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

Diplomová práce

Experimenty s Apache Mahout

Plzeň 2015 Lukáš Hain

Prohlášení

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

V Plzni dne 14. května 2015

Lukáš Hain

Abstract

This thesis focuses on the Apache Mahout toolkit. The goal is to create a tutorial and verify its usability on some tasks from natural language processing (NLP) field.

The text shows a solution to three NLP problems: classification of e-mails in a support forum, extracting discussion feeds from web pages and clustering semantic vectors. All these examples are accompanied with rich explanations that cover data loading, training, testing and using the trained models.

Very promising results were achieved for the first and third task. Classification of emails achieved approximately 91% accuracy. Clustering into 25 000 clusters based upon 400 000 vectors finished in 43 hours on a common laptop.

Tato práce je zaměřena na knihovnou Apache Mahout. Úkolem je vytvoření návodu a ověřit její použitelnost na úlohách z oblasti zpracování přirozeného jazyka (NLP).

V textu je znázorněno řešení tří problémů z oblasti NLP: klasifikace emailů z podpory, extrakce diskuzních příspěvků z webových stránek a shlukování sémantických vektorů. Všechny tyto úlohy jsou doprovázeny podrobným popisem, který zahrnuje načítání dat, trénování, testování a použití natrénovaného modelu.

Velice slibných výsledků bylo dosaženo v první a třetí úloze. Klasifikace emailů dosahuje úspěšnosti kolem 91%. Shlukování do 25 000 shluků, z 400 000 vektorů trvalo 43 hodin na běžném notebooku.

Obsah

1 Úvod

Cílem této práce je vytvoření uceleného návodu pro knihovnu Apache Mahout. Zatím neexistuje mnoho literatury na toto téma a často využívají stále stejné ukázkové příklady. Úkolem je tedy prozkoumat možnosti knihovny, sestavit stručný návod k použití a stanovit, do jaké míry je knihovna použitelná v praxi.

V úvodu je popsána oblast strojového učení. Jsou zde popsány různé typy strojového učení. Ke každému typu je uvedeno několik základních algoritmů.

Zbytek práce je věnován návodu a řešeným příkladům. Jsou zde popsány základní principy knihovny a implementované algoritmy. Dále jsou popsány oblasti úloh, které lze pomocí této knihovny řešit. Během výzkumu byly zadány vedoucím práce tři úlohy z oblasti NLP. Tyto úlohy jsou následně použity jako výukové příklady a je na nich demonstrována funkce knihovny Mahout.

První úloha vznikla na požadavek CIVu, jako prostředek pro usnadnění a výrazného urychlení činnosti podpory. Slouží k rozpoznávání obsahu mailů, které rozřazuje do předem daných skupin.

Druhá úloha využívá existující program FeedExtractor, který slouží k rozpoznávání komentářů na webových fórech. Úkolem je výměna stávající knihovny pro strojové učení za knihovnu Mahout.

Poslední úloha se týká shlukování sémanticky navzájem podobných slov. Ve zdrojových datech je 400 000 slovních vektorů o dimenzi 300, které obsahují kontextovou informaci o slovech. Je experimentálně ověřeno, že sémanticky podobná slova mají podobné vektory. Spojením slov s podobnými vektory získáme shluky slov s podobným významem.

V závěru jsou zhodnoceny výsledky a přínosy této práce.

2 Strojové učení

Strojové učení je jedna z disciplín umělé inteligence. Jde o schopnost programů učit se nebo vykonávat věci, ke kterým nebyly přímo naprogramovány.

2.1 Motivace

Na začátku bude zmíněno, co je vlastně strojové učení, kde pomáhá dnes a kde může pomoci v budoucnosti.

2.1.1 BigData

V dnešní době je hodně diskutované téma tzv. BigData. Rychlost generování dat aktuálně roste exponenciálně a podle předpokladů[1] budeme v roce 2020 mít cca 35 ZB (zettabyte = 1 000 000 000 000 TB) dat, která budou z části uložena v „cloudu“. Nově generovaná data již z podstatné části nejsou strukturovaná (obrázky, videa, CT skeny ...). Například na servery YouTube[2] je každou minutu nahráno 300 hodin videa a toto číslo se stále zvyšuje. Dalšími významnými producenty dat jsou meteorologové nebo astronomové. I komerční sektor má mnoho velkých přispěvatelů, patří k nim komunikace, energetika nebo farmacie. Část takto získaných dat je nutno zpracovávat v reálném čase, jinak ztratí smysl.

2.1.2 Jak může pomoci strojové učení

Je dobré data zpracovat rychle a vytvořit nějaký „očividný“ výsledek, ale takové obrovské množství informací určitě obsahuje i něco víc. Na tato data lze pak pohlížet z různých perspektiv a objevovat nové informace a řešení

doposud neřešitelných problémů. Například zdravotní informace budou uloženy na servech spolu s postupem léčby a jejími výsledky. Aplikací některých algoritmů strojového učení a umělé inteligence bude například možné objevit doposud nepoznané „vedlejší“ efekty některých léčiv a nepřímo vyvinout léky nebo změnit léčebné postupy.

Dále bude možné přecházet dopravním zácpám a nehodám. Okolo roku 2020 budou pravděpodobně existovat automobily, které nebudou potřebovat řidiče a samy najdou nejvhodnější cestu vzhledem k dopravní situaci. Již v blízké budoucnosti se začnou zavádět technologie pro komunikaci mezi automobily a některé automobilky[3] oznámily dostupnost této technologie již na rok 2016/2017.

2.1.3 Strojové učení dnes

Většina uživatelů počítačů používá strojové učení mnohokrát každý den, aniž by si to uvědomili. Dokonce i při použití něčeho tak "zastartalého", jako je papírová pošta, jsou využívány principy strojového učení pro rozpoznávání PSČ. K tomuto účelu jsou často trénovány neuronové sítě. Dalším příkladem je rozpoznávání řeči v moderních telefonech (Siri, Cortana, Google Now ...) a odpovídající reakce hlasového asistenta. Nebo překlad z jednoho jazyka do jiného. Dokonce i během nakupování, když se na webové stránce vyskytuje oddíl jako „Mohlo by se Vám líbit“ nebo podobné, tak jsou v pozadí využity algoritmy a principy strojového učení.

Strojové učení pomáhá a bude čím dál významněji pomáhat i v dalších oblastech, jako je věda, soudnictví, vzdělání, ekonomie a další. Je to jedna z oblastí informačních technologií, která má v nejbližších letech velkou budoucnost.

2.2 Přístupy

V oblasti strojového učení existují čtyři základní přístupy, jak stroj učit.

- Učení s učitelem (Supervised learning)
- Učení bez učitele (Unsupervised learning)
- Kombinace předchozích (Semi-supervised learning)
- Zpětnovazebné učení (Reinforcement learning)

Každá z těchto oblastí bude nyní krátce zmíněna.

2.2.1 Učení s učitelem

Jde o nejpoužívanější[4] přístup ve strojovém učení. Princip spočívá v tom, že je dostupná přiměřeně velká množina dat, která byla označena. To znamená, že každý prvek množiny má u sebe informaci, do které třídy patří. Část dat bude použita pro trénování algoritmu a na zbytku se bude testovat jeho kvalita.

2.2.2 Učení bez učitele

Jelikož data dodaná tomuto algoritmu nejsou nijak označena, snaží se algoritmus najít „skryté“ struktury a informace. Učení bez učitele se používá například při shlukování (clusteringu). Algoritmu se předají vstupní data, u kterých předem nemusí být znám počet ani obsah skupin, které jsou výstupem.

2.2.3 Kombinace předchozích

Používá se, pokud je na počátku dostupná pouze malá trénovací množina ručně označených dat. Algoritmus se natrénuje na této malé množině a dál se trénuje na neoznačených datech. Používá se, pokud je z nějakých důvodů nemožné nebo nákladné označit větší množství dat. Často se to stává například ve zdravotnictví.

2.2.4 Zpětnovazebné učení

Podstatně se liší od předchozích dvou přístupů. Využívá poznatků psychologie, specificky pozitivních a negativních stimulů. Algoritmus se snaží během času dosáhnout co nejvyššího počtu pozitivních hodnocení. Je úzce spjato s teorií rozhodování (ve statistice) a teorií řízení (v kybernetice).

2.3 Klasifikace

Pravděpodobně nejpoužívanější oblast strojového učení. Jde o problém identifikace (třídění) dat (dokumenty, obrázky ...) do předem známých tříd. Klasickým příkladem je spam filtr, který dělí příchozí poštu do dvou kategorií. Na tomto příkladu staví většina knih a skript o strojovém učení.

Klasifikace využívá principů učení s učitelem (popř. kombinaci učení s a bez učitele). V případě spam filtru je pro vytvoření počátečního modelu nutné mít dostupnou množinu vyžádané a nevyžádané pošty. Tuto množinu je pak nutné ručně označit, aby na ní mohl algoritmus trénovat.

Algoritmus sloužící ke klasifikaci se nazývá klasifikátor. Klasifikátor má za úkol rozdělit množinu dat podle sledovaných vlastností (features) do jednotlivých tříd. Tyto vlastnosti mohou být libovolné. Na jejich výběru a zpracování však stojí celá kvalita klasifikátoru. Sledované vlastnosti musí být tedy

vybírány tak, aby co nejlépe reprezentovali jednotlivé třídy.

Klasifikátor pracuje v N rozměrném prostoru, kde N je počet sledovaných vlastností. Pokud tedy sleduji pouze jednu vlastnost, pak je dělícím prvkem bod na přímce. Pokud jsou sledované prvky dva, pak je dělícím útvarem přímka (křivka) v rovině. Dále by to byla rovina v prostoru atd. Od tohoto rozměru dál je to jen těžko představitelné, ale výpočty fungují stále stejně. Výstupem klasifikátoru je tedy funkce, která co nejlépe vymezuje tyto třídy. Tato funkce pak pro každý vstupní vektor vlastností vrátí identifikátor třídy, do které příslušný vektor patří.

2.3.1 Bag of words

[5]Je model reprezentace dokumentu jako vektoru statické délky. Reprezentuje dokument jako množinu slov s jejich četností. To znamená, že jsou ukládána pouze samotná slova a počet, kolikrát se tato slova v dokument vyskytla. Neudržuje se pořadí slov ani kontext, ve kterém se slova vyskytla. Například pokud by byly vstupem dokumenty:

(a) `Problém byl přednesen na poradě. Řešení bude prezentováno na příští poradě.`

(b) `Dovolenou probereme na poradě.`

Nejdříve je vytvořen slovník unikátních slov:

1. `problém`
2. `byl`
3. `přednesen`
4. `na`

5. `poradě`
6. `řešení`
7. `bude`
8. `prezentováno`
9. `příští`
10. `dovolenou`
11. `probereme`

Slovník obsahuje jedenáct unikátních výrazů. Velikost výstupních vektorů tedy bude jedenáct. Výše zmíněné dokumenty mají následující vektorovou reprezentaci:

(a) `1, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0`

(b) `0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1`

Pro každý výskyt slova v dokumentu je na příslušný index, určený pořadím ve slovníku přičtena jednička.

Váha slov

Při klasifikaci dokumentů je nutné přidělovat jednotlivým slovům váhy. Je rozdíl, jestli se v dokumentu vyskytují slova jako "a, nebo, když", která se vyskytují v téměř každém dokumentu nebo slova jako "NLP, klasifikace, regrese". Druhá množina se jistě bude objevovat mnohem méně často a zároveň téměř výhradně v textech na dané téma. Proto je nutné dát těmto slovům vyšší váhové ohodnocení.

Pro tyto účely se často používá tzv. TF-IDF (Term frequency-inverse document frequency). Pro každé slovo je vypočítán následující vzorec:

$$TFIDF(t, d, D) = \frac{|d(t)|}{|d|} \cdot log\left(\frac{|D|}{|D(t)|}\right) \tag{2.1}$$

Kde `|d(t)|` je počet výskytů slova `t` v dokumentu `d` a `|d|` vyjadřuje počet slov v dokumentu `d`. Výraz `|D|` vyjadřuje celkový počet dokumentů a `|D(t)|` reprezentuje počet dokumentů, ve kterých se vyskytuje slovo `t`.

Příklad Nejčastějším slovem dle [6] je slovo "a". Pokud by se slovo "a" vyskytovalo v dokumentu sedmkrát, dokument měl celkem 1000 slov a celkem bylo dostupných 500 dokumentů, kde "a" by bylo v 480 z nich, byla by váha slova "a" následující:

$$TFIDF = \frac{7}{1000} \cdot log\left(\frac{500}{480}\right) = 0.000124 \tag{2.2}$$

Jak je vidět, váha je velmi nízká. Nyní je uvažováno slovo "NLP", které je dle [6] mnohem méně časté. Pro stejný dokument, kde se toto slovo vykytuje pětkrát a celkem se vyskytuje ve čtyřech dokumentech je váha následující:

$$TFIDF = \frac{5}{1000} \cdot log\left(\frac{500}{4}\right) = 0.01048 \tag{2.3}$$

Je vidět, že slovo NLP má v uvažovaných dokumentech zhruba 100x vyšší váhu. Takto se oddělují pro klasifikaci významná slova od těch méně významných.

2.3.2 Naivní Bayes

Prvním příkladem klasifikačního algoritmu je Naivní Bayes. Naivní Bayes se v NLP často používá pro klasifikaci textových dokumentů. Dokumenty jsou reprezentovány metodou bag of words viz 2.3.1. K určení třídy používá Naivní Bayes Bayesovu větu.

$$P\left(T|D\right) = \frac{P\left(D|T\right)P\left(T\right)}{P\left(D\right)} \tag{2.4}$$

P(T|D) je podmíněná pravděpodobnost že nastane jev T za podmínky že nastal jev D. Pokud T je tedy třída a D je dokument, pak vzorec vyjadřuje s jakou pravděpodobností patří dokument D do třídy T. Pokud má být nový dokument zařazen do nějaké třídy, je nutné vypočítat výše zmíněnou pravděpodobnost pro každou třídu. Vybrána bude třída s nejvyšší pravděpodobností.

Výhodou tohoto algoritmu je výpočetní nenáročnost, jednoduchá implementace a schopnost učit se i během používání modelu. Používá se např. jako spam detektor nebo k tématické klasifikaci článků.

2.3.3 Lineární regrese

Je dalším zástupcem klasifikačních algoritmů. K řádné funkci tohoto algoritmu je nutné sestavit funkci, která každý klasifikovaný objekt převede na tzv. vektor featur. Volba featur má zásadní vliv na úspěšnost klasifikace. Pro jednoduchý dvourozměrný případ, vypadá rovnice takto:

$$h_\theta\left(x\right) = \theta_0 + \theta_1\left(x\right) \tag{2.5}$$

Algoritmus lineární regrese musí určit parametry θ_1 a θ_2 tak, aby došlo

k co nejpřesnějšímu oddělení tříd. Proměnná x je hodnota, podle které se má rozhodnout, do které třídy objekt patří (např. při určování ceny bytu, podle jeho plochy by to byla plocha). Aby parametry mohly být vypočteny, je nutné mít hodnotící funkci, která řekne jak dobré nebo špatné jsou zvolené parametry.

Cenová funkce

Pro tyto potřeby je často využívaná metoda nejmenších čtverců, kdy se počítá rozdíl mezi funkční hodnotou regresní přímky a správnou (trénovací) hodnotou. Výslednou hodnotu se snažíme minimalizovat. Hodnotící funkci lze vyjádřit vzorcem:

$$\underset{\theta_1,\theta_2}{argmin}\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})+y^{(i)}\right)^2 \tag{2.6}$$

Toto je cenová funkce vypočtená metodou nejmenších čtverců. Cenovou funkci lze zapsat v následujícím tvaru.

$$\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})+y^{(i)}\right)^2 = J(\theta_1,\theta_2) \tag{2.7}$$

Gradientní sestup

Pro minimalizaci funkcí existuje metoda zvaná gradientní sestup, která v každém kroku vyřeší derivaci příslušné funkce a podle výsledku pokračuje dále ve výpočtu.

Postup:

1. Na počátku jsou zvoleny hodnoty pro parametry θ_1 a θ_2 např. nula, nebo náhodné číslo.

2. Dále jsou určovány nové hodnoty parametrů ϑ1 tak, aby se snižovala funkční hodnota cenové funkce J(θ_1,θ_2). Tento krok se opakuje, dokud funkční hodnota nedosáhne (lokálního) minima.

To samé v pseudokódu:

```
Dokud není dosaženo konvergence {
```

$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta_1, \theta_2)$ `(pro j = 0 a j = 1)`

```
}
```

α = míra učení (learning rate)

Je-li α příliš malé, pak bude sestup trvat dlouho. Naopak, pokud bude příliš velké, nemusí funkce nalézt minimum a může dokonce divergovat. Neexistuje univerzálně správná míra učení. Vždy je potřeba vyzkoušet na zadaném příkladu.

2.3.4 Logistická regrese

Dalším příkladem klasifikačního algoritmu je logistická regrese. Logistická regrese může být dvou typů binominální, kdy data klasifikuje do dvou množin (např. ano/ne), nebo multinomiální, kde existuje více rozpoznávaných stavů. Oproti lineární regresi se změní hypotéza na.

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.8}$$

Oproti lineární regresi, jejíž výstupem je reálné číslo, je výstupem pravděpodobnost třídy, do které objekt patří. V nejjednodušším případě je uvažována pouze jedna výstupní proměnná, která se nachází v intervalu <0, 1>.

2.4 Shlukování

Je příkladem algoritmu bez učitele. Jeho úkolem je shlukovat objekty s podobnými vlastnostmi do shluků (clusterů). Problém je, že algoritmus nemá vektory nijak označeny výstupním shlukem atp. Smysl algoritmu bude popsán na příkladu.

Mnoho uživatelů internetu čte zprávy pouze z jednoho zdroje. Pokud vyjde článek o nějakém zajímavém tématu, navštíví pouze jeden web, který může obsahovat nepřesnou informaci, nebo je příliš stručný. Existují ovšem postupy, které další články o stejném tématu najdou. Toto je účel služby Google News. Shromáždit na jednom místě články o stejném tématu, aby čtenář mohl informace porovnat a utvořit si tak lepší obraz skutečnosti. Podstatou je, že Google nemá skupinu zpravodajců, kteří články sdružují. Předem zároveň není známo jaké články vyjdou a jaké události nastanou.

2.4.1 K-means

Jednoduchý, ale velmi známý algoritmus shlukování. Ze zadaných dat, reprezentovaných pomocí d-dimenzionálních vektorů, se snaží vytvořit k shluků, které budou mít minimální sumu vnitřních čtverců dle vzorce:

$$\underset{S}{argmin} \sum_{i=1}^{k} \sum_{x \in S_i} \| x - \mu_i \|^2 \tag{2.9}$$

Kde S je množina nalezených shluků, tato množina má velikost k.μ je geometrický střed shluku.

Algoritmus se skládá ze dvou základních kroků. Prvním je přiřazení každého prvku z datové množiny do právě jednoho clusteru. Prvek bude přiřazen do clusteru, od jehož středu má nejmenší vzdálenost (viz minimální čtverec výše). Ve druhém kroku jsou vypočítány nové geometrické středy shluků.

Nový střed pro shluk je vypočítán jako průměr všech prvků přiřazených do daného shluku v předchozím kroku.

$$\mu_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2.10}$$

Jednou z největších nevýhod tohoto algoritmu je nutnost specifikovat k (počet shluků) při spuštění algoritmu. Tento problém částečně řeší Canopy clustering.

2.4.2 Canopy clustering

Algoritmus se často používá pro předzpracování dat pro k-means (nebo pro jiný algoritmus). Algoritmus má dvě nastavitelné hodnoty označované T1 a T2, kde mezi těmito hodnotami platí vztah T1 > T2.

Pseudokód:

1. Vyber náhodný vektor ze seznamu dostupných vektorů, označ ho jako střed nové canopy a odeber ho ze seznamu.
2. Projdi všechny vektory v seznamu dostupných vektorů a spočítej jejich vzdálenost od zvoleného středu. Pokud je tato vzdálenost menší než T1, pak vektor přidej do aktuální canopy. Pokud je vzdálenost zároveň menší než T2, odeber ho ze seznamu dostupných vektorů.
3. Opakuj body 1 a 2, dokud je množina dostupných vektorů neprázdná.

Jak je vidět z pseudokódu, jeden vektor může být součástí více canop. Z toho vyplývá, že počet shluků v k-means bude větší nebo stejný jako v případě canopy clusteringu. Poslouží však jako dobrá dolní hranice.

3 Mahout

Knihovna pro škálovatelné strojové učení. Je vyvíjena jako open-source v Apache Software Foundation. Mahout je dle oficiálních stránek[7] používán ve více než 30 společnostech a projektech. Mezi společnosti, které používají Mahout patří i taková jména jako Adobe, Linked.in, Twitter nebo Yahoo. Hlavní výhoda Mahoutu oproti jiným knihovnám pro strojové učení (Weka[1], R[2], ...) je právě škálovatelnost. Pokud je trénovací množina dat obsáhlá, projeví se silné stránky Mahoutu. Naopak mezi nevýhody patří relativně malá komunita kolem projektu a menší počet implementovaných algoritmů.

Pro dosažení škálovatelnosti využívá Mahout framework Hadoop a jeho implementaci MapReduce. Vývojáři jsou odhodláni výkon algoritmů dále zlepšovat, a tak se rozhodli nahradit stávající Hadoop, novým projektem Apache Spark. Dne 25. dubna 2014 vydali prohlášení[8], že nebudou přijímány žádné nové algoritmy na bázi MapReduce. Stávající algoritmy však v knihovně zůstanou.

3.1 Hadoop

Je framework umožnující provádět distribuované výpočty na rozsáhlých datech s využitím jednoduchých programovacích modelů vyvíjený společností Apache. Je vytvořen na základě článku[9] publikovaném společností Google v roce 2004. Dnes je Hadoop pravděpodobně nejznámější MapReduce framework na světě. Dle Apache Wiki je využíván ve více jak 200 společnostech a projektech[10] jako např. Facebook, Twitter nebo Yahoo. Skládá se z několika základních částí:

- Hadoop Common – Obsahuje základní a podpůrné funkce pro ostatní

[1] Weka: Data Mining Software, `http://www.cs.waikato.ac.nz/ ml/weka/`
[2] The R Project for Statistical Computing, `http://www.r-project.org/`

části.

- Hadoop Distributed File System (HDFS) – Distribuovaný souborový systém.
- Hadoop YARN – Plánovač a správa zdrojů.
- Hadoop MapReduce – Systém využívající YARN, vhodný pro zpracování velkého množství dat.

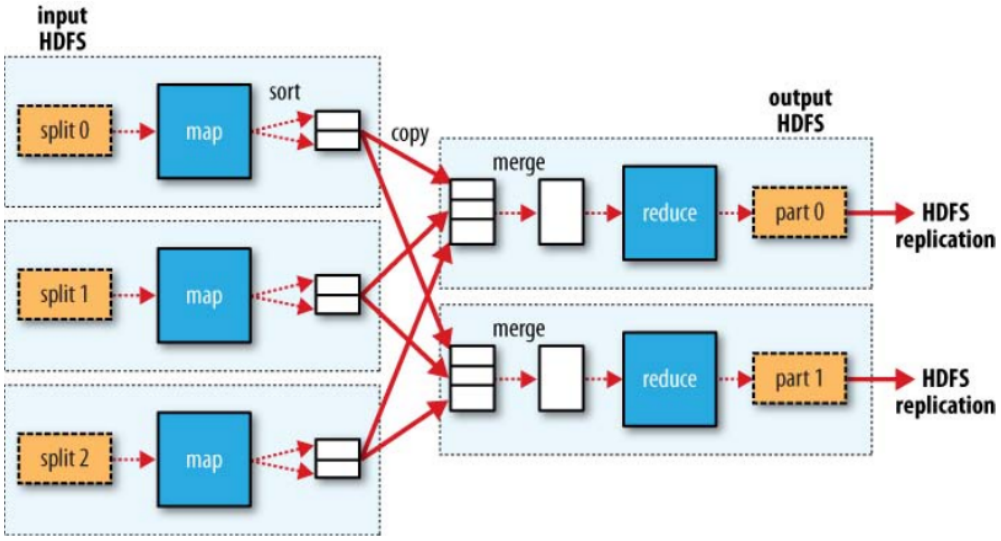
3.1.1 MapReduce

Hadoop podporuje MapReduce, jako systém pro efektivní zpracování velkého množství dat (až v řádech PB). Tento model se skládá ze dvou fází:

- Map – Namapování vstupních dat do formátu <klíč, hodnota>.
- Reduce – Třídění dat dle klíče mezi jednotlivé PC v clusteru a jejich závěrečné sloučení do jednoho souboru.

Klíčovou myšlenkou MapReduce je možnost rozdělit vstup mezi více strojů a tyto vstupy nezávisle na sobě zpracovat ve funkci Map. Data vystupující z funkce Map jsou dle klíče seřazena. Výsledná data se sloučí a jsou uložena jako textový soubor do HDFS.

MapReduce vznikl, protože mnoho úloh potřebuje výpočetní výkon clusteru a zároveň je lze reprezentovat pomocí modelu znázorněného na obrázku 3.1. Vývojáři tedy pro každou úlohu nemusejí psát velké množství kódu, který přímo nesouvisí s jejich problémem. Problémy, jako distribuce dat na jednotlivé nody, zajištění lokality výpočtu, replikace dat, výpadky nodů nebo konečné sloučení výsledků za něj řeší Hadoop. Jediné, co je nutné naprogramovat je obsah funkcí Map a Reduce. Je tedy minimalizován čas potřebný na vývoj a riziko vytvoření chyby.



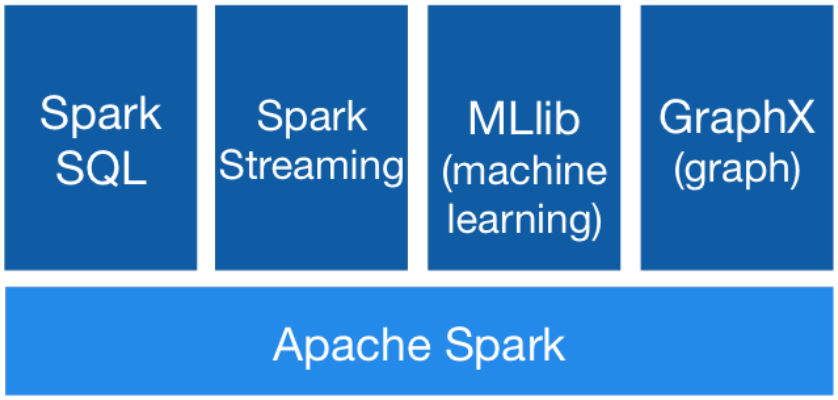
Obrázek 3.1: Průběh MapReduced algoritmu.

3.2 Spark

Hadoop samozřejmě není jediným frameworkem pro distribuované výpočty. Dalším příkladem je Apache Spark. Spark je vyvíjen od roku 2009. V roce 2013 se stal součástí Apache Foundation. O rok později se z něj stal Top-Level projekt společnosti Apache. V Listopadu 2014 se podařilo stanovit nový rekord[11] v rychlosti řazení dat a to právě pomocí Apache Spark (4.27 TB/min).

Spark oproti Hadoopu slibuje až 100x vyšší výkon a to zejména v iterativních výpočtech. Umožňuje totiž pracovat s daty mnohem efektivněji. Spark využívá Resilient Distributed Datasets (RDD), se kterým lze provádět nejen operace Map a Reduce, ale také mnoho dalších např. filter, union, intersection, group by nebo distinct. V aktuální verzi (1.2) je více než 30 možných akcí. Jak je vidět z názvů metod, s RDD se pracuje podobně jako s klasickou SQL databází. Na rozdíl od MapReduced algoritmu není potřeba mezivýsledky ukládat do HDFS, ale lze vytvořit efektivní pipeline. Další urychlení spočívá v možnosti určit, kde budou uchovávána, jestli na disku

nebo v paměti RAM.



Obrázek 3.2: Struktura Sparku.

Jak je vidět na obrázku 3.2, Spark se skládá z pěti částí. Jelikož je to relativně nový projekt, je podstatná část stále v rané fázi vývoje. A dokonce i části, které jsou již mimo své první alfa verze se mohou dost podstatně měnit.

3.3 Přehled algoritmů

	Single Machine	MapReduce	Spark
Collaborative Filtering with CLI Drivers			
User-Based Collaborative Filtering	♦		♦
Item-Based Collaborative Filtering	♦	♦	♦
Matrix Factorization with ALS	♦	♦	
Matrix Factorization with ALS on Implicit Feedback	♦	♦	
Weighted Matrix Factorization, SVD++	♦		
Classification with CLI Drivers			
Logistic Regression - trained via SGD	♦		
Naive Bayes / Complementary Naive Bayes		♦	♦
Random Forest		♦	
Hidden Markov Models	♦		
Multilayer Perceptron	♦		
Clustering with CLI Drivers			
Canopy Clustering	depracated	depracated	
k-Means Clustering	♦	♦	
Fuzzy k-Means	♦	♦	
Streaming k-Means	♦	♦	
Spectral Clustering		♦	

Tabulka 3.1: Přehled algoritmů doporučení, klasifikace a shlukování implementovaných v Mahoutu.

3.4 Sekvenční soubor

Sekvenční soubor (SequenceFile) je struktura definovaná ve frameworku Hadoop a Mahout ji využívá jako vstupní a výstupní formát MapReduced algoritmů. Struktura je obdobou mapy v Jave. Skládá se z dvojice klíč-hodnota, kde klíč musí být unikátní. Formát sekvenčního souboru je následující:

```
/Klíč           Hodnota (např. vektor)
příklad:
/101054         [5.3218, 4.6548, -0.1378]
```

Dokument lze převést do formy sekvenčního souboru dvěma způsoby:

1. automaticky, využitím utility `seqdirectory`
2. ručně, vytvořením programu v Jave

3.4.1 Automatický přístup

Pokud je na vytvoření vektorů použíta utilita `seqdirectory`, je vytvořen sekvenční soubor ve formátu <Text, Text>. Klíč bude v tomto případě obsahovat jména všech podsložek, včetně názvu koncového souboru. Hodnotou bude obsah koncového souboru. Proto musí být v každém souboru pouze jeden záznam.

3.4.2 Manuální přístup

Manuální přístup se používá, pokud jsou data v takovém tvaru, že je nejde snadno převést do výše zmíněné souborové struktury. Například pokud jsou uložena v XML, HTML nebo dokonce ve formě binárního souboru.

Pro vytvoření sekvenčního souboru v Jave slouží třída `SequenceFile.Writer`.

`Writer` obsahuje metodu `append(Writable key, Writable value)`, která na konec sekvenčního souboru přidá novou dvojici klíč-hodnota.

4 Doporučení

První ze tří hlavních oblastí, které Mahout pokrývá, je doporučení. Tato skupina algoritmů je dnes velmi populární. Doporučovací engine Mahoutu je využíván mnoha velkými společnostmi, jako je Foursquare[12] nebo AOL[7].

Doporučení je sada algoritmů, jež dokážou předpovědět, které objekty (knihy, filmy, vtipy ...) se budou uživateli líbit na základě jeho hodnocení dalších objektů a porovnání těchto hodnocení s dalšími uživateli.

4.1 Vstupní data

Vstupní data by měla být v tomto formátu (Pokud tento formát dodržen není, je nutné napsat vlastní parser.):

```
<long>,<long>,<float>
```

První číslo vyjadřuje unikátní identifikátor osoby, která hodnotila předmět. Druhé číslo je identifikátor předmětu, který byl hodnocen. Poslední je samotné hodnocení. Tato hodnota je nepovinná a při její neexistenci se předpokládá využití true/false doporučení (viz dále). Příklad:

```
547,84,5
```

Uživatel s ID 547 přiřadil položce s ID 84 hodnotu 5 (hvězdiček, bodů ...).

4.2 True/false doporučení

Ne vždy je u doporučení možné použít číselnou hodnotu jak moc se uživateli objekt líbí/nelíbí. Příkladem mohou být webové stránky, které se snaží

doporučit další články pomocí historie již přečtených. Pokud uživatel články nijak nehodnotí, může web pracovat pouze s tím, že uživatel článek navštívil. Existuje tedy pouze dvojice Uživatel ID, Předmět ID. Tato skutečnost nemusí být nutně negativní.

4.3 Dataset

Pro následující příklady je využit volně dostupný dataset MovieLens[13]. Konkrétně jsou použita data ze souboru MovieLens 100k. Vybraný dataset obsahuje 100 000 hodnocení, což je pro demonstrační účely dostatečná množina dat. Samotný soubor s hodnocením je v archivu pojmenována jako „u.data“. Data byla převedena z původního formátu do formátu definovaném v kapitole 4.1.

4.4 User-based doporučení

Prvním ze dvou přístupů k doporučením je tzv. user-based doporučení. Jde především o to, jak jsou nalezeny doporučené předměty. V tomto případě jsou vyhledáváni uživatelé, kteří mají podobný vkus (jejich hodnocení jsou podobná). Předpoklad je, že když stejné předměty hodnotili podobně, budou se jim pravděpodobně líbit stejné věci.

4.4.1 Algoritmy doporučení

Ve verzi Mahoutu 0.9 existuje několik algoritmů doporučení.

Generic User/Item Based Recommander

Používá se v případě, že existuje číselné ohodnocení každé dvojice uživatel-předmět. Je využitelný ve většině případů.

Generic Boolean Pref User/Item Based Recommender

Pokud není dostupné číselné ohodnocení prvků, je možné použít boolean doporučení. To znamená, že neexistuje číselná hodnota, která popisuje hodnocení předmětu. Existuje pouze vazba uživatel-předmět.

Item Average Recommender

Tato implementace vrátí pro každý předmět jeho průměrné ohodnocení z celého datasetu. Není zde brán ohled na jakoukoliv podobnost nebo blízkost předmětů a uživatelů. Jeho výhodou je rychlost, nevýhodou je to, že neposkytuje použitelná doporučení. Používá se k testovacím účelům.

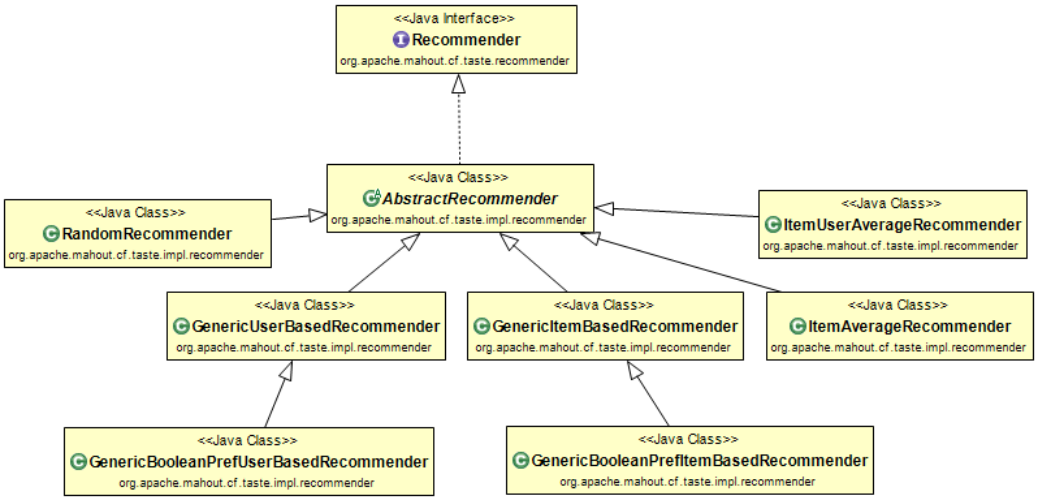
Item User Average Recommander

Stejné jako Item Average Recommander s tím rozdílem, že přizpůsobuje jednotlivé návrhy průměru uživatele. Např. Průměrné hodnocení předmětu P je 3,5. Uživatel U má průměrné hodnocení předmětů 3,8 a celkové průměrné hodnocení uživatelů je 3,6. Uživatel U má o 0,2 vyšší průměrné hodnocení než celek. Pak bude pro uživatele U a předmět P vrácena preference 3,5 + 0,2 = 3,7.

Random Recommander

Vrací náhodně doporučené předměty, včetně náhodně doporučených hodnot. Pro jednoho uživatele tedy může doporučit jakýkoliv předmět. Tento předmět se může v doporučení vyskytnou několikrát a pokaždé s jinou hodnotou.

Všechny tyto třídy jsou potomky abstraktní třídy `AbstractRecommender` a první dvě navíc implementují rozhraní `UserBaseRecommender` resp. `ItemBasedRecommender`.



Obrázek 4.1: UML diagram tříd doporučení.

4.5 Základní doporučení

Nejjednodušší možný program pro doporučení vypadá následovně.

Listing 4.1: Základní příklad doporučení.

```
DataModel model = new FileDataModel(new File("data.csv"));
UserSimilarity similarity = new
    EuclideanDistanceSimilarity(dataModel);
```

```
UserNeighborhood neighborhood = new NearestNUserNeighborhood(10,
    similarity, dataModel);
Recommender rec = new GenericUserBasedRecommender(dataModel,
    neighborhood, similarity);
```

Jelikož ve vstupních datech byl dodržen formát, který je zmíněn v kapitole 4.1, je možné použít připravenou třídu `FileDataModel`, která načte soubor hodnocení. Následuje definice objektu, který určuje podobnost mezi uživateli dle jejich hodnocení jednotlivých objektů (viz kapitola 4.5.1). Dále je vytvořen objekt sousedství, který pomocí podobnosti mezi uživateli vybere pouze relevantní (viz kapitola 4.5.2). V tomto případě je to deset nejpodobnějších uživatelů. Nakonec je vytvořen objekt `Recommander`. Tento objekt lze pro doporučení použít následujícím příkazem.

```
rec.recommend(ID uživatele, počet doporučených prvků);
```

Příklad výstupu pro `rec.recommend(10, 5)`:

```
RecommendedItem[item:880, value:5.0]
RecommendedItem[item:316, value:5.0]
RecommendedItem[item:292, value:4.591372]
RecommendedItem[item:125, value:4.5089555]
RecommendedItem[item:299, value:4.5]
```

Jak je vidět bylo vygenerováno 5 doporučení pro uživatele s ID 10. K těmto doporučením jsou přiřazeny i hodnoty, kterými by (dle algoritmu) uživatel předměty ohodnotil.

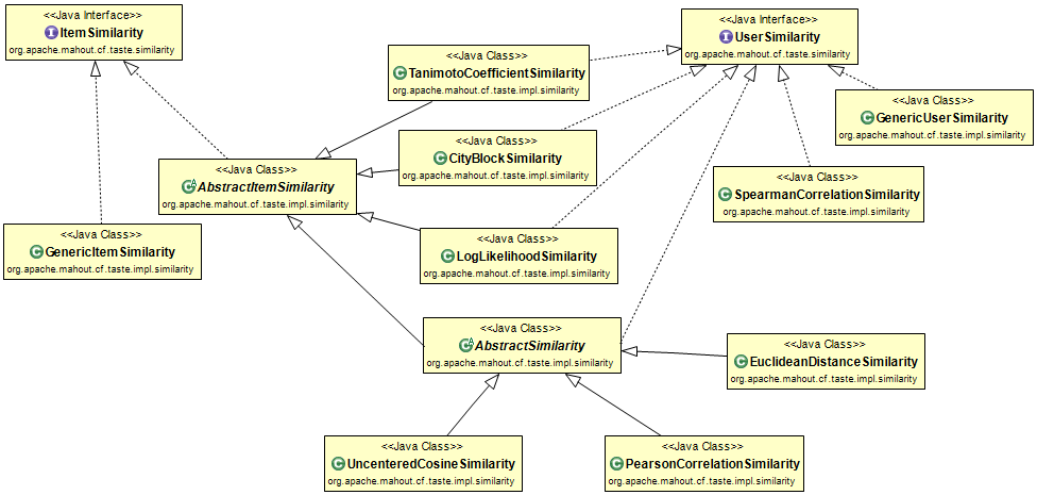
4.5.1 Podobnost

Jelikož řešené úlohy obecně jsou unikátní a každá má jiné specifické vlastnosti, je nutné mít nějaké nástroje, které tyto specifika respektují. Proto je v Mahoutu definováno hned několik algoritmů, jak měřit podobnost (simila-

rity) dvou uživatelů.

- `GenericUserSimilarity`
- `EuclideanDistanceSimilarity`
- `LogLikelyHoodSimilarity`
- `PearsonCorrelationSimilarity`
- `SpearmanCorrelationSimilarity`
- `TanimotoCoefficientSimilarity`
- `UncenteredCosineSimilarity`
- `CityBlockSimilarity`

Po provedení těchto algoritmů je určena podobnost mezi každými dvěma uživateli z datasetu.



Obrázek 4.2: UML diagram tříd podobnosti.

4.5.2 Sousedství

Využívá data podobnosti a ty dle parametrů specifických pro každou implementaci využije k získání nejpodobnějších uživatelů pro každého uživatele. V Mahoutu existují dvě implementace:

- `NearestNUserNeighborhood` – Vybere N nejpodobnějších uživatelů.
- `ThresholdUserNeighborhood` – Vybere pouze uživatele jejichž podobnost je vyšší něž zadaná hodnota (0 – 1).

Po provedení každého z těchto algoritmů je každému uživateli přiřazen seznam uživatelů, kteří jsou mu podle zadaného algoritmu nejpodobnější.

4.6 Item-based doporučení

Druhý možný přístup k doporučení. Oproti user-based doporučení, kde se řeší problém nalezení uživatele s podobným hodnocením a zjištění co mají/nemají rádi. Zde je na problém nahlíženo z jiného úhlu. Jsou zde hledány předměty podobné těm, které má dotyčný rád a ty jsou mu doporučeny. Nehledá se tedy podobnost mezi uživateli, ale mezi předměty, proto označení item-based.

4.7 Testování

Aby bylo možné zjistit, zda navržená doporučení odpovídají skutečnosti, je nutné testování. Mahout obsahuje velmi snadno použitelný nástroj pro evaluaci doporučení. Evaluační rozhraní se jmenuje `RecommenderEvaluator` existují dvě implementace tohoto rozhraní, které je možné využít:

- `AverageAbsoluteDifferenceRecommenderEvaluator`
- `RMSRecommenderEvaluator`

Rozhraní `RecommenderEvaluator` obsahuje pouze jednu metodu. Metoda se jmenuje `evaluate()` a jejím návratovým typem je double. Toto číslo hodnotí rozdíl mezi hodnotou, kterou navrhl Mahout a reálnou hodnotou, kterou zadal uživatel. Výpočet pro jednotlivé implementace:

AverageAbsoluteDifferenceRecommenderEvaluator

$$score = \frac{1}{N} \sum_{i=1}^{N} |m_i - r_i| \tag{4.1}$$

RMSRecommenderEvaluator

$$score = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (m_i - r_i)^2} \tag{4.2}$$

Kde N je velikost testovací množiny, m_i je hodnota doporučení i-tého prvku vrácená Mahoutem a r_i je reálná hodnota, kterou uživatel ohodnotil předmět. Cílem je tedy v obou případech co nejnižší odchylka.

4.7.1 Volba ideálních parametrů

Ukázka testování a volby nejlepšího doporučení proběhne na demonstračním příkladu použitém v kapitole 4.5. Pro trénování bude použito 80% dat a na 20% se bude algoritmus testovat. Jako evaluační výpočet bude použit `RMSRecommenderEvaluator`. Jelikož výpočet v sobě obsahuje druhou mocninu chyby, odhalí lépe větší (> 1) chyby.

Podobnost	**RMS**
`EuclideanDistanceSimilarity`	1.1333
`PearsonCorrelationSimilarity`	1.1862
`LogLikelihoodSimilarity`	1.0688
`SpearmanCorrelationSimilarity`	1.2048
`TanimotoCoefficientSimilarity`	1.0751
`UncenteredCosineSimilarity`	1.1821
`CityBlockSimilarity`	1.1089

Tabulka 4.1: Vliv výběru podobnosti na kvalitu doporučení.

Původní příklad má hodnotu RMS rovnou 1.133. Tato chyba je již značná. To znamená, že se hodnocení průměrně liší o více něž jeden stupeň. Pro různé volby měřítka podobnosti jsou výsledky následující:

Jak je vidět na ukázce, volba podobných uživatelů má na výsledek nezanedbatelný vliv. Pro nalezení ideálních parametrů je nutné vyzkoušet všechny kombinace algoritmů a jejich parametrů. Je tedy nutné vyzkoušet všechny míry podobnosti. Pro každou míru vyzkoušet obě možnosti výběru sousedů a pro tyto možnosti vyzkoušet nastavení jejich parametrů. To v praxi znamená v případě `ThresholdUserNeighborhood` vyzkoušet čísla od nuly do jedné (např. po jedné setině) a v případě `NearestNUserNeighborhood` hodnoty od nuly do celkového počtu uživatelů.

To je však výpočetně celkem náročné a v reálných situacích většinou i zbytečné. V mnoha případech nezáleží na tom, jaké přesně hodnocení uživatel předmětu udělí. Často se chce pouze omezená množina několika málo prvků, které by uživatel hodnotil kladně, ale už nezáleží, jestli by předmětu dal 4 nebo 5 bodů.

Jen pro zajímavost byl proveden výše zmíněný pokus o nalezení nejlepšího možného nastavení doporučení. Při statickém počtu sousedů byly voleny hodnoty po deseti (1, 11, 21 ...). V případě mezního výběru po jednom procentu.

Nejlepší nalezené nastavení:

Recommender: `GenericUserBasedRecommender`
UserSimilarity: `EuclideanDistanceSimilarity`
UserNeighborhood: `ThresholdUserNeighborhood`, threshold = 0.47
Skóre: 0.9430

Otestování trvalo více než 15 hodin. A to byl testován pouze jeden ze šesti možných doporučovacích enginů (`GenericUserBasedRecommender`). Jinak by testování trvalo mnohem déle. Některé sice nemá smysl vůbec testovat, jelikož jejich vypovídající hodnota je nulová (`RandomRecommander`). Úsilí k nalezení nejlepšího nastavení doporučení je velmi vysoké a ovlivněné výběrem trénovací a testovací množiny. Ideálně by tedy testy měly být provedené několikrát a výsledky průměrovány. Tento požadavek opět násobně prodlužuje dobu nalezení ideálních parametrů.

4.8 Zhodnocení

V tomto oddíle byla představena první ze tří oblastí, které Mahoutem lze řešit. Byl definován standardní formát vstupních dat. Dále byl popsán rozdíl mezi user-based a item-based přístupem k doporučení. Jako demonstrační příklad byl použit filmový dataset MovieLens[13], který obsahuje 100 000 unikátních hodnocení. Na závěr byl popsán proces nalezení optimálních parametrů pro doporučení.

5 Klasifikace

Klasifikace je druhou oblastí, kterou Mahout zpracovává. Jak bylo probráno v úvodu, klasifikace slouží k rozdělení dat (např. dokumentů, mailů ...) do předem známých tříd. Pro tyto potřeby je v knihovně Mahout implementováno několik algoritmů.

- Naivní Bayes
- Logistická regrese
- Náhodný les (Random Forrest)
- Skryté Markovovy modely
- Vícevrstvý perceptron

V textu této práce jsou popisovány/využity první dva jmenované. V dalších kapitolách (6 a 7) jsou zadány příklady k řešení. V kapitole 6 je využit Naivní Bayes, jako algoritmus vhodný k řešení úlohy, a také jako zástupce MapReduce algoritmu. V kapitole 7 je zvolena logistická regrese jako jeden z vhodných algoritmů, dále jako zástupce single machine algoritmu a také z důvodu porovnání s původním algoritmem Maximum Entropy.

5.1 Vektory

Pro zpracování dat je nutná jejich jednotná reprezentace. Tato reprezentace má formu N-rozměrných vektorů, kde N je počet featur. Vektorizace a volba featur je podstatná část klasifikace. V Mahoutu existují tři různé implementace vektoru:

- `DenseVector` – Je implementován jako pole proměnných typu double. Pro každou hodnotu je tedy alokován prostor, bez ohledu na to, zda má nulovou hodnotu.
- `RandomAccessSparseVector` – Implementován jako upravený `HashSet`. Třída se nazývá `OpenIntDoubleHashMap` a je součástí balíčku `math`. Uchovává pouze nenulové hodnoty a tím šetří paměť u řídkých vektorů.
- `SequentialAccessSparseVector` – Implementován třídou z balíčku `math` `OrderedIntDoubleMapping`. Tato třída obsahuje dvě pole. Jedno typu int pro uchování indexů a druhé pro uchování hodnot. Optimalizován pro sekvenční čtení.

Dále existují obalové třídy pro proměnné typu vektor, jako například `NamedVector`. Tato třída uchovává ke každému vektoru řetězec s jeho jménem. Toto jméno nemusí splňovat žádné vlastnosti co se týče délky, unikátnosti nebo čehokoliv jiného.

5.2 Vektorizace

Existuje několik základních typů hodnot, které je nutné převádět do vektorové reprezentace:

- spojité – Hodnoty jako teplota, nebo hmotnost jsou spojité. Teoreticky existuje nekonečné množství hodnot, které mohou nabývat.
- kategorické – Mohou nabývat jedné z množiny předdefinovaných hodnot. Tento počet může být od dvou (hodnota typu bool) a může nabývat vysokých počtů (například počet různých směrovacích čísel = 15 515 [14]).
- slovní – Jednoslovná proměnná má podobné charakteristiky jako kategorická proměnná, ale může nabývat téměř neomezeného počtu různých hodnot.

- textové – Množina slovních hodnot. Příkladem může být text této práce, obsah emailu atd.

Ke každé z těchto hodnot se při vektorizaci přistupuje různý způsobem. Mahout obsahuje několik tříd, které se o zakódování výše zmíněných hodnot do vektorů postarají.

5.2.1 Feature Vector Encoders

Pro každý z výše zmíněných typů proměnných existuje odpovídající enkodér. Jejich použití vypadá následovně:

Listing 5.1: Příklad použití ContinuousValueEncoder .

```
FeatureVectorEncoder encoder = new
    ContinuousValueEncoder("teplota");
Vector v = new RandomAccessSparseVector(100);
double teplota = 21.854;
encoder.addToVector(teplota, v);
```

Mahout obsahuje vlastnost feature hashing. To mu umožňuje zakódovat do vektoru více hodnot než je velikost vektoru. Při tomto procesu pochopitelně dochází ke kolizím. Pokud je ale množství kolizí malé, má na výsledek zanedbatelný dopad.

5.3 Vstupní data

Pokud je vytvořen program na čtení souboru a jeho převod na vektory, lze samozřejmě jako vstup použít jakýkoliv soubor. Pokud je Mahout ovládán z příkazové řádky, existuje několik formátů, které dokáže převést na vektorovou reprezentaci. Mahout zvládne převést textové soubory umístěně ve složkách

na jejich vektorovou reprezentaci (z každého souboru je vygenerován právě jeden vektor) více viz praktické příklady. Dále umí využívat CSV a ARFF[1] soubory jako zdroj dat. Jelikož čtení CSV ani ARFF souborů se v praktických příkladech nevyskytuje, budou tyto metody předvedeny zde.

5.3.1 CSV

CSV soubor je možné použít jako zdroj dat např. logistické regrese. V případě logistické regrese je vstupem CSV soubor s daty, názvy hodnot, které mají být použity jako prediktory, jejich datové typy (numeric/word/text), dále počet možných výstupů klasifikátoru a jméno sloupce, který tyto výstupy obsahuje.

Příkladem může být CSV soubor obsahující smyšlená data o lidech. Natrénování Logistické regrese by proběhlo následovně:

```
mahout trainlogistic --input vstup.csv --output model --target
cil --categories 2 --predictors vek zamestnani rodiny_stav
vzdelani --types numeric word word word --features 50
```

Příkaz načte ze souboru vstup.csv data. Výsledný model bude uložen do souboru model. Výstupem klasifikátoru má být hodnota uvedená ve sloupci s názvem cil. Počet tříd, do kterých se klasifikuje je dva. Jako prediktory souží sloupce věk, zaměstnání, rodinný stav a vzdělání. Tyto hodnoty mají odpovídající typy. Počet featur je stanoven na 50, aby mohly být hodnoty zakódovány s malým počtem konfliktů.

5.3.2 ARFF

Textový formát využívaný knihovnou Weka. Mahout obsahuje utilitu, která tento soubor převede na vektory kompatibilní s Mahoutem. Utilita se

[1] Attribute-Relation File Format, `https://weka.wikispaces.com/ARFF`

jmenuje `arff.vector`. Její použití je velmi jednoduché a vypadá následovně:

```
mahout arff.vector --input vstup.arff --output vystup
```

5.4 Měření kvality

Každý klasifikační algoritmus je nutné po natrénování otestovat. Pro testování klasifikačních algoritmů je základem tzv. matice chybovosti (confusion matrix). Matice obsahuje informaci o třídě, kterou určil klasifikátor a která to je ve skutečnosti. Příklad matice pro dvě třídy:

Klasifikováno jako			
A	B		
25	1	A	Skutečná třída
4	14	B	

Ukázková matice říká, že ve skutečnosti existuje 26 instancí třídy „A“. 25 z nich bylo klasifikováno správně. Pouze jedna byla klasifikována jako třída „B“. V ideálním případě jsou všechny hodnoty mimo diagonálu nulové.

Při použití testovacích funkcí v Mahoutu je matice chybovosti často součástí finálního výpisu (např. při použití utility `testnb`).

Z matice chybovosti lze spočítat mnoho užitečných ukazatelů kvality klasifikace. Mimo počet správně a špatně klasifikovaných objektů, lze vypočítat také precision(5.1) a recall(5.2).

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{5.1}$$

$$recall = \frac{true\ positive}{true\ positive + false\ nagative} \tag{5.2}$$

Vysvětlení hodnot - příklad pro třídu A
True positive = určeno jako třída A, ve skutečnosti třída A
False positive = určeno jako tato třída A, ale ve skutečnosti je to jiná třída
False negative = určeno jako jiná třída, ve skutečnosti je to třída A

Tyto hodnoty se společně používají pro výpočet F-míry (F-measure).

$$F1 = 2\frac{presision \cdot recall}{presision + recall} \tag{5.3}$$

Mahout neposkytuje metody pro výpočet F-míry. Tuto hodnotu musí uživatel sám počítat a testovat.

6 Klasifikace mailů pro CIV

Tato úloha byla zadána jako projekt, který měl otestovat, zda by bylo možné využít principů strojového učení a zpracování přirozeného jazyka pro urychlení činnosti HelpDesku na CIVu.

6.1 Úkol

Pokud mají studenti, nebo zaměstnanci ZČU nějaké problémy, nebo požadavky týkající se univerzitní sítě, JIS karet, IS/STAG atd., obrací se na HelpDesk CIV. Pokud napíší email na adresu operator@service.zcu.cz, musí operátor zprávu přečíst a vložit ji do jedné ze 4 definovaných front, kde administrátor již požadavek vyřídí. Problém je v tom, že tento systém vyžaduje čas na přečtení zprávy a zkušenosti operátora, aby správně zařadil došlý požadavek.

Z těchto a dalších důvodů vznikl požadavek na automatizaci procesu třídění požadavků do front.

6.2 Navržené řešení

Jelikož je požadavkem třídění zpráv do předem známých tříd, kde každý požadavek patří právě do jedné, je toto klasická úloha pro klasifikaci. Jako první bylo navrženo řešení pomocí Naivního Bayese. Toto řešení se nakonec ukázalo tak efektivní, že bylo doporučeno, jako vhodné pro řešení úlohy.

6.3 Dodaná data

Jelikož jsou data považována za citlivá, bude zde popsán jejich formát a tvar, ale nebudou uváděny kompletní ukázky. Zároveň také nejsou data přiložena Za dobu činnosti HelpDesku se v systému nashromáždilo téměř 10 000 požadavků. Požadavky jsou rozděleny do 4 složek, dle front, do kterých měly být zařazeny. Fronty jsou následující:

- rt-aaa – 2 580 zpráv
- rt-iss – 470 zpráv
- rt-konta – 670 zpráv
- rt-operator – 6 162 zpráv

Každá složka obsahuje výše zmíněný počet požadavků. V souboru není uveden pouze požadavek samotný, ale celý průběh jeho řešení. To znamená, že je zde uvedena veškerá komunikace mezi uživatelem a CIVem, včetně informací o změnách front a stavu řešení. Příklad obsahu souboru:

```
===> Ticket created by obal on Tue Apr 27 11:10:01 [num]
Ahoj,
potrebuji, prosim, pridat uzivatele [uživatelské jméno] do
skupiny LDAPu
safeq-uk.
Dik
[podpis]
      [num]: untitled (87b)
===> Outgoing email recorded by RT_System on Tue Apr 27 11:10:02
[num]
      [num]: untitled (163b)
===> Outgoing email recorded by RT_System on Tue Apr 27 11:10:02
```

```
[num]
      [num]: untitled (155b)
===> Correspondence added by valtri on Tue Apr 27 12:07:00 [num]
Ahoj,
je tam.
[podpis]
On Tue Apr 27 11:10:01 [num], obal wrote:
> Ahoj,
> potrebuji, prosim, pridat uzivatele [uživatelské jméno] do
 skupiny LDAPu
safeq-uk.
> Dik
> [podpis]
      [num]: untitled (162b)
===> Outgoing email recorded by RT_System on Tue Apr 27 12:07:01
[num]
      [num]: untitled (230b)
===> Status changed from 'new' to 'open' by RT_System on Tue Apr
 27 12:07:01 [num]
===> Status changed from 'open' to 'resolved' by valtri on Tue
Apr 27 12:07:01 [num]
===> Outgoing email recorded by RT_System on Tue Apr 27 12:07:01
[num]
      [num]: untitled (524b)
```

Zde je možné vidět ukázku formátu dodaných dat. Všechna 4 a vícemístná čísla byla nahrazena řetězcem „[num]“ a všechny řetězce, které mají formu emailové adresy, byly nahrazeny sekvencí „e-mail@example.com“. Toto jsou jediné změny provedené na produkčních datech.

Ve zde zobrazené ukázce jsou nahrazeny citlivé informace jako podpisy a uživatelská jména. Tyto náhrady v původních datech nejsou a byl provedeny, aby bylo možné ukázku zveřejnit.

6.4 Transformace dat

Jak je možné vidět z ukázky, každý soubor obsahuje mnohem více dat než samotný email. Prvním krokem tedy bude vytvoření jednoduchého programu na vyfiltrování všeho, kromě textu samotného mailu.

Naivní Bayes v Mahoutu očekává jako vstup pro trénování algoritmu bag of words reprezentaci dat. Toho lze dosáhnou převedením všech souborů do tvaru sekvenčního souboru, který využívá Mahout a jeho následným zpracováním.

6.4.1 Formát sekvenčního souboru

Pokud je cílem vytvoření vektorů z textu použitím utility `seq2sparse` (viz 6.5.1), je nutné dodržet předdefinovaný formát klíče sekvenčního souboru. Formát:

```
/kategorie/ID           Hodnota (např. text mailu)
```

příklad:

```
/rt-konta/101054        Dobrý den, chtěl bych... [zbytek textu]
```

6.5 Vytvoření modelu

Nyní je přípravený sekvenční soubor s texty emailových zpráv. Jak již bylo zmíněno, jako vstup Naivního Bayese slouží jejich bag of words reprezentace. Nejjednodušším možným postupem, je využití utility `seq2sparse`. Další možností je ruční vytvoření potřebných souborů (viz výstup utility `seq2sparse` v následující kapitole).

6.5.1 Seq2sparse

Utilita, která vytvoří ze sekvenčního souboru jeho vektorovou reprezentaci se jmenuje `seq2sparse`.

```
mahout seq2sparse -i seqFile/chunk-0 -o outTFIDF/vectors
```

Vstupem je sekvenční soubor vytvořený z textů emailů. Vstup je označen přepínačem `-i`. Do výstupní složky označené, jako `-o` jsou zapsány tyto soubory:

- `df-count` – obsahuje ID jednotlivých slov a počet dokumentů, ve kterých se toto slova vyskytují
- `tfidf-vectors` – Vektorizovaný sekvenční soubor (metodou TF-IDF).
- `tf-vectors` – Vektorizovaný sekvenční soubor (metodou TF).
- `tokenized-documents` – obsahuje dvojici ID dokumentu - seznam slov
- `dictionary.file-0` – obsahuje všechna unikátní slova v dokumentech a k nim přiřazené ID
- `frequency.file-0` – obsahuje četnosti jednotlivých slov ze slovníku (dvojice ID-četnost)
- `labelindex` – seznam možných tříd dokumentů

6.5.2 Trénování klasifikátoru

Před samotným tréninkem klasifikátoru bude nutné rozdělit vygenerované vektory do dvou množin. První množina bude trénovací a druhá bude sloužit k testování kvality klasifikátoru. Opět využijeme konzolové utility. Tentokrát se utilita jmenuje `split`.

```
mahout split -i outTFIDF/vectors/tfidf-vectors -tr
outTFIDF/train-vectors -te outTFIDF/test-vectors -sp 20
-ow -seq -xm sequential
```

Parametr `i` stejně jako u všech předchozích definuje vstupní soubor. V tomto případě jsou to tf-idf vektory vytvořené ze sekvenčního souboru. Dále jsou definovány přepínače `tr` a `te`, které nastavují výstupní cesty pro trénovací, resp. testovací vektory. `Sp` říká, že pro testování bude náhodně vybráno 20% ze všech vektorů.

Dalším krokem je samotné trénování klasifikátoru. Tréning probíhá opět pomocí příkazu v konzoli. Příkaz má název `trainnb`. Po provedení je vytvořen binární soubor, který obsahuje natrénovaný model. Tento model lze již použít pro klasifikaci nových vektorů.

```
mahout trainnb -i outWTIDF/train-vectors -el -li
outWTIDF/labindex -o outWTIDF/model -ow -c
```

Mahoutu jsou jako vstup předány trénovací vektory. Přepínač `el` říká, že labely se mají získat ze vstupního souboru. `Li` udává soubor, do kterého se mají labely uložit. Soubor pro výstup natrénovaného modelu je označen `-o`.

6.6 Testování modelu

Před použitím je většinou nutné otestovat kvalitu modelu. Toho lze docílit příkazem `testnb`. Vstupem je model, množina vektorů k otestování a seznam labelů. Výstupem je matice chybovosti (confusion matrix) spolu s dalšími statistikami, které hodnotí kvalitu modelu.

6.6.1 Výsledky

Při použití výše popsaných příkazů a postupů, bylo dosaženo následujících výsledků:

```
Summary
-------------------------------------------------------
Correctly Classified Instances          :       1773    91.2037%
Incorrectly Classified Instances        :        171     8.7963%
Total Classified Instances              :       1944

=======================================================
Confusion Matrix
-------------------------------------------------------
a     b     c     d     <--Classified as
463   6     3     21     |  493    a     = rt-aaa
1     61    8     17     |  87     b     = rt-iss
0     14    87    49     |  150    c     = rt-konta
1     11    40    1162   |  1214   d     = rt-operator

=======================================================
Statistics
-------------------------------------------------------
Kappa                                       0.8234
Accuracy                                   91.2037%
Reliability                                63.5493%
Reliability (standard deviation)            0.3895
```

Při testu bylo dosaženo přesnosti 91,2%. Koeficient Kappa, který popisuje homogennost shody mezi skutečností a výstupem klasifikátoru, má hodnotu 0,82. V literatuře[15] jsou hodnoty vyšší než 0,8 – 0,9 považovány za téměř perfektní shodu. Pro další hodnocení byla vypočtena mikro a makro F-míra. Výsledky jsou uvedeny v tabulce 6.1:

Třída	a	b	c	d	Průměr
SUMA	465	92	138	1249	
recall	0,939	0,701	0,580	0,957	0,794
precision	0,996	0,663	0,630	0,930	0,805
F1	0,967	0,682	0,604	0,944	0,800

Tabulka 6.1: Výsledky testu modelu Naivního-Bayese

Mikro F-míry jsou pro každou třídu v příslušném sloupci. Na těchto hodnotách je vidět, že dvě třídy s nejnižším počtem trénovacích záznamů (b = rt-iss, c = rt-konta) mají nejnižší míru přesnosti. Oproti zbylým dvěma třídám, je u nich o cca 30 – 35% nižší míra přesnosti. Makro F-míra je po zaokrouhlení rovna hodnotě 0,8.

Úloha jako celek se podařila zpracovat velmi dobře. Hlavní problém je nevyváženost jednotlivých skupin, co se četnosti zpráv týká. Dle mého názoru by se úspěšnost dala zvýšit vyčištěním dat od zbytečností, jako automatické podpisy, reklamní maily a jiné zprávy, které nemají co do činění s činností CIVu jako takového.

6.7 Použití modelu

Nyní, když je klasifikátor natrénován a otestován, je možné ho použít pro klasifikaci. Použití Naivního Bayese je možné pouze z Java kódu. Pro tyto potřeby je nutné vykopírovat z HDFS několik souborů. Jsou to následující soubory:

- natrénovaný model – model vytvořený příkazem `trainnb`
- label index – vytváří se společně s modelem přepínačem `-el`
- slovník – vytvořen ze sekvenčního souboru pomocí `seq2sparse`
- četnost slov – vytvořen ze sekvenčního souboru pomocí `seq2sparse`

Postup pro využití modelu není tak přímočarý jako při jeho vytváření. Nyní jsou tedy dostupné nové emaily, které je potřeba klasifikovat. Nejdříve je nutné načíst všechny výše uvedené soubory. Nejsnazší je načtení samotného modelu:

Listing 6.1: Načtení naterénovaného modelu Naivního-Bayese.

```
Configuration configuration = new Configuration();
NaiveBayesModel model = NaiveBayesModel.materialize(new
    Path(modelPath), configuration);
StandardNaiveBayesClassifier classifier = new
    StandardNaiveBayesClassifier(model);
```

Pro převod nového mailu na jeho bag of words reprezentaci (viz 2.3.1), potřebnou ke klasifikaci, bude nutné načíst i zbývající soubory. Jak bylo řečeno dříve, sekvenční soubor je obdobou Mapy v Jave. Proto se tato struktura nabízí jako ideální reprezentace zbývajících souborů. Pro načtení label indexu (dvojice Integer-String) existuje připravená metoda.

```
Map<Integer, String> labels = BayesUtils.readLabelIndex(
configuration, new Path(labelIndexPath));
```

V případě ostatních kombinací, je nutné metodu pro načtení vytvořit.

Listing 6.2: Načtení sekvenčního souboru do mapy.

```
public static Map<String, Integer> nactiSlovnik(Configuration
    conf, Path p) {
   Map<String, Integer> dictionnary = new HashMap<>();

   for (Pair<Text, IntWritable> pair : new
       SequenceFileIterable<Text,   IntWritable>(p, true, conf))
   {
      dictionnary.put(pair.getFirst().toString(),
          pair.getSecond().get());
   }
   return dictionnary;
```

```
}
```

Dalším úkolem je načtení mailu, který má být klasifikován. Email bude načten ze souboru jako řetězec. Tento řetězec je nutné rozdělit na jednotlivá slova. Každé slovo, které je zároveň v načteném slovníku je uchováno, včetně počtu kolikrát se v emailu vyskytlo.

Nyní k vytvoření samotného vektoru. Vektor má velikost uchovaného počtu slov. Pro každé uchované slovo je získáno jeho ID ze slovníku a jeho četnost z frequency souboru. Tyto hodnoty jsou použity jako parametry pro metodu `calculate()` třídy TFIDF (je nutné použít stejnou metodu jako při vytváření trénovacích vektorů). Metoda vrátí vypočtenou hodnotu TFIDF pro dané slovo. Hodnota je pak uložena do vektoru.

Po zpracování všech uložených slov, vznikne klasifikovatelný vektor, reprezentující nový email. Tento vektor lze klasifikovat následujícím způsobem:

Listing 6.3: Klasifikace vektoru.

```
Vector resultVector = classifier.classifyFull(vector);
int tridaId = resultVector.maxValueIndex();
String nazevTridy = labels.get(new Integer(tridaId));
```

V proměnné tridaId je uložen identifikátor třídy v mapě labelů. Její název lze snadno získat přes zmíněný klíč.

Použití modelu není úplně přímočaré. Ani jedna ze dvou knih týkajících se Mahoutu, které v současné době existují[16][17] se problémem použití modelu nezabývá. Obě končí u testování modelu příkazem `testnb`.

6.8 Zhodnocení

V této části práce bylo demonstrováno využití Naivního Bayese v knihovně Mahout. Byl zde popsán proces od získání dat, přes úpravu jejich formátu, trénování algoritmu, otestování až k využití natrénovaného modelu. Tento proces byl z větší části vykonán v příkazové řádce, pouze poslední krok byl demonstrován v Jave.

Jak je vidět v kapitole dosažené výsledky, úspěšnost klasifikace se pohybuje nad 90%. Tuto úspěšnost považuji s ohledem na původ dat (reálná data, ne testovací dataset) za velmi dobrou. Konečná implementace (nasazení na server CIVu) není cílem této práce. Cílem bylo otestovat, jak si Mahout poradí s touto úlohou.

7 Klasifikace komentářů na webových fórech

Další úloha zadaná pro ověření schopností Mahoutu v oblasti klasifikace. Tentokrát jsou klasifikované objekty HTML elementy na webové stránce.

7.1 Úkol

Úkolem je přepsat aplikaci FeedExtract[1] využívající knihovnu pro strojové učení vyvíjenou na KIVu na verzi využívající Mahout. A to s co nejmenším možným zásahem do programového kódu. Původní aplikace byla vyvinuta v rámci předmětu ZSWI týmem pěti studentů. Dále byla upravována a doplňována v rámci dalších semestrálních prací Davidem Steinbergerem, který je jedním z původních tvůrců aplikace.

Účelem je porovnání kvality klasifikace mezi oběma knihovnami a dále srovnání jednoduchosti implementace.

7.2 Dodaný program

Program je podrobně popsán v jednotlivých dokumentech, které byly vytvořeny během semestrálních prací. V této části bude popsán pouze systém hodnocení kvality klasifikace.

[1] FeedExtractor, Na DVD přiložena dokumentace z předmětu KIV/TKS

7.2.1 Testování

Testování probíhá metodou, která se nazývá Křížová validace (Cross validation). Specificky je to metoda Leave-one-out. Program obsahuje 34 různých webových fór. Při leave-one-out metodě se algoritmus natrénuje na N - 1 typech stránek (v tomto případě 33) a na posledním se testuje. Výhoda je v tom, že přesnou strukturu dat vynechaného fóra nemá v trénovacích datech. Lze tedy pozorovat, jak se klasifikátor chová na neznámých datech. Další velkou výhodou tohoto přístupu je opakovatelnost. Oproti náhodnému výběru dat, které obsahuje Mahout například v metodě split. Opakovatelnost zaručí stejné výsledky při každém spuštění. Lze tedy snadno porovnat provedené změny a není nutné algoritmus spouštět několikrát a sledovat průměr.

Testovací data

Program obsahuje celkem 34 různých webových fór a diskusí jako například fórum živě, autoweb, gentoo atd. Data jsou ve formátu HTML dokumentu staženého z příslušných stránek. Počty komentářů v jednotlivých typech stránek se pohybují od 2 do 2141. Jak je vidět rozptyl je značný. Tento fakt může ovlivňovat některé výsledky.

7.2.2 Hodnocení kvality klasifikace

Pro hodnocení kvality testování je ve stávajícím programu zavedena jistá stupnice. Její základní význam bude popsán zde, bližší popis je dostupný v dokumentaci původního programu.

Klasifikace probíhá v programu ve dvou fázích. Nejdříve se hledá obalující kontejner. Správně rozpoznaný kontejner je nejbližší element nadřazený elementům obsahující následující informace:

- uživatelské jméno přispěvatele

- datum vložení příspěvku
- text příspěvku

Ve druhé fázi jsou v rámci kontejneru klasifikovány jednotlivé elementy vypsané výše. Pro hodnocení jednotlivých částí byl v původní práci zaveden bodový systém, který klasifikaci hodnotí následovně:

- správně rozpoznaný kontejner příspěvku – 1 bod
- správné jméno přispěvatele – 3 body (1 bod)
- správné datum – 2 body (1 bod)
- správný text příspěvku – 5 bodů (2 body)

Maximální možné ohodnocení příspěvku je tedy **11 bodů**.

V závorkách je uvedeno bodové ohodnocení tzv. částečné evaluace. U částečné evaluace je podmínka, že klasifikovaný element obsahuje korektní data a zároveň délka vráceného řetězce je méně než desetinásobná oproti správnému řešení. Element tedy obsahuje správná data, ale i něco navíc, proto je hodnocení sníženo.

7.2.3 Výsledky původního programu

Před provedením jakýchkoliv úprav byla otestována kvalita klasifikace původního programu. Výsledky jsou následující:

```
------------------------------------------------------------
Vysledky 1.faze:
Celkem rozpoznano: 6323/11972/12040
------------------------------------------------------------
```

```
Vysledky 2.faze:
Celkem nicku: 3352/12040
Celkem dat: 9684/12040
Celkem obsahu: 8347/12040
Skore: 77550/132440
Pocet celych rozpoznanych: 2825/12040
------------------------------------------------------------
Vysledky samostatne 2.faze:
Celkem nicku: 5206/12040
Celkem dat: 10654/12040
Celkem obsahu: 9422/12040
Skore: 84761/120400
Pocet celych rozpoznanych: 4610/12040
```

V případě trojice čísel je význam: bylo rozpoznáno korektně/bylo rozpoznáno celkem/celkem bylo k rozpoznání.

7.3 Volba algoritmu

Mahout obsahuje dva vhodné algoritmy pro tuto úlohu. Jsou to Random Forrest a Logistická regrese. Jelikož principem práce je sestavit návod pro Mahout, byl zvolen sériový algoritmus Logistická regrese. Random Forrest funguje na podobném principu jako Naivní Bayes, který byl použit v minulé úloze.

Algoritmus logistická regrese se velmi snadno používá uvnitř Java kódu. Jeho hlavní nevýhodou je to, že algoritmus probíhá sériově, čímž uživatel přichází o jedinou výhodu Mahoutu oproti jiným knihovnám, kterou je možnost masivního paralelismu.

7.4 Provedené změny v kódu

Kód byl upraven pouze na nezbytně nutných místech. Většina popisovaných změn se týká třídy `CommentClassifier`. Stejné změny proběhly i ve třídě `FinalClassifier`, proto nebudou zmiňovány několikrát.

7.4.1 Vytvoření klasifikátoru

Algoritmus logistické regrese se v Mahoutu nachází ve třídě nazvané `OnlineLogisticRegression`. Parametry konstruktoru jsou:

- počet tříd
- počet featur
- priorní funkce

Nyní je možné vytvořit novou instanci klasifikátoru.

```
classifier = new OnlineLogisticRegression(LABEL_COUNT, FEATURES,
new L2());
```

7.4.2 Vytvoření vektorů z HTML dokumentů

Pro klasifikaci je nutné vytvořit vektorovou reprezentaci jednotlivých HTML elementů. Jak bylo probíráno v kapitole 5.2, v Mahoutu existují 4 typy featur. Všechny featury definované v původním programu jsou typu bool, tedy kategorické.

7.4.3 Trénink

Když jsou data vektorizována, je možné začít s tréninkem klasifikátoru. Pro všechny dostupné trénovací vektory se jednoduše provede příkaz `train()`.

```
classifier.train(labels[i], trainingVectors.get(i));
```

Parametrem příkazu je označení třídy, do které vektor patří (label) a vektor samotný. Po natrénování klasifikátoru je vhodné zavolat metodu `close()`. Není to nutné, ale dle dokumentace by tento příkaz měl zrychlit klasifikaci. V praxi nebyl pozorován žádný rozdíl.

```
classifier.close();
```

7.4.4 Klasifikace

Klasifikované HTML elementy opět musí být převedeny na vektory. Tyto vektory pak lze klasifikovat velice jednoduše použitím některé z variant metody `classify()`. Možnosti jsou:

```
classifier.classify(v);
classifier.classifyScalar(v);
classifier.classifyFull(v);
```

Metoda `classify()` vrací vektor o délce N – 1, kde N je počet tříd (labelů). Pravděpodobnost nulté kategorie je možné získat jako 1 - suma návratového vektoru. `ClassifyScalar()` je použitelná pouze pokud jsou objekty klasifikovány do dvou tříd. Aby se zabránilo zbytečným inicializacím vektorů, vrací tato metoda pouze pravděpodobnost první ze dvou tříd jako hodnotu typu double. Metoda `classifyFull()` vrací vektor o rozměru N. Obsahuje tedy pravděpodobnosti všech tříd a není nutné nic dopočítávat.

7.4.5 Další úpravy

Tyto úpravy se již netýkají samotné klasifikace. Původně použitá knihovna samozřejmě vrací data v jiném formátu než Mahout. Protože zásah do programu má být minimální, musely být vytvořeny některé třídy, které budou data z Mahoutu uchovávat ve formátu, kterému aplikace rozumí. Vytvořeny byly třídy `Matrix` a `ClassificationResult`.

7.5 Výsledky upraveného programu

Po provedení všech zmíněných úprav byl program otestován na stejných datech se stejným bodovým systémem jako původní program. Výsledky byly následující:

```
------------------------------------------------------------
Vysledky 1.faze:
Celkem rozpoznano: 0/0/12040
------------------------------------------------------------
Vysledky 2.faze:
Celkem nicku: 0/12040
Celkem dat: 0/12040
Celkem obsahu: 0/12040
Skore: 0/132440
Pocet celych rozpoznanych: 0/12040
------------------------------------------------------------
Vysledky samostatne 2.faze:
Celkem nicku: 2/12040
Celkem dat: 60/12040
Celkem obsahu: 2044/12040
Skore: 25087/120400
Pocet celych rozpoznanych: 0/12040
```

Jak je vidět z výpisu, Mahout nebyl schopen rozpoznat žádný komentář. Pouze v samostatné 2. fázi (kde jsou předány správně určené kontejnery) jsou některá data rozpoznána korektně. Oproti původnímu programu je však úspěšnost mimořádně nízká.

7.6 Vylepšení kvality rozpoznávání

Logistická regrese v Mahoutu je mimořádně citlivá na své parametry. Tyto parametry neumí nastavit ani pro tyto účely speciálně navržená třída `AdaptiveLogisticRegression`. Při využití této třídy není opět rozpoznán žádný komentářový kontejner. Je tedy nutné ručně metodou pokusu a omylu nastavit parametry logistické regrese. Tento algoritmus má v Mahoutu pět nastavitelných hodnot:

- alpha (1 - 10-3) – používá se pro výpočet hodnoty currentLearningRate
- decayExponent (-0.5) – používá se pro výpočet hodnoty currentLearningRate
- lambda (1 * 10-5 = 0.00001) – určuje míru regularizace
- learningRate (1) – používá se pro výpočet hodnoty currentLearningRate
- stepOffset (10) – používá se pro výpočet hodnoty currentLearningRate

V závorkách za hodnotami jsou uvedeny výchozí hodnoty v aktuální verzi Mahoutu (0.9). Hodnoty je možné nastavit setrem s odpovídajícím pojmenováním například:

```
classifier.decayExponent(-0.01);
```

Nyní je tedy nutné nalézt takové parametry, které budou produkovat dobré výsledky. Po mnoha pokusech o nalezení odpovídajících hodnot se výsledky zlepšily následovně:

```
------------------------------------------------------------
Vysledky 1.faze:
Celkem rozpoznano: 4515/6298/12040
------------------------------------------------------------
Vysledky 2.faze:
Celkem nicku: 1743/12040
Celkem dat: 4203/12040
Celkem obsahu: 4252/12040
Skore: 39876/132440
Pocet celych rozpoznanych: 942/12040
------------------------------------------------------------
Vysledky samostatne 2.faze:
Celkem nicku: 3049/12040
Celkem dat: 9739/12040
Celkem obsahu: 7452/12040
Skore: 69264/120400
Pocet celych rozpoznanych: 1815/12040
```

Jak je vidět, zlepšení je radikální. Více než třetina kontejnerů je korektně rozpoznána. Výsledky samostatné 2. fáze jsou již celkem blízké původnímu programu.

Samotné nalezení hodnot parametrů nemusí být dostačující. Jak už bylo několikrát zopakováno, Mahout je stavěn na velké objemy dat. Některým jeho algoritmům nemusí stačit přibližně 12 000 pozitivních případů ke konvergenci. Bohužel množství trénovacích dat se nedá snadno například desetinásobně zvětšit. Lze ale projít trénovací množinu více než jednou a umožnit klasifikátoru konvergovat. Problém vzniká při příliš malém počtu trénovacích příkladů a příliš velkém počtu opakování.

Tento problém se nazývá overfitting. Při něm se model natrénuje příliš specifiky pro daná trénovací data a zavleče se do něj „šum“. Tento problém se dá odstranit testováním různých hodnot počtu opakování, dokud se nedocílí dobrých hodnot v přesnosti klasifikace na testovací množině.

Pro další testy byl v první fázi zvolen počet opakování tréninkových vektorů 50. Ve druhé fázi bylo ponecháno jedno opakování. Výsledky jsou následující:

```
------------------------------------------------------------
Vysledky 1.faze:
Celkem rozpoznano: 6326/11215/12040
------------------------------------------------------------
Vysledky 2.faze:
Celkem nicku: 2718/12040
Celkem dat: 8319/12040
Celkem obsahu: 7439/12040
Skore: 68938/132440
Pocet celych rozpoznanych: 1593/12040
------------------------------------------------------------
Vysledky samostatne 2.faze:
Celkem nicku: 3049/12040
Celkem dat: 9739/12040
Celkem obsahu: 7452/12040
Skore: 69264/120400
Pocet celych rozpoznanych: 1815/12040
```

Jak je na výsledcích vidět, úspěšnost klasifikace v první fázi je téměř shodná s původní aplikací (6 318 vs. 6 326). Dále mírně klesl počet špatně klasifikovaných elementů (false positive) z 5 649 v původní aplikaci na 4 889.

7.6.1 Porovnání výsledků

Po odladění parametrů klasifikátoru bylo dosaženo následujících výsledků.

	Původní program	Mahout	Mahout s parametry	Mahout s parametry a opakováním
1. fáze				
Rozpoznáno	6 323/11 972	0/0	4 515/6 298	6 326/11 215
2. fáze				
Nicků	3 352	0	1 743	2 718
Dat	9 684	0	4 203	8 319
Obsahu	8 347	0	4 252	7 439
Skóre	77 550 z 132 440	0 z 132 440	39 876 z 132 440	68 938 z 132 440
Celé rozp.	2 852	0	942	1 539
Samostaná 2. fáze				
Nicků	5 206	2	3 049	3 049
Dat	10 654	60	9 739	9 739
Obsahu	9 422	2 044	7 452	7 452
Skóre	84 761 z 120 400	25 087 z 120 400	69 264 z 120 400	69 264 z 120 400
Celé rozp.	4 610	0	1 815	1 815

Tabulka 7.1: Porovnání výsledků jednotlivých verzí klasifikátoru s původním programem.

Jak je vidět z tabulky, tak logistická regrese s výchozími hodnotami není schopna správně klasifikovat žádný komentář. Po nastavení parametrů je situace mnohem lepší. Problém spočívá v tom, že je nutné tyto parametry nastavovat více méně hrubou silou. Při zvýšení počtu opakování tréninku u první fáze je vidět, že tato část dosahuje shodných výsledků s původním programem. Jistě by šlo dosáhnout lepších výsledků i ve druhé fázi. Cílem této práce ale není vylepšení stávajícího programu, ale demonstrace funkcí

Mahoutu.

7.7 Zhodnocení

V této úloze bylo popsáno použití logistické regrese. Oproti minulé úloze, kde byl použit Naivní Bayes je použití tohoto algoritmu velmi odlišné. Logistická regrese je implementována jako single machine algoritmus. To znamená, že nevyužívá Hadoop jako běhové prostředí a není tedy nutné vytvářet sekvenční soubory atp.

I když z počátku to vypadalo, že Mahout není vhodným kandidátem a s úlohou si neporadí, nakonec se podařilo dosáhnout uspokojivých výsledků. Při použití logistické regrese je u některých úloh nutné ručně nastavovat některé parametry. Pro tyto účely existuje v Mahoutu připravená třída `AdaptiveLogisticRegression`, ale na daných datech měla stejné výsledky jako obyčejná logistická regrese bez zadaných parametrů.

8 Shlukování

Poslední ze tří oblastí, které Mahout zpracovává. Na rozdíl od předchozích dvou jde o algoritmy učení bez učitele. To znamená, že nejsou předem známy výstupní hodnoty. Důsledkem těchto faktů je velmi obtížná validace výstupu úloh. Mahout obsahuje pět shlukovacích algoritmů:

- Canopy Clustering
- k-Means Clustering
- Fuzzy k-Means
- Streaming k-Means
- Spectral Clustering

8.1 Canopy Clustering

Do verze Mahoutu 0.9 byl využíván pro předzpracování dat pro k-means. Standardní postup byl canopy => k-means. Od verze 0.10.0 je Canopy clustering deprecated[18]. Problém je v tom, že obyčejný k-Means má počet clusterů jako povinný vstupní parametr. Canopy se tyto středy pokoušel vhodně zvolit. U něj bylo zase nutné zadat hodnoty T1 a T2 (viz kapitola 2.4.2). Tímto byla nahrazena jedna „magická“ hodnota za dvě jiné. Tento algoritmus bude při vydání finální verze knihovny pravděpodobně odstraněn.

8.1.1 Použití

Jelikož je odstranění tohoto algoritmu plánováno již delší dobu, nedochází u něj k odstranění některých částí, jako u ostatních algoritmů (viz později k-Means). Umožňuje tedy jak práci in-memory tak Hadoop MapReduce.

in-memory

Listing 8.1: Ukázka použití algoritmu Canopy clustering.

```
GloVeFileReader reader = new GloVeFileReader();
ArrayList<Vector> data = reader.readToPlainVectors(inputData);
EuclideanDistanceMeasure dm = new EuclideanDistanceMeasure();
List<Canopy> canopies = CanopyClusterer.createCanopies(data, dm,
    12, 8);
```

Velmi pohodlná práce s malým množstvím dat, která se vejdou do paměti. Pokud je dat větší množství, je nutné využít MapReduce verzi a případně výpočet distribuovat na více stanic.

MapReduce

V tomto případě musí být opět vstupem mnohem méně pohodlné sekvenční soubory oproti obyčejným vektorům výše. Všechny vstupní soubory musí být nahrány v HDFS.

a) volání z kódu

Listing 8.2: Ukázka použití MapReduce verze algoritmu Canopy clustering v Jave.

```
Configuration conf = new Configuration();
Path input = new Path("canopy/vstup/part-r-00000");
Path output = new Path("canopy/vystup/");
EuclideanDistanceMeasure dm = new EuclideanDistanceMeasure();
CanopyDriver.buildClusters(conf, input, output, dm, 12, 8, 0, 0,
    0, true);
```

b) příkazový řádek

```
mahout canopy -i canopy/vstup/part-r-00000 -o canopy/vystup/
-dm org.apache.mahout.common.distance.EuclideanDistanceMeasure
```

```
-t1 12 -t2 8
```

8.2 K-means

K-means, stejně jako všechny ostatní shlukovací algoritmy v Mahoutu, které nejsou deprecated, neumožňuje in-memory použití, které je při testování velmi pohodlné a urychluje práci. Tato možnost byla odebrána již ve verzi 0.7.

Vstupem k-means je množina vektorů, které se mají shlukovat. Dále je to buď počet clusterů (k), které se mají vytvořit, anebo soubor se zvolenými středy (např. z Canopy clusteringu). Pokud není dodán soubor se zvolenými středy, jsou vybrány náhodně ze vstupních vektorů.

8.2.1 Použití

Způsob použití je velmi podobný Canopy clusteringu a platí stejná pravidla.

a) volání z kódu

Listing 8.3: Ukázka použití MapReduce verze algoritmu k-means v Jave.

```
Configuration conf = new Configuration();
Path input = new Path("kmeans/vstup/part-r-00000");
Path clustersIn = new Path("kmeans/vystup/");
Path output = new Path("kmeans/vystup/");
KMeansDriver.buildClusters(conf, input, clustersIn, output, 20,
    "0.1", false);
```

Poznámka: V této verzi nebyl nalezen způsob, jak definovat počet clusterů a distance measure při tomto způsobu volání. Složka clustersIn musí

tedy skutečně obsahovat připravené středy clusterů. Oficiální stránky uvádějí metodu s neplatnými parametry (pravděpodobně starší verze). Celková nelogičnost této metody (např. načítání hodnoty typu double jako řetězec) je zarážející.

b) příkazový řádek

```
mahout kmeans -i kmeans/vstup/part-r-00000 -o kmeans/vystup/
-dm org.apache.mahout.common.distance.EuclideanDistanceMeasure
-c kmeans/vystup/ -k 1500 -x 20
```

8.3 Fuzzy k-means

V některých zdrojích se nazývá také fuzzy c-means. Jde o variantu tzv. soft clusteringu, kdy jednotlivé vektory nejsou vždy součástí právě jednoho clusteru, ale mají pravděpodobnost příslušnosti k 1 až N clusterům. Vektor tedy například může příslušet clusteru A s pravděpodobností 0.7 a clusteru B s pravděpodobností 0.3.

Způsob použití je stejný jako v případě k-means. Rozdíl je pouze ve jméně volané třídy/funkce (fuzzykmeans) a dvou parametrech vztahujících se k fuzzy části clusteringu. Těmito parametry jsou:

- fuziness – Jak moc „fuzzy“ má clustering být. Hodnota musí být větší nebo rovna jedné. Hodnoty blízké jedničce znamenají výstup velmi podobný k-means.
- emitMostLikely - Vektor bude zařazen k tomu nejpravděpodobnějšímu clusteru ostatní budou ignorovány. V případě nastavení na false bude výstupem matice s váhou každé dvojice vektor-cluster.

9 Shlukování slov dle sémantické podobnosti

Jako třetí úloha bylo zvoleno shlukování slov dle jejich sémantické podobnosti. Zpracování úlohy má ověřit schopnosti knihovny Mahout v oblasti clusteringu.

9.1 Zadání

Aplikace clusteringu na množinu vektorů, reprezentujících slova. Na výstupu bude ověřeno, zda jsou významově podobná slova ve stejných clusterech.

9.2 Vstupní data

Vstupem jsou před-připravené vektory z projektu univerzity Stanford. Projekt se nazývá GloVe (Global Vectors for Word Representation)[19], a poskytuje veřejně dostupné datasety. Pro tento projekt byl použit dataset „Wikipedia 2014+ Gigaword 5“. Konkrétně jeho 300d model.

Dataset obsahuje 400 000 unikátních slov/výrazů, které byly extrahovány z celkem šesti miliard slov. Při tomto procesu byla sledována jejich četnost a kontext ve kterém se vyskytovaly. Následně byla tato data vektorizována a poskytnuta veřejnosti.

9.2.1 Formát dat

Datový soubor dosahuje velikosti téměř 1 GB (989 MB). Formát uložených dat je velmi prostý. Každý řádek je uvozen slovem, které reprezentuje následující vektor. Začátek souboru:

```
the 0.04656 0.21318 -0.0074364 ... celkem 300 hodnot
```

9.3 Zpracování

Pro vypracování byla zvolena metoda k-means. Metoda exkluzivního shlukování, kde každý vektor (slovo) náleží do právě jednoho clusteru.

Bohužel při pokusu o spuštění fuzzy k-means se zaplnila celá dostupná paměť (16GB) a program se ani po více než třech hodinách nepohnul ze stavu map 0% reduce 0%. Postupně byl snižován počet prvků, dokud nebylo dosaženo rozumného času dokončení. Výstup fuzzy k-means tedy obsahuje pouze 1 000 clusterů z 10 000 vektorů (Bylo použito prvních 10 000 vektorů z původního souboru.). Výsledky tedy nejsou srovnatelné s obyčejným k-means, ale cílem bylo ukázat rozdíl ve výstupu fuzzy clusteringu.

9.3.1 Převod dat

Jak bylo zmíněno výše, k-means neumožňuje obyčejné načtení vektorů do paměti a jejich následný clustering. Jako u předchozích příkladů je tedy nutné převést data do formy sekvenčního souboru. Protože data nejsou ve formátu, který by Mahout uměl jednoduše na sekvenční soubor převést, je nutné vytvořit program, který datový soubor přečte a vytvoří datový soubor ručně. Více v kapitole Manuální převod **??**.

9.3.2 Příprava

Výsledný sekvenční soubor je o něco menší, než původní soubor (934 MB). Tento soubor je nutné nahrát do HDFS. Před spuštěním clusteringu je v případě této úlohy nutné zvýšit využitelné množství paměti. Před úpravou hodnot je doporučeno ukončit Hadoop příkazem `stop-all.sh`.

Nejdříve je nutné zvýšit množství paměti, kterou může využít Hadoop pro běh jednotlivých úloh (job). To lze provést editací souboru `hadoop-env.sh`, který se nachází v Hadoopu ve složce conf. Výchozí množství využitelné paměti je nastaveno na 1 000 MB. Pro potřeby této úlohy bylo vyhrazeno 8 000 MB paměti (nižší hodnoty nebyly testovány). V uvedeném souboru je na řádce 15 zakomentovaný export velikosti haldy. Příkaz odkomentujeme (popřípadě napíšeme sami kamkoliv jinam) a zadáme požadovanou hodnotu v MB.

```
export HADOOP_HEAPSIZE=8000
```

Dále je nutné upravit množství paměti pro jednotlivé Mappery. Ve stejné složce se nachází soubor `mapred-site.xml`. Do tohoto souboru je nutné přidat následující hodnotu:

```
<property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx8192m</value>
</property>
```

Nyní lze Hadoop opět spustit.

9.3.3 K-means clustering

Samotný clustering je spuštěn následujícím příkazem:

```
mahout kmeans -i glove/seqFile -o glove/kmeans/out -c
```

```
glove/kmeans/centers -k 25000 -x 50 -cd 0.1 -cl -dm
org.apache.mahout.common.distance.EuclideanDistanceMeasure
```

Přepínače `-i`, `-o` definují vstupní a výstupní složky. `-c` určuje, kam se vygenerují počáteční středy clusterů. `-dm` udává distanční funkci, která se má použít pro výpočet vzdálenosti mezi vektory. `-k` určuje počet clusterů. `-x` definuje maximální počet iterací algoritmu. Přepínač `-cd` stanovuje konvergenční hranici, pokud se středy clusterů mezi iteracemi neposunou o více než toto delta, je algoritmus ukončen. Přítomnost přepínače `-cl` indikuje, že po nalezení clusterů má proběhnout ještě přiřazení jednotlivých vektorů ze vstupu do příslušných clusterů.

V každé iteraci algoritmu je ve složce `glove/kmeans/out` vytvořena podsložka `clusters-i`, kde i je číslo iterace. Tato podsložka obsahuje aktuální polohy středů v dané iteraci. Po konvergenci středů (nebo po vyčerpání maximálního počtu opakování) je vytvořen soubor `clusters-i-final`, který obsahuje konečné polohy středů clusterů. Pokud je nastaven přepínač `-cl`, je nakonec ve složce `glove/kmeans/out` vytvořena složka `clusteredPoints` a v ní sekvenční soubor který mapuje jednotlivé vektory na nalezené clustery.

Při použití příkazu uvedeného výše, trvalo nalezení clusterů pro všech 400 000 vektorů více než 43 hodin. Nebylo dosaženo konvergenční hranice 0.1 a bylo provedeno všech 50 iterací. Při tomto testu bylo využito standardní nastavení počtu Mapperů (2) a Reducerů (1).

9.3.4 Získání výsledků

Pro získání „lidsky“ čitelných výsledků je nutné použít utilitu `clusterdump`. Její použití vypadá následovně:

```
mahout clusterdump -i glove/kmeans/out/clusters-50-final/ -o
vystup/glove -of TEXT -p glove/kmeans/out/clusteredPoints
```

Vstupem algoritmu jsou středy nalezených clusterů. Výstupní formát

(`-of`) může nabývat tří různých hodnot:

- TEXT – Nejobsáhlejší výpis, u každého clusteru uvádí jeho střed a rozměry, u vektorů uvádí jejich vzdálenost od středu clusteru a vektor samotný.
- CSV – Obsahuje pouze unikátní identifikátor clusteru, a unikátní identifikátor vektorů, které do daného clusteru patří.
- GRAPH_ML – Vytvoří soubor, který lze prohlížet v programu, který podporuje GRAPHML formát. Umožňuje vizualizovat clustery.

Pokud byl původní clustering spuštěn s přepínačem `-cl` a je tedy dostupné mapování mezi vektory a clustery, lze tato data připojit přepínačem `-p`. Ke každému clusteru je tak přidána množina vektorů, která do něj patří.

Výstup utility je proveden do běžného filesystému, nikoliv do HDFS. Cesta zadaná v -o je tedy cestou v běžném filesystému. Při prvním použití to může být matoucí.

Výstupní data byla získána ve všech dostupných formátech. Následují ukázky výstupu:

TEXT

```
VL-43097{n=10 c=[0.169, 0.218, ...] r=[0.302, 0.246, ...]}
Weight :[props - optional]: Point:
1.0 :[distance=5.014305335101661]: frog =[-0.240, 0.213, ... ]
1.0 :[distance=3.945374839658038]: frogs =[-0.086, 0.721, ... ]
1.0 :[distance=4.917794468932951]: toad =[0.200, 0.165, ... ]
...
```

Cluster dostal automatické označení VL-43097. N=10 značí, že cluster obsahuje 10 vektorů, c označuje polohu středu clusteru a r jeho rozměry.

Dále jsou vypsány samotné vektory, zde pouze první tři. U každého je uvedena váha (v případě použití fuzzy k-means by váha mohla nabývat různých hodnot, zde vždy jedna), vzdálenost od středu a na konec slovo a vektor, který ho reprezentuje.

Velikost výstupního souboru je 975 MB. Jeho prohlížení je nutné mít textový editor, který zvládne otevřít soubory této velikosti.

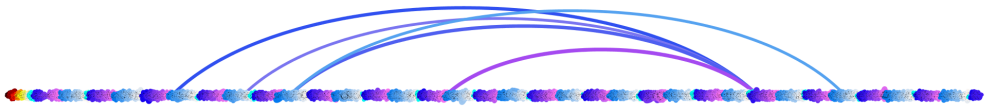
CSV

```
43097, frog, frogs, toad, horned, amphibians, toads, burrowing,
salamanders, tadpoles, newts
```

CSV výpis je mnohem úspornější (3 MB oproti 975 MB). Obsahuje pouze ID clusteru a ID vektorů. Ve vstupním sekvenčním souboru bylo slovo použito jako ID vektoru. Zde je vidět všech deset slov v clusteru.

GRAPH_ML

Ukázka výstupu (k-means) ve formátu GRAPH_ML byla provedena na 10 000 prvcích v 1 000 clusterech. Při testu na všech datech měly programy problémy se zobrazením.



Obrázek 9.1: Výstup k-means zobrazený v programu Gephi (otočeno o 90 stupňů ve směru hodinových ručiček).

9.3.5 Fuzzy k-means

Fuzzy k-means se spouští téměř totožným příkazem jako obyčejný k-means:

```
mahout fkmeans -i input/seqFile10000 -o outputFuzzy10000/ -dm
org.apache.mahout.common.distance.EuclideanDistanceMeasure -c
outputFuzzy10000/startClusters/ -k 1000 -cd 0.5 -x 20 -ow -cl
-m 1.75 -e false
```

Příkaz je rozdílný pouze v hodnotách parametrů (souvisejících se změnou počtu vektorů a clusterů) a dvou přepínačích. Zmíněné přepínače jsou na konci příkazu a vyjadřují míru fuzzy (fuziness) a informaci, zda budou výstupem pouze nejpravděpodobnější clustery.

9.3.6 Získání výsledků

Export výsledků lze provést opět příkazem clusterdump, ovšem je tu jiný problém. Výstupem fuzzy k-means ve složce clusteredPoints není, na rozdíl od obyčejného k-means, vektor a k němu id clusteru, do kterého patří. Výstupem je teoreticky matice, kde ke každé dvojici vektor-cluster je uvedena váha s jakou patří do daného clusteru. Toto se podepisuje na velikosti výstupního souboru. Soubor clusteredPoints pro obyčejný k-means se 400 000 vektory v 25 000 clustrech má 994 MB. Fuzzy k-means s omezenou množinou (viz výše) má velikost 23,14 GB. Při pokusu použít utilitu clusterdump na testovacím stroji vždy dojde paměť a utilita spadne. Bude proto potřeba přečíst soubor programově.

Výstupem jsou jako vždy sekvenční soubory. Ty lze přečíst pomocí třídy `SequenceFile.Reader`. Soubor clusteredPoint tvoří dvojice `IntWritable` a `WeightedPropertyVectorWritable` (WPV). WPV v sobě zapouzdřuje hodnotu typu double, reprezetující váhu příslušnosti ke shluku a samotný vektor.

Tento soubor obsahuje početVektorů * zadanýPočetShluků (v tomto případě 10 000 * 1 000 = 10 000 000) záznamů. Každý vektor je zopakován tolikrát, jaký je počet clusterů, vždy je k němu přiřazena váha, kterou má k příslušnému clusteru.

Listing 9.1: Načtení výstupu fuzzy k-means.

```
SequenceFile.Reader reader = new SequenceFile.Reader( ... );
IntWritable key = new IntWritable();
WeightedPropertyVectorWritable data = new
    WeightedPropertyVectorWritable();
while(reader.next(key, data)) {
   double weight = data.getWeight(); //mira prislusnosti clusteru
   Vector v = data.getVector();
   // zpracovani dat
}
```

Soubor se středy clusterů (clusters-i-final) je tvořen dvojicí `IntWritable` a `ClusterWritable`. Kombinací těchto dvou souborů je možné získat stejný soubor, který produkuje `clusterdump`, bez nutnosti mít enormní množství paměti.

9.4 Výsledky

V této kapitole budou uvedeny pouze výsledky obyčejného k-means, který zpracoval celý vstupní soubor.

Jak již bylo zmíněno kromě samotných shluků, které jsou výstupem clusteringu byly k těmto shlukům přiřazena i jednotlivá slova (vektory), které do nich spadají. Problém je, že oproti klasifikaci, kde jsou jednotlivé vektory označeny třídou, do které mají být zařazeny, tady tato vlastnost chybí.

Velikost clusteru	Četnost	Velikost clusteru	Četnost
1	13 754	14	107
2	3 018	15	101
3	1 285	16	110
4	830	17	106
5	532	18	88
6	367	19	100
7	297	20	85
8	204	21	93
9	214	22	81
10	189	23	76
11	181	24	77
12	142	25	62
13	115	26 a více	2 786

Tabulka 9.1: Tabulka četnosti clusterů, dle velikosti.

9.4.1 Velikost clusterů

Jako první se podíváme na velikosti jednotlivých clusterů. Přesněji na to, kolik vektorů obsahují:

Většina clusterů má velikost jedna. To znamená, že cluster tvoří právě jedno slovo. 942 clusterů obsahuje 100 nebo více slov. Dva obsahují více než 1000 slov a největší jich má 1 174. Střední hodnota velikosti clusteru by dle zadání měla být 16 (400 000/25 000) a výpočty tuto hodnotu potvrzují. Rozptyl má hodnotu 3495.767 a směrodatná odchylka 59.125.

Clustery velikosti jedna obsahují v ideálním případě pouze slova, která nejsou blízká žádným jiným slovům. Dokonce i tyto clustery, které neshlukují množinu slov mají svůj užitek. Velké clustery často sdružují výrazy v cizích jazycích (jiných než angličtina). Na obsahy menších a středních clusterů se zaměříme v další kapitole.

9.4.2 Kontrola clusterů

Jelikož byl použit algoritmus učení bez učitele, není tedy možné data spolehlivě zkontrolovat a ověřit kvalitu clusteringu. Náhodně tedy byla vybrána množina deseti clusterů a prověřena, zda obsahuje slova sémantický blízká.

Mnoho slov má v angličtině (stejně jako v jiných jazycích) více významů, které spolu často vůbec nesouvisí (např. cane znamená hůl, ale také třtina/rákos). Proto je zvláště těžké hodnotit takovouto úlohu. Dále je také problémem určení hranice, kdy jsou si slova ještě blízká, a kdy už ne.

Vybrané clustery obsahovaly jednoduché anglické výrazy, ke kterým byly popřípadě na internetu vyhledány alternativní významy. Celkem bylo tedy prověřeno 201 slov v deseti clusterech. Z nichž bylo 165 sémanticky blízkých ostatním slovům. 36 slov bylo shledáno již příliš vzdálených celkovému významu clusteru. Tento vzorek tedy dosahuje úspěšnosti $165/201 = 0.8209$ (82.09%).

Příklad "dobrého" clusteru

```
87213,inches,inch,cm,centimeters,millimeters,centimetres,
centimeter,millimeter,millimetres,centimetre,millimetre
```

Obsahem clusteru jsou jednotky délky (palce, milimetry a centimetry). Tento cluster byl hodnocen jako bezchybný.

Příklad "chyby" clusteru

```
318370, clustering, quicksort, k-means
```

Slova clustering a k-means jsou významově blízko, ale quicksort jako obecný řadící algoritmus již do tohoto clusteru nepatří.

9.5 Zhodnocení

V této úloze bylo představeno použití shlukovacích algoritmů k-means a částečně i fuzzy k-means. Algoritmy byly použity na volně dostupný dataset GloVe, který umožňuje shlukovat slova dle jejich sémantické podobnosti.

V úloze bylo popsáno jak upravit data do formátu použitelného v Mahoutu. Dále aplikace samotných algoritmů a získání výstupu. Byly popsány různé formy výstupu shluků včetně příkladů. V případě fuzzy k-means byl vytvořen program, který spojuje jednotlivé výstupní soubory a vytváří tak výpis podobný obyčejnému k-means.

Na konec byly prozkoumány samotné clustery. Byla zjišťována jejich velikost a to, zda opravdu obsahují slova sémanticky podobná. Ukázalo se, že shluky skutečně obsahují slova s podobným významem a shlukování na data fungovalo dobře. Velmi těžko se ovšem rozhoduje, zda jsou výrazy podobné. Velmi to závisí na osobě, která data hodnotí.

10 Závěr

V této práci byly stručně představeny základy strojového učení a algoritmů v něm používaných. Bylo zmíněno, kde je strojové učení a zpracování přirozeného jazyka využíváno dnes, a kde může pomáhat v budoucnosti.

Dále je v práci stručně popsána funkce algoritmu MapReduce, který je použit jako základ mnoha algoritmů v knihovně Mahout.

V hlavní části práce je popisována samotná knihovna Mahout. Jsou popisovány její funkce a oblasti, které umožňuje řešit. Tyto funkce jsou nejdříve obecně popsány v každé z úvodních kapitol a následně prakticky demonstrovány na příkladech.

První příklad je zpracován pro CIV a týká se reálného problému, který potřebují řešit. Zadání je z oblasti klasifikace dokumentů. Týká se automatického třídění mailů na podpoře mezi jednotlivé fronty pro zpracování. Úloha byla vypracována s použitím Naivního Bayesova algoritmu. Úspěšnost třídění mailů mezi jednotlivé fronty dosahuje velmi dobrých 92%. Tento výsledek bude prezentován na CIVu a bude nabídnuto nasazení do produkčního prostředí.

Další příklad je opět klasifikačním problémem. Tentokrát byly klasifikovány komentáře ve webových diskusích. Práce upravovala existující program nahrazením původní knihovny pro strojové učení knihovnou Mahout. Ukázalo se, že Mahout vyžaduje mnohem více úsilí při ladění klasifikátoru logistické regrese. Při nalezení dobrých parametrů dosahuje podobných výsledků jako původní knihovna.

Poslední příklad je z oblasti clusteringu. Jako zdroj dat byl použit volně dostupný dataset GloVe. Zadáním bylo vytvoření shluků slov, která mají podobný sémantický význam. Pro vypracování byl použit velmi populární algoritmus k-means. Výsledkem jsou shluky, které obsahují slova podobného

významu nebo popisující stejné téma. Velmi zajímavá je v tomto případě rychlost zpracování úlohy. Dataset obsahující 400 000 vektorů byl rozdělen do 25 000 clusterů během zhruba 43 hodin.

V každém příkladu je uveden postup od získání dat přes jejich úpravu, nasazení samotného algoritmu, až po jeho testování a použití, případně jeho ladění. Práce může velmi dobře sloužit jako úvodní dokumentace pro nového uživatele knihovny Mahout.

Přínosy práce:

- Přehledný návod popisující základy knihovny Mahout.
- Demonstrace těchto vlastností na třech reálných úlohách.
- Jedna z vytvořených úloh bude nabídnuta CIVu pro produkní nasazení.
- Dosažení velmi dobrých výsledků (především co se výkonu týče) v oblasti shlukování. Po konzultaci s vedoucím práce budou tyto výsledky použity pro další výzkum na katedře KIV.

Možné rozšíření práce:

- Popsání a otestovaní dalších algoritmů implementovaných v Mahoutu, jako například Náhodný les, Vícevrstvý perceptron, Streaming k-Means atd.
- Porovnání běhových prostředí Mahoutu (Hadoop vs. Spark).

Seznam obrázků

Seznam tabulek

Literatura

[1] Big data universe beginning to explode. CSC [online]. 2012. [cit. 25.10.2014]. Dostupné z: http://www.csc.com/insights/flxwd/78931-big_data_universe_beginning_to_explode.

[2] YouTube Statistics. YouTube [online]. 2015. [cit. 17.3.2015]. Dostupné z: https://www.youtube.com/yt/press/statistics.html.

[3] 2017 Cadillac CTS to be first with talking car capability. Chicago tribune [online]. 2014. [cit. 25.10.2014]. Dostupné z: http://www.chicagotribune.com/classified/automotive/sns-wp-blm-news-bc-gm-cadillac07-20140907-story.html.

[4] B. Liu, *Web data mining*. New York: Springer, 2nd ed. ed., c2011.

[5] K. P. Murphy, *Machine learning*. Cambridge: MIT Press, c2012.

[6] Český národní korpus: Abecední a retrográdní slovníky. Ústav Českého národního korpusu FF UK [online]. 2005. [cit. 4.5.2015]. Dostupné z: http://ucnk.ff.cuni.cz/retrograd10.php.

[7] Powered by Mahout. Mahout homepage [online]. 2014. [cit. 5.11.2014]. Dostupné z: https://mahout.apache.org/general/powered-by-mahout.html.

[8] Goodbye MapReduce. Mahout JIRA [online]. 2014. [cit. 7.11.2014]. Dostupné z: https://issues.apache.org/jira/browse/MAHOUT-1510.

[9] *MapReduce: Simplified Data Processing on Large Clusters. Google Research Publication [online]. 2004.* Dostupné z: http://static.googleusercontent.com/media/research.google.com/cs//archive/mapreduce-osdi04.pdf.

[10] PoweredBy. Hadoop Wiki [online]. 2014. [cit. 25.10.2014]. Dostupné z: https://wiki.apache.org/hadoop/PoweredBy.

[11] Top Results. Sort Benchmark Home Page [online]. 2014. [cit. 9.1.2015]. Dostupné z: http://sortbenchmark.org/.

[12] Building a recommendation engine, foursquare style. Foursquare Engineering [online]. 2011. [cit. 22.11.2014]. Dostupné z: http://engineering.foursquare.com/2011/03/22/building-a-recommendation-engine-foursquare-style/.

[13] MovieLens dataset. GroupLens [online]. [cit. 30.11.2014]. Dostupné z: http://grouplens.org/datasets/movielens/.

[14] Seznam PSČ. Seznam PSČ [online]. 2015. [cit. 4.2.2015]. Dostupné z: http://www.seznampsc.cz/.

[15] H. L. Kundel and M. Polansky, "Measurement of observer agreement1," in *Radiology*, vol. vol. 228, pp. 303–308, 2003.

[16] R. Anil, T. Dunning, and E. Friedman, *Mahout in Action*. New York: Manning Publications, 2012.

[17] P. Giacomelli, *Apache Mahout cookbook*. Birmingham: Packt Publishing, 2013.

[18] Deprecate Canopy Clustering. Mahout JIRA [online]. 2014. [cit. 7.11.2014]. Dostupné z: https://issues.apache.org/jira/browse/MAHOUT-1513.

[19] GloVe: Global Vectors for Word Representation. Stanford University [online]. 2015. [cit. 9.4.2015]. Dostupné z: http://nlp.stanford.edu/projects/glove/.

[20] Hadoop Java Versions. Hadoop Wiki [online]. 2014. [cit. 18.10.2014]. Dostupné z: http://wiki.apache.org/hadoop/HadoopJavaVersions.

Příloha A - Instalace

V této kapitole bude popsán proces instalace a nastavení frameworku Hadoop a knihovny Mahout. Předpokladem je operační systém Linux (v návodu a příkladech je využit OS Lubuntu 14.10). Dále je předpokladem nainstalovaná Oracle Java ve verzi 1.6.0_20 a vyšší. Dle oficiální wiki[20], mohou mít předchozí verze problémy (testy a reporty od uživatelů).

Postup instalace

1. Vytvoření nové skupiny hadoop a přidání nového uživatele hadoop. Tento krok slouží hlavně k oddělení souborů a procesů z budoucího HDFS od aktuálního uživatele.

```
sudo groupadd hadoop
sudo useradd -g hadoop -d home/hadoop -m hadoop
sudo passwd hadoop
[zadejte heslo pro uživatele hadoop]
```

2. Nyní si otevřete druhou konzoli a přepněte se v ní na uživatele hadoop pomocí následujícího příkazu.

```
su -l hadoop
[zadejte heslo, které jste vytvořili v předchozím kroku]
```

3. Dále vytvořte autentizační ssh klíč pro localhost. Ten pak přidejte mezi

autorizované klíče. Pro otestování správnosti, můžete vyzkoušet ssh připojení na localhost. Pokud dojde ke spojení, je tato část splněna.

```
ssh-keygen -t rsa -P ""
cat  /.ssh/id_rsa.pub »  /.ssh/authorized_keys
ssh localhost
exit
```

4. Nyní stáhněte Hadoop z oficiálního repozitáře. Pro Mahout 0.9 je doporučena verze 1.2.1. Verze 2.x nejsou kompatibilní s aktuální verzí Mahoutu. Tato chyba bude opravena v připravované verzi 1.0.

5. Stažený soubor (`hadoop-1.2.1.tar.gz`) přesuňte do domovské složky vytvořeného uživatele. Tam ho rozbalte a přidělte uživateli oprávnění pro práci s rozbalenou složkou.

```
sudo cp Downloads/hadoop-1.2.1.tar.get /home/hadoop
sudo tar -zxvf /home/hadoop/hadoop-1.2.1.tar.gz
sudo chown hadoop: /home/hadoop/hadoop-1.2.1
```

6. Upravte konfigurační soubor profile v jakékoliv textovém editoru (zde použito `vi`).

```
vi  /.profile
```

7. Na konec tohoto souboru přidejte následující hodnoty:

```
export HADOOP_INSTALL=/home/hadoop/hadoop-1.2.1
export PATH=$PATH:$HADOOP_INSTALL/bin:$HADOOP_INSTALL/sbin
```

8. Stejné hodnoty přidejte na konec souboru `/.bashrc`.

9. Přejděte do složky s konfigurací Hadoopu. Tato složka se bude nacházet v následujícím umístění: `/home/hadoop/hadoop-1.2.1/conf`

```
cd /home/hadoop/hadoop-1.2.1/conf
```

10. Zde v textovém editoru upravte soubor `hadoop-env.sh`. Na řádce 9 by se měl nacházet zakomentovaný řádek s umístěním `JAVA_HOME`. Odkomentujte tento řádek a vyplňte skutečné umístění instalace Javy.

Pozn. Umístění Javy můžete zjistit příkazem:
`ls -la /etc/alternatives/java`
V mém případě je navrácena hodnota:
`/etc/alternatives/java->/usr/local/java/jdk1.7.0_75/jre/bin/java`.
Do `JAVA_HOME` je tedy zadána hodnota: `/usr/local/java/jdk1.7.0_75/`

11. V konfigurační složce dále upravte soubor `core-site.xml`. Do tohoto souboru přidejte mezi tagy `<configuration></configuration>` následující hodnoty:

```
<property>
     <name>fs.default.name</name>
     <value>hdfs://localhost</value>
</property>
```

12. Stejným způsobem upravte i soubor `hdfs-site.xml`. Do souboru přidejte následující:

```
<property>
     <name>dfs.replication</name>
     <value>1</value>
</property>
```

13. Posledním souborem je `mapred-site.xml`. Do tohoto souboru přidejte následující:

```
<property>
     <name>mapred.job.tracker</name>
     <value>localhost:8021</value>
</property>
```

14. Aby bylo možné používat Hadoop, je nutné znovu načíst soubor bashrc,

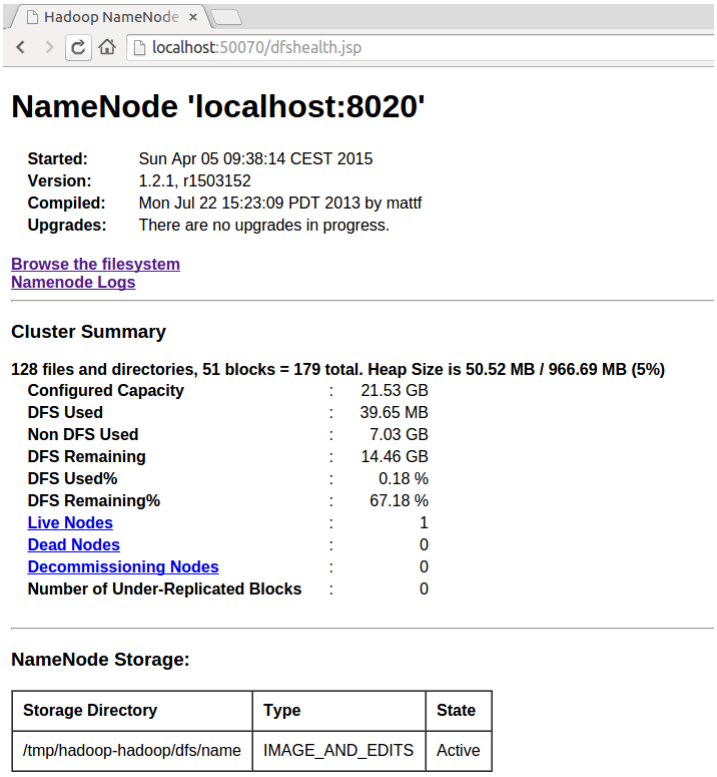
který jsme upravili. Toho docílíme příkazem:

```
source  /.bashrc
```

15. Nyní je Hadoop připraven k použití. Před prvním zapnutím je nutné naformátovat namenode. Po naformátování je možné Hadoop spustit.

```
hadoop namenode -format
start-dfs.sh
start-mapred.sh
```

16. Ve webovém prohlížeči na adrese `localhost:50070` můžete nyní sledovat stav nodu, můžete procházet obsah souborového systému (Browse the filesystem).



Obrázek 1: Webové prostředí Hadoopu.

17. Nyní zbývá instalace Mahoutu. Nejprve je nutné Mahout stáhnout z

oficiálních stránek. Následně postupujte stejně jako u Hadoopu v bodě 5.

18. Následně přidejte do souborů `/.profile` a `/.bashrc` cestu k Mahoutu.

19. Nyní lze Mahout používat.

Příloha B - Doporučení

Pro oblast doporučení byly vytvořeny tři ukázkové programy.

- `BasicExample` - základní příklad user-based doporučení
- `Testing` - test kvality doporučení
- `BestParams` - nalezení nejlepších parametrů doporučení

Všechny se nacházejí v balíčku `Mahout.Recommendation`. Jejich povinným atributem při spuštění je cesta k datovému souboru s hodnocením. Předpokladem je využití datasetu MovieLens [13], ale pokud je dodržen formát vstupu z kapitoly 4.1,lze použít jakákoliv data.

BasicExample

Jedná se o příklad použitý v kapitole 4.5. Nejjednodušší možný příklad doporučení. Výstupem je pět doporučení pro uživatele s ID 10.

Testing

Program, který otestuje kvalitu doporučení. Výstupem je odchylka hodnoty doporučení vypočtená metodou RMS viz 4.7

Listing 1: Otestování doporučení metodou RMS.

```
RecommenderEvaluator eval = new RMSRecommenderEvaluator();
RecommenderBuilder build = new RecommenderBuilder() {

   @Override
   public Recommender buildRecommender(DataModel model) throws
       TasteException {

      UserSimilarity similarity = new
          EuclideanDistanceSimilarity(model);
      UserNeighborhood neighborhood = new
          NearestNUserNeighborhood(10, similarity, model);
      Recommender rec = new GenericUserBasedRecommender(model,
          neighborhood, similarity);

      return rec;
   }
};

double score = eval.evaluate(build, null, dataModel, 0.8, 1);
System.out.println("Recommender score: " + score);
```

BestParams

Program obsahuje základní kód z ukázky výše. Tento kód je vykonáván v cyklech. Výstupem je nastavení, které má na zadaných datech minimální RMS. Příklad výstupu:

```
BEST SCORE: 0.943035262, BestAlg: 0, bestNum: 0.47, bestType: 1
```

`BestAlg` udává algoritmus výpočtu podobnosti (viz tabulka 1). `BestNum` udává parametr použitý při inicializaci sousedství. Pokud je `bestType` ro-

Podobnost	BestAlg
`EuclideanDistanceSimilarity`	0
`PearsonCorrelationSimilarity`	1
`LogLikelihoodSimilarity`	2
`SpearmanCorrelationSimilarity`	3
`TanimotoCoefficientSimilarity`	4
`UncenteredCosineSimilarity`	5
`CityBlockSimilarity`	6

Tabulka 1: Tabulka algoritmů výpočtu podobnosti a jejich indexy použité v programu `BestParams`.

ven jedné, pak `bestNum` udává počet nejpodobnějších sousedů použitých pro výpočet. Jinak určuje minimální podobnost mezi uživateli použitými pro výpočet.

Příloha C - Klasifikace

V této kapitole byly řešeny dva demonstrační příklady.

- Klasifikace mailů pro CIV
- Úprava programu FeedExtractor

Klasifikace mailů pro CIV

Upozornění: Jelikož vstupní data obsahují citlivé informace, nejsou tato data nahrána na dodaném DVD. O jejich případné poskytnutí je nutné požádat CIV.

CivDataToSeq

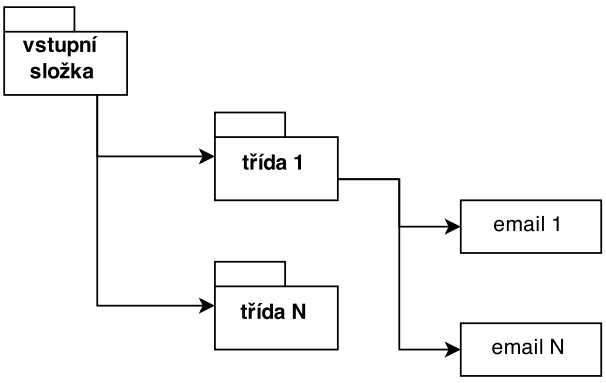
Vstupní data byla převedena do formy sekvenčního souboru programem `CivDataToSeq`, který se nachází v balíčku `mahout.classification.civ`. Jako text mailu byl brán obsah datového souboru mezi prvními dvěma escape sekvencemi (`"===> "`). Před zapsáním byly vymazány záznamy o velikosti mailu a jeho příloh (sekvence `"[num]: untitled"`). Po těchto úpravách byla data zapsána do souboru.

Použití

Program má dva povinné vstupní atributy:

- vstupní složka - Složka, která obsahuje všechna vstupní data.
- výstupní soubor - Cesta, kam má být zapsán sekvenční soubor.

U vstupní složky je předpokládáno následující rozložení souborů a složek:



CivClassifier

Program demonstruje použití natrénovaného Bayesova klasifikátoru. Má pět povinných atributů:

- natrénovaný model – model vytvořený příkazem `trainnb`
- label index – vytváří se společně s modelem přepínačem `-el`
- slovník – vytvořen ze sekvenčního souboru pomocí `seq2sparse`
- četnost slov – vytvořen ze sekvenčního souboru pomocí `seq2sparse`

- emaily - soubor obsahující emailové zprávy ke klasifikaci (jedna zpráva na řádek)

Pro spuštění je nutné zadat cesty k výše zmíněným souborům.

Úprava programu FeedExtractor

Upravený program nemá žádné vstupní atributy. Provedené úpravy viz 7.4.

Příloha D - Obsah přiloženého DVD

V kořenové složce dodaného disku jsou tři složky:

- data - Obsahuje vstupní datasety (MovieLens 100k, GloVe) a výstupy programů.
- FeedExt - Obsahuje původní a upravený program FeedExtract viz 7.
- ukazky_navody - Obsahuje demonstrační programy využité v textu práce (kromě FeedExtract).

data

Pro každou řešenou oblast existuje vlastní složka s daty.

- doporuceni - Obsahuje výstup programu BestParams a dataset MovieLens 100k.
- klasifikace - Obsahuje výstupy upraveného programu FeedExtract.
- shlukováni - Obsahuje výstupy shlukování na datasetu GloVe. A to jak pro k-means (400 000 vektorů, 25 000 shluků), tak pro fuzzy k-means (10 000 vektorů, 1 000 shluků)

FeedExt

Obsahuje dokumentaci k programu z předmětu KIV/TKS[1], dále původní a upravený program FeedExtract. Pro spuštění stačí program otevřít v obíleném IDE. Maven se postará o získání potřebných knihoven a závislostí.

ukazky_navody

Obsahuje Maven projekt s vytvořenými ukázkami programů.

[1] Teorie kognitivních systémů