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

Diplomová práce

Sumarizace rozdílů v recenzních textech

Plzeň 2018 Michal Veverka

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 14. května 2018

Michal Veverka

Abstract

Summarization of differences in review texts

The evolution of e-commerce brought with it a new type of automatic summarization. Contrastive opinion summarization is mainly used on product review texts. Its aim is the creation of two summaries, positive and negative, which capture the differences in the sentiment of reviewers for the purpose of better informativeness about the quality of the product in question.

I designed an extractive approach for contrastive opinion summarization based on known methods of sentence selection via the Latent Semantic Analysis (LSA). I also created a new method of sentence selection using LSA which is specialized for this type of summarization. I evaluated these methods on manualy created summaries of czech review texts using the ROUGE metric.

Abstrakt

S rozvojem obchodování po internetu se v posledních letech dostal do popředí nový typ automatické sumarizace. Kontrastní názorová sumarizace se zaměřuje především na recenzní texty a jejím cílem je z těchto textů vytvořit dva souhrny, pozitivní a negativní, které by zachycovaly rozdíly v cítění recenzentů za účelem lepší informovanosti zákazníků o kvalitě produktu.

V této práci jsem navrhl extraktivní způsob kontrastní názorové sumarizace založený na známých způsobech výběru vět metodou Latentní sémantické analýzy (LSA) a také jsem navrhl nový způsob výběru vět specializovaný na tento konkrétní problém sumarizace. Všechny navrhnuté metody jsem ověřil na ručně vytvořených souhrnech českých recenzních textů metrikou ROUGE.

Obsah

1 Úvod

Rozvoj internetu a mobilních zařízení v posledních dvou dekádách vedl k obrovskému nárůstu informací v podobě textů dostupných přes internet. Tento obrovský objem informací je neocenitelným zdrojem znalostí. Díky jeho každodennímu růstu jsou však lidé zaplaveni informacemi. Je tak potřeba objem těchto informací zmenšit vytvořením krátkých, věcných souhrnů, které by zachytily ty nejdůležitější informace. Z tohoto důvodu je automatická sumarizace jednou z nejintenzivnějších výzkumných oblastí zpracování přirozeného jazyka. První pokusy o automatickou sumarizaci pocházejí z 60. let 20. století a od té doby bylo navrhnuto mnoho různých přístupů k automatické sumarizaci. Ideální sumarizátor přesto stále neexistuje a oblast výzkumu je pořád aktivní.

V posledních letech se dostala do popředí zejména kontrastní názorová sumarizace. Jejím cílem je vytvoření dvou souhrnů, pozitivního a negativního, které zachycují rozdíly v cítění autora. Potřeba pro tento typ sumarizace plyne z rozvoje obchodování po internetu. Nárůst internetových recenzí produktů způsobil zahlcení zákazníků. Cílem kontrastní názorové sumarizace je poskytnout zákazníkovi krátké shrnutí těchto recenzí, za účelem lepší informovanosti zákazníka o kvalitě poskytovaných produktů a služeb.

Kontrastní názorová sumarizace je obvykle prováděna extraktivní metodou, kdy jsou souhrny skládány z vět, vyskytujících se v původním textu. Jeden z možných přístupů je založen na zkoumání aspektů, tedy diskutovaných vlastností produktu. Aspekty jsou však často příliš detailní pro správnou tvorbu souhrnu. Typy aspektů navíc silně závisí na druhu produtku, resp. služby, a sumarizátor tak musí být speciálně trénován pro určitý typ produktu. Druhým přístupem je sumarizace na základě globálních metrik důležitosti, sentimentu a podobnosti. Latentní sémantická analýza (LSA) dovoluje zkoumat významnost i podobnost jednotlivých vět a je velice vhodná pro použití při sumarizaci.

Tato práce si klade za cíl navrhnout způsob kontrastní názorové sumarizace pomocí metody LSA a ověřit kvalitu takového řešení na recenzích restaurací v českém jazyce metrikou ROUGE.

2 Data

Data obsahující recenze restaurací k sumarizaci pocházejí z práce K. Ježka a M. Campra [4]. Tato data byla zvolena za účelem porovnání zde vytvořeného sumarizátoru s výsledky dosaženými v této práci. Pro účely jejich práce vznikl dataset recenzí restaurací z české stránky *www.fajnsmekr.cz*. Z této stránky nejprve stáhli celkem 6008 recenzí na 1242 restaurací. Z tohoto souboru recenzí následně vybrali 50 restaurací a ke každé z nich několik recenzí tak, aby dohromady recenze každé z 50 restaurací obsahovaly minimálně 1000 slov [4].

Následně 3 nezávislí anotátoři vytvořili pro každou restauraci dvě abstraktní souhrnná hodnocení, pozitivní a negativní, každé s přibližně 100 slovy. V případech, kdy nebyly recenze dostatečně obsáhlé, byly vytvořeny kratší souhrny.

Nastala-li však situace, kdy vybrané recenze restaurace neobsahovaly dostatek pozitivních nebo negativních hodnocení, byly do vytvořených souhrnů vybrány i věty se slabším sentimentem. Průměrné ROUGE skóre shody mezi jednotlivými anotátory bylo následující:

- ROUGE-1: 0.61101
- ROUGE-2: 0.44537
- ROUGE-SU4: 0.41752

Tyto recenze jsou použity k zhodnocení funkčnosti sumarizátoru a jeho porovnání s jinými metodami.

3 Předzpracování

Předzpracování dat je jádrem zpracování přirozeného jazyka a předchází téměř všem úkolům. Jedná se o úpravu textu, která napomohá větší efektivitě algoritmů zpracování přirozeného jazyka. Samotné předzpracování se skládá z dvou hlavních částí a to:

- tokenizace,
- normalizace.

Pro označení částí textu v průběhu předzpracování používáme následující výrazy:

- slovo - samostatná posloupnost znaků, tak jak se vyskytuje v textu
- term - normalizované (předzpracované) slovo
- token - výskyt slova nebo termu v textu

Konkrétně použité metody popisuje sekce 9.2

3.1 Tokenizace

Tokenizace je členění textu na menší celky (odstavce, věty, slova) a je to první část předzpracování. V této práci půjde o dělení textu na věty a následné dělení vět na slova. Jako základní oddělitele vět se používají znaky, které ukončují věty. Tedy tečky, otazníky a vykřičníky. To může v některých případech činit problémy. Uvažme např. následující větu:

Pracoval společně s P. E. Richardsonem na nové metodě.

Tečka uprostřed věty zde není použita jako ukončující symbol. Pro správné dělení je tak zapotřebí složitějších metod.

Dělení textu na slova se provádí obdobně. Základní dělící jednotkou jsou mezery, čárky a pomlčky. Problémové jsou zde však zejména víceslovné názvy, datumy a čísla. Pro tokenizaci se nejčastěji používají regulární výrazy.

3.2 Normalizace

Máme-li text rozdělený na slova, přichází na řadu normalizace. Normalizace je proces, při kterém jednotlivá slova převedeme na termy. Normalizace se skládá z několika částí, jejich pořadí však závisí na daném problému, v některých případech mohou být některé části vynechány. Jednotlivé části normalizace jsou popsány následovně v pořadí, v jakém jsou obvykle použity.

3.2.1 Odstranění diakritiky a kapitálek

Jako úplně první část normalizace se provádí odstranění diakritiky. V českém jazyce se jedná o háčky a čárky. Následně se provede odstranění kapitálek. I zde však může dojít k nejednoznačným situacím, např. zkratka *MIT* (*Massachusetts Institute of Technology*) může být po redukci na malá písmena zamněněna za německé zájmeno *mit* (*s, čím*). V praxi se však toto neřeší a všechna slova projdou převodem na malá písmena. Tyto dvě transformace se provádí z toho důvodu, že v neformálních textech není obecně dbáno na pravopis a zejména na internetu lidé zpravidla nepoužívají diakritiku ani velká písmena.

3.2.2 Stemming a lemmatizace

Lemmatizace je proces převodu slova do základního tvaru, tzv. *lemma*. V češtině se například jedná o převod podstatných slov do 1. pádu (slovo *barvě* na tvar *barva*) nebo převod sloves do neurčitého tvaru (*uděláme* na tvar *udělat*). Problémová jsou však slova mnohovýznamová, např. slovo *tancích* je možné převést na slova *tanec* i *tank*. Z tohoto důvodu, a také protože tvorba dobrého lemmatizátoru je velice náročná, se namísto lemmatizace používá stematizace. Stematizace (angl. *stemming*) je hrubý proces, při kterém jsou konce slov odřezány, za účelem dosažení podobného výsledku jako při lemmatizaci. V češtině se tak ořezávají známé koncovky, případně předpony, např. *ne-*. Stematizace sice nedosahuje tak dobrých výsledků jako lemmatizace, avšak vytvoření dostatečně dobrého stemmeru je mnohem jednodušší, než vytvoření dobrého lematizátoru.

3.3 Odstranění nevýznamových slov

Nevýznamová slova, neboli *stop slova*, jsou velmi častá slova, která samostatně nenesou význam a nevypovídají tak o obsahu textu. Odstraňují se za účelem zrychlení algoritmů zpracování přirozeného jazyka a dosažení lepších

výsledků. Jedná se o spojky, zájmena a další nevýznamová slova. V češtině jsou to například slova *ale, anebo, ačkoli, se*. Taková slova jsou z textu odstraněna a nepokračují tak do další části zpracování.

4 Významnost textu

Při sumarizaci bychom chtěli do výsledných souhrnů zahrnout především nejvýznamnější části textu. Jak však v textu najít tyto části? Pokud na text nahlížíme jako na *bag-of-words*, pak je významnost textu dána vyznamností jednotlivých termů, které daný text tvoří. Jednou z nejpoužívanějších metrik pro hodnocení významnosti termů je tf-idf skóre.

4.1 TF-IDF

Tf-idf (term frequency - inverse document frequency) je jeden z nejpoužívanějších způsobů pro vážení termů a využívá se v celé řadě metod zpracování přirozeného jazyka. Základní myšlenkou je předpoklad, že termy vyskytující se v textu méně častěji mají větší výpovědní hodnotu, než termy vyskytující se velmi často. Vážením se tak snažíme přidat těmto termům větší význam. Při vážení se využívají následující hodnoty:

- *term frequency* $tf_{t,d}$ je počet výskytů termu t v dokumentu d
- pro lepší vážení se používá $tf_{t,d} = 1 + log_{10} tf_{t,d}$.
- *document frequency* df_t je počet dokumentů, ve kterých se term vyskytuje
- *inverse document frequency* je měřítko informativnosti termu t a získá se jako $idf_t = log_{10}(N/df_t)$, kde N je počet dokumentů
- celková váha $w_{t,d} = tf_{t,d} * idf_t$

Díky tomuto vážení můžeme jakýkoliv text reprezentovat jako vektor tf-idf vah a takto s ním pracovat.

5 Polarita textu

Jelikož provádíme kontrastní sumarizaci a chceme vytvářet souhrny pozitivní a negativní, je třeba použít některou z metod pro určení sentimentu vět. Cílem určení sentimentu textu je odhadnout, jak ho jeho autor vnímal. Obecně lze sentiment rozdělit dle řady emocí:

- radost, nadšení
- pobavení
- naštvání
- odpor

Toto rozdělení je však stále příliš složité pro současné metody a většinou si postačíme s polaritou textu, tedy rozdělením na:

- pozitivní
- neutrální
- negativní

I zde je možné dělit polaritu do několika stupňů: např. velmi pozitivní a pozitivní.

5.1 Slovníková metoda

Slovníková metoda se opírá o slovník ohodnocených slov. Většinou se jedná o slovník pozitivních a negativních lemmat, někdy ale také může mít pozitivita a negativita více stupňů. Každé slovníkové lemma tak má např. přiřazenu hodnotu od -2 do 2. Slovník pozitivních slov by obsahoval slova jako *skvělé, výborné, chutnat, líbit*. Slovník negativních slov by zase obsahoval slova záporného cítění.

Následující skóre pochází z práce [4]. Zde navrhli dva způsoby skórování pomocí slovníků. První skóre S_s s názvem Lex odpovídá sumě polaritních vah jednotlivých slov věty s:

$$S_s = \sum_{i=1}^{n} w[i]_s \tag{5.1}$$

Druhý způsob skórování věty se jmenuje LexWeight a každé slovo je zde váženo svou pozicí ve větě následovně:

$$S_s = \sum_{i=1}^{n} \frac{w[i]_s}{i}, \tag{5.2}$$

kde n je počet slov ve větě S a $w[i]$ je polaritní váha i-tého slova. Tímto způsobem dostane největší váhu první slovo věty.

5.2 Metody založené na slovních vektorech

Použití slovních vektorů k určování polarity bylo předvedeno v [4]. Autoři využili slovní vektory natrénované sítí Word2Vec a navrhli zde dva způsoby skórování. Při návrhu této metody vycházeli z předpokladu, že vektorový prostor Word2Vec dokáže zachytit nejen význam a podobnost slov, ale i jejich polaritu.

5.2.1 Word2Vec skóre 1

První skóre se zakládá na měření vzdálenosti jednotlivých slov věty od slov *dobrý* a *špatný*. Vzdálenost měřili ve vektorovém prostoru Word2Vec pomocí kosínové podobnosti. Tímto způsobem lze získat dvě polaritní skóre slova w: pozitivní skóre w_p a negativní skóre w_n. Polaritní skóre celé věty s, tedy S_s je:

$$S_s = \sum_{i=1}^{m} w[i]_p^r - w[i]_n^r, \tag{5.3}$$

kde m je počet slov ve větě a r je volitelný parametr pro vážení skóre.

5.2.2 Word2Vec skóre 2

Druhé skóre je velice podobné prvnímu, ale využili zde slovníku pozitivních a negativních slov (sek. 5.1). Namísto měření vzdálenosti od slov *dobrý* a *špatný* měřili průměrnou vzdálenost od všech slov ve slovníku pozitivních a negativních slov.

5.3 Klasifikační metody

Další možností pro určení polarity je použití klasifikačních metod. Existuje celá řada těchto metod. Cílem klasifikátoru je přiřadit objektu, který reprezentujeme pomocí příznaků x, jednu ze tříd y. Hledáme tedy funkci f

takovou, že

$$f : x \rightarrow y. \tag{5.4}$$

Při klasifikaci dokumentů máme dokument reprezentován vektorem příznaků a snažíme se tento dokument přiřadit do jedné z k tříd. Volba tříd zavisí na účelu klasifikace. Novinovým článkům můžeme přiřazovat témata, emailové zprávy dělit na spam a ne-spam. V případě polarity dělíme dokumenty nejčastěji na pozitivní a negativní.

5.3.1 Příznaky

Nejpoužívanějšími příznaky v textové klasifikaci jsou přímo termy vyskytující se v dokumentu. V některých případech však můžeme některé typy termů seskupovat do skupin a reprezenzovat je jako jeden. Například všechna čísla v dokumentu můžeme nahradit příznakem *číslo*, podobně můžeme přistupovat k chemickým vzorcům. Často používané jsou také tzv. *n-gramové* příznaky. *N-gram* je posloupnost n slov vyskytující se přímo v textu. Nejčastěji používané n-gramy jsou unigramy, bigramy a trigramy.

5.3.2 Maximum Entropy Classifier

Jedním z významných zástupců klasifikačních metod je Maximum Entropy Classifier. Jedná se o metodu strojového učení, která je založena na principu *maximální entropie*. Tento princip říká, že

Pravděpodobnostní rozdělení, které nejlépe odpovídá současnému stavu vědění, je to s největší entropií.

Maximum Entropy Classifier se tak snaží ze všech modelů, které reprezentují naše data, vybrat ten s maximální entropií. Maximalizací entropie dosáhneme toho, že do klasifikačního systému nezavedeme nechtěný informační bias.

Pravděpodobnostní distribuce s maximální entropií má exponenciální formu:

$$P(c|d) = \frac{1}{Z(d)} exp(\sum_i \lambda_i f_i(d, c)), \tag{5.5}$$

kde d je dokument ke klasifikaci, c je hledaná třída, $P(c|d)$ je podmíněná pravděpodobnost třídy c máme-li dokument d, $f_i(d, c)$ je příznaková funkce, λ_i je parametr, který je třeba odhadnout a $Z(d)$ je normalizační faktor ve tvaru:

$$Z(d) = \sum_c exp(\sum_i \lambda_i f_i(d, c)). \tag{5.6}$$

Nalezení parametrů λ se provádí pomocí metod gradientního sestupu, např. *Improved Iterative Scaling*. [9]

6 Podobnost

Při zkoumání podobnosti textů zkoumáme jejich sémantickou podobnost. Sémantická podobnost vypovídá o podobnosti významů obou textů. Zkoumání sémentické podobnosti je u sumarizace velice důležité, jelikož požadujeme, aby vytvořené souhrny neobsahovaly sémanticky podobné informace. Metody zkoumání sémantické podobnosti jsou založené na dvou předpokladech: *bag-of-words* hypotéza a *co-occurrence* hypotéza. *Bag-of-words* hypotéza zní následovně:

Význam textu je dán pouze slovy, které se v něm vyskytují. Nezávisí na jejich pořadí.

Hypotéza souvýskytu (*co-occurrence*) vypovídá o následujícím:

Slova vyskytující se v okolí stejných slov mají podobný význam.

Jako příklad můžeme uvést slova *hora*, *údolí*, *řeka*, která se budou v textu pravděpodobně vyskytovat v okolí stejných slov. To samé platí o slovech *kočka*, *myš* a *pes*.

6.1 HAL

Hyperspace Analoge to Language (HAL) je metoda postavená na statistickém modelu sémantické paměti a byla vyvinuta Kevinem Lundem a Curtem Burgessem na Kalifornské univerzitě. Základní myšlenkou této metody je právě hypotéza souvýskytu (*co-occurence*).

HAL používá čtvercovou matici $n \times n$ pro n jedinečných slov, která se vyskytují v textu. Označíme-li tuto matici H, pak prvek H_{ij} vyjadřuje vztah mezi termem i a j. Jak je vidět, matice je symetrická. Každé slovo je tak vyjádřeno řádkem a sloupcem, které reprezentují jeho vztahy s ostatními slovy. Při analýze čteme text slovo po slovu a pro každé slovo vyhledáme jeho řádek v matici H. Poté se v textu podíváme na x slov nalevo a napravo od tohoto slova a pro každé takové sousední slovo upravíme příslušný prvek ve vyhledaném řádku. Tomuto způsobu procházení textu se říká *posuvné okénko*. Posuvné okénko tvoří oněch x slov nalevo a napravo. Při samotné úpravě daného prvku matice můžeme vzít v potaz blízkost slov. Bližším slovům dáme větší váhu a vzdálenějším nižší. Např. slovo, které se od středu

vyskytuje ve vzdálenosti 5 bude mít váhu 2, zatímco slovo ve vzdálenosti 1 bude mít váhu 6.

Po zpracování celého textu budou mít souvyskytující se slova podobné řádky v matici H. Tyto řádky jsou poté reprezentovány jako vektory a mohou být porovnány pomocí kosínové podobnosti. Výsledná vzdálenost mezi řádky pak odpovídá podobnosti mezi slovy, které řádky reprezentují. [13]

6.1.1 Algoritmus

Začínáme s matici H o rozměrech $n \times n$, kde n je počet unikátních slov v textu. Pokud tedy např. text obsahuje 100 000 slov, z toho 10 000 jedinečných, pak bude mít matice H rozměry 10000×10000. Každý prvek matice je sumou všech hodnot získaných z průchodu posuvného okénka textem. Velikost posuvného okénka x je volitelná, obvyklé hodnoty jsou 5 nebo 8. Následně pro každé slovo v textu na pozici t přidáme do matice hodnoty získané z párů slova t se slovy $t - x$ až $t + x$. Pro každý takový pár pak do matice přidáme hodnotu určenou následujícím vzorcem:

$$x - |t - j| + 1, \tag{6.1}$$

kde t je pozice středového slova, j je pozice druhého slova (např. *t-3*) a x je velikost posuvného okénka. Takto dostanou slova bližší středu větší váhu, než slova vzdálenější středu. Celý algoritmus vypadá takto:

1. Inicializace všech prvků matice H na 0.
2. Pro každé slovo textu w_i na indexu i:
   (a) Pro každé slovo w_j na indexu j v rozsahu $i - x$ až $i + x$:
      i. $H_{ij} += x - |i - j| + 1$

Po skončení tohoto algoritmu matice H obsahuje výsledné vektory určující podobnost jednotlivých slov. Tyto podobnosti poté mohou být použity k určení podobnosti větších částí textu, jako jsou věty, odstavce nebo celé dokumenty. [12]

6.2 LSA

Latentní sémantická analýza (LSA) je metoda zpracování přirozeného jazyka patentovaná v roce 1988. Slouží k získání a reprezentaci kontextového významu slov aplikováním statistických výpočtů na soubor textů. Po zpracování velkého objemu strojově čitelných dat reprezentuje LSA slova, věty a

odstavce jako body ve vysoce dimenzionálním (50-1500) *sémantickém prostoru*. [6]

6.2.1 Inspirace

LSA je stejně jako metoda HAL založena na hypotéze *bag-of-words* a hypotéze souvýskytu. Díky těmto dvěma principům je LSA schopna odhalit skryté sémantické vztahy a koncepty.

Způsobem, jakým toho LSA dosahuje, je redukce dimenzionality. Redukce dimenzionality je matematický nástroj, jakým při aplikaci na text získáváme znalosti o vztazích mezi slovy, které nejsou na první pohled zřejmé z jejich distribuce v textu. Představme si matici slov a úryvků textu, ve kterých se tato slova vyskytují. V neredukovaném stavu mohou být data reprezentována v libovolném hyperdimenzionálním prostoru, který má nejvýše tolik dimenzí, kolik máme různých slov a pasáží textu. Pokud reprezentujeme data v jejich maximální možné dimenzionalitě, získáme distribuci slov tak, jak se v textu vyskytují. V redukovaném dimenzionálním prostoru budou data reprezentována nejlépe jak je to možné, ale ne perfektně. Některá data se tak budou vyskytovat blíže nebo dále než v původní distribuci. Sémanticky podobná data se budou vyskytovat blíže. Matematickým aparátem pro redukci dimenzionality je metoda singulárního rozkladu (*singular value decomposition*). [14]

Úspěch metody LSA je někdy vysvětlen přirovnáním LSA k psychickým procesům lidí při získávání sémantických znalostí pouze z přímého kontaktu s jazykovými daty. LSA nepředpokládá žádné předem získané znalosti, které by napomáhaly při rozpoznávání významu slov a vět. LSA pouze předkládá obecný postup získávání sémantické struktury textů, který je obdobný tomu, čím lidé mohou procházet při stejném procesu.

6.2.2 Singular value decomposition

Singular value decompostion (SVD - singulární rozklad) je matematický aparát pro faktorizaci matic. Jedná se o teorém lineární algebry, říkající, že každá $n \times m$ matice M, jejíž prvky jsou reálná čísla, může být rozložena do matic U, Σ, V^T tak, že

$$M = U\Sigma V^T. \tag{6.2}$$

U je matice o rozměrech $m \times m$ a V^T je matice o rozměrech $n \times n$. Obě matice mají ortonormální sloupce. Matice Σ je diagonální matice o rozměrech $m \times n$ a vypadá takto:

$$\Sigma = \begin{bmatrix} s_1 & 0 & \cdots \\ 0 & s_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, \tag{6.3}$$

Hodnoty na diagonále matice Σ se nazývají singulární čísla (hodnoty) matice M a jsou seřazena od nejvyššího po nejmenší. Singulární čísla jsou odmocniny vlastních čísel matice $M^T M$.

Singulární hodnoty reprezentují "dimenze významu" pro slova a části textu. Matice M může být znovu sestrojena roznásobením matic U, Σ a V^T. [14]

Dimenzionalita sémantického prostoru může být snížena nahrazením některých singulárních čísel nulou. Typicky se postupuje od nejnižších singulárních čísel po nejvyšší. Nechť ponecháme pouze k-nejvyšších singulárních čísel, redukujeme tak matici Σ na matici Σ_k a podobně redukujeme matice U, V^T na matice U_k, V_k^T tak, že ponecháme pouze prvních k sloupců, resp. řádků. Matice M je tak aproximována svojí k-dimenzionální rekonstrukcí

$$M_k = U_k \Sigma_k V_k^T. \tag{6.4}$$

Jelikož mají matice U_k, Σ_k, V_k^T rozměry $m \times k$, $k \times k$, $k \times n$, výsledek jejich násobení bude mít rozměry $m \times n$, stejně jako originální matice M.

Nyní se podíváme na význam jednotlivých matic pro LSA. Matice U má rozměry $n \times m$ a každý její sloupec může být považován ze reprezantaci *tématu* (někdy také *konceptu*), tedy kombinaci slov ze vstupních dat danou vahami, které jsou vyjádřeny jednotlivými hodnotami sloupců. Matice Σ je diagonální a jednotlivé hodnoty vyjadřují váhy patřičných *témat*. Matice V^T je vyjádřením jednotlivých vět (odstavců, pasáží, ...) v prostoru *témat*. Každý sloupec odpovídá jedné větě. [1] V těchto maticích tak máme reprezentována slova a věty v prostoru konceptů a zároveň koncepty v prostory slov a vět.

Zkoumáním těchto matic můžeme tvořit úvahy o podobnosti slov a vět. Věty a slova lze reprezentovat jejich vektory v příslušných maticích, jejichž prvky vypovídají o vztahu slov, resp. vět k jednotlivým konceptům. Tyto vektory lze dále porovnávat pomocí kosínové podobnosti nebo korelační metody.

6.3 Slovní vektory

Pro hodnocení sémantické podobnosti lze také využít slovní vektory. Slovní vektory jsou vektorové reprezantace slov natrénované neuronovými sítěmi

zkoumáním souvýskytu slov ve velkých souborech textu. Takové vektory poté tvoří vektorový prostor, ve kterém lze porovnávat sémantickou podobnost slov a textů. Každé slovo je v tomto vektorovém prostoru reprezentováno svým vysokodimenzionálním vektorem (50-300). Vektory sémanticky podobných slov jsou v tomto prostoru blízko sebe. Nejznámějším modelem pro trénování takových slovních vektorů je model *Word2Vec*.

K vytvoření vektorové reprezentace celé věty je nutné slovní vektory jednotlivých slov věty zkombinovat. Nejčastějším způsobem takové kombinace je průměrování. Poté můžeme určit podobnost dvou dokumentů d_1 a d_2 reprezantovaných jejich příznakovými vektory v_1 a v_2 pomocí kosínové podobnosti:

$$sim(v_1, v_2) = \frac{\sum_{i=1}^{n} v_1[i] * v_2[i]}{\sqrt{\sum_{i=1}^{n} v_1[i]^2} * \sqrt{\sum_{i=1}^{n} v_2[i]^2}} \tag{6.5}$$

7 Sumarizace

Automatická sumarizace je podúlohou zpracování přirozeného jazyka. Jejím úkolem je redukce velkého objemu textu za účelem snadnější zpracovatelnosti uživatelem. Hlavním aspektem sumarizace je vybrání těch nejdůležitějších informací, které nejlépe reprezentují původní text. Vytvoření krátkého souhrnu umožní uživatelům získání představy o podstatě dokumentu, aniž by museli číst dokument v celé jeho podobě. Výstupem sumarizace by měl být plynulý, čitelný a pochopitelný text. V jednodušší podobě je souhrn tvořen z jednoho dokumentu, ve složitějším případě je souhrn tvořen z více dokumentů. V takovém případě je nutné použití metrik sémantické podobnosti, abychom zabránili redundanci informací v souhrnu.

Problém sumarizace není triviální. Na začátku sumarizace je nutné si položit následující dvě otázky. Co jsou původní dokumenty? A co je důležitá informace? Odpověďi na tyto dvě otázky závisí na doméně daného problému sumarizace a není vždy jednoduché je nalézt. Doména určuje vlastnosti tvořených souhrnů, způsob výběru vět, a hraje důležitou roli v návrhu a nastavení sumarizátoru. Některé domény automatické sumarizace zahrnují:

1. *Zpravodajské články*
2. *Výsledky webových vyhledávačů*
3. *Recenze*
4. *Vědecké články*

7.1 Typy sumarizace

Typ sumarizace lze rozlišovat dle řady kritérií. Kam konkrétní problém sumarizace spadá, záleží především na doméně sumarizovaných dokumentů, požadavcích na výsledné souhrny a cílovém uživateli.

7.1.1 Dělení dle účelu

Pro správnou tvorbu automatického sumarizátoru je nutné určit účel samotné sumarizace. Od účelu sumarizace se odvíjí podoba tvořených souhrnů, jejich délka a kompozice.

Indikativní sumarizace

Účelem indikativní sumarizace je naznačit uživateli typ informací, které může v dokumentu očekávat. Uživatel se podle této informace rozhoduje, zda bude dokument dále číst. Tento typ sumarizace je častý zejména v oblasti vyhledávání informací.

Informativní sumarizace

Informativní sumarizace má za úkol do souhrnu zahrnout ty nejpodstatnější informace, které dokument obsahuje. Takový souhrn poté nahrazuje původní dokument.

Názorová sumarizace

Smyslem názorové sumarizace je zachycení autorova názoru na předmět dokumentu. Názorová sumarizace je důležitá pro prodejce a zákazníky. Často je sumarizace rozdělena na tvorbu pozitivních souhrnů a negativních souhrnů.

Komparativní sumarizace

Cílem komparativní sumarizace je analýza více dokumentů a následné vytvoření souhrnu, který bude obsahovat nejvýznamnější rozdíly mezi těmito dokumenty.

Kontrastní sumarizace

Kontrastní sumarizace je velice podobná komparativní sumarizaci. I zde chceme do souhrnu zahrnout rozdíly mezi sumarizovanými dokumenty. Kontrastní sumarizace však bere v potaz i sentiment autorů dokumentů a snaží se tvořit dva souhrny: pozitivní a negativní. Tento způsob sumarizace se často používá právě při sumarizaci produktových recenzí.

7.1.2 Dělení dle způsobu

Kromě účelu sumarizace lze také souhrny dělit dle způsobu jejich vytváření. Jedná se o základní rozdělení sumarizátorů do dvou skupin: *abstraktivní* a *extraktivní*.

Abstraktivní sumarizace

Abstraktivní sumarizace vytváří souhrny, které neobsahují věty vyskytující se v původním dokumentu. Namísto toho je souhrn tvořen novými větami. V

některých případech lze převzít výrazy a spojení z původního dokumentu, ale celkový souhrn je stále považován za nový text. Abstraktivní sumarizátory využívají jazykových modelů pro tvorbu smysluplných vět. I přesto však často dochází ke ztrátě smysluplnosti, jelikož jazykové modely mají stále svá omezení. Z tohoto důvodu je abstraktivní sumarizace výrazně složitější a méně často používaná než sumarizace extraktivní.

Extraktivní sumarizace

Při extraktivní sumarizaci jsou z původních dokumentů vybírány celé věty a pomocí nich jsou poté tvořeny souhrny. Častým krokem je skórování důležitosti jednotlivých vět za účelem výběru těch nejpodstatnějších částí. Extraktivní sumarizace se skládá ze tří částí [1]:

1. **Získání reprezentace dokumentu nebo dokumentů** - i ty nejjednodušší metody sumarizace potřebují ke své práci reprezentaci sumarizovaného textu, pomocí které mohou určit významné informace. Reprezentace pomocí *témat* (angl. *topic*) se snaží reprezentovat dokument jako kolekci témat, která se v textu vyskytují. Tento způsob využívají ty nejpopulárnější sumarizátory a jednotlivé přístupy se mohou velice lišit ve složitosti a síle reprezentace. *Indikátorové přístupy* narozdíl od toho reprezentují jednotlivé věty množinou indikátorů jejich významnosti: délka věty, pozice v dokumentu, výskyt důležitých frází atp.

2. **Hodnocení vět** - jakmile máme vytvořenou reprezentaci dokumentu, můžeme jednotlivým větám přiřadit skóre. Při reprezentaci pomocí témat je skóre odvozeno od schopnosti věty vyjádřit téma nebo kombinaci témat, často svou váhou v jednotlivých tématech. Indikátorové metody tvoří skóre věty váženou kombinací jednotlivých indikátorů.

3. **Výběr vět** - nakonec je do souhrnu vybrána kombinace nejdůležitějších vět. V přístupu *nejlepších n* je vybráno n vět s nejvyšším skóre, tak aby měl souhrn požadovanou délku. Tento přístup je možné obohatit o uvažování redundance vět v souhrnu. V takovém případě je výběr vět iterativní, v každém kroku dojde k přepočítání skóre, které je lineární kombinací ohodnocení významnosti věty a její podobnosti se zbytkem souhrnu. Cílem výběru je maximalizace důležitosti, minimalizace redundance a maximalizace konzistence. Tyto požadavky jsou mnohdy protichůdné.

7.2 Sumarizace pomocí LSA

Jak již bylo řečeno v kapitole 6.2, aplikací singulárního rozkladu na matici výskytu slov M získáme tři matice

$$M = U\Sigma V^T. \tag{7.1}$$

Tyto matice lze použít nejenom pro určení podobnosti dvou vět, ale také pro určení významnosti tématu a korelaci vět s tématem. Významnost tématu je dána patřičnou hodnotu na diagonále matice Σ. Dále je možné zjistit, jakou váhou přispívá věta k libovolnému tématu prozkoumáním matice V^T. Tyto tři matice tak poskytují bohatý základ informací o podstatě dokumentu a jeho částech, který lze využít při sumarizaci. Postup sumarizace metodou LSA odpovídá krokům extraktivní sumarizace, Význam jednotlivých kroků je následující:

1. Reprezentace dokumentu - stavba matice souvýskytu M a její rozklad singulární dekompozicí dle $M = U\Sigma V^T$.
2. Hodnocení významnosti vět - liší se dle daného přístupu, viz. 7.2.2
3. Výběr vět - liší se dle daného přístupu, viz. 7.2.2

7.2.1 Stavba matice

Vstupem singulárního rozkladu je u LSA matice výskytu slov M. Řádky odpovídají jednotlivým slovům a sloupce větám. Způsob tvorby matice M je velice důležitý a přímo určuje podobu matic vypočítaných singulárním rozkladem. Jelikož je singulární rozklad výpočetně náročný, je vhodné před tvorbou matice redukovat počet použitých slov, např. procesy předzpracování popsanými v kap. 3. Existuje několik způsobů jak určit jednotlivé prvky matice M [11].

- Frekvence slova: hodnota prvku matice je určena počtem výskytu slova v dané větě (slova se však ve větách často neopakují).
- Binární výskyt: prvek matice obsahuje 1, pokud věta obsahuje slovo, jinak 0.
- Tf-Idf váha: prvky matice jsou určeny tf-idf vážením jak bylo popsáno v sekci 4.1.
- Log-entropie: matice je vyplněna hodnotou log-entropie slova, což podává informaci o důležitosti slova ve větě.

- Modifikovaná tf-idf: navrhnuto v [10] za účelem redukce hluku ve vstupní matici. Prvky matice jsou nejprve doplněny klasicky svými tf-idf vahami. Následně je pro každý řádek matice spočítán průměr a ponechány jsou jen ty hodnoty prvků řádku, které jsou větší než jeho průměr.

7.2.2 Jednotlivé přístupy použití LSA

Věty je možné do výsledných souhrnů vybírat řadou způsobů za použití výsledků singulárního rozkladu. Následuje popis těch nejznámějších přístupů.

Gong a Liu (2001)

Algoritmus navrhnutý v [3] je prvním navrženým způsobem výběru vět do souhrnů pomocí LSA. Je založen na reprezentaci vět pomocí témat. Po provedení singulárního rozkladu je použita matice *témata* × *věty*. V této matici jsou témata seřazena od nejdůležitějšího po nejméně důležité. Jedna věta je vybrána z nejdůležitějšího téma, další věta z druhého nejdůležitějšího téma a tak dále, dokud nemá souhrn požadovanou délku. Algoritmus tvorby souhrnu je následující:

1. Sestav matici výskytu M a proveď rozklad $M = U\Sigma V^T$.
2. Nastav i=1.
3. Dokud nemá souhrn požadovanou délku:
   (a) Vyber větu s největší hodnotou prvku v i-tém řádku matice V^T.
   (b) $i++$

Redukce dimenze v tomto případě nemá své opodstatnění, jelikož počet dimenzí musí být alespoň tak velký, jako počet vět generovaných souhrnů. Větší počet dimenzí se při výběru neprojeví, jelikož bereme v potaz jen prvních n témat, resp. dimenzí, kde n je počet vět souhrnu.

Dále má tento přístup dvě nevýhody. První nevýhodou je to, že pokud je délka tvořeného souhrnu příliš veliká, jsou do souhrny vybrány i věty z málo významných témat. Druhým problémem je výběr pouze jedné věty z každého tématu. Některá témata, obzvláště ta nejdůležitější, je vhodné reprezentovat v souhrnu více větami.

Steinberger a Ježek (2004)

Přístup Steinbergera a Ježka [15] se podstatně liší od přístupu Gong a Liu. Gong a Liu hodnotí věty dle patřičnosti k jedinému tématu. Steinberger a Ježek hodnotí větu dle její patřičnosti k více vybraným tématům (jejich počet závisí na zvolené dimenzi redukovaného prostoru). Za tímto účelem používají k hodnocení vět délku jejich vektorů v matici ΣV^T. Prostor je možné redukovat, abychom omezili vliv méně významných témat. Pokud například chceme hodnotit věty na základě jejich patřičnosti k pěti nejvýznamnějším tématům, redukujeme počet dimenzí na pět. Postup tvorby souhrnu vypadá takto:

1. Sestav matici výskytu M a proveď rozklad $M = U\Sigma V^T$.
2. Proveď redukci na požadovaný počet dimenzí n.
3. Vyber požadovaný počet vět s největší délkou získanou jako

$$s_k = \sqrt{\sum_{i=1}^{n} v_{k,i}^2 * \sigma_i^2}, \tag{7.2}$$

kde s_k je délka vektoru k-té věty, $v_{k,i}$ je prvek matice V^T a σ_i^2 je prvek na diagonále matice Σ.

Výhodou této metody je nezávislost počtu ponechaných dimenzí na počtu vybíraných vět. Nevýhodou je zanedbání vlivu negativních hodnot v matici V^T. Tyto hodnoty se vlivem druhé mocniny projeví stejně, jako hodnoty pozitivní, což může být nežádoucí účinek.

Murray a spol. (2005)

Murray a spol. [8] se rozhodli vylepšit původní metodu výběru Gong a Liu a odstranit problém výběru pouze jedné věty na téma. Tímto způsobem lze vybrat více než jednu větu z nejvýznamnějších témat. Počet vět vybraných z daného téma je určen poměrem jeho singulární hodnoty v matici Σ vůči sumě singulárních hodnot všech témat. Vzorec pro určení této distribuce je:

$$d_i = \frac{s_i}{\sum_{j=1}^{n} s_j} * k, \tag{7.3}$$

kde s_i je i-tá singulární hodnota na diagonále matice Σ, n je ponechaný počet dimenzí a k je požadovaná délka souhrnu. Celý algoritmus výběru je:

1. Sestav matici výskytu M a proveď rozklad $M = U\Sigma V^T$.
2. Nastav i=1.

3. Spočítej distribuci d vět v tématech.

4. Dokud nemá souhrn požadovanou délku:

   (a) Vyber d_i (počet vět pro výběr v i-tém tématu) vět s největší hodnotou prvku v i-tém řádku matice V^T.

   (b) $i++$

Ozsoy a spol. (2010)

Metoda navržená v [10] nazvaná *cross method* je modifikací metody Steinbergera a Ježka [15]. Úprava spočívá v odstranění vlivu vět, které sice souvisí s tématem, ale nemají v něm hlavní význam. Pro každý řádek matice V^T je spočítán průměr jeho hodnot. V řádcích jsou poté ponechány jen ty hodnoty, které jsou větší než průměr daného řádku. Tento krok odstraní z tématu věty, které s ním souvisí méně. Algoritmus vypadá takto:

1. Sestav matici výskytu M a proveď rozklad $M = U\Sigma V^T$.

2. Proveď redukci na požadovaný počet dimenzí n.

3. Spočítej průměry řádků matice V^T. V matici V^T nastav na nulu všechny prvky, jejichž hodnota je nižší než průměr daného řádku.

4. Vyber požadovaný počet vět s největší délkou získanou jako

$$s_k = \sqrt{\sum_{i=1}^{n} v_{k,i}^2 * \sigma_i^2}, \tag{7.4}$$

7.3 Sumarizace rozdílů v recenzních textech

Problém sumarizace rozdílů v recenzních textech spadá do kategorie kontrastní názorové sumarizace (v angličtině *contrastive opinion summarization*). Cílem sumarizace je analyzovat dokument po stránce sémantické a polaritní a vytvořit dva souhrny: pozitivní a negativní. První by měl zachycovat ty nejdůležitější pozitivní názory, zatímco ten druhý ty nejdůležitější negativní názory. Zároveň by však vytvářené souhrny neměly obsahovat sémanticky podobné části. Kontrastní názorová sumarizace se obvykle provádí extraktivním přístupem. Pojmem sumarizace se tedy v následujích částech myslí extraktivní sumarizace. Ta se dělí do následujících kroků.

7.3.1 Segmentace dokumentu

Stejně jako v klasické extraktivní sumarizaci je dokument rozdělen na menší části. Tyto stavební jednotky jsou odvozeny od gramatických pravidel (věty, odstavce). Drtivá většina extraktivních přístupů využívá jako základní stavební jednotku celé věty.

7.3.2 Hodnocení vět

Jestliže máme dokument rozdělen na jednotlivé věty, potřebujeme větám přiřadit skóre, podle kterého se budeme dále rozhodovat při výběru vět. Požadujeme, aby tvořené souhrny obsahovaly ty nejdůležitější věty, ale zároveň neobsahovaly redundantní informace. Klasické skóre věty se tak skládá ze dvou částí: funkce pro určení významnosti věty a funkce pro určení její podobnosti s ostatními větami souhrnu. V případě tvorby pozitivních a negativních souhrnů lze věty rozdělit na pozitivní a negativní a při výběru brát v potaz věty pouze z dané třídy. Druhým způsobem je hodnotit polaritu pomocí funkce a přidat ji jako třetí část skóre. Možnou formulí pro výpočet skóre S_s věty s je:

$$S_s = \frac{imp(s) * pol(s)}{sim(s)}, \tag{7.5}$$

kde $imp(s)$ je funkce důležitosti věty s, $pol(s)$ je funkce polarity věty s. $sim(s)$ je funkce podobnosti věty s se zbytkem souhrnu, která se určí jako:

$$sim(s) = \max_{s_2 \in sum} (sim(s, s_2)), \tag{7.6}$$

kde sum je množina vět, které se momentálně nachází v souhrnu a $sim(s, s_2)$ je podobnost věty s a s_2 (např. kosínová vzdálenost). Funkce skóre by měla splňovat následující vlastnosti:

1. její absolutní hodnota roste s důležitostí věty
2. její absolutní hodnota klesá s mírou podobnosti s větami jíž zařazenými v souhrnu
3. její znaménko udává polarita věty

7.3.3 Stavba souhrnu

Máme-li vytvořené skóre pro jednotlivé věty, můžeme přistoupit k samotnému výběru vět. Algoritmus výběru je následující:

1. Výběr první věty: vybrána nejdůležitější věta s požadovanou polaritou.
2. Dokud nemá souhrn požadovaný počet vět nebo jsme nevyčerpali všechny věty v dokumentu:
   (a) Vyber větu s nejvyšší hodnotou dle skóre výše $S_s = \frac{imp(s)*pol(s)}{sim(s)}$

Délka souhrnů se liší dle účelu sumarizace a může být i proměnlivá. V takovém případě vybíráme věty, dokud nalezneme takovou, která má hodnotu skóre větší než zvolená hraniční hodnota. U kontrastní názorové sumarizace se obvykle vytvářejí souhrny o délce 10 vět nebo 100 slov. Konktrétně použité metody jsou popsány v kapitole 9 a jejich implementace v kapitole 10.

8 Hodnocení sumarizace

Hodnocení sumarizace je stálou částí výzkumné činnosti. Hlavním přístupem pro hodnocení kvality sumarizace je zhodnocení obsahu proti ideálnímu souhrnu, který je vytvořený lidskými anotátory. Pro extraktivní sumarizaci je často porovnán počet vět, které se vyskytují v souhrnu vytvořeném automatickým sumarizátorem a v souhrnu vytvořeném anotátorem. Je tak určen počet ideálních vět, které automatický souhrn obsahuje. Pokud jsou však lidské souhrny tvořeny abstraktivním přístupem, je nutné porovnávat souhrny na základě jednotlivých slov. Jednou z nejpoužívanějších metod pro takové hodnocení je metrika ROUGE.

8.1 ROUGE

Recall-Oriented Understudy for Gisting Evaluation [7], neboli ROUGE, je měřítkem pro automatické hodnocení kvality souhrnu jeho porovnáním s ideálním (lidským) souhrnem. ROUGE měří počet společných slov, sekvencí slov, párů slov a n-gramů mezi počítačově vygenerovaným souhrnem a ideálním souhrnem vytvořeným člověkem. Metoda obsahuje řadu metrik pro měření podobnosti, nejpoužívanější jsou ROUGE-N, ROUGE-S a ROUGE-SU. [7]

Původní balíček ROUGE byl vytvořen v jazyce *perl*. Od té doby se dočkal svého zpracování i v ostatních programovacích jazycích, např. Java a Python. Je však nutné zmínit, že vysledky poskytnuté různými implementacemi ROUGE se mohou lišit.

8.1.1 ROUGE-N

Metrika ROUGE-N měří *recall* n-gramů mezi hodnoceným souhrnem a množinou referenčních souhrnů, tedy počet kolik procent n-gramů v referenčních souhrnech se vyskytuje v měřeném souhrnu. Skóre vypadá následovně:

$$ROUGE - N = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} count_{match}(gram_n)}{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} count(gram_n)}, \quad (8.1)$$

kde n je délka n-gramu $gram_n$ a $count_match(gram_n)$ je počet společných n-gramů v měřeném souhrnu a kolekci referenčních souhrnů. [7] Jelikož *recall* je závislý na délce měřených souhrnů a s rostoucí délkou měřeného souhrnu se zvyšuje, byla do novějších verzí balíčku ROUGE přidána také *precision*,

určující kolik procent n-gramů v měřeném souhrnu se vyskytuje také v referenčních souhrnech. Dále byla také přidána f-míra:

$$F_\beta = \frac{(1+\beta^2) * (precision * recall)}{\beta^2 * precision + recall}, \tag{8.2}$$

kde β je faktor balancující *precision* a *recall*. Pro $\beta > 1$ má větší váhu *precision*, pro $\beta < 1$ má větší váhu *recall*.

Nejpoužívanějšími verzemi jsou ROUGE-1, která měří počet společných unigramů a ROUGE-2 měřící počet společných bigramů.

8.1.2 ROUGE-S

ROUGE-S měří počet společných skip-bigramů. Skip-bigram je dvojice slov v pořadí, v jakém se vyskytují ve větě při vynechání libovolného počtu slov mezi nimi. Metrika je velice podobná ROUGE-2, jelikož měří počet společných bigramů, které tvoříme přeskočením daného počtu prostředních slov.

8.1.3 ROUGE-SU

Problémem ROUGE-S je fakt, že pokud se v hodnoceném souhrnu nevyskytuje žádný společný skip-bigram, bude výsledné hodnocení nulové. To se však stane i u vět, které obsahují stejná slova, ale v opačném pořadí. Tyto věty bychom chtěli odlišit od vět, která stejná slova nemají. Za tímto účelem byla vytvořena metrika ROUGE-SU, která rozšiřuje ROUGE-S o počítání společných unigramů.

8.1.4 Nastavení

Kromě výběru použitých typů ROUGE je také důležitým aspektem jeho samotné nastavení. ROUGE využívá řadu metod pro úpravu hodnocených souhrnů. Používaná je lemmatizace, stemming, odstranění nevýznamových slov, ale také např. rozpoznávání jmenných entit a synonym. Při použití ROUGE je tak vhodné uvést také přesný způsob, jakým bylo použito, jelikož v opačném případě se výsledky dvou různých ROUGE měření mohou velice lišit.

9 Použité metody

V této kapitole jsou popsány použité metody. Jejich implementace je v kapitole 10.

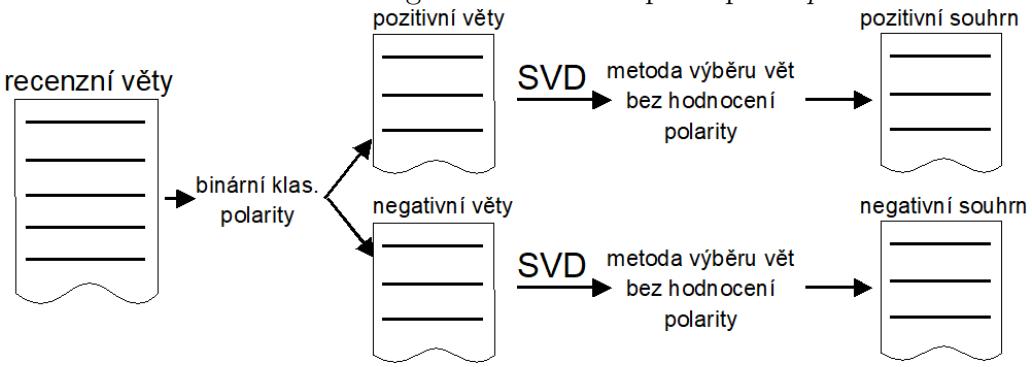
9.1 Návrh sumarizace recenzí pomocí LSA

Ke kontrastní názorové sumarizaci lze přistoupit dvěma způsoby, které se liší použitím polarity.

9.1.1 Přístup *split*

Přístup *split* využívá pro hodnocení polarity binární klasifikátor, který věty rozdělí na pozitivní a negativní. Tyto věty poté putují odděleně do metody SVD, po jejímž provedení jsou z nich vybrány jedním z přístupů popsaných v sekci 7.2.2. Diagram postupu sumarizace je vidět na obr. 9.1.

Obrázek 9.1: Diagram sumarizace přístupem *split*.



Tento přístup trpí jednou závažnou chybou a tou je binární klasifikace polarity. Klasifikace na pouze pozitivní a negativní věty nedovoluje věty rozlišit na např. více pozitivní a méně pozitivní a do souhrnu poté zahrnout věty více pozitivní. Druhým problémem může být nedostatek pozitivních, resp. negativních vět. Recenze skvělého produktu budou obsahovat malé množství negativních vět, do souhrnu bychom tedy chtěli zahrnout i věty slabšího sentimentu, abychom dodrželi délku souhrnu.

9.1.2 Přístup *score*

Přístup *score* využívá pro hodnocení polarity funkci, která nabývá hodnot obecně v rozmezí $(-m, +n)$. Věty nejsou před provedením SVD rozděleny na pozitivní a negativní, nýbrž putují do SVD všechny. Po provedení singulárního rozkladu jsou následně vybrány věty dle kompozitního skóre

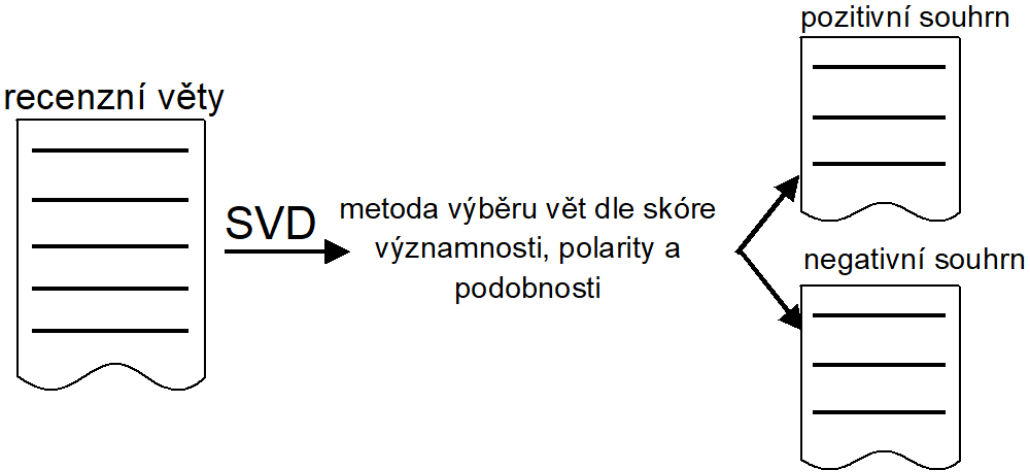
$$S_s = pol(s) * (1 + imp(s)) * (1 - sim(s))^p \tag{9.1}$$

pro pozitivní věty a

$$S_s = -1 * pol(s) * (1 + imp(s)) * (1 - sim(s))^p \tag{9.2}$$

pro negativní věty, kde $pol(s)$ je funkce polarity, $imp(s)$ je funkce důležitosti, p je vážící faktor podobnosti a $sim(s)$ je funkce podobnosti věty se zbytkem souhrnu. Diagram postupu sumarizace je vidět na obr. 9.2. Výsledky funkce polarity jsou normalizovány na rozsah $< -1, +1 >$ a výsledky funkce důležitosti jsou normalizovány na rozsah $< 0, +1 >$.

Obrázek 9.2: Diagram sumarizace přístupem *score*.



9.2 Předzpracování

Předzpracování se skládá z kroků, které jsou popsány v kapitole 3. Jedná se o kroky v následujícím pořadí:

1. Rozdělení textu na věty pomocí regulárního výrazu.
2. Rozdělení vět na slova pomocí regulárního výrazu.

3. Odstranění diakritiky a převod na malá písmena.
4. Lemmatizace pomocí slovníku lemmat.
5. Odstranění stop slov.

9.3 Stavba matice souvýskytu

Implementoval jsem dvě metody pro stavbu matice souvýskytu M:

- *binary* - stavba matice dle způsobu binárního výskytu, viz. 7.2.1.
- *tfidf* - stavba matice s tf-idf vážením, viz. 7.2.1.

Dokumenty jsou u tfidf vážení jednotlivé věty. Hodnota *document frequency* je tedy počet vět, které obsahují dané slovo. Důležité je toto vážení natrénovat na recenzích všech restaurací. Pokud bychom *tfidf* vážení trénovali pro každou restauraci zvlášť, došlo by k potlačení efektu významnosti některých slov. Např. u pizzérie bude slovo pizza časté a bude proto mít nízkou hodnotu *idf*, avšak při sumarizaci je toto slovo velice důležité. Proto je vhodnější vážení trénovat na recenzích všech restaurací, tímto způsobem bude mít slovo *pizza* větší význam, jelikož se bude vyskytovat především u pizzérií.

9.4 Hodnocení polarity

V této sekci jsou popsány metody hodnocení polarity vět.

9.4.1 lex

Hodnocení polarity je provedeno slovníkovou metodou a skórem *lex* (viz. sekce 5.1). Slovník polaritních termů tvoří dva seznamy pozitivních a negativních slov. Pokud se hledané slovo vyskytuje v seznamu pozitivních termů, je mu přidána hodnota 1, pokud se vyskytuje v seznamu negativních termů, je mu dána hodnota -1. Polaritní hodnocení věty je součtem polaritních vah jednotlivých slov.

9.4.2 maxEnt

Během analýzy se ukázalo, že metoda *lex* velkou mírou preferuje dlouhé věty. V delších větách se dle očekávání vyskytuje více pozitivních, resp. negativních slov. Proto jsem se rozhodl použít pro ohodnocení polarity jiný způsob, který by nezávisel na délce věty. Vytvořil jsem binární klasifikátor, který věty

klasifikuje na pozitivní a negativní. Ke klasifikaci jsem použil klasifikátor s maximální entropií (sekce 5.3.2), který jsem natrénoval na seznamech pozitivních a negativních recenzních vět restaurací. Jako příznaky jsem použil unigramy. Tuto metodu jsem nazval *maxEnt*.

9.5 Hodnocení významnosti

9.5.1 Existující metody

Významnost vět je hodnocena výhradně využitím výsledků singulárního rozkladu, konkrétně pomocí součinu matic ΣV^T. Použité metody jsou popsány v sekci 7.2.2. Implementovány jsou tyto existující metody

- *topic* - metoda vychází z přístupu Gong a Liu (2001), pokud je po redukci dimenze ponechán menší počet témat, než je počet vybíraných vět, pokračuje výběr znovu od prvního tématu
- *length* - založeno na přístupu Steinberger a Ježek (2004). Metoda je však rozšířena o možnost vynulování negativních hodnot z matice ΣV^T.
- *subtractlength* - velice podobné jako metoda *length*, ale po výběru věty je její sloupec odečten od všech sloupců matice ΣV^T. To zabraňuje redundanci, jelikož podobné věty budou mít podobné sloupce, které budou po odečtení téměř vynulované a budou tak mít velice malou délku. Také umožňuje vynulování negativních hodnot.

9.5.2 InvertLength

Při analýze výsledků výše zmíněných metod získaných metrikou ROUGE (kapitola 11) a také analýzou vytvořených souhrnů jsem zjistil, že všechny tyto metody preferují delší věty. Pisatelé recenzí však často napíšou celou svou recenzi do jedné věty. Takové věty poté tvoří velkou část souhrnu, který je svou délkou omezen na 100 slov. Z tohoto důvodu jsem se rozhodl vytvořit nový způsob skórování vět, který by věty penalizoval na základě jejich délky. Algoritmus této metody je následující:

1. Sestav matici výskytu M a proveď rozklad $M = U\Sigma V^T$.
2. Proveď redukci na požadovaný počet dimenzí n.
3. Spočítej délku d_i vektoru věty i v matici ΣV^T.

4. Pokud je $d_i < 1$ vrať d_i, v opačném případě vrať $\frac{1}{d_i}$

Tento přístup se může na první pohled zdát kontraproduktivní, jelikož věty penalizujeme hodnotou, kterou jsme u předchozích metod považovali za ohodnocení důležitosti. Při testování však tento způsob hodnocení poskytl zdaleka nejlepší výsledky.

9.6 Hodnocení podobnosti

Hodnocení podobnosti je založeno na počítání kosínové vzdálenosti sloupců matice M_k. Jedná se o matici souvýskytu v redukovaném prostoru po provedení singulárního rozkladu. Jednotlivé sloupce této matice reprezentují věty v prostoru termů. Podobnost věty se zbytkem souhrnu je hodnocena dvěma způsoby.

9.6.1 Varianta č. 1

Tato varianta, nazvaná sim_{sent} počítá podobnost zkoumané věty se souhrnem určením maximální podobnosti s větami, které se aktuálně nacházejí v souhrnu. Vzorec, již dříve uvedený pod označením 7.6, vypadá následovně:

$$sim_{sent}(s) = \max_{s_2 \in sum}(sim(s, s_2)), \tag{9.3}$$

kde sum je množina vět, které se momentálně nachází v souhrnu a $sim(s, s_2)$ je kosínová vzdálenost mezi sloupci vět s a s_2 v matici M_k.

9.6.2 Varianta č. 2

Druhá varianta s názvem sim_{sum} zkoumá podobnost také pomocí kosínové vzdálenosti, ale namísto hledání maximální podobnosti s jednotlivými větami počítá celkovou podobnost s celým souhrnem. Souhrn je reprezentován průměrem vět, které se v něm vyskytují. Následně je kosínovou vzdáleností spočítána podobnost se zkoumanou větou.

Tento způsob by měl dosahovat lepších výsledků, než způsob sim_{sent}, jelikož podobnost věty a souhrnu není výsledkem porovnání s jednou jedinou větou (tou nejpodobnější), ale vypovídá o podobnosti s celým souhrnem.

10 Implementace

Sumarizátor jsem od počátku zamýšlel a vytvářel jako znovupoužitelnou knihovnu, která bude sloužit pro testování automatické sumarizace. Z tohoto důvodu jsem se rozhodl ji vytvořit v jazyce Java. K výběru tohoto jazyka také vedl fakt, že celá řada *state-of-art* metod zpracování přirozeného jazyka je implementována právě v Javě. Jádrem knihovny je kontrastní sumarizace a metoda LSA, avšak pouhým rozšířením by mohla knihovna sloužit k testování dalších metod a typů sumarizace.

10.1 Architektura knihovny

Nejdůležitější částí knihovny je kontrastní sumarizace metodou LSA, resp. přístupem *score* (sekce 9.1.2). UML diagram tříd, které jsou součástí tohoto jádra je na obr. 10.1.

Jednotlivé třídy, které lze vidět v tomto diagramu, jsou popsány dále. Hlavním balíčkem knihovny je *summarization*. Veškeré ostatní balíčky jsou jeho podbalíčkem a jsou popsány v dalších sekcích.

10.1.1 data

Balíček *data* obsahuje třídy a metody pro čtení a zápis textových dat a třídy pro interní reprezentaci načtených dat.

io.InOutSettings

Tato třída je přepravkou pro uchování cesty ke složce se vstupními soubory a uchování cesty ke složce, kam budou po dokončení sumarizace zapsány její výsledky.

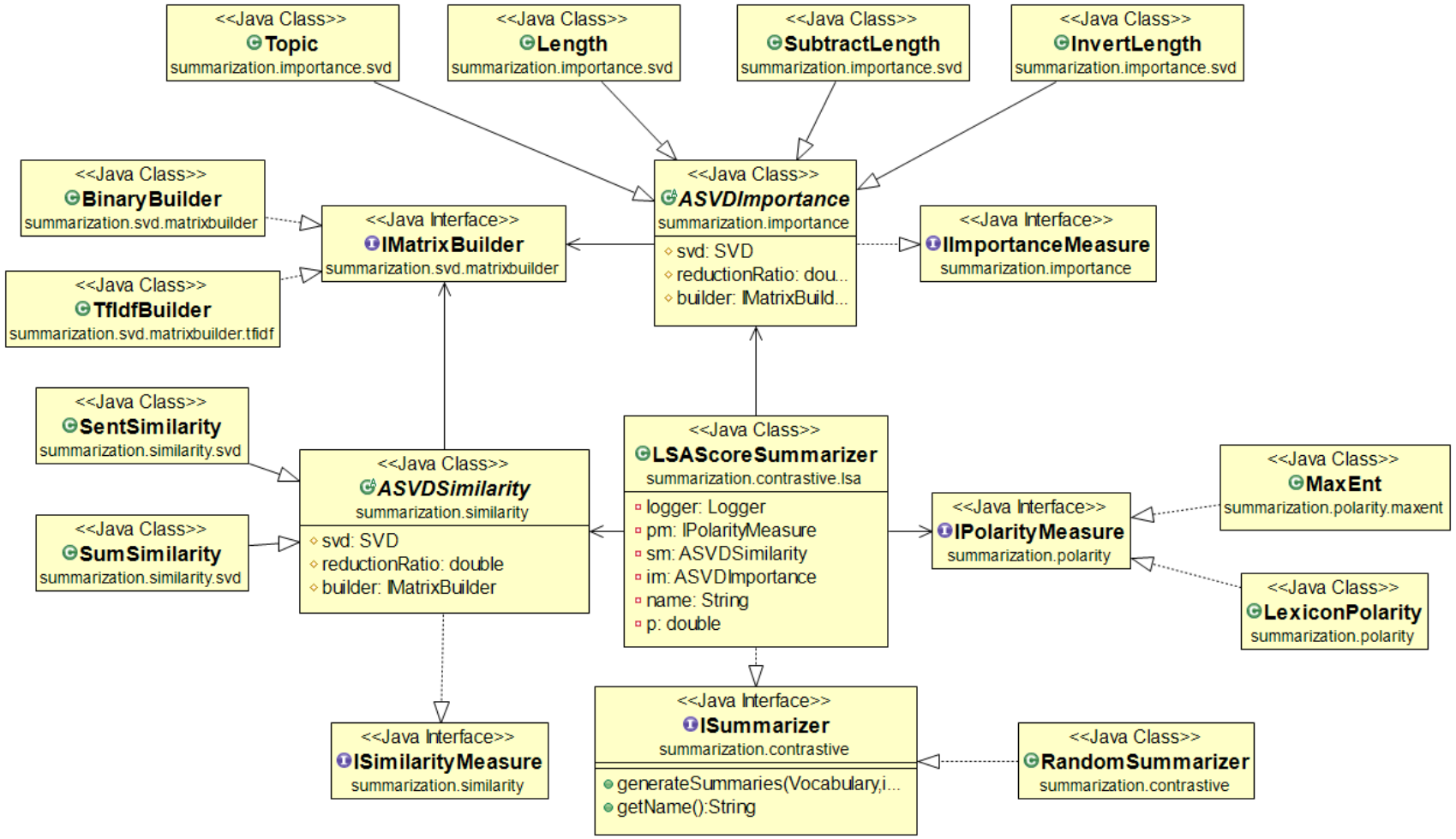
io.DataProvider

Rozhraní pro čtení dat. Obsahuje hlavičky metod pro čtení souboru po řádcích, případně pro přečtení celého souboru a pro uzavření souboru.

io.BasicDataProvider

Implementuje rozhraní DataProvider. Pro čtení souboru využívá třídu BufferedReader. Umožňuje čtení souboru po řádkách a načtení celého souboru

Obrázek 10.1: UML diagram jádra sumarizace *score*.



najednou.

io.DataWriter

Tato třída slouží k zápisu výsledků sumarizace. Obsahuje metody pro vytvoření složky, zápis souhrnů do souboru a také zápis normalizovaného souhrnu do souboru.

Sentence

Třída Sentence je jednou z nejdůležitějších tříd celé knihovny. Jelikož při sumarizaci používáme jako hlavní stavební jednotku větu, je sumarizovaný text rozdělen na věty, které jsou reprezentovány instancemi této třídy. Instance třídy Sentence obsahují následující instanční atributy:

- *int id* - identifikační číslo věty, věty jsou číslovány v pořadí, v jakém jsou načteny ze souboru
- *String sentence* - celá věta tak, jak se vyskytuje v textu
- *Map<String, Integer> words* - mapa obsahující normalizovaná slova (bez stop slov) a jejich počet, která se ve větě vyskytují
- *String[] wordsArray* - celá věta v původním znění rozdělená na slova

Třída dále obsahuje statickou metodu *readSentences*, pomocí které lze celý soubor rozdělit na věty a vrátit seznam instancí Sentence.

Vocabulary

Instance třídy Vocabulary slouží k interní reprezentaci načtených dat. Obsahuje veškeré potřebné instanční atributy pro reprezentaci načtených dat a pro práci s nimi. Mezi tyto atributy patří:

- *List<String> words* - seznam objevených slov
- *Map<String, Integer> wordsToKeyMap* - mapuje slova na jejich pozici v seznamu slov *List<String> words*
- *List<Sentence> sentences* - seznam načtených vět

Třída obsahuje metody pro přečtení celého souboru, jeho rozdělení na věty a následné naplnění instance *Vocabulary*.

10.1.2 preprocessing

Tento balíček obsahuje třídy a metody pro předzpracování.

Normalizer

Rozhraní pro normalizaci slov. Obsahuje hlavičku jediné metody *String norm(String input)*, která má vrátit normalizovanou podobu slova.

Lemmatizer

Implementuje rohraní Normalizer a slouží k lemmatizaci. Ke svému fungování potřebuje slovník lemmat. Pro rychlé vyhledávání lemma při lemmatizaci jsou načtená lemmata uložena v instanci *HashMap<String, String>*.

HPSWrapper

Také implementuje rozhraní Normalizer a slouží ke stematizaci. Ke stematizaci volá metody knihovny HPS (High Precision Stemmer), která byla vytvořena na Západočeské univerzitě v Plzni.

Preprocessing

Hlavní třída předzpracování. Její konstruktor má jako parametr instanci Normalizer pro normalizaci a boolean hodnotu, zda chceme odstranit stop slova. Pro normalizaci slouží metoda *String normalize(String word)*, která slovo nejdříve převede na malá písmena, odstraní diakritiku, zkontroluje, zda se jedná o stop slovo. Pokud se jedná o stop slovo a u instance Preprocessing je nastaveno odstranění stop slov, je slovo zahozeno. V opačném případě je slovo normalizováno pomocí instance Normalizer a vrácen jeho nový tvar. Seznam stop slov je čten ze souboru.

BasicTokenizer

Tato třída slouží k tokenizaci textu. Umožňuje tokenizaci textu na věty a vět na slova. K tomu využívá regulární výrazy.

10.1.3 svd

Balíček *svd* obsahuje třídy pro provedení singulárního rozkladu a pro stavbu matice souvýskytu.

matrixbuilder.IMatrixBuilder

Rozhraní pro třídy implementující stavbu matice souvýskytu. Obsahuje hlavičku metody *double[][] buildOccurrenceMatrix(Vocabulary voc)*, která slouží k vytvoření matice souvýskytu, ve které řádky reprezentují slova a sloupce věty. K vytvoření této matice je nutná instance třídy Vocabulary.

matrixbuilder.BinaryBuilder

Implementuje rozhraní IMatrixBuilder. Slouží ke stavbě matice metodou *binary* (sekce 7.2.1).

matrixbuilder.tfidf.TfIdf

Instance třídy TfIdf umožňují použití *tfidf* vážení termů. Konstruktor instancí této třídy má jako jediný parametr instanci *Vocabulary*, která by měla obsahovat věty recenzí všech restaurací.

matrixbuilder.tfidf.TfIdfBuilder

Implementuje rozhraní IMatrixBuilder. Slouží ke stavbě matice metodou *tfidf* (sekce 7.2.1). Konstruktor potřebuje instanci třídy TfIdf.

SVD

Třída SVD slouží k provedení singulárního rozkladu. Pro provedení rozkladu využívá třídy SingularValueDecomposition knihovny apache.commons.math3. Obsahuje následující instanční atributy:

- *double[][] M, U, S, V* - matice souvýskytu M, a matice U, $Sigma$ a V^T po provedení rozkladu
- *double[][] Mk, Uk, Sk, Vk* - redukované verze matic M, U, Σ, V^T
- *double[][] SV, SVk* - neredukovaná verze a redukovaná verze součinu ΣV^T

Konstruktor instance SVD má jediný parametr *double[][] matrix*, který představuje matici souvýskytu. Při vytváření rozkladu je proveden singulární rozklad zavoláním metody *doSVD()*. Tato metoda provede singulární rozklad a naplní všechny výše zmíněné matice. Matice V a Vk jsou přenásobeny hodnotou -1, jelikož knihovna apache.commons.math3 vrací po rozkladu tyto dvě matice s opačnými znaménky. Pro redukci slouží metoda *reduce(double reductionRatio)*, kde *reductionRatio* je procento ponechaných dimenzí. Tato

matice provede redukci dimenze a nastaví výše zmíněné redukované matice *Mk, Uk, Sk, Vk, SVk*.

Třída navíc obsahuje statické metody pro práci s maticemi, které jsou využívány v metodách výběru vět. Jedná se o metody pro získání délky vektoru, získání sloupce matice a odečtení sloupce matice.

10.1.4 polarity

Tento balíček obsahuje třídy pro určení polarity věty.

IPolarityMeasure

Toto rozhraní implementují třídy, které poskytují metody pro hodnocení polarity. Konkrétně se jedná o hlavičky metod *double getPolarity(Sentence s)* a *public double[] getPolarities(List<Sentence> sentences)*.

LexiconPolarity

Třída LexiconPolarity poskytuje metodu *lex* (sekce 9.4) pro hodnocení polarity. Umožňuje načtení polaritních slov metodou *readWords(String file, HashMap<String, Integer> map, int val)*, kde *map* je mapa obsahující načtená polaritní slova a *val* je ohodnocení čtených polaritních slov. Načtená polaritní slova jsou držena v instančním atributu *HashMap<String, Integer> words*. Konstruktor umožňuje nastavení instance třídy Preprocessing pro normalizaci čtených polaritních slov.

maxent.NGramFeature

Tato třída umožňuje extrakci n-gramových příznaků z dat.

maxent.MaxEnt

Implementace klasifikátoru maximální entropie. Implementuje rozhraní IPolarityMeasure. Pro klasifikaci využívá knihovnu Brainy [5]. Konstruktor má jako parametr instanci třídy Preprocessing, pro předzpracování dat, na kterých se trénuje klasifikátor.

10.1.5 similarity

Třídy pro hodnocení podobnosti se nacházejí v balíčku *similarity*.

ISimilarityMeasure

Toto rozhraní implementují třídy poskytující metody hodnocení podobnosti vět. Obsahuje následující hlavičky:

- *double getSimilarity(int id1, int id2)* - podobnost mezi dvěma větami
- *double getSimilarity(int id, List<Integer> ids)* - podobnost věty s množinou vět
- *double[][] getSimilarities()* - matice podobnosti všech vět

ASVDSimilarity

Abstraktní třída ASVDSimilarity je základem pro třídy, které poskytují hodnocení podobnosti pomocí metody singulárního rozkladu. Implementuje rozhraní ISimilarityMeasure. Třída má následující instanční atributy:

- *SVD svd* - instance třídy SVD pro provedení singulárního rozkladu
- *double reductionRatio* - redukční faktor singulárního rozkladu
- *IMatrixBuilder builder* - instance pro stavbu matice souvýskytu

Konstruktor této třídy má dva parametry, *double reductionRatio* a *IMatrixBuilder builder*. Pro provedení singulárního rozkladu slouží metoda *train(Vocabulary voc)*. Ta nejprve sestaví matici souvýskytu pomocí instance IMatrixBuilder a následně nad touto maticí provede singulární rozklad. Pokud je také nastaven redukční parametr na hodnotu menší než 1, bude provedena redukce.

svd.SentSimilarity

Implementace metody sim_{sent}. Dědí od ASVDSimilarityMeasure.

svd.SumSimilarity

Implementace metody sim_{sum}. Dědí od ASVDSimilarityMeasure.

10.1.6 importance

Tento balíček obsahuje třídy a metody pro hodnocení důležitosti.

IImportanceMeasure

Toto rozhraní implementují třídy poskytující metody hodnocení důležitosti vět. Obsahuje následující hlavičky:

- *double getImportance(int id1)* - vrací důležitost věty
- *double[] getImportances()* - vrací důležitost všech vět

ASVDImportance

Abstraktní třída ASVDImportance je základem pro třídy, které poskytují hodnocení důležitostí pomocí metody singulárního rozkladu. Implementuje rozhraní IImportanceMeasure. Třída má následující instanční atributy:

- *SVD svd* - instance třídy SVD pro provedení singulárního rozkladu
- *double reductionRatio* - redukční faktor singulárního rozkladu
- *IMatrixBuilder builder* - instance pro stavbu matice souvýskytu

Konstruktor této třídy má dva parametry, *double reductionRatio* a *IMatrixBuilder builder*. Pro provedení singulárního rozkladu slouží metoda *train(Vocabulary voc)*, která funguje stejně jako metoda *train(...)* třídy ASVDSimilarity. Dále třída obsahuje následující abstraktní metodu:

- *boolean select(int id)* - slouží ke komunikaci sumarizátoru s metodou podobnosti. Metodě podobnosti říká, že byla do souhrnu vybrána věta *id*. Metoda tak může na tento výběr libovolně reagovat upravením svého fungování a zpětně informuje sumarizátor, zda je třeba přepočítat hodnocení důležitosti návratovou hodnotou boolean.

svd.Topic

Třída Topic dědí od abstraktní třídy ASVDImportance a obsahuje implementaci metody *topic*. Kromě zděděných atributů třídy ASVDImportance třída obsahuje instanční atribut *int topicId*, který slouží k aktuálnímu označení tématu, ve kterém je hledaná důležitost věty. Při výběru věty metodou *select(...)* dojde k inkrementaci *topicId*. Pokud je hodnota *topicId* větší, než je počet témat, dojde k nastavení jeho hodnoty na 0. Výběr následující věty tak začíná znovu od prvního nejdůležitějšího tématu.

svd.Length

Také dědí od třídy ASVDImportance a implementuje metodu *length*. Kromě atributů zděděných ze třídy ASVDImportance obsahuje atribut *boolean nonNegative*, který říká, zda do délky vektoru započítat i negativní hodnoty. V případě, že je tento atribut *true*, jsou negativní hodnoty při výpočtu délky přeskočeny. Zavolání metody *select(...)* neprovede žádnou změnu a metoda proto vrací hodnotu *false*.

svd.SubtractLength

SubtractLength opět dědí od třídy ASVDImportance a implementuje metodu *subtractLength*. Kromě atributů zděděných ze třídy ASVDImportance také obsahuje atribut *boolean nonNegative*, který říká zda při výpočtu délky vektoru a odečítání vektoru od zbytku matice započítat i negativní hodnoty. V případě, že je tento atribut *true*, jsou negativní hodnoty při výpočtu délky a odečtu přeskočeny. Zavolání metody *select(int id)* provede odečtení vektoru věty *id* od zbytku matice SV^T. Z tohoto důvodu vrací metoda *select(int id)* hodnotu *true*, která informuje sumarizátor o nutném přepočtu hodnot důležitosti.

svd.InvertLength

Třída InvertLength opět dědí od třídy ASVDImportance a implementuje metodu *invertLength*. Zavolání metody *select(...)* neprovede žádnou změnu a metoda proto vrací hodnotu *false*.

10.1.7 contrastive

V balíčku *contrastive* se nacházejí třídy pro kontrastní sumarizaci.

ISummarizer

Rozhraní ISummarizer implementují kontrastní sumarizátory. Rozhraní má následující hlavičky:

- *List<List<Sentence» generateSummaries(Vocabulary voc, int count)* - metoda by měla vrátit instanci *List* s pozitovním a negativním souhrnem, každý s počtem slov daným parametrem *count*.
- *String getName()* - sumarizátory lze pojmenovat, aby je bylo možné rozlišit při testo

RandomSummarizer

Třída RandomSummarizer obsahuje náhodný sumarizátor, který může posloužit při testování navržených metod. Během sumarizace si sumarizátor udržuje pole *int[] used* o velikosti počtu vět k sumarizaci. Toto pole slouží k poznamenání vět, které již byly do souhrnu vybrány, aby nemohly být vybrány vícekrát.

lsa.LSAScoreSummarizer

Třída LSAScoreSummarizer implementuje rozhraní ISummarizer a slouží ke kontrastní sumarizaci metodou *score* s využitím LSA podle skóre

$$s_s = pol(s) * (1 + imp(s)) * (1 - sim(s))^p \tag{10.1}$$

pro pozitivní věty a

$$s_s = -1 * pol(s) * (1 + imp(s)) * (1 - sim(s))^p \tag{10.2}$$

pro negativní věty, kde $pol(s)$ je funkce polarity, $imp(s)$ je funkce důležitosti, p je vážící faktor podobnosti a $sim(s)$ je funkce podobnosti věty se zbytkem souhrnu. Instance třídy LSAScoreSummarizer využívají k výpočtu tohoto skóre následující tři instanční atributy:

- *IPolarityMeasure pm* - hodnocení polarity *pol*
- *ASVDImportance im* - hodnocení důležitosti *imp*
- *ASVDSimilarity sm* - hodnocení podobnosti *sim* věty se souhrnem

Zjednodušený kód metody *generateSummary(List<Sentence> sentences, int count, boolean positive)* pro stavbu souhrnu je následující:

```
1  List<Sentence> summary = new ArrayList<Sentence>(count);
2  double[] polarities = Utils.normalize(pm.getPolarities());
3  double[] importances = Utils.normalize(im.getImportances());
4  int words = 0;
5  int used[] = new int[sentences.size()];
6  int polarityChange = (positive) ? 1 : -1;
7  while(words<count){
8              int best = -1;
9              double bestScore = 0;
10             for(int i = 0; i < sentences.size(); i++){
11                 if(used[i]==1)continue;
12                 double maxSimilarity = 0;
13                 if(sm!=null) maxSimilarity = sm.getSimilarity(i,
                       getSummaryIds(summary));
```

```
14                  double score = (polarityChange*polarity[i]*(1+
                        importances[i]))*Math.pow(1-maxSimilarity,p);
15                  if(score>bestScore){
16                      bestScore = score;
17                      best = i;
18                  }
19              }
20              if(best==-1)break; // no other sentences to choose
21              summary.add(sentences.get(best));
22              boolean changed = im.select(best);
23              if (changed) importances = Utils.normalize(im.
                    getImportances()); // if changed, recalculate
                    importance
24              words += sentences.get(best).words;
25              used[best] = 1;
26          }
27          return summary;
```

Sumarizátor nejrve získá hodnoty důležitosti a polarity, poté iteračně vybírá nejlepší věty do souhrnu. Po každém výběru notifikuje metodu důležitosti o výběru věty, která zpětně informuje sumarizátor o případně nutném přepočítání důležitosti.

10.1.8 summarization

Toto je hlavní balíček celé knihovny. Obsahuje všechny výše zmíněné balíčky.

SimpleRunner

Spouštěcí třída knihovny pro ověření funkčnosti sumarizace. K nastavení sumarizace je použit framework Spring a konfigurace beanů pomocí XML konfiguračních souborů (viz. sekce 10.2). Běh sumarizace je následující:

1. Získej instanci InOutSettings čtením beanu *data*.
2. Vytvoř složky pro výstup definované v instanci InOutSettings.
3. Získej instanci Preprocessing *pr* čtením beanu *preprocessing* pro předzpracování souborů před sumarizací.
4. Získej instanci Preprocessing *pr_rouge* čtením beanu *rouge_preprocessing* pro normalizaci vytvořených souhrnů pro hodnocení metrikou ROUGE.
5. Získej seznam instancí ISummarizer *pr* čtením beanu *summarizers*, kterými bude provedena sumarizace.

6. Pro každý soubor *f* obsahující recenze jedné restaurace ze vstupní složky definované v instanci InOutSettings:
   (a) Vytvoř nad daty *f* souboru instanci Vocabulary.
   (b) Pro každý sumarizátor *s* v seznamu sumarizátorů:
      i. Vygeneruj pozitivní a negativní souhrn souboru *f* sumarizátorem *s*.
      ii. Zapiš do výstupní složky nenormalizovanou verzi obou souhrnů.
      iii. Zapiš do výstupní složky normalizovanou verzi obou souhrnů pomocí instance *pr_rouge*.
7. Připrav referenční souhrny pomocí instance *pr_rouge*.

10.2 Konfigurace

Knihovna využívá pro svou konfiguraci XML konfigurační soubory Spring Bean. Pro spuštění je nutné definovat následující beany:

- *data* - instance InOutSettings určující vstupní a výstupní složky.
- *preprocessing* - instance Preprocessing určující typ předzpracování recenzí před sumarizací.
- *rouge_preprocessing* - instance Preprocessing určující typ předzpracování výsledných systémových souhrnů a referenčních souhrnů pro vyhodnocení metrikou ROUGE.
- *summarizers* - instance List<ISummarizer>, definující jednotlivé systémové sumarizátory.

V příloze 1 je vidět ukázka takového konfiguračního souboru.

10.3 ROUGE metrika

Původní balíček ROUGE byl vytvořen v jazyce perl. Perl však není tolik dostupný jako jazyk java a ani použití původního balíčku není úplně jednoduché. Pro hodnocení systémových souhrnů jsem proto použil implementaci ROUGE v javě. Konkrétně se jedná o knihovnu RxNLP Rouge 2.0 [2]. Tuto knihovnu jsem zvolil z toho důvodu, že autoři ověřili výsledky jejich implementace oproti výsledkům původní perl implementace.

10.4 Spuštění sumarizace a jejího ověření

Sumarizátor potřebuje pro svůj běh sumarizované recenze. Složku, ve které se tyto recenze nacházejí, je možné specifikovat v konfiguračním XML souboru (viz. sekce 10.2) jako parametr *inputFolder* beanu *data*. Tato složka by měla obsahovat jednotlivé recenzní věty restaurací, jeden soubor pro každou restauraci. Tyto soubory jsou předpřipravené ve složce `data/reviews`. Nastavení sumarizátoru proběhne také pomocí zmíněných konfiguračních souborů. Je možné použít více sumarizátorů najednou. V takovém případě bude souhrn vygenerován každým definovaným sumarizátorem. Názvy souborů jsou vytvářeny dle následujícího formátu:

summary<*číslo-souboru*>_<*jméno-sumarizátoru*>.txt

Sumarizaci je možné spustit zadáním příkazu v příkazové řádce

`java -jar dplsa.jar <konfigurační soubor>`,

kde `<konfigurační soubor>` je cesta ke zvolenému konfiguračnímu XML souboru. Předpřipravené konfigurační soubory jsou ve složce `config`. Po spuštění a dokončení sumarizace bude složka *outputFolder* definovaná v beanu *data* obsahovat vytvořené a předzpracované systémové souhrny spolu s referenčními souhrny ve složce *normed*. Nenormalizované systémové souhrny budou ve složce *outputFolder/raw*.

Nad souhrny ve složce *outputFolder/normed* je poté možné provést evaluaci metrikou ROUGE. Knihovnu ROUGE RxNLP je možné nastavit konfiguračním souborem `rouge.properties`. Důležitou částí tohoto souboru je řádka

`project.dir=`*`outputFolder`*`/normed`

pro definici evaluované složky. Spustit evaluaci je možné zavoláním

`java -jar rouge2-1.2.jar`,

Po dokončení budou výsledky zapsány do souboru, který je definovaný poslední řádkou

`outputFile=`*`jméno souboru s výsledky`*

Jedná se o CSV soubor, ve kterém jsou hodnoty recall, precision a f-measure pro každý systémový souhrn a každou použitou metriku. Pro získání cel-

kových průměrných výsledků slouží prográmek *averager.jar*, který je možné spustit z příkazového řádku zavoláním

`java -jar averager.jar <soubor s výsledky>`,

po dokončení jsou průměrné výsledky vypsány do konzole. Ukázky souboru s recenzními větami a výsledného souhrnu jsou k vidění v přílohách 2 a 3. Výpis prográmku *averager.jar* je k vidění v příloze 4. Řádky odpovídají jednotlivým systémům. Pro každý systém je vypsána hodnota recall metrik ROUGE-1, ROUGE-2 a ROUGE-SU4 v tomto pořadí.

11 Testování a výsledky

Funkčnost navržených metod jsem otestoval na datech popsaných v kapitole 2 metodou ROUGE, konktrétně metrikami ROUGE-1, ROUGE-2 a ROUGE-SU4. Systémové i referenční souhrny jsem před měřením předzpracoval jednoduchou lemmatizací. Žádné jiné úpravy nebyly použity. Pro všechny tři metriky jsem získal hodnoty recall, precision a f-míru. V následujícím popisu dosažených výsledků je záměrně uváděn pouze recall, jelikož ten je pro ROUGE metriky tradičně uváděn a také z toho důvodu, že délka souhrnů byla omezena na 100 slov, což vedlo k podobnosti výsledků recall, precision a f-míry.

Před samotným popisem testování je třeba ještě uvést některé vlastnosti dat, použitých k testování. Délka sumarizovaných recenzí byla 1000 slov a délka tvořených souhrnů byla 100 slov. Sumarizací tak došlo k redukci 90% informace. Počet odhalených témat metodou SVD byl roven počtu vět sumarizovaných recenzí. Průměrný počet vět recenzí jedné restaurace (a tedy i témat a hodnot matice Σ) byl 75. Při redukci dimenze na 10% ($r = 0{,}1$) tak bylo ponecháno v průměru 7-8 témat (hodnot matice Σ).

Testoval jsem možné kombinace navržených metod (viz. sekce 9) hodnocení polarity, podobnosti a důležitosti dle skóre v sekci 9.1.2. Tam, kde to mělo smysl, jsem testoval závislost výsledků na velikosti redukčního faktoru. Ten jsem nastavoval na hodnotu $r \in \{1; 0{,}9; 0{,}8; 0{,}7; 0{,}6; 0{,}5; 0{,}4; 0{,}3; 0{,}2; 0{,}1\}$ (hodnota r určuje podíl ponechaných dimenzí). Jako výchozí bod (*baseline*) pro porovnávání navržených metod jsem použil metodu výběru využívající jen polaritní hodnocení, dle metody *lex* (viz. sekce 9.4). Dále jsou výsledky porovnávány s nejlepším výsledkem dosaženým v [4] (*Ježek + Campr + Nykl*) a metodou náhodného výběru (*random*).

Tabulka 11.1: Výsledky srovnávacích metod.

metoda	ROUGE-1	ROUGE-2	ROUGE-SU4
random	0,3243	0,1045	0,1678
lex	0,3734	0,1692	0,2321
Ježek + Campr + Nykl	0,4059	0,2273	0,2128

Tyto výsledky budou dále používány při analýze ostatních metod. Metody sumarizace jsou označovány jako kombinace jednotlivých metod stavby matice, hodnocení polarity, důležitosti a podobnosti v tomto pořadí násle-

dovně:

builder + polarity + importance + similarity,

kde jednotlivé části mohou být:

- *builder* - *binary*, *tfidf*, dle sekce 9.3
- *polarity* - *lex*, *maxEnt*, dle sekce 9.4
- *importance* - *topic*, *length*, *subtractLength*, *invertLength*, dle sekce 9.5
- *similarity* - sim_{sent}, sim_{sum}, dle sekce 9.6

11.1 Podobnost

Nejprve jsem otestoval kombinaci polarity a podobnosti, pro určení nejlepšího nastavení metody podobnosti, která bude použita při testování metod důležitosti. Pro hodnocení podobnosti dvou vět, příp. podobnosti věty a souhrnu jsou použity funkce sim_{sent} a sim_{sum} (sekce 9.6). Nastavení podobnosti, které dávalo nejlepší výsledky, jsem poté použil při testování ostatních metod. Při analýze funkčnosti podobnosti jsem se soustředil na podíl redukce dimenze u metody SVD, na způsob stavby matice M a na vážící faktor p (viz. vzorec 9.1).

Nejprve jsem začal se zkoumáním závislosti výsledků na velikosti faktoru p, který jsem nastavil na hodnoty $\{0{,}5; 1; 3; 5\}$. Se zvyšující se hodnotu p se výsledky mírně horšily. Zvyšující se faktor p více penalizuje podobné věty, u kontrastní názorové sumarizace může však mírná podobnost být žádoucí, jelikož silná témata bychom chtěli mít zastoupená více větami. Např. pro pizzerii bychom v souhrnu chtěli mít 2-4 věty o kvalitě pizzy. Se snižujícím se p výsledky sumarizace mírně stoupaly. Nejlepších výsledků bylo dosaženo pro $p = 0{,}5$.

Následně jsem provedl analýzu obou metod sim_{sent} a sim_{sum} pro stavbu matice souvýskytu metodami *binary* a *tfidf* a velikost redukčního faktoru r. Výsledky metody sim_{sent} v kombinaci s metodou polarity *lex* jsou vidět v tabulce 11.2. Při stavbě matice M pomocí metody *tfidf* byla dosaženo obecně lepších výsledků než metodou *binary*.

Tabulka 11.2: Výsledky metody $lex + sim_{sent}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3706	**0,1674**	**0,2296**	**0,3738**	**0,1697**	**0,2324**
0,9	0,3621	0,1603	0,2222	0,3712	0,1682	0,2305
0,8	0,3668	0,1630	0,2259	0,3715	0,1668	0,2301
0,7	0,3638	0,1583	0,2218	0,3699	0,1661	0,2286
0,6	0,3682	0,1644	0,2269	0,3657	0,1625	0,2252
0,5	0,3670	0,1644	0,2267	0,3696	0,1656	0,2288
0,4	0,3656	0,1627	0,2255	0,3701	0,1671	0,2295
0,3	0,3669	0,1633	0,2260	0,3693	0,1663	0,2292
0,2	0,3634	0,1605	0,2228	0,3692	0,1648	0,2281
0,1	**0,3720**	0,1659	0,2292	0,3713	0,1690	0,2314

Výsledky metody sim_{sum} v kombinaci s metodou polarity *lex* se stejným nastavením parametrů jsou vidět v tabulce 11.3.

Tabulka 11.3: Výsledky metody $lex + sim_{sum}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3656	0,1626	0,2253	0,3672	0,1639	0,2267
0,9	0,3678	0,1657	0,2279	0,3728	0,1683	0,2313
0,8	0,3684	0,1624	0,2256	0,3736	0,1674	0,2308
0,7	0,3681	0,1619	0,2254	0,3726	0,1644	0,2277
0,6	0,3692	0,1647	0,2274	0,3715	0,1651	0,2288
0,5	0,3684	0,1644	0,2272	0,3682	0,1630	0,2261
0,4	0,3670	0,1639	0,2269	0,3688	0,1635	0,2268
0,3	0,3684	0,1626	0,2256	0,3695	0,1665	0,2287
0,2	0,3701	0,1657	0,2286	0,3706	0,1672	0,2310
0,1	**0,3741**	**0,1703**	**0,2333**	**0,3747**	**0,1690**	**0,2330**

U metody $lex + sim_{sum}$ bylo stejně jako u $lex + sim_{sent}$ dosaženo lepších výsledků pro tvorbu matice M pomocí vážení *tfidf*. Nejlepších výsledků pro hodnocení podobnosti dosáhla funkce $lex + sim_{sum}$ s redukcí $r = 0,1$ a stavbou matice M metodou *tfidf*. Toto nastavení hodnocení podobnosti bude použito u všech dalších testů a bude označeno sim_{sum}.

11.2 Významnost

Hodnocené metody důležitosti jsou popsané v sekci 9.5. Zkoumané parametry jsou stejné jako u metod podobnosti. Metody jsou zkoumané v kombinaci s hodnocením polarity *lex* a s/bez hodnocení podobnosti sim_{sum}.

11.2.1 topic

Výsledky metody *topic* při použití metody podobnosti sim_{sum} a polarity *lex* jsou v tabulce 11.4.

Tabulka 11.4: Výsledky metody $lex + topic + sim_{sum}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3683	0,1651	0,2293	0,3704	0,1670	0,2305
0,9	0,3683	0,1651	0,2293	0,3704	0,1670	0,2305
0,2	0,3682	0,1652	0,2293	0,3702	0,1673	0,2304
0,1	**0,3736**	**0,1692**	**0,2340**	0,3677	0,1641	0,2275

Výsledky metody *topic* s hodnocením polarity *lex* bez použití metody podobnosti jsou v tabulce 11.5.

Tabulka 11.5: Výsledky metody $lex + topic$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3684	0,1636	0,2276	0,3668	0,1638	0,2277
0,9	0,3684	0,1636	0,2276	0,3668	0,1638	0,2277
0,2	0,3695	0,1655	0,2297	0,3674	0,1636	0,2276
0,1	**0,3731**	**0,1671**	**0,2319**	0,3692	0,1657	0,2297

V obou tabulkách jsou zobrazeny pouze výsledky pro krajní hodnoty redukčního parametru. Redukcí matice V^T odstraníme spodní r-část řádků. Redukce se tak projeví až v případě, kdy ponecháme méně řádků, než kolik je vybíraných vět. Nejlepších výsledků bylo dosaženo při ponechání 10% témat. V takovém případě je v průměru ponecháno 7-8 témat. Při výběru 10-11 vět (100 slov) to ve výsledku znamená, že z prvních třech témat budou vybrány dvě věty, abychom dosáhli na požadovaný počet slov. Distribuce vět

v tématech bude následující:

$$\{2, 2, 2, 1, 1, 1, 1\}$$

Z důležitějších témat je vybráno více vět, než z méně důležitých. Takové rozdělení připomíná metodu Murray a spol. (2005), viz. sekce 7.2.2 a naznačuje, že při použití této metody bychom dosáhli lepších výsledků.

Ani jedna z verzí však nedosáhla lepších výsledků, než samotná metoda polarity a podobnosti $lex + sim_{sum}$. To je nejspíše zapříčeněno tím, že vybíráme věty z prvních n témat, aniž bychom zkoumali, jak jsou si témata podobná. Druhé téma může být například podtématem prvního. Druhým důvodem je, že věty hodnotíme dle významnosti k jedinému tématu, což dostatečně nevypovídá o celkové důležitosti věty ve vztahu ke všem tématům a tato metoda je z tohoto důvodu značně primitivní. A třetím důvodem je fakt, že i tato metoda preferuje delší věty. Delší věta má větší pravděpodobnost, že ve sledovaném tématu bude mít větší hodnotu, jelikož se týká více témat. Také jsem zkoušel nahradit hodnocení polarity *lex* metodou *maxEnt*. Tato změna však přinesla zhoršení výsledků.

11.2.2 length

Metoda *length* byla testována v celkem čtyřech konfiguracích:

1. $lex + length_{negative}$
2. $lex + length_{negative} + sim_{sum}$
3. $lex + length_{nonnegative}$
4. $lex + length_{nonnegative} + sim_{sum}$

Každá z těchto konfigurací byla testována vzhledem k různým nastavením parametru r. Výsledky první konfigurace jsou vidět v tabulce 11.6.

V tabulce 11.6 je vidět, že žádné nastavení nedosáhlo výrazně lepších ani horších výsledků než ostatní. To je dáno principem metody $length_{negative}$. Metoda pracuje s velikostmi vektorů, kde kvůli druhé mocnině ve vzorci dojde k eliminaci významu negativních hodnot. Tyto hodnoty však reprezentují významnost tématu ve větě a jejich potlačení, ba dokonce obrácení jejich významu, je nežádoucí. Eliminací negativních hodnot dojde k eliminaci informace o příslušnosti věty k tématu, resp. tématům a všechna nastavení tak produkují podobné výsledky.

Tabulka 11.6: Výsledky metody $lex + length_{negative}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3700	0,1662	0,2302	0,3664	0,1638	0,2275
0,9	0,3709	0,1680	0,2322	0,3667	0,1640	0,2277
0,8	0,3713	0,1682	0,2324	0,3659	0,1634	0,2271
0,7	0,3706	0,1678	0,2320	0,3658	0,1636	0,2272
0,6	0,3709	0,1679	0,2321	0,3663	0,1645	0,2278
0,5	0,3708	0,1681	0,2322	**0,3677**	**0,1650**	**0,2287**
0,4	**0,3716**	**0,1688**	**0,2329**	0,3669	0,1648	0,2285
0,3	0,3695	0,1655	0,2298	0,3654	0,1637	0,2275
0,2	0,3716	0,1673	0,2315	0,3633	0,1609	0,2247
0,1	0,3715	0,1653	0,2304	0,3674	0,1635	0,2279

Tabulka 11.7: Výsledky metody $lex + length_{negative} + sim_{sum}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3721	0,1687	0,2335	0,3687	0,1665	0,2309
0,9	0,3713	0,1683	0,2330	0,3683	0,1664	0,2307
0,8	0,3720	0,1687	0,2335	0,3690	0,1673	0,2317
0,7	**0,3726**	**0,1695**	**0,2343**	0,3689	0,1672	0,2316
0,6	0,3707	0,1673	0,2324	**0,3699**	**0,1678**	**0,2324**
0,5	0,3714	0,1676	0,2327	0,3661	0,1648	0,2293
0,4	0,3699	0,1663	0,2314	0,3663	0,1640	0,2284
0,3	0,3688	0,1644	0,2295	0,3650	0,1615	0,2263
0,2	0,3697	0,1648	0,2304	0,3631	0,1593	0,2240
0,1	0,3679	0,1630	0,2281	0,3686	0,1661	0,2303

Tabulka 11.7 obsahuje výsledky metody $length_{negative}$ při použití hodnocení podobnosti sim_{sum}. Přestože přidání hodnocení podobnosti přineslo mírné zlepšení, z tabulky 11.7 je patrné, že metoda trpí stejným nedostatkem jako metoda v tabulce 11.6. Výsledky metody *length* s odstraněním negativních hodnot z matice V^T jsou vidět v tabulce 11.8.

Z tabulky 11.8 je patrné, že metoda $length_{nonnegative}$ odstranila problém negativních čísel. Nejlepších výsledků bylo dosaženo při ponechání 10% témat, pomocí nichž je následně počítána velikost vektoru. V takovém případě

Tabulka 11.8: Výsledky metody $lex + length_{nonnegative}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3700	0,1647	0,2288	0,3639	0,1628	0,2250
0,9	0,3691	0,1639	0,2281	0,3708	0,1683	0,2320
0,8	0,3699	0,1646	0,2288	0,3710	0,1685	0,2323
0,7	0,3698	0,1646	0,2291	0,3709	0,1683	0,2322
0,6	0,3710	0,1652	0,2298	0,3695	0,1677	0,2314
0,5	0,3704	0,1644	0,2291	0,3702	0,1671	0,2311
0,4	0,3689	0,1647	0,2286	0,3689	0,1662	0,2299
0,3	0,3701	0,1634	0,2282	0,3694	0,1667	0,2306
0,2	0,3740	0,1666	0,2320	0,3715	0,1673	0,2317
0,1	**0,3775**	**0,1710**	**0,2365**	**0,3748**	**0,1709**	**0,2352**

jsou jednotlivé věty hodnoceny dle jejich příslušnosti k těmto 10% nejvýznamnějších témat.

Tabulka 11.9: Výsledky metody $lex + length_{nonnegative} + sim_{sum}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3681	0,1643	0,2289	0,3727	0,1688	0,2326
0,9	0,3680	0,1642	0,2288	0,3703	0,1668	0,2305
0,8	0,3685	0,1648	0,2295	**0,3717**	**0,1684**	**0,2322**
0,7	0,3707	0,1662	0,2314	0,3703	0,1676	0,2315
0,6	0,3704	0,1643	0,2297	0,3697	0,1668	0,2310
0,5	0,3706	0,1648	0,2301	0,3686	0,1667	0,2304
0,4	0,3692	0,1639	0,2289	0,3677	0,1654	0,2293
0,3	0,3654	0,1603	0,2254	0,3679	0,1651	0,2293
0,2	0,3692	0,1623	0,2281	0,3692	0,1651	0,2297
0,1	**0,3717**	**0,1645**	**0,2303**	0,3693	0,1669	0,2306

Snížení redundance použitím funkce sim_{sum} nepřineslo zlepšení, jak je vidět v tabulce 11.9.

11.2.3 subtractLength

Metoda *subtractLength* již obsahuje aparát pro snížení redundance v podobě odečtení vektoru vybrané věty od zbytku matice. Proto je přidání hodnocení podobnosti zbytečné. Testována byla pro dvě konfigurace:

- $lex + subtractLength_{negative}$
- $lex + subtactLength_{nonnegative}$

Testována byla konfigurace s ponecháním negativních hodnot i s jejich odstraněním. Obě konfigurace byly otestovány pro různé hodnoty redukčního parametru. Při vyhodnocení metody $lex + subtractLength_{negative}$ byl vidět stejný jev, jako u $lex + length_{negative}$, kdy negativní metody způsobily znehodnocení výsledků. Žádné z nastavení této metody nedosáhlo výrazně odlišných výsledků oproti ostatním. Proto zde nejsou výsledky záměrně uvedeny. Výsledky $lex + subtractlength_{nonnegative}$ jsou v tabulce 11.10.

Tabulka 11.10: Výsledky metody $lex + subtractlength_{nonnegative}$ pro $p = 0,5$.

	tvorba matice M					
	binary			tfidf		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,3688	0,1632	0,2264	**0,3729**	**0,1681**	**0,2291**
0,9	0,3686	0,1633	0,2268	0,3721	0,1676	0,2319
0,8	0,3674	0,1615	0,2251	0,3713	0,1670	0,2314
0,7	0,3687	0,1633	0,2275	0,3706	0,1668	0,2310
0,6	0,3662	0,1618	0,2258	0,3709	0,1667	0,2312
0,5	0,3667	0,1626	0,2263	0,3699	0,1655	0,2301
0,4	0,3706	0,1660	0,2301	0,3684	0,1623	0,2325
0,3	0,3707	0,1623	0,2274	0,3698	0,1647	0,2269
0,2	0,3666	0,1591	0,2238	0,3665	0,1636	0,2271
0,1	0,3708	0,1631	0,2276	0,3725	0,1676	0,2318

Nejlepší výsledek poskytlo nastavení s nulovou redukcí dimenze. Výsledky se snižovaly se zvyšujícím se procentem redukce. Zajímavé je, že při ponechání pouze 10% dimenzí došlo k průdkému zlepšení výsledků. To naznačuje, že je lepší ponechat dimenze všechny, nebo jich ponechat takové množství, kolik vybíráme vět.

11.2.4 invertLength

Metoda *invertLength* byla navržena za účelem preferování kratších vět. Metodu jsem nejprve otestoval v kombinaci s metodou *lex* pro porovnání s

předchozími výsledky. Metoda dosáhla lepších výsledků než pžedchozí metody. Následně jsem ji otestoval v kombinaci s metodou *maxEnt* pro ještě větší potlačení delších vět. Tato záměna dále zlepšila výsledky. Ty jsou k vidění v tab. 11.11.

Tabulka 11.11: Výsledky metody $tfidf + maxEnt + invertLength$.

	podobnost					
	žádná			sim_{sum}		
r	ROU-1	ROU-2	ROU-SU4	ROU-1	ROU-2	ROU-SU4
1,0	0,4112	0,1914	0,2421	0,4206	0,1948	0,2486
0,9	0,4099	0,1876	0,2401	0,4242	0,1991	0,2532
0,8	0,4109	0,1895	0,2423	0,4241	0,2003	0,2546
0,7	0,4166	0,1967	0,2498	0,4224	0,1983	0,2533
0,6	0,4170	0,1957	0,2505	**0,4259**	**0,2014**	**0,2568**
0,5	**0,4207**	0,1999	0,2563	0,4236	0,2020	0,2570
0,4	0,4175	**0,2035**	**0,2606**	0,4221	0,1993	0,2559
0,3	0,4098	0,1949	0,2545	0,4218	0,2085	0,2657
0,2	0,4062	0,1952	0,2554	0,4137	0,2014	0,2614
0,1	0,3991	0,1896	0,2530	0,4078	0,1936	0,2577

Jak je vidět kombinace $maxEnt + invertLength + sim_{sum}$ dosáhla mnohem lepších výsledků než veškeré předchozí metody.

11.2.5 Porovnání nejlepších výsledků

Tabulka 11.12 obsahuje výsledky nejlepšího nastavení pro veškeré testované kombinace sumarizace.

Tabulka 11.12: Přehled nejlepších výsledků pro všechny metody pro $p = 0,5$.

metoda	r	ROU-1	ROU-2	ROU-SU4
$random$		0,3277	0,1094	0,1739
lex		0,3734	0,1692	0,2322
$lex + sim_{sum}$	0,1	0,3747	0,1690	0,2330
$lex + topic + sim_{sum}$	0,1	0,3736	0,1692	0,2340
$lex + length_{nonnegative}$	0,1	0,3775	0,1710	0,2365
$lex + subtractlength_{nonnegative}$	1,0	0,3729	0,1681	0,2291
$maxEnt + invertLength + sim_{sum}$	0,6	**0,4259**	**0,2014**	**0,2568**
Ježek + Campr + Nykl		0,4059	0,2273	0,2128

Všechny metody dosáhly lepších výsledků než náhodný výběr. Stávající metody využívající singulárního rozkladu však nedosáhly výrazně lepších

výsledků, než samotná polaritní metoda *lex*. To může být zapříčeněno tím, že metoda *lex* sama o sobě dosahuje velice solidního výsledku. Navržená metoda *invertLength* dosáhla nejlepších výsledků. Její výsledky byly výrazně lepší než výsledky ostatních metod, což prokazuje, že řeší problém preference dlouhých vět. Vylepšení metody *invertLength* oproti metodě $lex + sim_{sum}$ jsem ověřil provedením párového t-testu statistické významnosti na ROUGE výsledcích souhrnů jednotlivých restaurací. Na 95% konfidenčním intervalu se vylepšení metody *invertLength* ukázalo jako statisticky významné.

Negativní vliv na výsledky měly zejména záporné hodnoty v matici V^T u metod, kde dojde k eliminaci jejich znaménka. Další nevýhoda metody LSA je patrná při zkoumání vytvořených souhrnů. Je to již zmíněné preferování delších vět, které se v recenzích často vyskytují. Jejich autoři často napíšou celé své hodnocení do jedné dlouhé věty a ta je proto vybrána jako velice důležitá. Svou dlouho délkou však zabere většinu souhrnu. Anotátoři tak ideální souhrny staví spíše z kratších vět, které poté vystihují cítění více osob.

12 Závěr

V první části práce jsem popsal problém hodnocení polarity, důležitosti a podobnosti a některé jejich metody. Dále jsem popsal metodu Latentní sémantické analýzy, shrnul problém sumarizace a popsal metriku ROUGE. Ve druhé části jsem navrhl a obohatil o modifikaci několik metod kontrastní názorové sumarizace, které jsou založené na známých použitích metody LSA. Vytvořil jsem knihovnu v jazyce Java pro automatickou sumarizaci a implementoval jsem navržené metody použití LSA a seznamů polaritních termínů pro kontrastní názorovou sumarizaci.

Všechny metody poskytly při ověření metrikou ROUGE solidní výsledky, přestože sumarizace textů v českém jazyce je obtížným úkolem. To platí speciálně pro recenzní texty, u kterých není vždy dodržena správná forma a gramatika. Při testování jsem objevil některé výhody i nevýhody metody LSA. Hlavní nevýhodou se ukázalo být zejména preferování dlouhých vět. Proto jsem navrhl metodu využití LSA pro potlačení tohoto vlivu. Tato metoda nazvaná $maxEnt + invertLength$ dosáhla při redukci dimenze na 60% nejlepších výsledků:

- ROUGE-1: 0,4259
- ROUGE-2: 0,2014
- ROUGE-SU4: 0,2568

Tento výsledek byl výrazně lepší než veškeré ostatní metody založené na současných přístupech použití LSA. Překonal i výsledek dosažený v práci [4]. Všechny úkoly vytyčené na začátku práce byly splněny.

V budoucnu by bylo dobré navržené metody ověřit na jiných testovacích datech, obzvláště anglických, pro lepší porovnání s konkurenčními metodami. Zároveň by bylo dobré otestovat jiné metody kontrastní názorové sumarizace pomocí LSA.

Přehled zkratek

LSA - latentní sémantická analýza
SVD - singulární dekompozice
ROUGE - Recall-Oriented Understudy for Gisting Evaluation

Přílohy

Listing 1: XML konfigurační soubor nejlepších konfigurací.

```
<?xml version = "1.0" encoding = "UTF-8"?>

<!--suppress SpringFacetInspection -->
<beans xmlns = "http://www.springframework.org/schema/beans"
       xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-3.0.xsd">

    <bean id="data" class="summarization.data.io.InOutSettings">
        <constructor-arg name="inputFolder" value="data/reviews/"/>
        <constructor-arg name="outputFolder" value="data/tests/test01/"/>
    </bean>

    <bean id="preprocessing" class="summarization.preprocessing.Preprocessing">
        <constructor-arg name="normalizer" ref="normalizer_rouge"/>
        <constructor-arg name="removeStopWords" value="true"/>
    </bean>

    <bean id="normalizer_rouge" class="summarization.preprocessing.Lemmatizer">

    </bean>

    <bean id="rouge_preprocessing" class="summarization.preprocessing.Preprocessing">
        <constructor-arg name="normalizer" ref="normalizer_rouge"/>
        <constructor-arg name="removeStopWords" value="false"/>
    </bean>

    <bean id="matrix_builder" class="summarization.svd.matrixbuilder.BinaryBuilder">
    </bean>

    <bean id="matrix_builder_tf" class="summarization.svd.matrixbuilder.tfidf.
         TfIdfBuilder">
        <constructor-arg name="tfidf" ref="tfidf"/>
    </bean>

    <bean id="tfidf" class="summarization.svd.matrixbuilder.tfidf.TfIdf">
        <constructor-arg name="voc" ref="vocabulary"/>
    </bean>

    <bean id="vocabulary" class="summarization.data.Vocabulary">
        <constructor-arg name="file" value="data/reviews/"/>
        <constructor-arg name="folder" value="true"/>
        <constructor-arg name="pr" ref="preprocessing"/>
    </bean>


    <!-- ============== SUMMARIZERS ================== -->

    <util:list id="summarizers" list-class="java.util.ArrayList">
        <ref local="random"/>
        <ref local="length"/>
        <ref local="topic"/>
        <ref local="sublength"/>
        <ref local="invertlength"/>
        <ref local="pol_sim"/>
        <ref local="pol"/>
    </util:list>

    <bean id="pol_sim" class="summarization.contrastive.lsa.LSAScoreSummarizer">
        <constructor-arg name="name" value="pol-sim"/>
        <constructor-arg name="im"><null/></constructor-arg>
        <constructor-arg name="sm" ref="svd_sm_1"/>
        <constructor-arg name="pm" ref="lex_pm"/>
        <constructor-arg name="p" value="0.5"/>
    </bean>

    <bean id="pol" class="summarization.contrastive.lsa.LSAScoreSummarizer">
        <constructor-arg name="name" value="pol"/>
        <constructor-arg name="im"><null/></constructor-arg>
```

```
        <constructor-arg name="sm"><null/></constructor-arg>
        <constructor-arg name="pm" ref="lex_pm"/>
        <constructor-arg name="p" value="0.0"/>
    </bean>

    <bean id="length" class="summarization.contrastive.lsa.LSAScoreSummarizer">
        <constructor-arg name="name" value="length"/>
        <constructor-arg name="im" ref="length_im"/>
        <constructor-arg name="sm"><null/></constructor-arg>
        <constructor-arg name="pm" ref="lex_pm"/>
        <constructor-arg name="p" value="0.5"/>
    </bean>

    <bean id="topic" class="summarization.contrastive.lsa.LSAScoreSummarizer">
        <constructor-arg name="name" value="topic"/>
        <constructor-arg name="im" ref="topic_im"/>
        <constructor-arg name="sm"><null/></constructor-arg>
        <constructor-arg name="pm" ref="lex_pm"/>
        <constructor-arg name="p" value="0.5"/>
    </bean>

    <bean id="sublength" class="summarization.contrastive.lsa.LSAScoreSummarizer">
        <constructor-arg name="name" value="sublength"/>
        <constructor-arg name="im" ref="sublength_im"/>
        <constructor-arg name="sm"><null/></constructor-arg>
        <constructor-arg name="pm" ref="lex_pm"/>
        <constructor-arg name="p" value="0"/>
    </bean>

    <bean id="invertlength" class="summarization.contrastive.lsa.LSAScoreSummarizer">
        <constructor-arg name="name" value="invertlength"/>
        <constructor-arg name="im" ref="invert_im"/>
        <constructor-arg name="sm"><null/></constructor-arg>
        <constructor-arg name="pm" ref="maxent_pm"/>
        <constructor-arg name="p" value="0"/>
    </bean>

    <bean id="length_im" class="summarization.importance.svd.Length">
        <constructor-arg name="builder" ref="matrix_builder"/>
        <constructor-arg name="reductionRatio" value="0.1"/>
        <constructor-arg name="nonNegative" value="true"/>
    </bean>

    <bean id="sublength_im" class="summarization.importance.svd.SubtractLength">
        <constructor-arg name="builder" ref="matrix_builder_tf"/>
        <constructor-arg name="reductionRatio" value="1.0"/>
        <constructor-arg name="nonNegative" value="true"/>
    </bean>

    <bean id="topic_im" class="summarization.importance.svd.Topic">
        <constructor-arg name="builder" ref="matrix_builder"/>
        <constructor-arg name="reductionRatio" value="0.1"/>
    </bean>

    <bean id="invert_im" class="summarization.importance.svd.InvertLength">
        <constructor-arg name="builder" ref="matrix_builder_tf"/>
        <constructor-arg name="reductionRatio" value="0.6"/>
    </bean>


    <bean id="svd_sm_1" class="summarization.similarity.svd.SumSimilarity">
        <constructor-arg name="builder" ref="matrix_builder_tf"/>
        <constructor-arg name="reductionRatio" value="0.1"/>
    </bean>

    <bean id="maxent_pm" class="summarization.polarity.maxent.MaxEnt">
       <constructor-arg name="proc" ref="preprocessing"/>
    </bean>

    <bean id="lex_pm" class="summarization.polarity.LexiconPolarity">
        <constructor-arg name="pr" ref="preprocessing"/>
    </bean>

    <bean id="random" class="summarization.contrastive.RandomSummarizer">
        <constructor-arg name="name" value="random"/>
    </bean>

</beans>
```

Listing 2: Recenzní text ukázkové restaurace.

```
Restaurace Balaboosta je součástí penzionu Lion v Modřicích u Brna. Umístění ve vilové
    čtvrti na okraji města jí sice sluší, má však jednu zásadní nevýhodu: dostupnost
    jinak než autem není nejlepší. Chodníky tady nejsou, zastávky MHD jsou daleko a nav
    íc jsou mimo základní zóny. Samotná restaurace je pak jakýsi vystrčený altán, v
    jehož interiéru převládají zeleno-bílé tóny. Moderní design se snaží vzbudit pocit
    luxusu, ale daří se to jen napůl. Víc než v přepychovém prostředí si totiž připadám
     jako v pokojíku pro panenky. S tím koresponduje i nevelká kapacita restaurantu a
    atmosféra. Obsluhu zvládá jediný člověk, jehož profesionálnímu přístupu se nedá nic
     vytknout. Je znalý, všímavý, pozorný a pohotový. Jako pozornost podniku dostávám
    na úvod domácí husí sádlo s krajíčkem domácího chleba. Jemné a dobré. Stejné je i
    paté z husích jatýrek s cibulkovým džemem a švestkovým coulis. Vše se pěkně doplň
    uje, úprava na talíři vypadá dobře a porce je tak akorát, aby navnadila, ale zárove
    ň nechala prostor pro hlavní chod. Ten obstarávají medailonky z vepřové panenky s
    grilovanou cuketou a restovanými bramborami. A znovu nezbývá než chválit. Maso je
    krásně šťavnaté, brambory opečené do zlatova, přílohy není přehnaně moc, aby
    zahltila pozitivní chuťový vjem ostatních součástí tohoto menu. Jako dezert na závě
    r vybírám čokoládové brownies se zmrzlinou. Jednoduché a lahodné. Samotné brownies
    jsou krásně vláčné a podpořeny vanilkovou zmrzlinou a několika malinami přinášejí v
    ýborný zážitek. Espresso značky Hausbrandt s plným servisem je také bez výhrad. Růž
    ové mladé víno Cabernet Cortis z vinařství pana Františka Spěváka z Dubňan je krásn
    ě bohaté, výrazné ve vůni i chuti. Jediné, co mi trochu nedává smysl, je jeho servír
    ování a účtování po 1,5dcl. Celkový dojem z návštěvy restaurace Balaboosta je nicm
    éně pozitivní a stojí za námahu spojenou s cestováním do těchto vzdálených končin.
Tentokrát jen rychlý páteční oběd z denního menu. V poledne kolem 12. hodiny beznadějně
     plno, ale tady se chvilku počkat fakt vyplatí:-) Polední menu rychlé, moc dobré,
    jak kuřecí kousky, tak i burger. Vedlejší stůl moc chválil ptáčka s rýží :-) Pití
    nehodnotím, měli jsme jen vodu a nealko pivo. Nabídnutý dezert (už nás tam znají) -
     čokoládový dortík s makovou zmrzlinou s kafíčkem prostě super! Chodíme se poměrně
    často a ani super obsluha nikdy nezklamala!
Na návštěvu této restaurace se vždy velice těšíme! Jídlo výborné, atmosféra příjemná,
    obsluha skvělá! Nejde opomenout vtip a šarm servírky :-).
Po první návštěvě, která byla OK, byla večeře celkem zklamání... Možná to bylo smutkem
    po porážce se Švédy, pátek večer, ale to profesionály neomlouvá... Začněme interié
    rem: podvečer - světlo jen nad některými stoly, takže se musíte posadit, kde zrovna
     svítí... Stůl: trochu připomíná vietnamské bistro - bez ubrusu, bambusové prostírá
    ní - hygienicky neočistitelné... čest zachraňují jen látkové ubrousky... Obsluha
    docela v korektní, i když se po servírování nápojů nadlouho neukázali... A ná
    sleduje dlouhé čekání - lokál skoro prázdný a na 2 ryby čekáme skoro 3/4 hodiny (
    kuchař je asi obzvláště smutný). Čekání není zkráceno ani pečivem a máslem ani
    jinou pozorností podniku, což je zcela jinak, než minule. Když losos a candát koneč
    ní dorazí, musíme konstatovat, že kuchař je minimalista - žádné zdobení, ok, dnes
    držíme smutek. Ryby jsou obě v pořádku, také zelenina, kterou má manželka jako pří
    lohu... zato moje bramborové pyré s mascaropone je více než podprůměrné - a ty
    hrudky tam asi nepatří... ne? Když k tomu přidám fakt, že byly připraveny nože,
    které se hodí spíše na steak než rybu... je to další kapka, která mi po uvedeném zá
    žitku posouvá tento podnik daleko jinam, než jsem si myslel a nijak mě neláká se
    zde znovu zastavit...
Zatím jsme zašli spíš na něco malého a drink. Rozhodně umí polévky (hříbkový krém, i
    boršč byl chutný) a také mají dobré a originální vlastní dezerty (např. pomerančový
     želé dortík s mátou a kysanou smetanou). Jiný večer jsme zase vsadili na sýrový
    talíř a mile mě potěšilo, že to nebyla v ČR obligátní sestava Eidam, uzený Eidam a
    hermelín, ale pestřejší výběr sýrů. Nabízejí zajímavé týdenní nabídky, takže se urč
    itě chystáme na vícechodovou návštěvu. Dle lístku trochu dražší ceny. Souhlasím s t
    ím, že detaily interiéru jsou trochu "levnější" a bambusové prostírání není moc pří
    jemné.
Už po několikáté jsme se zastavili na oběd (dva dospělí, 8mi letý kluk :-). Jídlu,
    obsluze, atmosféře prostě není co vytknout. Číšník se servírkou milí, nevtíraví,
    ochotní, skvěle nabídl týdenní nabídku (neví totiž, že tam často jezdí právě kvůli
    ní :-)) Udělat poloviční porci pro dítě není nikdy problém, zatím nám vyšli vstříc
    u jakéhokoli jídla. Na víno nejsem znalec, ale dvojka červeného Portugalu byla moc
    dobrá. Prostě skvělé, vydržte prosím, rádi k vám jezdíme!
Skvělý zážitek a nebylo to poprvé co jsme tam byli. Vřele všem doporučuji!!!
Naprosto luxusní zážitek, nejlepší Carpaccio na světě a to už jsem jich ochutnala hodně
    ! Pánové, jen tak dále! Gratuluji...:-)))
```

Listing 3: Výsledný souhrn metodou *maxEnt+invertLength* pro ukázkovou restauraci.

```
V poledne kolem 12.
A znovu nezbývá než chválit.
Jídlo výborné, atmosféra příjemná, obsluha skvělá!
Na návštěvu této restaurace se vždy velice těšíme!
Prostě skvělé, vydržte prosím, rádi k vám jezdíme!
Skvělý zážitek a nebylo to poprvé co jsme tam byli.
Vřele všem doporučuji!
Jídlu, obsluze, atmosféře prostě není co vytknout.
Jemné a dobré.
Naprosto luxusní zážitek, nejlepší Carpaccio na světě a to už jsem jich ochutnala hodně
    !
```

```
Nejde opomenout vtip a šarm servírky :-).
Chodíme se poměrně často a ani super obsluha nikdy nezklamala!
Ryby jsou obě v pořádku, také zelenina, kterou má manželka jako přílohu.
Jako dezert na závěr vybírám čokoládové brownies se zmrzlinou.
```

Listing 4: Výpis prográmku *averager.jar* pro sumarizaci dle konfiguračního souboru *config/best.xml*.

```
SUBLENGTH.TXT 0.37291920000000006 0.16814230000000002 0.23249200000000006
RANDOM.TXT 0.3223669 0.10349650000000002 0.16813859999999997
INVERTLENGTH.TXT 0.42586460000000004 0.20137800000000003 0.25681429999999994
LEX–SIMSUM.TXT 0.37474969999999996 0.16899769999999997 0.23299459999999994
LENGTH.TXT 0.3774559 0.17097170000000003 0.23647719999999997
TOPIC.TXT 0.3735800999999999 0.16924309999999995 0.23396170000000002
LEX.TXT 0.37343070000000017 0.16923449999999998 0.23215470000000007
```

Literatura

[1] Aggarwal, C. C. – Zhai, C. *Mining text data*. Springer Science & Business Media, 2012.

[2] Ganesan, K. ROUGE 2.0: Updated and Improved Measures for Evaluation of Summarization Tasks. 2015.

[3] Gong, Y. – Liu, X. Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis. 01 2001, s. 19–25.

[4] Ježek, K. – Campr, M. Summarization of Contrastive Opinion in User Reviews, interní dokument.

[5] Konkol, M. Brainy: A machine learning library. In *International Conference on Artificial Intelligence and Soft Computing*, s. 490–499. Springer, 2014.

[6] Landauer, T. K. – Foltz, P. W. – Laham, D. An introduction to latent semantic analysis. *Discourse processes*. 1998, 25, 2-3, s. 259–284.

[7] Lin, C.-Y. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*. 2004.

[8] Murray, G. – Renals, S. – Carletta, J. Extractive summarization of meeting recordings. 2005.

[9] Nigam, K. – Lafferty, J. – McCallum, A. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, 1, s. 61–67, 1999.

[10] Ozsoy, M. G. – Cicekli, I. – Alpaslan, F. N. Text summarization of turkish texts using latent semantic analysis. 2010, s. 869–876.

[11] Ozsoy, M. G. – Alpaslan, F. N. – Cicekli, I. Text summarization using latent semantic analysis. *Journal of Information Science*. 2011, 37, 4, s. 405–417.

[12] Petersen, U. HAL example. Dostupné z: `https://emdros.org/examples/HAL/HALGuide.pdf`. [Online; 28. srpna 2017].

[13] Prokopp, C. Hyperspace Analgoue to Language Introduction. Dostupné z: `http://www.semantikoz.com/blog/hyperspace-analogue-to-language-hal-introduction/`. [Online; 28. srpna 2017].

[14] Reidy, P. An Introduction to Latent Semantic Analysis. Dostupné z: http://www.ling.ohio-state.edu/~reidy.16/LSAtutorial.pdf. [Online; 28. srpna 2017].

[15] Steinberger, J. – Jezek, K. Using latent semantic analysis in text summarization and summary evaluation. *Proc. ISIM*. 2004, 4, s. 93–100.