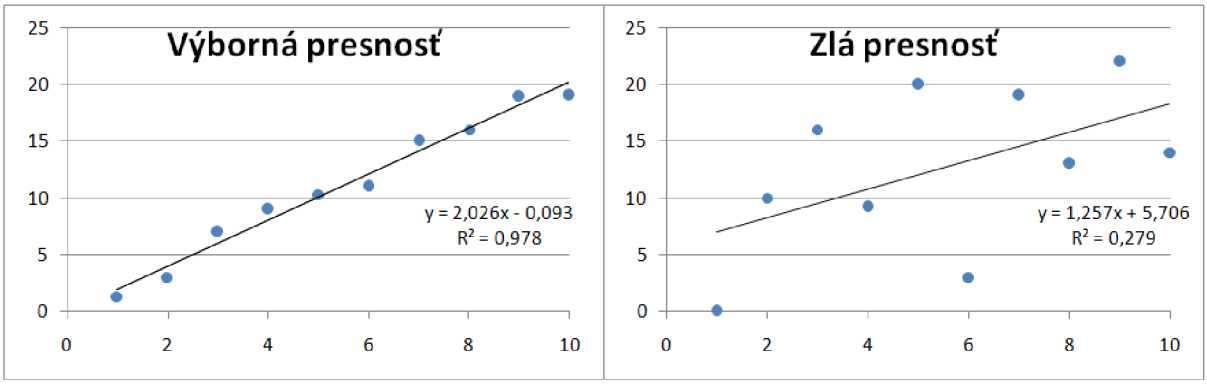


Obr. 2.3: Proces strojového učenia

reprezentuje rovnica vzťahy medzi premennými. Na obr. 2.4 sú znázornené situácie s rôznou presnosťou. Výhodami lineárnej regresie sú rýchlosť výpočtov, nenáročnosť na výpočetný výkon a pamäť počítača, vysoká presnosť predikcií pri lineárne závislých hodnotách. Z toho vyplýva, že tento algoritmus nedokáže pracovať s nelineárnymi vzťahmi, môže pracovať iba s číselnými hodnotami, a presnosť je veľmi ovplyvniteľná malým počtom hodnôt, ktoré ležia mimo grafu (outlier). Preto ich treba pred samotným výpočtom eliminovať.

Klasifikácie

Cieľom klasifikačného algoritmu je rozhodnúť, do akej skupiny alebo kategórie nový prvok patrí, teda klasifikovať ho. Program sa rozhoduje podľa modelu, ktorý bol vytvorený na základe predošlých pozorovaní známych dát. Typickým príkladom je



Obr. 2.4: Presnosť zhody

spam filter. Klasifikačných algoritmov existuje niekoľko, táto práca sa venuje rozhodovaciemu stromu a náhodnému lesu.

Rozhodovací strom je veľmi jednoduchý a ľahko pochopiteľný algoritmus klasifikácie. Model predstavuje samotný rozhodovací strom. Ten je vytvorený na základe historických dát. Program môže napríklad klasifikovať, či má človek cukrovku alebo nie podľa jeho váhy a veku. Strom by mal v tomto prípade ako koreň rozhodovací blok váhy. Ten sa rozvetví a každá vetva smeruje do rozhodovacieho bloku veku. Po spojení týchto dvoch štatistických predpovedí strom rozhodne, či sa jedná o cukrovkára alebo nie. O poradí parametrov v strome rozhoduje algoritmus sám, manuálne by to predstavovalo neľahkú úlohu, keďže je za tým veľa komplexnosti. Úrovní stromu môže byť väčšie množstvo, záleží na počte prediktorov. Výhody rozhodovacieho stromu sú, že dokáže pracovať s chýbajúcimi dátami a jeho rýchlosť a jednoduchosť [29]. Tá má význam napríklad pri rozhodovaní o poskytnutí pôžičky zákazníkovi. Ak chce vedieť prečo mu nebola poskytnutá, dá sa to veľmi ľahko vyhľadať. Nevýhodou je obmedzená presnosť a nevhodnosť pre veľké počty prediktorov.

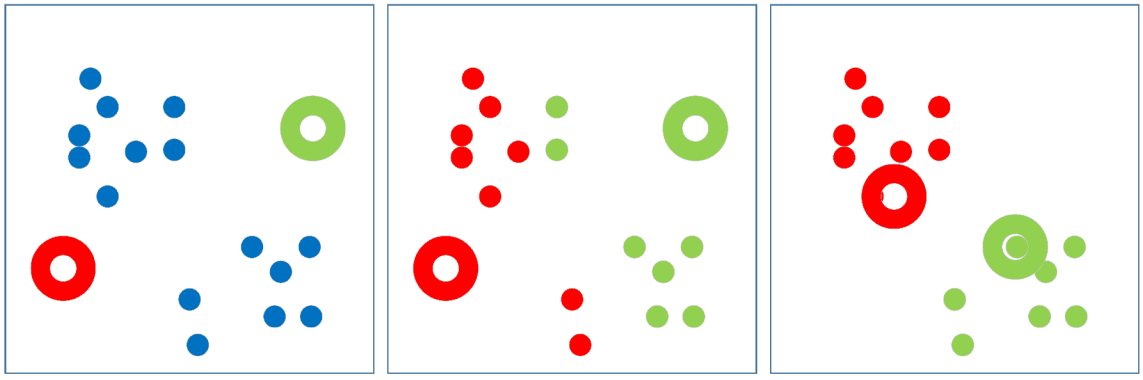
Náhodný les je jedným z najobľúbenejších a najpresnejších algoritmov klasifikácie. Je postavený na rozhodovacom strome. Narozdiel od neho ale využíva viacero modelov (stromov). Rozhodnutia z každého stromu porovná, a väčšinové zvolí ako výsledné. Jednoducho sa to dá vysvetliť na kúpe monitoru. Spýtate sa 10 kamarátov, či si kúpiť určitý monitor. Každý kamarát predstavuje jeden model. Podľa toho, čo povie väčšina, sa rozhodnete. Veľkou výhodou náhodného lesa je jeho presnosť a schopnosť pracovať s veľkým počtom prediktorov. Každý strom pracuje iba s časťou hodnôt (väčšinou s odmocninou všetkých prediktorov) a hodnoty si vyberá náhodne. Vďaka tomu dokáže pracovať aj s chýbajúcimi dátami. To spôsobuje ale veľkú náročnosť na výpočtové prostriedky a čas. Využíva sa často vo vedeckých výskumoch,

zdravotných diagnózach a všade, kde je kľúčová presnosť a nezáleží na rýchlosti.

Zhlukovanie dát

Zhlukovanie dát je jednou z techník strojového učenia. Je to metóda pre zoskupovanie dát do určitého počtu skupín alebo podmnožín, a to podľa ich vzájomných podobností. Podobnosť dát určuje podobnosť hodnôt parametrov, ktoré obsahujú. V tejto technike sa nerozdeľujú parametre na prediktory a predikované hodnoty, ale všetky možno považovať za prediktory.

Táto práca sa venuje zhlukovaniu KMeans. Všetky vstupné prvky alebo riadky sa zoskupujú do `k` zhlukov, pričom každý prvok sa vo výsledku nachádza len v jednom zhluku. Vzorový priebeh KMeans zhlukovania je znázornený na obr. 2.5. Najskôr je vytvorený m-rozmerný priestor, kde m je počet údajov alebo stĺpcov a do tohto priestoru sú umiestnené všetky prvky. Následne sa náhodné umiestni do priestoru požadovaný počet centrálnych bodov reprezentujúcich centrum zhlukov [30]. V tomto prípade je na obrázku dvojrozmerný priestor s dvomi zhlukmi. Medzi každým prvkom je vypočítaná vzdialenosť k obom centrálnym bodom a podľa nej je priradený k jednému z nich. K meraniu vzdialenosti je možné použiť viaceré metódy, najpoužívanejšia je Euklidova vzdialenosť, teda po priamke. Rozdelené prvky teraz predstavujú nové zhluky a v ďalšom kroku je vypočítané centrum týchto zhlukov, kde sa premiestnia centrálne body. Po ich premiestnení nastáva ďalšia iterácia, teda prepočítanie vzdialeností, vytvorenie nových zhlukov a premiestnenie centrálnych bodov. Tým sa niektoré prvky premiestnia do druhého zhluku. Tento proces sa opakuje až kým sa centrálne body nestabilizujú a prestanú sa premiestňovať.



Obr. 2.5: KMeans zhlukovanie

Výhodou Kmeans zhlukovania oproti iným metódam je jeho rýchlosť, efektivita pre veľký počet premenných a je pomerne jednoducho pochopiteľný. Nevýhodou je,

že počet zhlukov musí byť dopredu známy. To môže byť problém, pretože nie vždy sa dá jednoducho odhadnúť koľko logických skupín dáta obsahujú. Zhlukovanie sa často používa ako prvotné zoskupenie dát. Potom sú na jednotlivé skupiny použité ďalšie variácie klasifikácií.

Početné vzory

Jedným z krokov pri analýze veľkých objemov dát je hľadanie početných prvkov, podmnožín alebo vzorov. Metóda strojového učenia, ktorá odhaľuje takéto vzťahy medzi prvkami vo veľkých dátových množinách sa nazýva učenie asociačných pravidiel (Association rule learning). Typické využitie tejto metódy sú systémy odporúčania na internete. Pri prezeraní produktov si systém zaznamenáva položky, a spája si získané dáta so zákazníkmi, ktorí si prezerali podobné produkty. Podľa ich ďalších vyhľadávaní ponúka aktuálnemu zákazníkovi produkty, ktoré by ho mohli zaujímať. Ďalším príkladom sú nákupné košíky. Vernostné kartičky zákazníkov sa využívajú na zaznamenávanie položiek v jednotlivých nákupoch, a systém je následne schopný odhadnúť pravidlo, že pri kúpe masla a syru je veľká pravdepodobnosť, že si kúpite aj chlieb. Podľa týchto pravidiel sú supermarkety zvyšovať zisky správnym usporiadaním produktov v regáloch, alebo vyberaním akciových produktov.

Dôležitými parametrami, s ktorými sa v tejto metóde pracuje sú podpora (support) a dôveryhodnosť (confidence). Podpora určuje, ako často sa prvok alebo podmnožina nachádza v celej dátovej množine. Dôveryhodnosť určuje ako často sa dané pravidlo potvrdilo. Ak asociačné pravidlo udáva, že pri kúpe chleba a syru si zákazník kúpi aj maslo, a vo všetkých zaznamenaných nákupoch, kde je chleba a syr sa nachádza aj maslo, dôveryhodnosť pravidla je 100 percent [31].

Jedným z algoritmov, ktorý pracuje na základe asociačných pravidiel je FP-growth, parallel FP-growth v prostredí Sparku. Ten pri spracovaní dátovej množiny transakcií najskôr vypočíta frekvencie jednotlivých prvkov. Potom si vytvorí model (FP-tree), v ktorom si následne vyhľadáva opakujúce sa podmnožiny.

3 PROSTREDIE APACHE SPARK

Apache Spark je open-source framework postavený na programovacom jazyku Scala. Je to distribuovaný nástroj pre prácu s veľkými objemami dát. Vznikol na univerzite v Berkley a od roku 2013 je pod správou Apache Software Foundation. Od tej doby sa stal najpoužívanejším nástrojom v tejto oblasti [19]. Ponúka kvalitné API (Application Programming Interface) pre jazyky Java, Scala, Python, R a množstvo knižníc napr. pre strojové učenie a analýzu grafov.

3.1 Apache Hadoop a Apache Spark

Napriek značným výhodám Sparku sú situácie, kedy spoločnosti viac vyhovuje starší ekvivalent Hadoop. Pre správne pochopenie ich využitia sa treba zamerať na ich porovnanie, jedná sa totiž o veľmi odlišné technológie.

Obe technológie sú v podstate frameworky pre prácu s veľkými objemami dát, avšak Hadoop sa často považuje za databázu, softvér, ktorý umožňuje spoľahlivé a bezpečné ukladanie veľkého množstva dát. Hlavnou vlastnosťou Sparku je rýchle spracovanie týchto dát a analýza výsledkov.

V publikácii [21] a v mnohých ďalších sa uvádza, že Spark je až 100x rýchlejší ako klasická implementácia MapReduce. Nie je to však vždy pravda. Je to vďaka tomu, že medzi-výsledky operácií neukladá na disk ale využíva pri výpočtoch operačnú pamäť počítačov. Ukladanie a opätovné načítanie dát z disku totiž značne spomaľuje výpočet. To platí ale iba pri výpočtoch kedy pracuje Spark v in-memory režime. Pri riešení problémov veľkých objemov dát spočíva zrýchlenie v rozložení záťaže na viacero výpočtových strojov, čo nemožno označiť jednoducho ako stonásobné zrýchlenie. Takisto pri práci na lokálnom počítači, kde operačná pamäť nie je dostatočne veľká, dokáže Spark ukladať dáta na disk a postupne načítať do pamäte.

3.1.1 Spôsob spracovania dát

Jedna z podstatných odlišností týchto frameworkov je spôsob spracovania dát. Zatiaľ čo Hadoop využíva tzv. dávkové (batchové) spracovanie, Spark podporuje aj spracovanie prúdové (streamové). Práve vďaka tomu je mnohonásobne rýchlejší pri výpočtových úlohách.

Dávkové spracovanie

Dávkové (batchové) spracovanie je vhodné v prípadoch, kedy nie sú potrebné okamžité výsledky, ale dáta sa určitý čas ukladajú do databáz a následne sa spustí požadovaná úloha. To je typickým príkladom princípu MapReduce, na ktorom funguje

Hadoop.

Prúdové spracovanie

Prúdové (streamové) spracovanie, alebo spracovanie prúdov dát, má naopak najväčší význam, keď potrebujeme okamžitú reakciu na prichádzajúce dáta, čo umožňuje získať výsledky takmer v reálnom čase [20]. Vstupné hodnoty teda nie sú žiadne fixné súbory, ale neustály prúd dát.

Technológie, ktoré umožňujú takéto spracovanie sú napr. Apache Spark, Apache Storm alebo Apache Flink. Apache Flink je najnovší framework, ktorý podporuje prúdové spracovanie v pravom slova zmysle a dosahuje väčších rýchlostí ako Spark. Je však ešte len v počiatkoch vývoja. Apache Storm takisto ako Flink podporuje pravé prúdové spracovanie, avšak ako jediný žiadne iné. Pre svoju jednoduchosť a vyspelosť je určite najrozšírenejšou technológiou Spark. Pri prúdovom spracovaní však dochádza k niekoľko-sekundovému oneskoreniu. Preto pri potrebe výsledkov naozaj v reálnom čase sú vhodnejšími voľbami Flink alebo Storm.

3.1.2 Komponenty

Hadoop

Hadoop pozostáva z troch hlanvých komponentov - Hadoop MapReduce (konkrétna implementácia modelu popísaného v 2.2), Hadoop Distributed File System (HDFS) a Hadoop YARN.

HDFS je distribuovaný súborový systém, ktorý tvorí jadro Hadoop systému. Všetky časti systému sú s ním prepojené. Podporuje bežne operácie so súbormi a zabezpečuje bezpečné fungovanie aj pri poruche niektorého z diskov. Toto zabezpečenie dát je umožnené napríklad využitím RAIDu (redundant array of independent disks). Všetky dáta sú uložené redundantne aspoň na troch diskoch a v prípade zlyhania niektorej časti clustru dochádza k replikácii dát na ďalšie stroje.

O celkovú správu systému sa stará Hadoop YARN. Ten slúži k riadeniu celého systému, má na starosti napríklad rozdeľovaniu práce jednotlivým serverom v clustri.

Spark

Na obr. 3.1 sú znázornené hlavné komponenty Sparku. Hlavným riadiacim komponentom je Spark Core (jadro), na ktorom stavajú ostatné komponenty. Zaisťuje základné operácie a plánovanie.

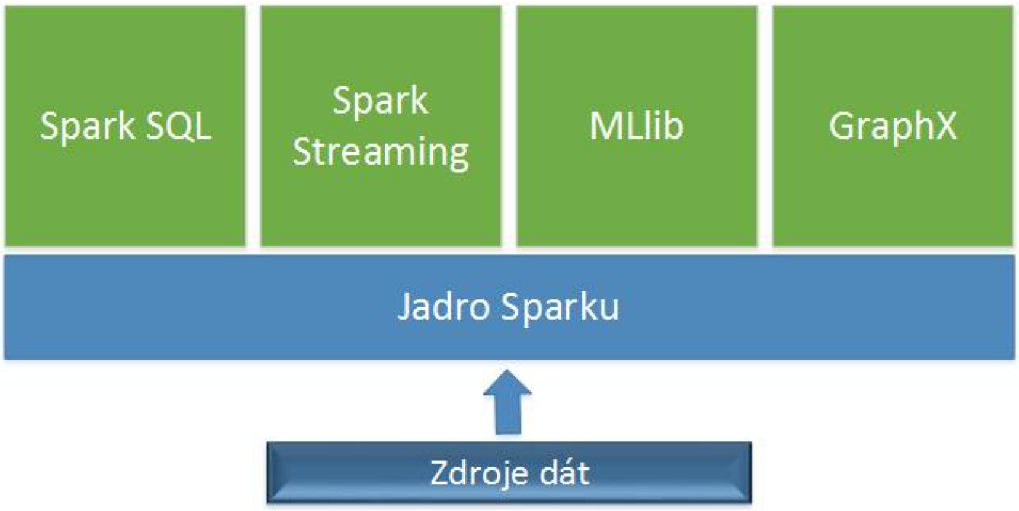
Komponent Spark SQL (pôvodne Apache Shark) je knižnica pre prácu so štrukturovanými dátami a umožňuje využívanie dotazov nad dátami v SQL jazyku. O výpočty nad dátami sa ale stará samotný Spark. Používa pevne danú štruktúru Data Frame, ktorá je rovnako ako RDD distribuovaná po clustri.

Spark Streaming je knižnica Sparku, ktorá zaobstaráva prúdové spracovanie dát. To môže vo všeobecnosti pracovať na základe niekoľko rôznych techník. Jednou z nich je zaznamenávanie každého prúdu individuálne a spracovávanie priamo za behu programu. Ďalšia možnosť je kombinovanie viacerých prúdov do tzv. mini-dávok (mini-batches). Tie môžu byť ohraničené časom, alebo počtom záznamov v jednej dávke. Spark využíva druhú spomínanú techniku, preto sa aj často označuje ako streamové spracovanie v nie úplne pravom slova zmysle, keďže dochádza k určitému oneskoreniu. Spark pri tom využíva štruktúru DStream (Discretized Stream). Je to postupnosť mini-dávok, kde každú mini-dávku reprezentuje RDD. Jedna dávka sa napĺňa určitý čas (napr 5s) a potom sa začne spracovávať. Počas toho sa začne napĺňať ďalšia dávka. Takýmto spôsobom možno pracovať s nekonečným prúdom dát takmer v reálnom čase [20]. Sú rôzne spôsoby ako sa dajú poslať na vstup Spark Stremingu dáta. Môžno k tomu využiť špeciálne platformy od Apache - Kafka alebo Flume, načítať dáta z databázy HDFS alebo spracovať dáta priamo z Twitteru. Vlastnosti Spark Streamingu:

- jednoduché používanie, programy sa v zásade nelíšia od tých, ktoré sú písané pre batchové spracovanie
- integrácia, programy písané pre batchové spracovanie možno znovu použiť pre streamové
- tolerancia chýb, batche (dávky) sú zapisované redundantne naprieč clustrom
- škálovateľnosť, možnosť definovať aká veľká časť DStreamu sa zapíše do jednotlivých RDD

Machine Learning Library (MLlib) je ďalšou veľmi podstatnou súčasťou Sparku. Je to knižnica obsahujúca algoritmy strojového učenia určené pre prácu v clustri. Spark je veľmi efektívnym nástrojom pre strojové učenie, najmä kvôli charakteristickej vlastnosti ukladania dát do operačnej pamäte serverov, takže nedochádza k načítavaniu dát z disku pri každom prechádzaní určitej štruktúry. Táto vlastnosť zaisťuje veľkú rýchlosť spracovania, je teda vhodnou voľbou pre všetky iteratívne výpočty.

GraphX je rozhranie pre programovanie grafov a paralelné grafové výpočty. Ponúka zobrazenie dát v podobe grafov alebo kolekcií, efektívne transformovanie či spájanie grafov s RDD 3.2, a písanie vlastných iteratívnych algoritmov pre grafy.



Obr. 3.1: Komponenty Sparku

3.2 Resilient Distributed Dataset

Základnou abstrakciou alebo objektom, ktorý sa používa pri vytváraní Spark aplikácií je Resilient Distributed Dataset (RDD). RDD sa dá jednoducho chápať ako zapuzdrenie veľkej kolekcie dát (datasetu). Môže obsahovať ľubovolný typ objektov, a to aj z užívateľom definovaných tried.

Dôležité vlastnosti pre prácu s RDD:

- Všetky požadované operácie v Sparku sú vyjadrené buď vytvorením nového RDD, transformáciou existujúceho RDD alebo prevedením akcie na RDD k získaniu výsledku. Na vytvorenom RDD možno teda vykonávať 2 typy operácií.
  - Transformácia predstavuje aplikáciu určitej funkcie na dáta v RDD za vzniku nového RDD. Najbežnejšou transformáciou je „filter“, ktorého výstup je nové RDD, ktoré obsahuje podmnožinu dát z povôdného RDD.
  - Aplikáciou akcie dostaneme výsledok na základe existujúceho RDD. Jednoduchým príkladom je akcia „first“, ktorej výstupom je prvý prvok v RDD.
- Transformácie sa nezačnú vykonávať hneď po zavolaní, ale až po zavolaní akcie na dané RDD, čo sa nazýva lenivé vyhodnocovanie (lazy evaluation). Táto vlastnosť redukuje počet pokynov, ktoré musí Spark spracovať tým, že ich zhlukuje dohromady.
- Spark automaticky distribuuje medzi všetky stroje dáta uložené v RDD a paralelne rozloží výpočty potrebné na prevedenie operácií zadaných na príslušné RDD.
- Vo všeobecnosti každý Spark program prebieha podobným spôsobom. Vytvorí

sa počiatočné RDD na základe vstupných dát, pomocou transformácií ako „filter“ či „map“ vzniknú zjednodušené RDD, z ktorých sa následne pomocou akcií získajú požadované výsledky.

- Keďže v reálnych situáciach sa často stretávame s dátami vo formáte meno – hodnoty k nemu priradené, je možné vytvoriť z normálneho RDD tzv. PairRDD, ktoré obsahuje páry kľúč – hodnoty. To umožňuje ľahšie a efektívnejšie zaobchádzanie s takýmito dátami a takisto podporuje všetky operácie používané pre klasické RDD.

3.3 Súčasné využitia

V súčastnej dobe vychádza veľké množstvo článkov, v ktorých sú popísané nové techniky a nápady pre zefektívnenie rôznych procesov, práve pomocou technológie Apache Spark. To potvrdzuje jeho skutočnú hodnotu.

Detekcia DDoS

Počet útokov DDoS stále narastá a ich frekvencia sa často mení. Útočníci používajú pre útok veľký objem dát v krátkom časovom úseku, aby znemožnili ďalšiu funkcionalitu danej služby. Niekedy však narúšajú jej chod aj menšími dátami počas dlhšej doby. Táto variabilita znemožňuje detekciu týchto útokov. Nová metóda založená na neurónových sietiach v prostredí Sparku však prekonáva tieto problémy a v reálnom čase rozoznáva až 94 percent DDoS útokov [22].

Magnetická rezonancia

V zdravotníctve sú stále potrebnejšie nástroje pre analýzu veľkých objemov dát. Pomocou nich dokážu lekári určovať presné diagnózy, či predikovať výskyt chorôb. Základným prostriedkom pre analýzu ľudského tela je magnetická rezonancia, ktorá sníma a ukladá celkový 3D obraz. To je vo väčšine prípadoch časovo veľmi náročný proces. V článku [23] je predstavená metóda využívajúca Spark, ktorá je časovo až štvornásobne efektívnejšia ako doteraz používané metódy. Dáta sú načítané do RDD, nasleduje paralelný výpočet a konvertovanie do obrazovej podoby.

Twitter

Twitter používajú ľudia po celom svete, aby vyjadrovali a zdieľali svoje pocity, udalosti, názory a iné. V tomto obrovskom objeme neštrukturovaných dát sa skrýva zlatá baňa informácií pre mnohé organizácie. Podľa krátkych správ dokážu určiť, ktoré produkty sú v danej chvíli najvhodnejšie pre online reklamy, predvídať vývoj

akciových trhov, alebo napr. predpovedať zisky nového filmu podľa reakcií na vydaný trailer. Správy sú však veľmi náročné na spracovanie, keďže obsahujú skratky, smajlíky a ďalšie neštandardné symboly. Práca [24] predstavuje novú klasifikačnú metódu, ktorá pracuje na technológii Spark a veľmi efektívne dokáže spracovávať milióny dátových inštancií tohoto typu.

Internet vecí

Internet vecí označuje koncept, ktorý prepája viaceré zariadenia cez internetovú sieť a umožňuje im navzájom komunikovať. Táto komunikácia produkuje veľké objemy dát, ktoré musia byť správne spracované pre ich skutočné využitie. Tieto dáta je nutné vo väčšine prípadov spracovávať v reálnom čase. Tradičné technológie analýzy dát však nepodporujú prúdové spracovanie dát [25]. Nové technológie preto využívajú ako riešenie Spark, ktorý podporuje prúdové spracovanie dát a pomocou paralelných výpočtov dosahuje výborných výsledkov.

4 NÁVRH LABORATÓRNYCH ÚLOH

Navrhnuté návody k laboratórnym úlohám budú písane oproti zbytku textu práce v českom jazyku, pretože predmet „Paralelní zpracování dat“ je takisto vyučovaný v českom jazyku.

4.1 Základní operace v Sparku

Cíl úlohy

Cílem laboratorní úlohy je seznámit se s prostředím Apache Spark, ověřit rychlost programu s využitím jednoho a více jader, na vzorovém programu se seznámit s využívanými koncepty a operacemi a na základě získaných informací navrhnou program pro řešení zadaného problému.

Zadání

- Seznamte se s prací s RDD, transformacemi, akcemi a agregacemi na vzorovém programu.
- Navrhnete programové řešení problému.
- Ověřte rychlost programu při využití různých počtů výpočetních prvků.

Teoretický úvod

Apache Spark je technologie pro jednoduché a rychlé zpracování a analýzu velkých objemů dat. Je to open-source framework určený pro paralelní výpočty v clustru, avšak možno ho využít i na lokálním počítači a pak napsaný program aplikovat na cluster s tisíckami strojů bez téměř jakýchkoliv změn. Využitím většího počtu počítačů dochází k značnému zrychlení výpočtů. Spark je založen na programovacím jazyku Scala, ale ponouká taky API pro Javu, Python a R. V průběhu tohoto kurzu se postupně seznámíte s jeho komponentami v jazyku Java, a vyzkoušíte si řešení praktických problému.

Tenhle úkol se zabývá základními operacemi v jádru Sparku. Všechny probíhají na objektu RDD (Resilient Distributed Dataset), který je paralelně zpracovávaný napříč jednotkami v clustru. Tyto operace se dělí na:

- transformace (např. map, flatMap, filter, reduce),
- akce (např. collect, count, first, take).

Rozdíl je ten, že výstupem transformace je nové upravené RDD a výsledkem akce jsou jenom požadované výstupní data. Na jednoduchém příklade si teď vysvětlíme některé operace a základní koncepty při práci se Sparkem, viz výpis 4.1.

Výpis 4.1: Úvodní program

```
1  public class Uloha1Uvod {
2
3    public static void main(String[] args) {
4
5      System.setProperty("hadoop.home.dir", new File("lib/")
6          .getAbsolutePath());
7
8      Logger.getLogger("org").setLevel(Level.ERROR);
9
10     SparkConf conf = new SparkConf().setAppName("uvod")
11         .setMaster("local[*]");
12
13     JavaSparkContext sc = new JavaSparkContext(conf);
14
15     List<Integer> inputNumbers = Arrays
16         .asList(1,123,25,385,0,514,62,92);
17
18     JavaRDD<Integer> inputRDD = sc.parallelize(inputNumbers);
19
20     JavaRDD<Integer> filteredRDD = inputRDD
21         .filter(number -> (number > 100));
```

Tento jednoduchý program vybere ze vstupních hodnot čísla větší než 100 a vypočítá z nich průměrnou hodnotu. Určitě by se dal napsat jednodušším způsobem a nepředstavuje typické využití Sparku, ale je vhodný pro vysvětlení principu operací.

Prvním příkazem na řádcích 5–6 nastavíte cestu k Hadoop knihovnám. Pak logovací level na „ERROR“, jinak Spark při kompilaci vypisuje značné množství informačních správ a výstup je nepřehledný. Úvodní konfigurace na ř. 10–11 obsahuje název aplikace a režim spuštění, kde local označuje, že Spark program poběží lokálně za využití počtu jader CPU zadaných v hranatých závorkách (* označuje maximální dostupný počet). Konfiguraci použijeme pro vytvoření objektu JavaSparkContext (ř. 13), který představuje vstupný bod pro práci se Spark Core (jádro Sparku).

Vstupní hodnoty jsou náhodná čísla uložené v Listu. Metodou `parallelize` je z nich vytvořený objekt RDD (ř. 18), kde každé číslo představuje jeden element. RDD je možné vytvořit taky z textového souboru, csv/tsv (comma/tab separated values) souboru, nestrukturovaných dat a pod. Pro získaní čísel větších než 100 je možné

použít transformaci filter. Ta projde každý jeden element/záznam v RDD, a jestli daný element splňuje zadanou podmínku, pošle ho na výstup, kterým je nové vyfiltrované RDD. Využitím akce count zjistíme počet záznamů v novém RDD, a akcí reduce součet těchto záznamů. Reduce představuje jednu z operací nazývaných agregace, které sdružují záznamy podle nastavených pravidel. Reduce v tomhle případe postupně prochází dvojice prvků a sčítá je dokud neprojde všechny záznamy. Sčítá tedy 123(x) + 385(y) = 508, 508(x) + 514(y) = 1022, a takhle by pokračoval dál.

V Sparku se často používají lambda funkce, co je zápis funkce předávané funkci filter ve výpisu 4.1. Je to zkrácený zápis anonymní třídy, která má v sobě jenom jeden příkaz, který se provede na každém prvku RDD.

Další často používané transformace jsou:

- Map – projde každý element RDD a vykoná na něm určitou operaci
- flatMap – funguje stejně jak Map, ale může vrátit větší počet elementů, vhodné na rozdělení záznamů
- set operace
  - union – spojí 2 RDD do jednoho, stejné data se duplikují
  - intersection – vrátí společné (stejné) elementy

Další často používané akce jsou:

- collect – vrátí všechny elementy RDD
- count – spočítá elementy RDD
- first – vrátí první element RDD
- take(n) – vrátí n elementů

RDD je možné taky změnit transformací MapToPair na PairRDD nebo KeyValue RDD. To má záznamy uložené pomoci objektů Tuple, kde jednomu klíči připadá hodnota, nebo seznam hodnot. V některých případech je práce s takovými záznamy jednodušší. Na PairRDD možno použít většinu operací jako na normální RDD, a taky některé další:

- countByKey – spočítá počet záznamů se stejným klíčem
- groupByKey – provede agregaci podle klíče, např. součet, průměr
- reduceByKey – reduce operace pro každý klíč
- aggregateByKey – agregace záznamů podle klíče

Pracovní postup

Chystáte se založit firmu v určitém odvětví a chcete zjistit konkurenci ve svém okolí, nebo oblast/město, kde se nachází málo podobným firem. Seženete si databáze firem

z dvou různých zdrojů (běžně dostupné na internetu, pro tenhle úkol jsou použité 2 vzorky po 5000 firem). Tyto záznamy by byly v případe velkých států nebo měst velmi rozsáhlé, tedy určitě vhodné pro paralelní zpracování. Vaším úkolem je upravit jednotlivé databáze na jednotný formát vhodný na další zpracování, spojit databáze do jedné a vyfiltrovat podle vámi zvolené kategorie. Výstup bude následovný:

- Výpis vyfiltrovaných firem seřazených abecedně podle města bez zbytečných údajů (zůstane jenom město, ulice, název, IČO/web) IČO nebo web pro případné ověření jestli firma existuje/funguje.
- Výpis jednotlivých měst a počtu firem v nich (město : počet).
- SQL tabulka (název,ulice,město,IČO/web).

pozn.: Pro jednoduchost se nebudou některé postupy shodovat se všeobecnými koncepty OOP, cílem tohoto kurzu je pochopit význam a použití Sparku.

1. Proveďte úvodní konfiguraci. Nastavte cestu do Hadoop binaries a logovací level na ERROR. V tomhle úkolu budete pracovat taky se SparkSQL, který využívá DataFrame místo RDD a pro práci s ním musíte vytvořit objekt SparkSession, a ne SparkContext jako v úvodní ukázce.

```
SparkSession spark = SparkSession
                .builder()
                .appName("nazev")
                .master("local[*]")
                .getOrCreate();
```

2. Dalším způsobem jak vytvořit RDD je načtení dat ze souboru. Vytvořte dvě RDD a do každého načtete jednu databázi (skDat.txt a czDat.txt). Jednotlivé řádky teď budou tvořit elementy uvnitř RDD. Do RDD vložíte data následujícím příkazem:

```
JavaRDD<String> skDat = spark.sparkContext()
                .textFile("data/skDat.txt" 1).toJavaRDD();
```

3. Nyní potřebujete upravit obě databáze na společný formát. Všimnete si, že se v mnohém liší: pořadí sloupců, zápis adresy, různé údaje, velkost písmen a taky některé neúplné záznamy. Tyto nepřesnosti je nutno postupnými úpravami ošetřit nebo odstranit. Začnete přepsáním obsahu na malé písmena pro snazší práci s filtrem. Využijte transformaci map a jednoduchou lambda funkci (metoda `toLowerCase`).

Teď pomocí transformace filter vyfiltrujte kategorii firem, s kterou chcete pracovat, co výrazně zmenší objem dat, s kterými budeme dále pracovat a taky odstraní hlavičku. Opět využijte lambda funkci, kde metodou split rozdělíte řádek (který

představuje jeden element RDD) a zjistíte jestli první prvek v řádku obsahuje danou kategorii.

Z vyfiltrovaných dat odstraňte přebytečné informace a vytvořte nové RDD, které bude obsahovat jenom název, ulici, město a IČO/web firmy v tomto pořadí. Použijte zase transformaci map, tentokrát však nemůžete použít lambda funkci, protože funkce bude obsahovat víc než jeden příkaz. Vytvoříme tedy anonymní třídu, kde definujeme, co se má s každým záznamem provést. Kód bude vypadat následovně:

Výpis 4.2: Filtrování

```
JavaRDD<String> clearedSkDat = filteredSkDat.map
            (new Function<String, String>(){

private static final long serialVersionUID = 1L;

@Override
public String call(String line)throws Exception {

/* rozdělení line, přiřazení vhodných částí,
ošetření chybějících údajů */

return StringUtils.join(new String[] {nazev,"\t",ulice,"\t",
                        mesto,"\t",ICO});
}
});
```

Tělo funkce bude hlavně práce s polem a metodou split. Obdobně upravte taky druhou databázi czDat. Na místo IČO z ní použijeme web. Pro kontrolu můžete používat

```
for ( String s~: clearedSkDat.collect()) {
System.out.println(s);
}
```

4. Upravené data spojíme do jednoho RDD pomocí transformace union, kterou použijte na jedno RDD a jako argument pošlete druhé. Teď když máte všechno v jedné databázi, vytvořte PairRDD, aby jste mohli pracovat s městem jako klíčem, a zbytkem informací jako hodnotami k němu přiřazenými. Z obyčejného RDD vytvořte KeyValue PairRDD transformací MapToPair, která implementuje rozhraní PairFunction, viz výpis 4.3. Funkce `call` přímá String na vstupu, projde každý záznam v RDD a výstup vrátí jako objekt Tuple2 s klíčem typu String a hodnotami v poli String[].

Výpis 4.3: Vytvoření PairRDD

```
JavaPairRDD<String, String[]>pairUnionDat = unionDat.mapToPair
        (new PairFunction<String, String, String[]>(){

private static final long serialVersionUID = 1L;

@Override
public Tuple2<String, String[]> call(String arg) {

/* rozdělte vstupní hodnoty (spojenou databázi) arg
a přiřaďte do vhodných proměnných, ne každý záznam
obsahuje IČO/web - ošetřte */

return new Tuple2<String, String[]>(mesto,values);
}
});
```

Vytvořené PairRDD seřaďte abecedně transformací sortByKey, a obsah následně vypište, čím bude splněn první výstup.

```
for (Tuple2<String, String[]> pairList : sortedPairUnionDat.take(20)){
          System.out.println(pairList._1 + "\t" + pairList._2[0] + "\t"
+ pairList._2[1] + "\t" + pairList._2[2]);
}
```

Pro druhý výstup (město:počet firem) použijte akci countByKey, která spočítá kolik je v PairRDD záznamů se stejným klíčem. Tato funkce vrací výsledek typu Map. V tomhle případě Map<String,Long>. Pak pro výpis použijte následovný cyklus:

```
for (Map.Entry<String,Long> entry : countedByKey.entrySet()) {
          System.out.println(entry.getKey() + " : " + entry.getValue());
}
```

5. Poslední krok je výpis do tabulky pomocí komponenty SparkSQL. Ta používá strukturu dataset, která usnadňuje práci se strukturovanými daty. Data jsou v ní uložené do datový abstrakce Row. V našem případě představuje Row jeden řádek RDD. Nejdřív vytvořte schému, která bude definovat sloupce tabulky.

```
String schemaString = "nazev ulice mesto ICO/web";
List<StructField> fields = new ArrayList<>();

for (String fieldName : schemaString.split(" ")) {
          StructField field = DataTypes.createStructField(fieldName,
```

```
DataTypes.StringType, true);
          fields.add(field);
}
StructType schema = DataTypes.createStructType(fields);
```

Teď překonvertujte RDD<String> na RDD<Row> a vytvořte dataset použitím vytvořeného RowRDD a schéma.

```
JavaRDD<Row> rowRDD = unionDat.map((Function<String, Row>) record ->{

/* rozdělte vstup record a přiřaďte vhodným proměnným, opět nutno
ošetřit neúplné záznamy (wwwICO) */

return RowFactory.create(nazev, ulice, mesto, wwwICO);
});
Dataset<Row> rowDataFrame = spark.createDataFrame(rowRDD,schema);
```

Obsah datasetu vypište následujícím příkazem:

```
rowDataFrame.show(20);
```

6. Teď, když máte program hotový, zkuste ověřit rychlost Sparku při využití více jader lokálního počítače. Ideální by bylo použít cluster s více stroji, ale pro ukázkové účely to bude stačit. Změňte v úvodní konfiguraci hodnotu v hranatých závorkách `"local[]"` a na konec programu přidejte následující cyklus, který zajistí, že program zůstane běžet:

```
while(true){
      try{
           Thread.sleep(10000);
      } catch(InterruptedException e) {
           e.printStackTrace();
      }
}
```

Nyní spusťte program a do webového prohlížeče zadejte <`http://localhost:4040/`>. Tato stránka je dostupná dokud Spark program běží alespoň v pozadí. Najdete zde množství informací a jednou z nich je čas výpočtů. Zastavte program, změňte počet jader, znovu ho spusťte a načtete stránku znovu. Porovnejte.

4.2 Lineární regrese

Cíl úlohy

Cílem laboratorní úlohy je seznámit se s koncepty strojového učení (ML - Machine Learning) a komponentami pro práci s nimi v prostředí Apache Spark a využití těchto znalostí při návrhu programu pro řešení zadaného problému.

Zadání

- Seznamte se s koncepty strojového učení v Sparku a algoritmem lineární regrese.
- Využijte data z předchozí úlohy.
- Navrhněte programové řešení problému.
- Optimalizujte program.

Teoretický úvod

Prostředí Apache Spark umožňuje praktickou a jednoduchou práci s algoritmy strojového učení, a to prostřednictvím knihoven `spark.mllib`, která je postavená na práci s RDD a `spark.ml`, novější verzi, která pracuje hlavně s objekty DataFrame a pipeline.

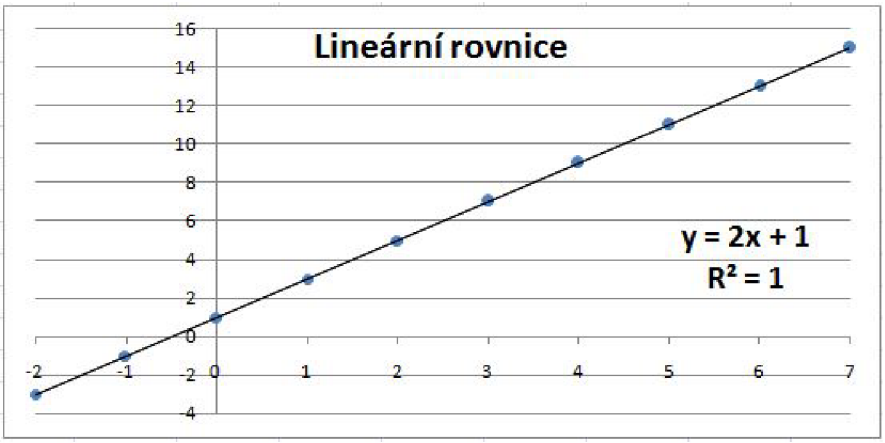
Vstupem algoritmu lineární regrese musí být data v numerickém formátu, proto je třeba je nejdřív převést z textového formátu. Spark dále poskytuje 2 speciální datové typy:

- Local Vector - vektor hodnot typu Double, může být:
  - Dense Vector - vektor nenulových hodnot v tvare $(13.0, 3.5, 0.3)$
  - Sparse Vector - vektor hodnot, kdy velká část dat má nulové hodnoty, originální zápis je nahrazen speciálním pro úsporu velikosti dat, zápis $(6, (1, 4), (3.5, 9.0))$ pak značí, že vektor má 6 prvků, a na pozicích 1 a 4 jsou nenulové hodnoty 3.5 a 9.0.
- Labeled Point - datová struktura, která obsahuje seznam (Local Vector) prediktorů, nebo-li hodnot na základe kterých se bude predikovat (features), a k nim asociovanou cílovou hodnotu, kterou chceme predikovat (label).

Každý algoritmus strojového učení očekává vstupní data tohoto typu, proto je potřebná data vždy nejdřív připravit a překonvertovat do této podoby. K tomu se využívají již připravené funkce v ML knihovnách.

Tenhle úkol se zabývá lineární regresí. Je to algoritmus strojového učení, jehož model představuje rovnice úsečky. Ta určuje vztah mezi závislými a nezávislými

proměnnými. Algoritmus je vhodný pouze pro případy, kdy je tento vztah lineární. Na obr. 4.1 je znázorněný příklad lineární rovnice.



Obr. 4.1: Lineární rovnice úsečky

Rovnice $y = 2x+1$ určuje vztah mezi proměnnými. Tohle je ovšem ideální případ. V reálných situacích jsou body na grafu (prediktory) rozptýlené mimo úsečku, a úseček může být taky více než jedna když máme více než jeden prediktor. Parametr R^2 ("Goodness of Fit") pak určuje, jak přesný je model, s klesající přesností se parametr blíží k nule.

To, do jaké míry prediktory ovlivňují výsledné hodnoty určuje úroveň korelace mezi těmito hodnotami. Čím se korelace víc přibližují k 1 nebo -1, tím víc budou ovlivňovat výsledek. V následujícím programu si vyzkoušíte vliv parametrů s různými korelacemi.

Pracovní postup

V předešlém úkolu jste si vyzkoušeli práci se základními operacemi Sparku a připravili jste si data pro další zpracování. Nyní využijete získané výstupní hodnoty. K dispozici máte připravený soubor uloha2.csv ve formátu čárkou-oddělené-hodnoty pro jednodušší práci, v kterém jsou vyfiltrovaná města a počty firem zabývajících se výpočetní technikou. Ke každému městu jsou taky přiřazeny hodnoty počtu obyvatel, rozlohy a nadmořské výšky.

pozn.: I když síla Sparku spočívá hlavně ve paralelním zpracování velkých objemů dat, jeho další výhodou je možnost vytvořit program lokálně na malém vzorku a pak ho použít bez větších změn na velkých objemech v klastrech. Ve všech úkolech tedy pracujeme pro jednoduchost s poměrně malými daty. Kdyby však chceme zpracovat údaje o všech firmách v rámci celého světa, lokální zpracování by už nebylo vhodné,

a náš algoritmus bychom mohli použít paralelně na více výpočetních strojích.

Takto předzpracovaná data jsou vstupem strojového učení. Vytvořte program, který bude na základě dostupných parametrů predikovat počet firem zabývajících se výpočetní technikou v jednotlivých městech. Vypočtěte a zobrazte korelace mezi jednotlivými parametry, koeficienty lineární rovnice, tabulku s predikovanými hodnotami a parametr test dobré shody.

1. Proveďte úvodní konfiguraci. Nastavte cestu do Hadoop binaries a logovací level na ERROR. Budete hodně pracovat s komponentou SparkSQL, vytvořte proto objekt SparkSession.

2. Načtěte soubor „uloha2.csv“ do datového rámce(dataset) s využitím hlavičky souboru a vytisknete pro kontrolu obsah a schéma.

```
Dataset<Row> firmyDf = spark.read()
          .option("header", "true")
          .csv("data/uloha2.csv");
firmyDf.show(5);
firmyDf.printSchema();
```

Na zobrazeném schématu je patrné, že všechna data jsou nyní načtené ve formátu String. Pro algoritmy strojového učení je nezbytné data převést do číselné podoby. To nejjednodušeji uděláte funkcí `map`, s kterou se ale pracuje na RDD. Vytvořte proto nyní schéma datového rámce pro pozdější zpětnou konverzi, a vytvořte `RDD<Row>` funkcí `.toJavaRDD()`.

Funkcí `map` překonvertujte číselné hodnoty na typ `Double`. Pamatujte, že pracujete s RDD pozůstávajícího z datových typů Row, nie String jak v předešlé úloze. Tělo vnitřní funkce `call` bude vypadat následovně:

```
Row convertedRow = RowFactory.create(
          iRow.getString(0),
          Double.valueOf(iRow.getString(1)),
          .
          .
          .
);
return convertedRow;
```

Přetransformované RDD překonvertujte pomocí schématu zpátky na Dataframe.

3. Data jsou ve vhodném formátu, teď je připravíte na strojové učení. Nejdřív vypočtěte korelace mezi jednotlivými parametry. Tím zjistíte, do jaké míry ovlivní výsledné predikce a dá se taky odhadnou jak přesné predikce budou.

```
for( StructField field : firmySchema.fields() ){
      if( ! field.dataType().equals(DataTypes.StringType)){
           System.out.println("Korelace mezi poctem firem a " + field.
           name()+ " = " + firmyDf2.stat().corr("FIRMY", field.name()));
      }
}
```

Další `map` funkcí překonvertujte data na LabeledPoint. K tomu musíte opět z datového rámce vytvořit `RDD<Row>`. Funkce `call` pak bude vracet typ LabeledPoint a její tělo bude vypadat následovně:

```
LabeledPoint lp = new LabeledPoint(iRow.getDouble(1),
      Vectors.dense(iRow.getDouble(2),
           iRow.getDouble(3),
           iRow.getDouble(4)
));
return lp;
```

Výraz iRow je parametrem funkce `call`, první hodnota v LabeledPointe je ta, kterou chceme predikovat, a hodnoty zapsané do „Dense vektoru“ jsou prediktory, na základě nich je pak vypočtený model.
RDD překonvertujte zpátky na datový rámec a vypište si několik hodnot. Hodnoty jsou nyní rozděleny na „features“, tedy prediktory, a sloupec „label“ představuje cílové hodnoty.

```
Dataset<Row> firmyLp = spark.createDataFrame(rdd4, LabeledPoint.class);
firmyLp.show(5);
```

V reálných situacích se strojové učení využívá tak, že se nejdřív na velikém počtu vzorků natrénuje model s vyhovující přesností, a ten se pak aplikuje na nové data, ze kterých chceme predikovat. Nám však bude stačit, když si naše data rozdělíme na trénovací sadu, na které model vypočteme a na testovací sade ověříme jeho přesnost. Náhodnost výběru je přitom klíčová, proto použijte následující funkci:

```
Dataset<Row>[] splits = firmyLp.randomSplit(new double[]{0.7, 0.3}, SEED);
Dataset<Row> trainingData = splits[0];
Dataset<Row> testData = splits[1];
```

Proměnnou SEED nastavte mimo funkci `main` jako statickou, a přiřaďte jí hodnotu 987. To zabezpečí, že náhodný výběr pro testovací a trénovací sadu zůstane stejný při každém spuštění.

4. Samotné strojové učení už je poměrně jednoduché, protože všechny potřebné funkce už jsou v Sparku definované. Vytvořte objekt lineární regrese. Použité parametry budou budou vysvětlené na závěr při optimalizaci.

```
LinearRegression lr = new LinearRegression()
      .setMaxIter(100)
      .setRegParam(0.0)
      .setElasticNetParam(0.2);
```

Vypočtete model lineární regrese. Z něj pak můžete zjistit koeficienty přímky (modelu) a dosadit libovolné hodnoty prediktorů. Proveďte predikci na testovací sadě dat a predikce vypište.

```
LinearRegressionModel lrModel = lr.fit(trainingData);

System.out.println("Koeficienty: " + lrModel.coefficients()
                   + " Beta: " + lrModel.intercept());

Dataset<Row> predikce = lrModel.transform(testData);

predikce.show();
```

Ve výpisu vidíte ve sloupci label skutečné a ve sloupci prediction predikované počty firem. Indikátor přesnosti pro lineární regresi se nazývá „Goodness of Fit“ (zn. R^2), přesnost roste, čím bliž je hodnota R^2 k 1. Vypočtěte přesnost a vypište.

```
RegressionEvaluator evaluator = new RegressionEvaluator()
      .setLabelCol("label")
      .setPredictionCol("prediction")
      .setMetricName("r2");
double r2 = evaluator.evaluate(predikce);
```

5. Jak vidíte, predikované hodnoty se dost liší od skutečných. To značí i malá hodnota R^2. Program je nutné vždy optimalizovat, často se tím dosáhnou značně lepší přesnosti. Ve výpisech celého programu jsou počty firem poměrně malé. Když si ale otevřete soubor uloha2.csv, všimnete si jednoho záznamu, který se značně odchyluje. Takový údaj se nazývá „outlier“ a pro přesnost lineární regrese představuje veliký problém, proto je třeba vždy ošetřit. Odstraňte záznam Bratislava a uložte soubor

jako uloha2opt.csv. Zkuste spustit program.

Lepší přesnosti se dá dosáhnout i parametry zadávanými při vytváření objektu lineární regrese. `setMaxIter` udává počet iterací. Platí, že čím víc iterací, tím lepší přesnost na úkor doby výpočtů. Při moc velikém počtu však může dojít k přetrénování dat. `ElasticNetParam` udává rozložení nepřesností a `RegParam` regularizaci. Zkuste změnit tyto hodnoty a otestovat jak to ovlivní přesnost modelu. Také zkuste na základě vypočtených korelací odstranit některé z údajů z LabeledPointu a sledujte změny přesnosti. Výsledky zdůvodněte.

4.3 Klasifikace

Cíl úlohy

Cílem úlohy je seznámit se s algoritmy klasifikace, s prací s nimi v prostředí Sparku, a s využitím znalostí z předchozí úlohy navrhnout řešení zadaného problému.

Zadání

- Seznamte se s algoritmy klasifikace rozhodovací strom a náhodný les.
- Navrhněte programové řešení problému.
- Porovnejte přesnost algoritmů.
- Aplikujte model na nová data.

Teoretický úvod

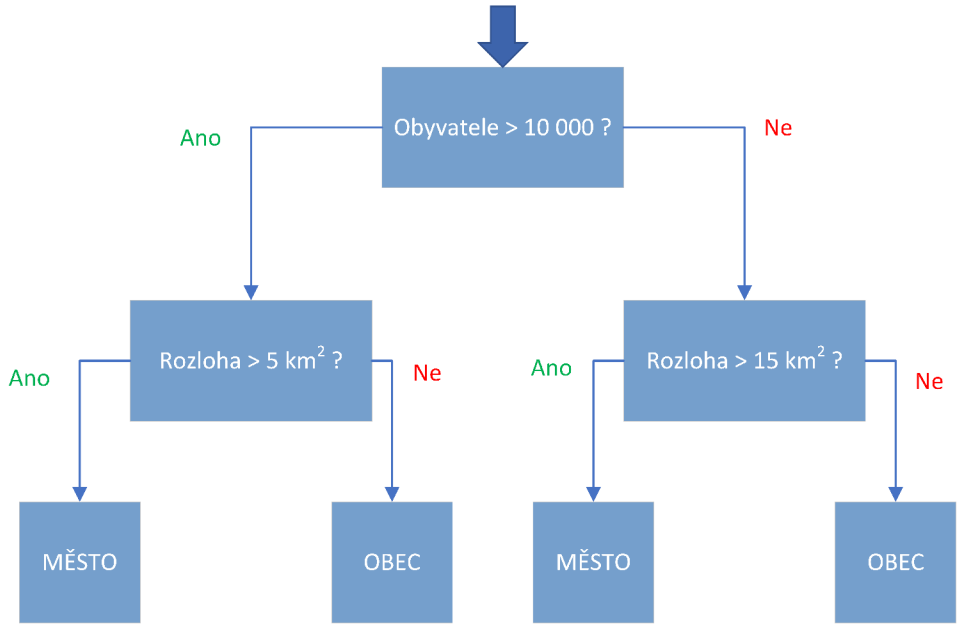
Tento úkol se věnuje další skupině algoritmů strojového učení, klasifikaci. V předešlém úkolu jste pracovali s regresním algoritmem, kde jste odhadovali hodnotu podle rovnice, která se co nejvíce shodovala s postupnými vstupními daty. Klasifikace nepredikuje dopředu neznámou číselnou hodnotu, ale rozhoduje, do jaké skupiny nebo kategorie vstupní prvek patří. Klasifikace je tedy vhodná pro případy, kdy potřebujeme rozdělit data do několika skupin. Typickým využitím jsou spamové filtry, programy rozhodující o udělení půjčky, určení, zda je pacient nemocen, nebo rozdělení dat do dvou skupin pro následné odlišné zpracování.

Nejpoužívanější algoritmy jsou rozhodovací strom (decision tree) a náhodný les (random forest). Oba pracují na podobném principu, až na výsledné rozhodování. Každý má pak určité výhody a nevýhody a oblasti použití.

Jednoduchý rozhodovací strom je na obr. 4.2. V tomto příkladě jsou parametry počet obyvatel a rozloha prediktory, podle kterých se model rozhoduje, zda se jedná o město nebo obec. V prvním rozhodovacím bloku se rozhodne o výsledku podle počtu obyvatel. Poté se model rozvětví na dvě větve a každá směruje do rozhodovacího bloku rozlohy města. Tady se opět rozhodne o výsledku, a jelikož to je poslední stupeň, rozhodnutí je konečné. Při více prediktorech by se strom větvil dál. Vytvoření modelu je poměrně komplexní problém, ale díky knihovnám Sparku ho nemusíme řešit, vytváří jsi jej sám na základě vstupních dat.

Náhodný les je velice přesný algoritmus. Jedná se o tzv. souborovou metodu (ensemble method) založenou na rozhodovacích stromech. To znamená, že výsledná predikce je výsledkem mnohých rozhodnutí, ne jenom jednoho. Program vytvoří

ze vstupních dat několik rozhodovacích stromů, porovná predikce každého z nich, a většinovou predikci zvolí jako výslednou. Všechny stromy jsou vytvořené z jiné podmnožiny dat vybrané náhodně, aby byl každý model rozdílný. V případě, že máme data o 1000 řádcích a 5 sloupcích, každý strom může být vytvořený ze 700 řádků a 3 sloupců. Náhodný les je velice náročný na čas a výpočetní výkon.



Obr. 4.2: Rozhodovací strom

Pracovní postup

V tomhle úkolu budete řešit problém znázorněný v teoretickém úvodu. Využijte data z předešlých úkolů s přidaným sloupcem město/obec. O tom, zda se jedná o město, rozhoduje několik faktorů, jako např. zabezpečení dopravního spojení s okolím, zabezpečení služeb pro obyvatele z okolních obcí, počet obyvatel, status hospodářského, administrativního nebo kulturního centra. V přiloženém souboru máte k dispozici údaje o počtu obyvatel, rozloze a nadmořské výšce. Vytvořte program, který bude predikovat zda se jedná o město a tím ověřte zda si algoritmus strojového učení poradí na dostatečně velké datové množině jenom s těmito údaji.

Úvodní načtení, zpracování a překonvertovaní dat je téměř stejné pro všechny programy strojového učení v tomhle kurzu. Části kódu se budou proto opakovat a budou k nim teď naposledy uvedené nápovědy.

1. Proveďte úvodní konfiguraci. Nastavte cestu do Hadoop binaries a logovací level na ERROR. Budete hodně pracovat s komponentou SparkSQL, vytvořte proto objekt SparkSession.

2. Načtěte soubor „uloha3.csv“ do datového rámce(dataset) s využitím hlavičky souboru a vytisknete pro kontrolu obsah a schéma.

```
Dataset<Row> firmyDf = spark.read()
          .option("header", "true")
          .csv("data/uloha3.csv");
firmyDf.show(5);
firmyDf.printSchema();
```

Data jsou nyní načtené jako hodnoty typu String. Je potřeba je překonvertovat na typ Double. To uděláte funkcí `map` na RDD. Dopředu si můžete vytvořit schéma datového rámce pro zpětnou konverzi.

```
StructType firmySchema = DataTypes.createStructType(new StructField[] {
          DataTypes.createStructField("MESTO", DataTypes.StringType, false),
          DataTypes.createStructField("FIRMY", DataTypes.DoubleType, false),
          .
          .
          .

});
```

Teď překonvertujte datový rámec (dataset) na `JavaRDD<Row>`, a použijte funkci `map` pro konverzi hodnot typu String na typ Double, stejně jako v předešlém úkolu. S použitím výsledného RDD a předem vytvořeného schéma, vytvořte nový datový rámec a vypište jeho obsah.

3. Korelační analýza probíhá vždy na číselných datech. Když chceme vypočítat korelace mezi jednotlivými parametry a údajem město/obec, musíme hodnotám město a obec přidat indexy. To znamená, že každé unikátní String hodnotě přiřadíme celočíselnou hodnotu. To lze udělat jednoduše pomocí objektu StringIndexer, který je zabudovaný v Sparku.

```
StringIndexer indexer = new StringIndexer()
          .setInputCol("MESTO/OBEC")
          .setOutputCol("IND_MESTO/OBEC");
```

Vytvořte StringIndexerModel a aplikujte ho na váš datový rámec. Do rámce se přidá sloupec s indexy.

```
StringIndexerModel indModel = indexer.fit(firmyDf2);
Dataset<Row> indFirmy = indModel.transform(firmyDf2);
```

Pro porovnání vypište oba sloupce s počtem hodnot.

```
indFirmy.groupBy("MESTO/OBEC","IND_MESTO/OBEC").count().show();
```

Vypočítejte a vypište korelace mezi jednotlivými parametry a údajem město/obec a vytvořte LabeledPoint pro strojové učení. Postup se téměř přesně shoduje s minulým úkolem. Pozor však na sloupec, který chcete predikovat. V zobrazené tabulce je to 5. sloupec, musíte však pracovat se sloupcem s indexovými hodnotami, který jste přidali na konec datového rámce. Z LabeledPointu vytvořte nový datový rámec a ten pak rozdělte na trénovací a testovací sadu v poměru 0.7/0.3.

```
Dataset<Row> firmyLp = spark.createDataFrame(rdd4, LabeledPoint.class);
firmyLp.show(5);
```

4. Data jsou připravená na strojové učení. Nejdřív vytvořte klasifikační objekt rozhodovacího stromu, a identifikujte sloupce s prediktory a predikovanou hodnotou.

```
DecisionTreeClassifier dt = new DecisionTreeClassifier()
          .setLabelCol("label")
          .setFeaturesCol("features");
```

Predikce bude probíhat na číselných datech, proto jste dělali indexaci. Ve výsledku by ale číselné hodnoty byly uživatelsky nevhodné, proto si vytvořte konvertory, které pro výsledný výpis převedou indexové hodnoty zpátky na String. Poté vytvořte model na trénovacích datech.

```
IndexToString labelKonvertor = new IndexToString()
          .setInputCol("label")
          .setOutputCol("labelString")
          .setLabels(indModel.labels());

IndexToString predKonvertor = new IndexToString()
          .setInputCol("prediction")
          .setOutputCol("predikceString")
          .setLabels(indModel.labels());

DecisionTreeClassificationModel dtModel = dt.fit(trainingData);
```

Model otestujte na testovacích datech. Nejdřív proveďte predikci na číselných hodnotách a pak je přetransformujte na textové hodnoty, a ty vypište ve výsledné tabulce.

```
Dataset<Row> ciselnePredikce = dtModel.transform(testData);
Dataset<Row> predikce = predKonvertor.transform(labelKonvertor
                    .transform(ciselnePredikce));

predikce.select("labelString","predikceString","features").show(15);

predikce.groupBy("labelString","predikceString").count().show();
```

Pro výpočet přesnosti použijte následující evaluator.

```
MulticlassClassificationEvaluator evaluator =
                new MulticlassClassificationEvaluator()
                .setLabelCol("label")
                .setPredictionCol("prediction")
                .setMetricName("accuracy");
double presnost = evaluator.evaluate(predikce);
System.out.println("presnost = " + Math.round(presnost * 100) + " procent");
```

Pokud jste postupovali správně, výsledná přesnost by měla vyjít 92 procent. Nyní vyzkoušejte použít algoritmus náhodný les a porovnejte jejich přesnost. Jediné, co musíte změnit je `DecisionTreeClassifier` na `RandomForestClassifier` a `DecisionTreeClassificationModel` na `RandomForestClassificationModel`.

5. Na závěr otestujte váš model na úplně nových datech. Ve složce máte k tomu připravený soubor „noveData.csv“, můžete si však vytvořit vlastní soubor s obcemi podle vašeho výběru. Všechny hodnoty, s kterými zde pracujeme jsou jednoduše dostupné na wikipedii o každé obci. S novými daty postupujte stejně jako v předešlém případě. Dávejte si jenom pozor na to, které hodnoty v nových datech jsou a které ne. Sloupec město/obec chybí, protože ten chceme predikovat. Nemusíte dělat indexování, a při tvorbě LabeledPointu si vytvořte lokální proměnnou id, kterou použijete jako predikovanou hodnotu. Při výpisu výsledků překonvertujte pomocí `predKonvertoru` jenom predikované hodnoty na String.

4.4 Shlukování dat, četné vzory

Cíl úlohy

Cílem úlohy je seznámit se s algoritmy shlukování dat, četnými vzory, s prací s nimi v prostředí Sparku a následně navrhnout řešení zadaného problému.

Zadání

- Seznamte se s algoritmy KMeans pro shlukování a FP-growth pro určení četných vzorů.
- Navrhněte programové řešení problému pomocí KMeans shlukování.
- Vyzkoušejte si práci s algoritmem FP-growth.
- Navrhněte vhodně vstupní data a nastavení algoritmu.

Teoretický úvod

Další technikou strojového učení je shlukování dat. Jejím výstupem nejsou predikované hodnoty, ale seskupení prvků dat do logických podmnožin nebo skupin. Logické podobnosti závisí na hodnotách parametrů prvků. V tomto případe můžeme všechny parametry považovat za prediktory.

V tomhle úkolu budete pracovat s algoritmem KMeans pro shlukování. Jeho výhodou je rychlost a efektivnost při práci s velikým počtem proměnných. Shlukování probíhá více iteracemi. Počtem shluků určíte počet centrálních bodů, které se postupně přemísťují po m-rozměrném prostoru vždy do centra aktuálních shluků. Po každém přemístění jsou přepočítané vzdálenosti prvků k centrálním bodům, a některé prvky jsou přemístěny do jiného shluku. Rozměr prostoru určuje počet parametrů prvků.

Primární využití shlukování dat je jakékoliv logické seskupení prvků. Může to být seskupení zákazníků, dokumentů, webových stránek, správ, nebo taky geolokační seskupení objektů. Právě tomu se budete věnovat v této úloze. Dalším využitím je dopředná kategorizace dat pro následné rozdílné zpracování skupin jinými algoritmy, např. klasifikaci.

Hledání četných vzorů je velice často využívaná metoda strojového učení v obchodní sféře. Typickými příklady jsou systémy odporoučení v internetových obchodech, nebo programy pro analýzu nákupních košíku.

Algoritmus, kterým se tato úloha zabývá, a který slouží právě k hledání četných vzorů je FP-growth. Ten si nejdřív při analýze jednotlivých transakcí vypočítá frekvenci prvků a z těch si vytvoří model FP-tree. Z něho pak získává asociační pravidla, podle kterých odporoučí další položky s určitou věrohodnostní.

Minimální věrohodnost (confidence) a podpora (support) jsou dva důležité parametry, které se v algoritmu FP-growth nastavují. Podpora určuje, jako často se percentuálně položka nachází ve všech transakcích. Když minimální podporu nastavíte moc velikou, a položky se ve vstupních datech opakují málo, program nebude brát do úvahy žádnou z nich. Když ji naopak nastavíte moc nízko, program bude řešit asociační pravidla prvků, které se vyskytli např. jenom jednou. To je z efektivního hlediska zbytečné.
Věrohodnost určuje, jak často bylo asociační pravidlo potvrzené v jiných transakcích. Když bude asociační pravidlo udávat, že při koupě sýru a másla si koupíte taky chléb, a toto pravidlo se potvrdí ve 2 ze 4 transakcí, věrohodnost bude 50 procent. Proto třeba při nastavování minimální věrohodnosti zvážit, jaká věrohodnost je pro vás dostačující.

Pracovní postup

V předchozích úkolech jste pracovali s daty o městech a obcích, predikovali nebo klasifikovali určité hodnoty. Teď by jste ale chtěli získat pohled z větší perspektivy o počtech a umístění firem v rámci celého Česka a Slovenska. Údaje o poloze obcí jsou jednoduše dostupné v rámci internetu nebo GPS. Pomocí geolokačního KMeans shlukování dat nyní navrhněte program, který rozdělí města a obce do vámi zvoleného počtu oblastí podle jejich zeměpisné šířky a výšky. Vypíšete si počet měst v jednotlivých oblastech, v kterých jsou firmy zabývající se výpočetní technikou. K dispozici máte data z předešlých úkolů doplněné o souřadnice obcí a měst.

Úvodní nastavení, načtení, a zpracování dat se téměř úplně shoduje s předešlými dvěma úkoly, proto k nim již nebudou uvedeny nápovědy.

1. Proveďte úvodní konfiguraci. Nastavte cestu do Hadoop binaries a logovací level na ERROR. Pro některé části kódu budete potřebovat kromě objektu SparkSession i objekt SparkContext. Oba však nemohou fungovat zároveň. Pro tento případ máte připravenou třídu Connection, vytvořte s její pomocí oba objekty.

```
JavaSparkContext spContext = Connection.getContext();
SparkSession spSession = Connection.getSession();
```

2. Načtěte soubor „uloha4.csv“ do datového rámce(dataset) s využitím hlavičky souboru a vytisknete pro kontrolu obsah a schéma.

Překonvertujte vstupní data na typ Double pomocí funkce `map` na RDD. Vytvořte schéma datového rámce pro zpětnou konverzi.

Překonvertujte data zpátky na datový rámec.

I když v algoritmech shlukování dat nepredikujeme žádné hodnoty, do struktury LabeledPoint musíte zadat proměnnou jako cílovou hodnotu. Jednou z možností je zadat lokální proměnnou id. Ve výsledném výpisu byste ale viděli pouze číselné hodnoty, bez uživatelsky přívětivých informací. Proto vytvořte stejně jako v minulé úloze objekt StringIndexer, který použijete k přiřazení unikátního indexu každému městu. Index použijte jako cílovou hodnotu a po vykonání strojového učení ho překonvertujete zpátky na názvy měst.

```
StringIndexer indexer = new StringIndexer()
          .setInputCol("MESTO")
          .setOutputCol("IND_MESTO");

StringIndexerModel indModel = indexer.fit(firmyDf2);
Dataset<Row> indFirmy = indModel.transform(firmyDf2);
```

3. Při shlukování dat je důležité, aby hodnoty v jednotlivých sloupcích byly ve stejném číselném rozsahu. Během výpočtů se měří Euklidova vzdálenost mezi prvky. Kdyby sloupce obsahovaly hodnoty v jiných řádech, mohlo by to působit nepřesnosti. Proto se před samotným strojovým učením dělá centrování a škálování (v tomhle případě to není úplně nevyhnutné, ale je dobré si to vyzkoušet pro další použití). Tím docílíte, že všechny hodnoty ve všech sloupcích proporcionálně převedete do rozsahu $(-3, 3)$. Nejdřív vypočtěte průměr a standardní odchylku všech sloupců. Využijte k tomu funkci `aggregate`.

```
Row priemRow = indFirmy.agg(avg(indFirmy.col("VYSKA")),
                            avg(indFirmy.col("SIRKA")))
               .toJavaRDD().takeOrdered(1).get(0);

Row stdRow = indFirmy.agg(stddev(indFirmy.col("VYSKA")),
                          stddev(indFirmy.col("SIRKA")))
               .toJavaRDD().takeOrdered(1).get(0);
```

Hodnoty uložte do Broadcast proměnné pro následnou práci při vytváření LabeledPointu.

```
Broadcast<Row> bcPriemRow = spContext.broadcast(priemRow);
Broadcast<Row> bcStdRow = spContext.broadcast(stdRow);
```

Vytvořte objekt LabeledPoint, kde jako predikovanou hodnotu zvolte sloupec s indexy měst, a do vektoru prediktorů zadejte proměnné `s`, `v`. Tyto proměnné se budou počítat pro každý řádek jako rozdíl průměrné hodnoty parametru a hodnoty v aktuálním řádku, děleno standardní odchylkou daného parametru. Ve funkci `call` zapište před vytvoření LabeledPointu:

```
double s~= (bcPriemRow.value().getDouble(0) - iRow.getDouble(2))
             / bcStdRow.value().getDouble(0);
double v~= (bcPriemRow.value().getDouble(1) - iRow.getDouble(3))
             / bcStdRow.value().getDouble(1);
```

4. Z LabeledPointu vytvořte nový datový rámec. Dále vytvořte objekt KMeans. V nastavení `.setK` si můžete nastavit vámi zvolený počet shluků/oblastí.

```
KMeans kmeans = new KMeans()
          .setK(10)
          .setSeed(1L);
```

Vytvořte model a aplikujte ho na centrovaný a škálovaný datový rámec.

```
KMeansModel model = kmeans.fit(firmyLp);
Dataset<Row> ciselnePredikce = model.transform(firmyLp);
```

Teď už zbývá jen překonvertovat indexy na názvy měst pomocí IndexToString konvertoru a vypsat výsledky.

```
IndexToString labelKonvertor =  new IndexToString()
          .setInputCol("label")
          .setOutputCol("labelString")
          .setLabels(indModel.labels());


Dataset<Row> predikce = labelConverter.transform(ciselnePredikce);


predikce.show();


predikce.groupBy(col("prediction")).count().show();
```

Ve výsledných výpisech vidíte ve sloupci „prediction“ číslo, které reprezentuje číslo shluku/oblasti, do kterého bylo dané město přiřazeno a taky počet měst v jednotlivých oblastech. Zkontrolujte zda jsou města v jedné oblasti geograficky blízko sebe.

5. Na závěr si vyzkoušíte práci s algoritmem FP-growth. Tyto programy jsou poměrně jednoduché, jde tady spíš o to, aby jste se seznámili s jeho možnostmi. Vytvořte program pro analýzu nákupních košíků.

Proveďte úvodní konfiguraci. Nastavte cestu do Hadoop binaries a logovací level na ERROR, vytvořte objekt SparkSession.

Vstupní data nebudete načítat ze souboru, ale napíšete si je přímo v programu pro jednoduchou editaci. V následujícím bloku si vytvořte transakce neboli-li nákupní košíky podle vašeho výběru.

```
List<Row> transakce = Arrays.asList(
     RowFactory.create(Arrays.asList("chleb maslo syr pivo".split(" "))),
     RowFactory.create(Arrays.asList("chleb syr sunka jablko".split(" "))),
     RowFactory.create(Arrays.asList("pivo chleb jablko".split(" ")))
     );
```

Vytvořte schéma a s jeho použitím datový rámec s transakcemi.

```
StructType schema = new StructType(new StructField[]
      new StructField("predmety", new ArrayType(DataTypes.StringType, true),
                          false, Metadata.empty())
);

Dataset<Row> predmetyDf = spark.createDataFrame(transakce, schema);
```

Nyní stačí už jenom vytvořit model a aplikovat ho na datový rámec. Vypište frekvence předmětů, asociační pravidla s hodnotami věrohodnosti a taky konkrétní predikce/odporoučení.

```
FPGrowthModel FPmodel = new FPGrowth()
           .setItemsCol("predmety")
           .setMinSupport(0.5)
           .setMinConfidence(0.6)
           .fit(predmetyDf);

FPmodel.freqItemsets().show();
FPmodel.associationRules().show();
FPmodel.transform(predmetyDf).show(false);
```

Zkuste měnit parametry `minSupport` a `minConfidence` a sledujte změny ve výsledcích. Změny zdůvodněte.

5 ZÁVER

Bakalárska práca je zameraná na technológie Apache Spark a Hadoop, princíp ich fungovania a typické využitie v reálnych situáciach. Riešenie problémov v praktickej časti je navrhnuté v prostredí Apache Spark, v programovacom jazyku Java. V súvislosti s týmito technológiami popisuje práca koncept veľkých objemov dát, problémy vznikajúce pri ich spracovaní, a súčastné využitia v rôznych oblastiach. Ďalej opisuje princíp paralelného spracovania dát, základné operácie, algoritmy strojového učenia a súčastné využitia Sparku.

Cieľom bolo navrhnúť laboratórne cvičenia do predmetu „Paralelní zpracování dat“. Prvá úloha zoznámi študentov s prostredím Sparku a so základnými konceptami. Nasleduje návrh programu, ktorý z dvoch rozdielnych databáz českých a slovenských firiem získa užitočné dáta pre ďalšie spracovávanie. Na výstupe programu sú údaje o počte firiem určitej kategórie v každom meste databáze. Tieto dáta sú využité v následujúcich úlohách. V závere úlohy je demonštrovaný rozdiel rýchlosti pri využití rôzneho počtu výpočtových strojov. V ostatných troch úlohách sa pracuje s algoritmami strojového učenia. Využitím lineárnej regresie je navrhnutý program, ktorý si na základe údajov z prvej úlohy vytvorí model, podľa ktorého predpovedá, koľko firiem sa bude v meste nachádzať iba podľa rozlohy a počtu obyvateľov. V závere úlohy sú demonštrované spôsoby optimalizácie programu. V tretej úlohe navrhnutý program klasifikuje, či sa jedná o obec alebo mesto podľa počtu obyvateľov, rozlohy a nadmorskej výšky. Porovnáva sa v nej presnosť algoritmov rozhodovací strom a náhodný les. Po dosiahnutí požadovaného výsledku si študenti môžu otestovať program na nimi zvolených nových dátach. Posledná úloha zoskupuje pomocou zhlukovania dát mestá a obce do územných celkov. Jedná sa o geolokačné zhlukovanie dát. Výsledok ukazuje, koľko firiem danej kategórie sa nachádza v jednotlivých oblastiach. K tejto úlohe sú k vstupným dátam pridané súradnice obcí. V závere úlohy je navrhnutá jednoduchá úloha na algoritmus pre početné vzory, a to typický príklad nákupného košíka, kde si možno meniť produkty v transakciách a sledovať zmeny vo výsledku.

Hlavným prínosom práce je postup pre oboznámenie študentov s technológiou Apache Spark v rámci predmetu „Paralelní zpracování dat“, s jeho významom a využitím na praktických príkladoch. Zoznámia sa takisto s funkcionálnym programovaním.

Výstupom práce sú štyri kompletné návody pre laboratórne cvičenia, vypracované so stručným teoretickým úvodom a postupnými inštrukciami pre vypracovanie. Úlohy riešia reálne problémy a tým navádzajú študentov k novým nápadom využitia Sparku. Jedným z možných spôsobov rozšírenia práce je vytvorenie aplikácie, ktorá by zahŕňala mimo iné funkcionality programov v tejto práci.

LITERATÚRA

[1] PRESS, Gil. A very short history of big data. *Forbes Tech Magazine*, May, 2013, 9.

[2] BUYYA, Rajkumar, Rodrigo N. CALHEIROS a Amir Vahid DASTJERDI, ed. *Big data: principles and paradigms*. Cambridge: Morgan Kaufmann, 2016. ISBN 978-0-12-805394-2.

[3] BOYD, danah a Kate CRAWFORD. *CRITICAL QUESTIONS FOR BIG DATA*. *Information, Communication* [online]. 2012, 15(5), 662-679 [cit. 2017-10-28]. DOI: 10.1080/1369118X.2012.678878. ISSN 1369-118x. Dostupné z: http://www.tandfonline.com/doi/abs/10.1080/1369118X.2012.678878

[4] KOMAROV, Mikhail. *Network Challenges of New Sources of Big Data* [online]. [cit. 2017-10-15]. DOI: 10.1109/CBI.2015.32. ISBN 10.1109/CBI.2015.32. Dostupné z: http://ieeexplore.ieee.org/document/7264764/

[5] NAIK, Kirtida a Abhijit JOSHI. *Role of Big Data in various sectors* [online]. [cit. 2017-10-15]. DOI: 10.1109/I-SMAC.2017.8058321. ISBN 10.1109/I-SMAC.2017.8058321. Dostupné z: http://ieeexplore.ieee.org/document/8058321/

[6] MARX, Vivien. Biology: The big challenges of big data. *Nature*, 2013, 498.7453: 255-260.

[7] HARČÁR, Ivan. *Problematika spracovania veľkých objemov údajov (Big Data) a financie* [online]. 2015 [cit. 2017-10-15]. Diplomová práce. Vysoká škola regionálního rozvoje a Bankovní institut – AMBIS, Bankovní institut vysoká škola SK. Vedoucí práce Juraj Pančík.

[8] LEAVITT, Neal. Storage challenge: Where will all that big data go?. *Computer*, 2013, 46.9: 22-25.

[9] HASHEM, Ibrahim Abaker Targio, et al. The rise of „big data“ on cloud computing: Review and open research issues. *Information Systems*, 2015, 47: 98-115.

[10] NADSCHLAGER, Stefan. *Introducing Design Patterns to Knowledge Processing Systems in the Context of Big Data and Cloud Platforms* [online]. [cit. 2017-10-15]. DOI: 10.1109/DEXA.2017.26. ISBN 10.1109/DEXA.2017.26. Dostupné z: http://ieeexplore.ieee.org/document/8049684/

[11] ZHANG, Xuyun, Chi YANG, Surya NEPAL, Chang LIU, Wanchun DOU a Jinjun CHEN. An evaluation of major threats in cloud computing associated with big data. 2013 *International Conference on Cloud and Green Computing* [online]. IEEE, 2013, , 105-112 [cit. 2017-10-27]. DOI: 10.1109/CGC.2013.24. ISBN 978-0-7695-5114-2. Dostupné z: http://ieeexplore.ieee.org/document/6686016/

[12] KAUR, Kamalpreet, Ali SYED, Azeem MOHAMMAD a Malka N. HALGAMUGE. Review: *An evaluation of major threats in cloud computing associated with big data.* 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)([online]. IEEE, 2017, , 368-372 [cit. 2017-10-27]. DOI: 10.1109/ICBDA.2017.8078842. ISBN 978-1-5090-3618-9. Dostupné z: http://ieeexplore.ieee.org/document/8078842/

[13] BRAEGER, Matthias a Manish DEVGAN. *Unlocking Big Data at CERN* [online]. Terracotta, 2014 [cit. 2017-05-18]. Dostupné z: http://blog.terracotta.org/wp-content/uploads/2014/10/Unlocking-Big-Data-at-CERN.pdf

[14] MAYER-SCHÖNBERGER, Viktor a Kenneth CUKIER. *Big Data: Revoluce, která změní způsob, jak žijeme, pracujeme a myslíme.* Brno: Computer Press, 2014. ISBN 978-80-251-4119-9.

[15] *Industry 4.0.* In: Wikipedia: the free encyclopedia [online]. San Francisco (CA):Wikimedia Foundation, 2017 [cit. 2017-05-18]. Dostupné z: https://en.wikipedia.org/wiki/Industry_4.0

[16] 5 Big Data Use Cases in Banking and Financial Services. *Ingram Micro Advisor* [online]. Ingram Micro Advisor, 2017 [cit. 2017-05-18]. Dostupné z: http://www.ingrammicroadvisor.com/data-center/5-big-data-use-cases-in-banking-and-financial-services

[17] LESKOVEC, Jure; RAJARAMAN, Anand; ULLMAN, Jeffrey David. *Mining of massive datasets.* Cambridge university press, 2014.

[18] DEAN, Jeffrey; GHEMAWAT, Sanjay. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 2008, 51.1: 107-113.

[19] SHANAHAN, James G.; DAI, Laing. Large scale distributed data science using apache spark. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2015. p. 2323-2324.

[20] STYPINSKI, Martin. *Data Stream Management Systems: Apache Spark Streaming.* Rapperswil, 2017. Master of Science in Engineering. Hochschule für Technik Rapperswil. Vedoucí práce Prof. Stefan F. Keller.

[21] SHYAM, R., et al. Apache spark a big data analytics platform for smart grid. *Procedia Technology*, 2015, 21: 171-178.

[22] 2016 International Conference on Applied System Innovation (ICASI) [online]. IEEE, 2016 [cit. 2018-05-15]. ISBN 978-1-4673-9888-6. Dostupné z: http://ieeexplore.ieee.org/document/7539833/

[23] SARRAF, Saman a Mehdi OSTADHASHEM. Big data application in functional magnetic resonance imaging using apache spark. 2016 Future Technologies Conference (FTC) [online]. IEEE, 2016, 2016, , 281-284 [cit. 2018-05-15]. DOI: 10.1109/FTC.2016.7821623. ISBN 978-1-5090-4171-8. Dostupné z: http://ieeexplore.ieee.org/document/7821623/

[24] CASTRO, Eduardo P. S., Saurabh CHAKRAVARTY, Eric WILLIAMSON, Denilson Alves PEREIRA a Edward A. FOX. Classifying Short Unstructured Data Using the Apache Spark Platform. 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL) [online]. IEEE, 2017, 2017, , 1-10 [cit. 2018-05-15]. DOI: 10.1109/JCDL.2017.7991567. ISBN 978-1-5386-3861-3. Dostupné z: http://ieeexplore.ieee.org/document/7991567/

[25] D'SILVA, Godson Michael, Azharuddin KHAN, Eric GAURAV, Siddhesh BARI a Edward A. FOX. Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework. 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL) [online]. IEEE, 2017, 2017, , 1804-1809 [cit. 2018-05-15]. DOI: 10.1109/RTEICT.2017.8256910. ISBN 978-1-5090-3704-9. Dostupné z: http://ieeexplore.ieee.org/document/8256910/

[26] ZAHARIA, Matei, et al. Apache spark: a unified engine for big data processing. Communications of the ACM, 2016, 59.11: 56-65.

[27] HONZÍK, Petr. Strojové učení. Elektronická skripta VUT Brno, 2006.

[28] SEBER, George AF, LEE, Alan J. Linear regression analysis. John Wiley and Sons, 2012.

[29] KOHAVI, Ronny; QUINLAN, J. Ross. Data mining tasks and methods: Classification: decision-tree discovery. In: Handbook of data mining and knowledge discovery. Oxford University Press, Inc., 2002. p. 267-276.

[30] KANUNGO, Tapas, et al. An efficient k-means clustering algorithm: Analysis and implementation. IEEE transactions on pattern analysis and machine intelligence, 2002, 24.7: 881-892.

[31] AGGARWAL, Charu C.; HAN, Jiawei (ed.). Frequent pattern mining. Springer, 2014.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

B	bajt
ACM	ACM Digital Library – kolekcia textov týkajúcich sa oblasti počítačovej vedy
IEEE	Institute of Electrical and Electronics Engineers – svetová technická organizácia
IoT	Internet of Things – Internet vecí
CERN	Európska organizácia pre jadrový výskum
SSD	Solid State Drive – disk pre ukladanie dát
KPS	Knowledge Processing Systems – systémy pre spracovanie znalostí
API	Application Programming Interface – rozhranie pre programovanie aplikácií
HDFS	Hadoop Distributed File System – distribuovaný súborový systém
RAID	Redundant Array of Independent Disks – pole nezávislých diskov
SQL	Structured Query Language – štrukturovaný vyhľadávací jazyk
csv	comma separated values – súbor, v ktorom sú hodnoty oddelené čiarkami
tsv	tab separated values – súbor, v ktorom sú hodnoty oddelené tabulátormi
OOP	Object-Oriented Programming – objektovo orientované programovanie
RDD	Resilient Distributed Dataset - kolekcia dát používaná v Sparku
conf	konfigurácia programu
sc	objekt JavaSparkContext
inputNumbers	vstupné hodnoty
inputRDD	RDD zo vstupných hodnôt
filteredRDD	RDD s filtrovanými záznamami
spark	objekt JavaSparkSession
skDat	slovenská databáza firiem
czDat	česká databáza firiem
line	aktuálne prechádzaný záznam RDD
kategoria	požadovaná kategória firmy
filteredSkDat	filtrovaná slovenská databáza firiem

clearedSkDat	slovenská databáza firiem s vyčisteným obsahom
nazev	názov firmy
mesto	mesto pôsobenia firmy
ulice	ulica pôsobenia firmy
ICO	Identifikačné číslo organizácie
unionDat	spojená databáza firiem
pairUnionDat	databáza vo formáte kľúč – hodnoty
values	hodnoty k jednotlivým kľúčom
sortedPairUnionDat	zoradená databáza vo vo formáte kľúč – hodnoty
countedByKey	databáza s počtom firiem v mestách
schemaString	reťazec pre vytvorenie schémy
fields	list označení stĺpcov
schema	schéma SQL tabuľky
rowRDD	RDD so záznamami vo formáte Row
record	záznam v RowRDD
wwwICO	hodnota obsahujúca webovú stránku alebo IČO firmy
rowDataFrame	výsledná databáza v dátovej štruktúre DataFrame
ICO	IČO firmy
DDoS	Distributed Denial of Service - útok obmedzujúci činnosť služby
3D	trojdimenzionálny
ML	Machine Learning - strojové učenie
firmyDf	dátový rámec so vstupnými dátami
convertedRow	konkrétny riadok dátového rámca
firmyDf2	prekonvertovaný dátový rámec
lp	Labeled Point - dátová štruktúra používaná pri strojovom učení
firmyLp	dátový rámec s upravenými dátami vo forme Labeled Pointu
splits	pole s rozdelenými hodnotami z firmyLp
SEED	premenná pre zhodu výsledkov
lr	objekt lineárnej regresie
lrModel	model lineárnej regresie
predikce	dátový rámec s výslednými hodnotami
evaluator	objekt hodnotiteľa regresie
firmySchema	schéma vstupných dát
indexer	objekt pre indexovanie textových hodnôt
indModel	model pre transformáciu na dáta s indexmi
indFirmy	dátový rámec s hodnotami s indexmi
dt	objekt rozhodovacieho stromu

dtModel	model rozhodovacieho stromu
labelKonvertor	objekt pre zmenu indexov späť do textovej podoby
predKonvertor	objekt pre zmenu indexov späť do textovej podoby
ciselnePredikce	dátový rámec s výslednými číselnými hodnotami
presnost	premenná výslednej presnosti
KMeans	algoritmus pre zhlukovanie dát
FP-growth	algoritmus pre hľadanie početných vzorov
FP-tree	model algoritmu FP-growth
GPS	Global Positioning System - lokalizačný systém
priemRow	premenná určujúca priemer hodnôt
stdRow	premenná určujúca štandardnú odchýlku
bcPriemRow	globálna premenná určujúca priemer hodnôt
bcStdRow	globálna premenná určujúca štandardnú odchýlku
s	štandardizovaná premenná zemepisnej výšky
v	štandardizovaná premenná zemepisnej šírky
kmeans	objekt algoritmu zhlukovania dát
transakce	list s jednotlivými transakciami
predmetyDf	dátový rámec s položkami nákupu
FPmodel	model algoritmu početných vzorov