Sem vložte zadání Vaší práce.

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

Bakalářská práce

Automatizované zkracování textu

Ivo Sklenář

Vedoucí práce: doc. RNDr. Ing. Marcel Jiřina, Ph.D.

10. května 2015

Poděkování

Děkuji doc. RNDr. Ing. Marcelu Jiřinovi, Ph.D. za vedení práce a za poskytnuté rady předané pro tuto práci. Dále bych chtěl poděkovat rodině a přátelům za podporu během studia a při tvorbě této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 10. května 2015 .....

Odkaz na tuto práci

Sklenář, Ivo. *Automatizované zkracování textu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem této práce je prozkoumat a implementovat některé metody pro automatizované zkracování textu. Práce důkladně zkoumá čtyři takové algoritmy a kromě algoritmů pro zkracování textu se také zabývá podpůrnými metodami pro zpracování přirozeného jazyka. V závěru byly jednotlivé algoritmy mezi sebou porovnány pro český a anglický jazyk a byly identifikovány nejlepší způsoby zkracování textů pro každý jazyk.

Klíčová slova zkracování textu, přirozený jazyk, strojové učení, evaluace textu

Abstract

Goal of this thesis is to research and implement several algorithms for automatic text summarization. This thesis does a detail analysis of four algorithms and apart from text summarization algorithms, it also analyzes algorithms and methods used for natural language processing. At the end of this work, these algorithms were compared between each other for Czech and English languages and for each language, the best method of summarization was identified

Keywords text summarization, natural language, machine learning, text evaluation

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešním informačním věku existuje spousta zdrojů, které nám chtěně i nechtěně poskytují informace. Ať už se jedná o aktuální dění ve světě, článek na online encyklopedii nebo finanční výsledky firem. Informace nás obklopují a je jich stále více.

S rostoucím počtem informací roste také čas nutný k jejich přečtení a zpracování. Procesory se zrychlují, disky zvětšují, ale lidská rychlost čtení zůstává stále stejná. Proto se v posledních letech rozšířily nástroje pro automatizované zkracování textu a extrahování pouze důležitých informací v nich obsažených.

Zkracování textu je velice široká disciplína, a proto také existuje spousta způsobů a algoritmů, které nám umožňují dosáhnout tohoto cíle. Jedná se o algoritmy zaměřené na obecné zkracování, ale i algoritmy přesně nastavitelné na jeden typ a téma textu.

V první kapitole je popsáno rozdělení algoritmů, popis podpůrných metod a existujících řešení. Ve druhé kapitole jsou popsány algoritmy vybrané v této práci. Kapitola třetí se zabývá popisem implementace těchto algoritmů a v závěrečné čtvrté kapitole je provedeno měření přesnosti jednotlivých algoritmů a jejich srovnání.

Kapitola 1

Analýza

Tato kapitola se zabývá taxonomií metod pro automatizované zkracování textu, podpůrnými metodami a algoritmy a analýzou existujících řešení.

Automatizované zkracování textu je postup realizovaný metodami a algoritmy pro redukci vstupního textu do kratší podoby. Cílem těchto algoritmů je, aby jejich výstup obsahoval co nejvíce informací a zároveň, aby byl co nejkratší. Snaží se tedy maximalizovat počet informací ve výstupním textu, zatímco minimalizují délku tohoto textu.

1.1 Taxonomie

Automatizované zkracování textu se dělí do několika kategorií [3].

1.1.1 Způsob výběru vět

Algoritmy založené na extrakci se snaží v textu identifikovat důležité věty a ty následně extrahovat do výstupu. Tyto věty jsou ve výstupu nezměněné a organizované stejně, jako ve vstupním dokumentu.

Algoritmy abstrakce generují nové věty. Tyto věty mohou obsahovat slova nebo fráze ze vstupního textu, ale v důsledku se snaží předávat informace a myšlenky z původního dokumentu do nově vytvořeného. Abstrakce teoreticky dosahuje lepších výsledků, ale kvůli náročnosti zpracování přirozeného jazyka počítačem je tento způsob těžko použitelný.

1.1.2 Počet dokumentů

Jedno-dokumentové algoritmy mají jako vstup jeden dokument. Z tohoto dokumentu se následně snaží vytvořit zkrácený text bez ostatních vnějších vstupů.

Více-dokumentové algoritmy na vstup přijímají více dokumentů a z nich tvoří jeden zkrácený. Tyto algoritmy se mohou dívat na dokumenty jako je-

den velký celek nebo je mohou považovat za různé a vytvářet jeden dokument obsahující nezávislé výtahy těchto dokumentů. Pokročile více-dokumentové algoritmy dokážou vstupní dokumenty klasifikovat do skupin podle témat a následně použít kombinaci obou přístupů k vytvoření jednoho cílového dokumentu.

1.1.3 Dotazové zkracování

Výstupem obecného zkracování je text, který se snaží pokrýt všechny myšlenky vstupního textu. Obecné zkracování neklade žádné specifické předpoklady na cílovou audienci čtenářů a ani na doménu nebo téma textu.

Naopak dotazové zkracování potřebuje kromě vstupního textu i dotaz od uživatele. Tento dotaz říká algoritmu na jaké informace se má soustředit, čímž je lépe specifikována doména textu. Výstupem je potom zkrácený text, který je zaměřený na myšlenky položené v dotazu uživatelem.

1.1.4 Vícejazyčnost

Mohou existovat dokumenty, u kterých je část textu psaná v jiném jazyce. Tento případ může také nastat u více-dokumentového zkracování, kde může být každý dokument psaný jiným jazykem. Algoritmy pak dokáží identifikovat jazyky obsažené ve vstupu a přizpůsobit se jim, ať už překladem nebo jinou metodou.

1.1.5 Cíl zkracování

Indikativní zkracování si klade za úkol seznámit čtenáře s obsahem dokumentu. Jedná se o krátký výtah, který má uživatele nalákat ke čtení zkracovaného dokumentu. Výstupem informativního zkracování je plná náhrada za původní text, a to pomocí eliminace redundantních informací.

1.1.6 Formát výstupu

Při obecném zkracování je výstupem zkrácený text o libovolné délce. Délka výstupu je nastavitelná pro některé algoritmy, popřípadě ji lze měnit pomocí vstupních dat. Klíčové zkracování z textu vybírá klíčová slova, která nejlépe reprezentují vstupní text. Titulové zkracování je speciální případ obecného, kde výstupem je jedna věta sloužící jako titulek pro vstupní text.

1.2 Algoritmy zkracování textu

Algoritmy pro zkracování textů lze dělit do dvou kategorií. První kategorie obsahuje obecné algoritmy, tedy algoritmy, které dostanou na vstup text, pomocí nějaké metody identifikují důležité věty a ty následně extrahují do výstupu.

Mezi tyto algoritmy patří TextRank [1], Latentní sémantická analýza (LSA) [4], MEAD [5], LexRank [6] nebo evoluční algoritmus založený na clustrování [7].

Druhou kategorií jsou algoritmy postavené na strojovém učení. Tyto algoritmy fungují na principu učení se z dat. Jelikož proces zkracování je vlastně identifikace, resp. klasifikace vět, můžeme těmto algoritmům předložit trénovací data a naučit je, jak a podle čeho klasifikovat jednotlivé věty. Zde lze použít téměř jakékoliv algoritmy učení s učitelem.

Algoritmy strojového učení potřebují pro svoji správnou funkci trénovací množinu dat. V případě zkracování textu se jedná o vstupní text a k němu referenční zkrácený text. Z této trénovací množiny extrahují hodnoty předem definovaných atributů a následně tyto atributy používají ke klasifikaci vstupu.

Poskytnutá trénovací data ovlivňují přesnost těchto algoritmů. To může být výhodou, neboť existuje možnost natrénovat klasifikátory úzce specializovanými daty a následně dosahovat velice vysokých přesností klasifikace.

1.3 Podpůrné algoritmy a metody zpracování přirozeného jazyka

Ke zlepšení přesnosti algoritmů pro zkracování textu je vhodné použít některé metody pro transformaci tohoto textu. Tyto metody mohou eliminovat redundantní informace nebo transformovat slova k dosažení větší přesnosti při porovnávání vět.

1.3.1 Transformace slov

V mnoha jazycích jsou slova skloňována, např. kvůli časování nebo životným rodům. Toto skloňování může zkreslit podobnost dvou vět, a proto může být lepší věty a slova transformovat a pracovat s jejich transformacemi.
Mezi tyto transformace patří:

- Stemming – nalezení kořenového tvaru slova. Tyto algoritmy jsou založené na suffix stripping algoritmu. Pro angličtinu je nejpoužívanější Porter Stemmer [8], pro ostatní jazyky Snowball Stemmer [9];

- Lemmatizace – hledání základního (slovníkového) tvaru slova. Nejrozšířenější lemmatizér je LemmaGen [10], který je založený na lemmatizační databázi.

1.3.2 Slovníkové databáze

Kromě transformace slov je také vhodné používat databáze slov, ve kterých máme pro každé slovo uložená upřesňující data. Tato data nám pak mohou

pomoci při provádění algoritmů, ať už při porovnávání vět nebo při získávání informačních dat z věty.
Takové databáze mohou obsahovat:

- Stop words – slova jako *the, a, an* v angličtině; *ale, je* v češtině. Nenesou žádnou informaci a při porovnávání jen zkreslují výsledek;
- Synonyma - databáze WordNet [11] pro anglická slova, která pro každé slovo obsahuje také jeho synonyma;
- Slovní druhy - databáze slovních druhů, známé také pod názvem POS (Part of Speech) tagger, kde ke každému slovu je uložený jeho slovní druh.

1.4 Existující řešení

Na internetu existuje spousta webových stránek, které dokáží zkrátit text (např. `http://textcompactor.com/` nebo `http://autosummarizer.com`). Tento způsob je vhodný pro zkracování malého počtu krátkých textů. Tyto stránky většinou neposkytují veřejné API a nebo poskytují placené, popř. omezené API. Mezi jejich další nevýhody patří použití velice jednoduchých a nepřesných algoritmů a závislost na jazyku vstupního textu (typicky angličtina).

Na repositáři pro open-source projekty `www.github.com` lze najít několik knihoven pro zkracování textu. Nejrozsáhlejší z nich je knihovna pro Python SUMY [12]. Tato knihovna obsahuje 5 algoritmů a dokáže zkracovat i text ve formátu HTML stránek. Ostatní knihovny obsahují implementaci jednoho nebo dvou jednoduchých algoritmů.

Existují komerční nástroje jako například Essential Summarizer [13]. Tento nástroj je placený, licence pro osobní užití začíná na 150 EUR.

Kromě těchto nástrojů existuje i populární mobilní aplikace Summly [14], která agreguje zprávy z více zdrojů a vytváří krátké zprávičky, které jsou následně doručené přímo do mobilního telefonu.

Kapitola 2

Návrh

V této kapitole jsou detailně popsány algoritmy vybrané v této práci. Jedná se o algoritmy TextRank, latentní sémantická analýza, naivní Bayesův klasifikátor a KNN.

2.1 Obecné algoritmy

Obecné algoritmy nepotřebují, kromě vstupního textu, dodatečné vstupy. Tyto algoritmy mají výhodu, že dokáží pracovat nad textem libovolného tématu a struktury. Jejich nevýhodou může být, že nemusí dosahovat tak dobrých výsledků jako algoritmy založené na strojovém učení.

2.1.1 TextRank

TextRank [1] je algoritmus pro zkracování textu založený na teorii grafů. Používá stejné principy jako PageRank [15], který je hlavním stavebním prvkem vyhledávání na Google. Jeho výhodou je fakt, že se jedná o algoritmus bez učitele a tedy nepotřebuje k vykonávání své funkce žádná vzorová data. Kromě extrakce vět ho lze upravit pro extrakci klíčových slov.

2.1.1.1 Výpočet vah

TextRank se na větu s dívá jako na seznam slov, $s = [w_1, w_2, \ldots, w_n]$. Potom porovnávání dvou vět s_1 a s_2, je podle počtu slov w_k, která jsou společná v obou větách.

$$Similarity(s_1, s_2) = \frac{|\{w_k | w_k \in s_1 \wedge w_k \in s_2\}|}{\log(|s_1|) + \log(|s_2|)}$$

Tento způsob však není vhodný, jelikož kvůli skloňování slov je možné, že dvě věty nesoucí stejnou nebo velmi podobnou informaci jsou vyhodnoceny jako různé. Pro lepší porovnávání je vhodné na každé slovo věty aplikovat nějakou metodu z 1.3 a tyto slova buď transformovat nebo zanedbat.

2.1.1.2 Konstrukce grafu

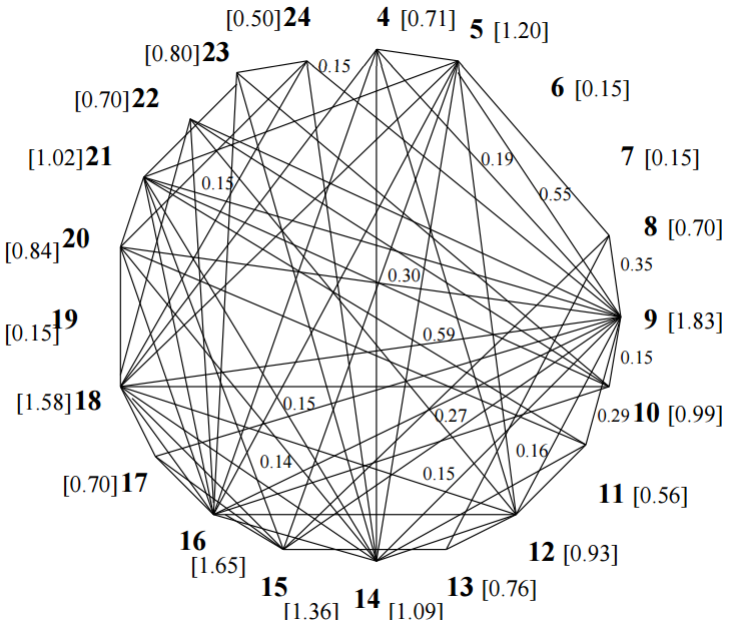
TextRank ohodnocuje jednotlivé věty pomocí grafu, kde každá věta je reprezentována jako jeden uzel neorientovaného ohodnoceného grafu. Pro propojení uzlů hranami je použita metrika z 2.1.1.1, která porovná dvě věty a určí hodnotu jejich podobnosti. Pokud tato metrika není nulová, je mezi dva uzly vložena hrana ohodnocená výsledkem této metriky, čímž postupně vzniká propojený graf.

Po sestavení grafu je každé větě vypočítáno skóre, označující jak moc je důležitá. Na začátku má každý uzel nastavené skóre na 1. Výpočet skóre S věty s_i reprezentovanou uzlem V_i vypadá následovně:

$$S_{V_i} = (1-d) + d \cdot \sum_{V_j \in N(V_i)} \frac{w_{ji}}{\sum_{V_k \in N(V_j)} w_{jk}} S_{V_j}$$

kde d je vhodně zvolená konstanta, w_{ij} značí váhu mezi uzly V_i a V_j a $N(V_k)$ je množina sousedů uzlu V_k. V algoritmu TextRank je doporučeno používat stejnou hodnotu $d = 0.85$ jako v algoritmu PageRank [15].

Postupně se iteruje a skóre každé věty se přepočítává, dokud všechny skóre nezkonvergují. Poznat, kdy skóre opravdu konvergovaly je velice obtížné. Iteruje se, dokud existuje alespoň jedna věta, které se mezi iteracemi změnilo skóre o více než 0.0001. Typicky skóre konverguje do 30 iterací.



Obrázek 2.1: Ukázka grafu vět s jejich skóre; zdroj [1]

2.1.1.3 Výběr vět

Po zkonvergování všech skóre vět se vyberou takové věty, které mají nejvyšší skóre. Počet těchto vět záleží na potřebách zkracování. Vybrané věty jsou sestaveny jako v původním dokumentu a vráceny jako výstup.

2.1.2 Latentní sémantická analýza

Latentní sémantická analýza (LSA) [4] identifikuje sémanticky důležité věty a následně jsou nejdůležitější z nich vybrány do zkráceného textu. Výhoda tohoto algoritmu je rychlost, schopnost pracovat nad libovolnými texty bez nutnosti učení a poměrně vysoké přesnosti.

2.1.2.1 Vážené vektory frekvencí termů

Vstupní text je rozdělen na věty a každá věta na části, tzv. termy. Term může reprezentovat cokoliv, typicky se jedná o slovo věty. Pro každou větu s_i je následně vypočítán vektor frekvence termů $T_{s_i} = [t_{1i}, t_{2i}, \ldots, t_{ni}]$, kde n je počet termů v dokumentu a t_{ji} označuje počet výskytů termu t_j ve větě s_i. Tento vektor je transformován na vážený vektor frekvencí termů věty s_i, $A_{s_i} = [a_{1i}, a_{2i}, \ldots, a_{ni}]$, kde a_{ji} je definováno jako $a_{ji} = L(t_j, s_i) \cdot G(t_j)$. $L(t_j, s_i)$ a $G(t_j)$ značí lokální, resp. globální váhu termu j.

$L(t_j, s_i)$ je počítáno pro větu s_i a term t_j jako počet výskytů termu t_j ve větě s_i. Na $G(t_j)$ je použita metrika inverse document frequency, $idf(t)$, definována $G(t_i) = idf(t_i) = \log \frac{N}{n_{t_i}}$, kde N je počet všech vět v dokumentu a n_{t_i} je počet vět obsahující term t_i.

2.1.2.2 Sestavení matice

Po vytvoření těchto vektorů je vytvořena matice termů a vět $A = [A_1, A_2, \ldots, A_N]$, kde A_i je sloupcový vážený vektor frekvencí termů věty s_i. Na tuto matici se lze dívat také jako na matici, kde řádky jsou termy a sloupce věty. Jelikož každý term nebývá obsažen v každé větě, je tato matice velice řídká.

Po sestavení této matice se provede singular value decomposition (SVD) matice A

$$A = U\Sigma V^T$$

kde U je $m \times n$ ortonormální matice; Σ je $n \times n$ diagonální matice, jejíž složky jsou seřazené podle velikosti a V je $n \times n$ ortonormální matice. Tuto dekompozici lze chápat jako organizaci vstupního textu na témata.

2.1.2.3 Extrakce vět

Po dekompozici se zajímáme o matici V^T. Pro extrakci k-té věty se hledá vektor v_i ze sloupcových vektorů matice V^T, jehož k-tá složka je největší. Po nalezení se k-tá věta extrahuje do zkráceného textu a vektor v_i se odebere z matice V^T. Tento postup se opakuje, dokud se nevybere požadovaný počet vět.

2.2 Algoritmy strojového učení

Algoritmy strojového učení jsou algoritmy, které se naučí jak klasifikovat data [16]. Tyto algoritmy pak dokáží elementy přiřadit do třídy kam patří. V této práci se rozhoduje, zda-li věta patří nebo nepatří do množiny zkráceného textu.

Algoritmy se dělí podle učení, konkrétně učení s učitelem a bez učitele. Učení s učitelem požaduje trénovací data, která jsou klasifikována do jednotlivých tříd. Podle těchto dat poté klasifikují vstupní data. Do této kategorie patří např. naivní Bayesův klasifikátor, algoritmus KNN nebo rozhodovací stromy.

Algoritmy bez učitele nepotřebují trénovací data. Místo toho se snaží ve vstupních datech najít skrytou vnitřní strukturu. Příkladem těchto algoritmů jsou k-means clustering nebo self-organizing map.

2.2.1 Atributy klasifikace

Pro potřeby klasifikace je nutné extrahovat atributy ze vstupních dat. V této práci se jako jedna instance dat považuje jedna věta textu. Počet a typ těchto atributů pak výrazně ovlivňuje přesnost klasifikace.
Použité atributy:

- Zkrácená množina
- Počet znaků
- Počet slov
- Počet čárek
- Počet slov začínajících velkým písmenem
- Počet akronymů (slova psaná pouze velkými písmeny)
- Počet závorek
- Počet interpunkčních znamének
- Počet fyzikálních jednotek
- Počet čísel
- Počet zkratek
- Počet apostrofů
- Počet slov kratších než 3 znaky (včetně)
- Počet slov delších jak 10 znaků (včetně)
- Počet podstatných jmen
- Počet přídavných jmen

- Počet sloves
- Počet odkazů
- Počet matematických symbolů
- Počet písmen/slov řecké abecedy
- Počet číslovek
- Zakončení věty
- Pozice v odstavci.

Tyto atributy byly navrženy společně s vedoucím práce a byly inspirovány atributy v [16].

První atribut je třída, neboli atribut, který se snažíme určit pomocí klasifikace. Poslední dva atributy jsou nominální a oba dosahují pouze tří hodnot. Zakončení věty má hodnotu 1 pro tečku, 2 pro otazník a 3 pro vykřičník. První věta v odstavci má hodnotu pozice v odstavci 1, poslední věta má hodnotu 3 a věty uprostřed odstavce hodnotu 2. Ostatní atributy jsou interpretovány jako diskrétní spojité náhodné veličiny s normálním rozdělením.

Spojité atributy mohou dosahovat hodnot $\langle 0, \infty)$. Tento interval však nemusí být ideální pro všechny algoritmy, proto může být vhodné tyto atributy normalizovat do intervalu $\langle 0, 1\rangle$. Toho lze dosáhnout pomocí několika normalizačních funkcí. V této práci byla zvolena soft-max normalizace [17].

Soft-max normalizace je založená na sigmoidální funkci a normalizuje hodnotu x_i na novou hodnotu y_i která je v intervalu $\langle 0, 1\rangle$ pomocí tohoto vzorce

$$y_i = \frac{1}{1 + e^{-(\frac{x_i - \mu_i}{\sigma_i})}}$$

kde μ_i je střední hodnota i-tého atributu v trénovací množině a σ_i je odmocnina z rozptylu i-tého atributu v trénovací množině.

2.2.2 Naivní Bayesův klasifikátor

Naivní Bayesův klasifikátor [16] je klasifikátor učení s učitelem založený na Bayesově větě. To znamená, že po trénovacích datech požaduje, aby měly přiřazenou třídu, do které patří. Tento klasifikátor počítá pravděpodobnosti z atributů trénovacích dat a následně podle těchto pravděpodobností rozhodne, do jaké třídy přiřadit právě klasifikovaný element.

Základním předpokladem tohoto klasifikátoru je, že jednotlivé atributy jsou na sobě nezávislé. Jedná se pouze o předpoklad, proto, pokud mezi atributy existuje pouze vzdálená závislost, je stále možné použít tento klasifikátor k dosažení velice dobrých a přesných výsledků.

Tento klasifikátor se používá při klasifikaci textu, ale má hojné využití v medicíně a v ostatních oblastech strojového učení a data miningu.

2.2.2.1 Klasifikátor

Tento klasifikátor získal své jméno z Bayesovy věty o podmíněné pravděpodobnosti

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

Tento vzorec můžeme použít pro klasifikování vektoru atributů $x = [x_1, x_2, \ldots, x_n]$ do třídy C následovně

$$P(C|x) = \frac{P(x|C) \cdot P(C)}{P(x)}$$

což lze upravit na klasifikaci pomocí hodnot atributů vektoru x jako

$$P(C|x) = \frac{P(C) \cdot \prod_{j=1}^{k} P(x_j|C)}{\sum_{i=1}^{m} \left(P(C_i) \cdot \prod_{j=1}^{k} P(x_j|C_i) \right)}$$

kde k je počet atributů vektoru x a m je počet tříd.

Jmenovatel tohoto vzorce je normalizace do intervalu $\langle 0, 1 \rangle$. Jelikož při klasifikaci není potřeba normalizovat tuto hodnotu a zároveň máme pouze dvě třídy, stačí pro klasifikaci věty s vypočítat tzv. posteriory pravděpodobnosti, jestli věta patří nebo nepatří do zkrácené množiny.

$$posteriory(s \in S) = P(s \in S) \cdot \prod_{j=1}^{k} P(s_j|s \in S)$$

a

$$posteriory(s \notin S) = P(s \notin S) \cdot \prod_{j=1}^{k} P(s_j|s \notin S))$$

kde $s \in S$ a $s \notin S$ značí, zda-li věta patří, resp. nepatří do množiny zkráceného textu, s_j je hodnota j-tého atributu věty s, k je počet atributů klasifikace, $P(s \in S)$ je hodnota odhadnutá z trénovacích dat a $P(s \notin S) = 1 - P(s \in S)$.

Následně se porovnají hodnoty $posteriory(s \in S)$ a $posteriory(s \notin S)$. Pokud $posteriory(s \in S) \geq posteriory(s \notin S)$, je věta s klasifikována jako že patří do množiny zkráceného textu, v opačném případě že nepatří.

V případě, že chceme zkrátit dokument, je tento proces proveden nezávisle pro každou větu dokumentu.

2.2.2.2 Pravděpodobnosti atributů

Pro nominální atributy se pravděpodobnost hodnoty atributů počítá pomocí četnosti výskytu této hodnoty v trénovací množině.

I přes to, že ostatní atributy jsou diskrétní, není vhodné je počítat pomocí četnosti výskytu. Pro každý atribut je vypočítána střední hodnota z trénovací množiny

$$\mu = E(X) = \sum_{i=1}^{n} x_i \cdot P(X = x_i)$$

a rozptyl

$$\sigma^2 = E(X^2) - (E(X))^2$$

Na tyto atributy je poté nahlíženo jako na náhodné veličiny z normálního rozdělení $N(\mu, \sigma^2)$. Poté se pravděpodobnost hodnoty atributu počítá pomocí hustoty pravděpodobnosti rozdělení $N(\mu, \sigma^2)$:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

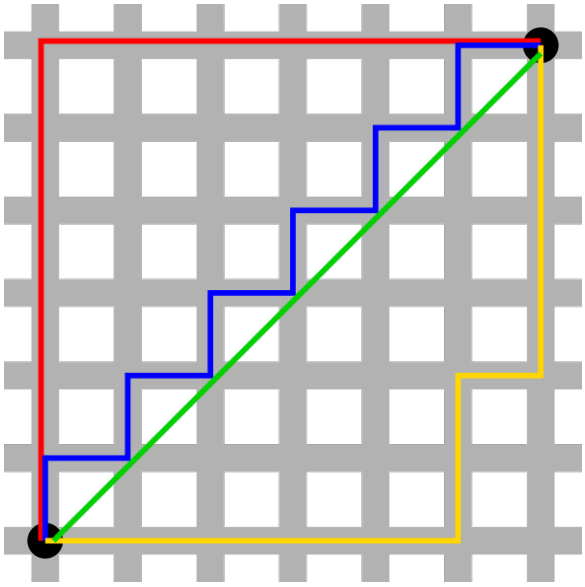
Pro každou třídu klasifikace je vypočítána vlastní střední hodnota a rozptyl. Při výpočtu posteriory pravděpodobností se poté použije μ a σ^2 asociované třídy klasifikace.

2.2.3 KNN

Algoritmus KNN (k nearest neighbours, k nejbližších sousedů) [18] patří do kategorie algoritmů strojového učení s učitelem. Funguje na podobném principu jako naivní Bayesův klasifikátor, liší se ale ve způsobu klasifikace.

2.2.3.1 Metriky počítání vzdálenosti

Pro počítání vzdálenosti je možné použít několik metrik. Tyto metriky se liší způsobem, jakým na elementy nahlížejí a změna metriky může pro stejná data dávat různé výsledky.



Obrázek 2.2: Vizualizace metrik. Zelená: Euklidova; Ostatní: Manhattanské; zdroj [2]

Manhattanská vzdálenost

Tato metrika byla inspirovaná čtvrtí Manhattan v americkém městě New York. Tato čtvrť je známá pravoúhlými ulicemi a vlastností, že cesta z bodu A do bodu B je stejně dlouhá, ať zvolíme jakoukoliv cestu.

Manhattanská vzdálenost je definována následovně

$$M(x, y) = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n|$$

Euklidova vzdálenost

Euklidova vzdálenost měří nejkratší vzdálenost v Euklidově prostoru mezi dvěma body. V dvoudimenzionálním prostoru je to také známé pod jménem cesta přímou čarou.

Nechť x a y jsou dva body v n-dimenzionálním prostoru, potom Euklidova vzdálenost x a y se počítá pomocí Pythagorovy věty jako

$$E(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

Tento způsob funguje pro spojité hodnoty atributů. Pro počítání Euklidovy vzdálenosti pro vektory obsahující spojité i nominální atributy, lze zkombinovat Euklidovu a Hammingovu vzdálenost. Tuto kombinovanou metriku lze vyjádřit následovně:

$$HE(x,y) = \sqrt{\sum_{i=1}^{n} f(x_i, y_i)}$$

$$f(a,b) = \begin{cases} 1 & \text{pokud } a,b \text{ jsou nominální} \wedge a \neq b \\ (a-b)^2 & \text{pokud } a,b \text{ jsou spojité} \\ 0 & \text{jinak} \end{cases}$$

Kosinová vzdálenost

Kosinová vzdálenost měří úhel mezi dvěma vektory v prostoru $\mathbb{R}^n$. Tato metrika je odvozena ze vzorce pro skalární součin dvou vektorů v Euklidově prostoru

$$\langle x, y \rangle = \|x\| \cdot \|y\| \cdot \cos\theta$$

jako

$$C(x,y) = \cos\theta = \frac{\langle x, y \rangle}{\|x\| \cdot \|y\|} = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2} \cdot \sqrt{\sum_{i=1}^{n} (y_i)^2}}$$

Kosinová vzdálenost se liší od ostatních v tom, že výsledky jsou v intervalu $\langle 0, 1 \rangle$. Hodnota 1 značí identické vektory, 0 úplné opaky. Proto je potřeba při klasifikaci hledat sousedy nejblíže hodnotě 1, místo sousedy s nejkratší vzdáleností.

2.2.3.2 Klasifikace

KNN při klasifikaci elementu hledá jeho k nejbližších sousedů a dívá se na jejich třídy. Vzdálenost mezi elementy lze ovlivňovat použitou metrikou. KNN operuje nad vektory z prostoru $\mathbb{R}^n$, v případě zkracování textu se jedná o vektory atributů věty.

Po nalezení těchto nejbližších sousedů, se spočítá počet těchto sousedů náležící jednotlivých třídám. Element je poté klasifikován třídou, do které patří nejvíce jeho nejbližších sousedů.

Kapitola 3

Realizace

Pro práci byl zvolen programovací jazyk Java, verze 1.8. Dále bylo použito několik knihoven pro zpracování přirozeného jazyka, konkrétně Snowball Stemmer [9], databáze slovních druhů StanfordPOS [19] a WordNet Grid [20] a lemmatizér LemmaGen [10]. Kromě těchto knihoven byla použita i knihovna Efficient Java Matrix Library (EJML) [21] pro SVD.

Účelem bylo vytvořit jednoduché rozhraní, které lze zakomponovat do již existující aplikace bez velkých problémů. Toho bylo docíleno pomocí procesu oddělení zodpovědnosti (anglicky Separation of Concers - SoC), kde modul poskytuje jen to nejnutnější rozhraní pro vykonávání své funkce.

Každý ze čtyř algoritmů je navržen jako jedna třída implementující rozhraní `Summarizer`, které poskytuje pouze jednu metodu, a to pro zkrácení textu. Parametry této metody jsou vstupní text a nastavení algoritmu. Algoritmy k-NN a naivní Bayes dále implementují rozhraní `TrainableSummarizer`, které kromě zkracování poskytuje metodu pro trénování klasifikátoru.

Jelikož algoritmy požadují různé nastavení, jsou tyto parametry obaleny třídou `SummarizationSettings`. Pro zjednodušení vytváření této třídy existuje i třída `SummarizationSettingsBuilder`, která pomocí tzv. fluent API umožňuje vytvářet a nastavovat parametry a pokud je potřeba, je inicializovat implicitní hodnotou.

3.1 Podpůrné třídy

Pro realizaci klasifikace je potřeba několika tříd společných všem algoritmům. Tyto třídy se dají dělit do několika kategorií.

3.1.1 Třídy pro zpracování přirozeného jazyka

Třída `SentenceUtils` obsahuje jednu metodu, která rozdělí text na věty. Detekce věty je složitý problém, pro který neexistuje univerzální, přesný algoritmus. Proto byl použit velice jednoduchý, naivní algoritmus, který předpokládá,

že věta začíná velkým písmenem nebo číslem a jednotlivé věty jsou odděleny tečkou, vykřičníkem nebo otazníkem následované mezerou.

Snowball stemmer má pro každý jazyk jednu třídu, proto je vytváření stemmeru pro cílový jazyk řešeno přes volání reflection API programovacího jazyka Java. Tento způsob ale není jednoduchý na používání, proto byla vytvořena třída `StemmerFactory`, která používá factory návrhový vzor. Tímto je vytváření stemmeru zjednodušeno a není potřeba řešit výjimky, které můžou vzniknout kvůli reflection API.

Pro POS tagování jsou použité dva taggery. Anglický tagger je databáze slov z americké univerzity Stanford [19]. Pro češtinu byl vytvořen jednoduchý tagger, pro který byla data získána z WordNet Grid [20]. Tato data obsahují jen pár tisíc slov, a proto jsou uloženy jejich lemmatizované varianty. Při vyhledávání slovního druhu slova je slovo nejdříve lemmatizováno, čímž je zvýšeno pokrytí taggeru. Způsob použití těchto taggerů se liší, proto byla vytvořena třída `POSTagger`, která obaluje tyto taggery a poskytuje jednoduše použitelné rozhraní k otaggování textů.

Tyto tagovací databáze obsahují velké množství dat. Třída `POSTaggerFactory` slouží k vytváření těchto taggerů a zároveň tyto vytvořené taggery uchovává v paměti. Tím se zmenší počet přístupů na disk, neboť jednotlivé instance tagerů jsou recyklovány.

Jelikož algoritmy pro strojové učení potřebují slovníky, třída `WordDatabases` v sobě obsahuje statické kolekce těchto slovníků. Jedná se o slovníky stop words, číslovek, řeckých písmen a fyzikálních jednotek. Slovníky byly vytvořeny ručně a existují varianty pro češtinu a angličtinu.

3.1.2 Třídy reprezentace věty

Každá věta je reprezentována jednou instancí třídy, jejíž typ záleží na použitém algoritmu. Algoritmus TextRank používá třídu `TextRankSentence`, která kromě textu věty obsahuje i její skóre, podle kterého se v poslední fázi algoritmu rozhoduje, zda-li věta patří nebo nepatří do zkrácené množiny.

Algoritmy KNN a naivní Bayes používají třídu `ClassificationSentence`. Kromě textu věty obsahuje hodnoty atributů pro klasifikaci, definované konstanty pro nominální atributy a sadu metod pro extrakci atributů z textu věty.

Algoritmus LSA nepotřebuje speciální reprezentaci věty, proto si je ukládá jako seznam objektů typu `String`.

3.1.3 Třídy pro strojové učení

Algoritmy KNN a naivní Bayes mají společnou část předzpracování textu. S použitím principu DRY (z anglického Don't Repeat Yourself) [22], byla proto navržena sada univerzálních tříd, které se o toto předzpracování starají.

Třída `ClassificationPreprocessor` obsahuje pouze statické metody, které slouží k předzpracování dat pro algoritmy strojového učení. Tyto metody dokáží vstupní text rozdělit na odstavce a následně do vět. Pro každou větu je vytvořena instance `ClassificationSentence`, jsou extrahovány její atributy a nastaveny příznaky. Z těchto metod se vrací seznam `ClassificationSentence` objektů, se kterými volající naloží podle potřeby. Tato třída je použitelná jak pro předzpracování dat pro trénování, tak pro klasifikaci.

Další důležitou třídou je `ClassificationUtils`, která obsahuje především matematické metody. Jedná se o třídu, jejíž účelem je počítat střední hodnotu a rozptyl atributů množiny vět. Kromě statistických metod také poskytuje metodu pro normalizaci atributů vět pomocí soft-max normalizace.

3.2 Realizace algoritmů

Každý algoritmus má svoji funkci zabudovanou ve své třídě. Tyto třídy používají třídy z předchozí kapitoly, ale i třídy specifické jednomu algoritmu.

3.2.1 TextRank

Třída `TextRank` slouží ke zkracování textu pomocí algoritmu TextRank. Tato třída implementuje výše zmíněné rozhraní `Summarizer`, kterým poskytuje dvě veřejné metody.

Stavebním prvkem této třídy je třída `Graph`, která reprezentuje ohodnocený graf. Veškeré operace, které provádí TextRank nějak modifikují nebo používají instanci této třídy. `Graph` má jako vrcholy objekty typu `TextRankSentence` a mezi nimi uchovává relaci určující váhu mezi dvěma vrcholy. Interní reprezentace tohoto grafu je pomocí `HashMap` objektu, kde klíčem je objekt `Edge` a hodnotou je váha hrany jako typ `Double`.

`Edge` má dva parametry typu `TextRankSentence`. Díky přetíženému `hashCode()` a `equals()` je možné z grafu získat informace o váze mezi dvěma vrcholy pomocí vytvoření nového dočasného objektu `Edge`, kterému jsou předané dva `TextRankSentence` objekty. To má minimální dopad na výkon, neboť JVM je optimalizované na vytváření dočasných POJO objektů.

Kromě reprezentace grafu podporuje `Graph` i metody pro získávání informací grafu, které jsou potřebné při průběhu TextRanku. Jedná se především o metodu pro získání množiny sousedních uzlů pro zadaný uzel.

Nedílnou součástí je také třída `SentenceComparator`, která provádí porovnávání vět na základě zvolených parametrů. Kromě naivního porovnávání umí používat stemmer nebo eliminovat stop words v porovnávaných větách a pracovat s jejich transformacemi.

3.2.2 LSA

Jelikož je LSA velice jednoduchý algoritmus, stačí pro jeho funkčnost pouze jedna třída, a to `LSASumarizer`. Tato třída je bezstavová, stejně jako `TextRank` a kromě `SentenceUtils` nepoužívá žádné jiné třídy a knihovny mimo EJML [21] a JDK.

Při zavolání metody `summarize()` je vytvořen `SimpleMatrix` objekt z EJML. Této matici jsou nastaveny sloupce váženými vektory frekvence termů a následně je nad ní provedeno SVD, které je součástí knihovny EJML. Jelikož LSA umožňuje zvolit poměr počtu vět zkráceného textu k vstupnímu, jsou následně extrahovány věty, dokud není dosaženo požadovaného množství.

3.2.3 Naivní Bayes

Naivní Bayes je implementován ve třídě `NaiveBayes`. Kromě rozhraní `Summarizer` implementuje i rozhraní `TrainableSummarizer`, které poskytuje metodu pro vložení trénovacích dat a metodu pro naučení. Metoda pro naučení začne proces extrakce hodnoty atributů ze všech doposud vložených trénovacích dat, napočítá střední hodnoty a rozptyly a odhadne pravděpodobnosti nominálních atributů.

Bayes je jediná třída, která interně používá `BigDecimal` pro reprezentaci čísel s desetinnou čárkou. Důvodem je, že násobením pravděpodobností mohou vznikat malá čísla. Jelikož implementace čísel s plovoucí desetinnou čárkou podle standardu IEEE 754 jsou přesné na 7 číslic (15 v případě dvojité přesnosti), je možné, že by tyto datové typy nebyly schopné přesně reprezentovat potřebné pravděpodobnosti. `BigDecimal` umožňuje reprezentovat desetinná čísla s neomezenou přesností, ale na úkor rychlosti.

Jelikož se `BigDecimal` používají pouze při klasifikaci a pro každou větu se provádí $2k$ aritmetických operací, kde k je počet atributů, je rozdíl v efektivitě `BigDecimal` a čísel v plovoucí desetinné čárce v tomto případě zanedbatelný.

3.2.4 KNN

KNN implementuje rozhraní `Summarizer` a `TrainableSummarizer`, které fungují stejně jako v naivním Bayesovi. Na rozdíl od ostatních algoritmů se jedná o jediný algoritmus, který vyžaduje do konstruktoru parametry. Jedná se o hodnotu k a metriku pro vzdálenosti mezi dvěma větami.

Metriky vzdálenosti jsou implementovány jako třídy implementující rozhraní `KNNMetric`, které poskytuje jednu metodu pro porovnání dvou vět. To umožňuje rapidní přidávání nových metrik a díky anonymním třídám v Javě je možné tyto metriky definovat pouze pro jednu konkrétní instanci KNN.

Při klasifikaci se pro každou vstupní větu počítá vzdálenost k větám v trénovací množině. Tyto vzdálenosti se ukládají jako pár věta-vzdálenost do seznamu. Tento seznam je poté seřazen podle vzdálenosti (se speciální rutinou pro kosinovou vzdálenost) od nejmenší po největší. Následně je spočítáno,

kolik vět v prvních k prvcích seznamu patří do množiny zkráceného textu. Pro klasifikaci elementu, jako že do této množiny patří, je potřeba více jak 50 % sousedů klasifikovaných stejně.

3.3 Realizace evaluace

Pro zjednodušení spouštění evaluací byl vytvořen speciální testovací framework. Tento framework obsahuje 8 metod a každá je přiřazena jedné kombinaci algoritmus - jazyk. Tyto metody slouží pro inicializaci kombinací specifických nastavení algoritmů a jsou způsobilé k vytvoření vláken pro každý test.

Dále byly vytvořeny 4 třídy, každá pro jeden algoritmus. Tyto třídy provádějí jednotlivé evaluace podle způsobu popsaného v 4.3. Kvůli potřebě logovat různé parametry v každé evaluaci, vytváří tyto třídy také reporty o proběhnutém testu, každý upravený podle potřeb algoritmu.

Kvůli časové náročnosti jsou testy v rámci jedné metody frameworku spouštěny paralelně. Každému testu je přiřazeno jedno vlákno, najednou však běží maximálně 6 vláken. Třídy v testovacím prostředí jsou chráněny klíčovým slovem `synchronized`, které zaručuje, aby k nim mělo výhradní přístup pouze jedno vlákno v jeden okamžik.

I přes paralelní evaluaci trvalo testování na procesoru Intel Core i5-2500k přes 4 hodiny. Nejdelší testy jsou testy algoritmu KNN v angličtině, které samy o sobě zaberou přes 3,5 hodiny.

Kapitola 4

Evaluace

Tato kapitola se věnuje evaluací jednotlivých algoritmů. Cílem evaluace je zjistit, jak jsou výstupy algoritmů pro automatizované zkracování textu přesné vůči zkráceným textům vytvořených člověkem.

Algoritmy jsou testovány s různými nastaveními na datech z jednoho zdroje. K porovnání algoritmů jsou použity 3 metriky ze souboru metrik ROUGE [23], která se mimo evaluace algoritmů zkracování textu používá i pro evaluaci strojových překladů.

4.1 Testovací data

Pro testování byly použity Multilingual summary evaluation data z European Commission - Joint Research Centre [24]. Jedná se o 20 dokumentů rozdělených do čtyř témat. Každý dokument je dostupný v 7 světových jazycích (včetně češtiny a angličtiny) a obsahuje v průměru 40 vět a 4 ručně vytvořené zkrácené reference.

Tato data jsou uložena ve formátu XML a témata jsou organizována podle jazyka do souborů. Témata dokumentů jsou články o malárii, genetice, konfliktu Izraele a Palestiny a o vědě a společnosti. Každá věta dokumentu je vložená do vlastního XML elementu, čímž odpadá nutnost detekovat věty a zlepšuje tedy přesnost zkracování. První dvě věty jsou rezervované pro nadpis a datum vytvoření článku a jelikož ani jedno nejsou relevantní informace k obsahu článku, jsou tyto věty ignorovány.

Věty patřící do reference mají XML atribut `annotators`. Hodnota tohoto atributu určuje anonymizované označení člověka, který tuto větu označil. Tito lidé jsou označeni písmeny A, B, C a D a od každého je označeno 20 - 30 % vět v každém dokumentu. Algoritmy TextRank a LSA, u kterých lze přímo ovlivňovat kolik vět patří do výsledku, mají proto nastaveno zkrácení na 30 %.

```
<document did="israel1" url="http://www.project-
    syndicate.org/commentary/fischer35/English">
<s sid="1">Palestinian Hopes for Barack Obama</s>
<s sid="2">2008-11-14</s>
<s sid="3" annotators="B">RAMALLAH - President-elect
    Barack Obama's defiantly positive campaign for change
     has inspired hope not only in the millions of
    Americans who voted for him, but also in the billions
     of others worldwide who could not.</s>
<s sid="4" annotators="B C">Across the Middle East, as
    elsewhere, expectations are building that his
    presidency will herald a new era for America's role
    in the world.</s>
<s sid="5" annotators="A C">Palestinians identify
    strongly with the civil rights movement in the United
     States.</s>
<s sid="6">Many recall the dark days when American
    society enforced racial segregation.</s>
<s sid="7">That the same society elected an African-
    American president only a few decades later renews
    Palestinian hopes that, in our ongoing struggle for
    justice and freedom, we, too, shall overcome.</s>
...
</document>
```

Obrázek 4.1: Ukázka struktury testovacího dokumentu

4.2 ROUGE

K porovnávání algoritmů se používá metrika ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [23]. Jedná se o sadu metrik, které porovnávají výstup algoritmu s referenčním výstupem, na základě několika parametrů a nastavení. Tyto metriky lze dle potřeb upravit na porovnáváni specifických dat.

4.2.1 ROUGE-N

Metrika ROUGE-N je založená na porovnávání n-gramů, které můžeme chápat jako sekvenci n slov v dokumentu. ROUGE-N je definováno jako

$$ROUGE - N(r, c) = \frac{|\{gram_n | gram_n \in c \wedge gram_n \in r\}|}{|\{gram_n | gram_n \in r\}|}$$

Tato metrika se dívá na počet n-gramů společných ve výstupu algoritmu c a referenci r . Toto číslo je pak normalizováno počtem n-gramů v referenci, čímž dostaneme číslo z intervalu $\langle 0; 1 \rangle$.

V případě, že máme k jednomu vstupnímu textu více referencí, bere se maximální ROUGE-N skóre všech párů výstupního textu a referencí.

$$ROUGE - N_{multi}(R, c) = argmax_i \, ROUGE - N(R_i, c)$$

4.2.2 ROUGE-LCS

ROUGE-LCS (longest common subsequence) je metrika pro porovnávání podle nejdelší společné subsekvence. Sekvence $Z = [z_1, z_2, \ldots, z_n]$ je subsekvecí jiné sekvence $X = [x_1, x_2, \ldots, x_m]$, pokud existuje rostoucí sekvence indexů $I = [i_1, i_2, \ldots, i_k]$ sekvence X, taková, že $\forall j \in 1 \ldots k$ platí $x_{i_j} = z_j$ [25]. LCS dvou sekvencí je subsekvece s největší délkou.

ROUGE-LCS dvou vět, reference r délky m a výstupu c délky n se počítá pomocí

$$ROUGE - LCS(r, c) = \frac{(1 + \beta^2) \cdot R_{lcs} \cdot P_{lcs}}{R_{lcs} + \beta^2 \cdot P_{lcs}}$$

$$R_{lcs} = \frac{LCS\,(X, Y)}{m}$$

$$P_{lcs} = \frac{LCS\,(X, Y)}{n}$$

β je vhodně zvolená konstanta, v [23] je doporučována hodnota 8.

Pro výpočet LCS je možné použít následující algoritmus založený na dynamickém programování [26].

Algorithm 1 Longest common subsequence

```
1:  function LCS(X, Y)
2:      m ← X.length
3:      n ← Y.length
4:      for i ∈ 0...m do
5:          c(i, 0) ← 0
6:      for i ∈ 0...n do
7:          c(0, i) ← 0
8:      for i ∈ 1...m do
9:          for i ∈ 1...n do
10:             if X(i) = Y(j) then
11:                 c(i, j) ← c(i − 1, j − 1) + 1
12:             else
13:                 c(i, j) ← max(c(i, j − 1), c(i − 1, j))
        return c(m, n)
```

V případě, že máme více referenčních textů, provádí se jackknifing [27], který je podobný křížové validaci.

Algorithm 2 Jackknifing

```
1: function JackKnife(referenceDocuments, candidateDocument)
2:     for leftOut ∈ referenceDocuments do
3:         D ← referenceDocuments − leftOut
4:         max ← argmax_{d∈D}(ROUGE − LCS(d, candidateDocument))
5:         accumulator ← accumulator + max
       return accumulator ÷ referenceDocuments.length
```

4.2.3 ROUGE-WLCS

ROUGE-WLCS (weighted longest common subsequence) se na rozdíl od LCS dívá i na to, aby subsekvence obsahovala co nejvíce prvků ihned za sebou. ROUGE-WLCS se počítá stejně jako ROUGE-LCS, rozdíl je v R_{lcs} a P_{lcs}

$$ROUGE - WLCS(r, c) = \frac{(1 + \beta^2) \cdot R_{wlcs} \cdot P_{wlcs}}{R_{wlcs} + \beta^2 \cdot P_{wlcs}}$$

$$R_{wlcs} = f^{-1}\left(\frac{WLCS\,(X, Y)}{f\,(m)}\right)$$

$$P_{wlcs} = f^{-1}\left(\frac{WLCS\,(X, Y)}{f\,(n)}\right)$$

$f\,(k)$ je vážící funkce, pro kterou musí platit že $f(x + y) > f(x) + f(y) \quad \forall x, y \in N$ a také musí existovat $f^{-1}\,(k)$. Pro tuto metriku je ideální

funkce $f(k) = k^2$, jelikož pro ní platí první vlastnost a její inverzní funkce je lehce vyjádřitelná.

V případě, že máme více referenčních dokumentů, se používá algoritmus jackknifing 2, stejně jako v ROUGE-LCS. Pro výpočet WLCS se používá algoritmus 1 upravený o počítání vah [23].

Algorithm 3 Weighted longest common subsequence

```
1:  function WLCS(X, Y)
2:      m ← X.length
3:      n ← Y.length
4:      for i ∈ 0...m do
5:          c(i, 0) ← 0
6:          w(i, 0) ← 0
7:      for i ∈ 0...n do
8:          c(0, i) ← 0
9:          w(0, i) ← 0
10:     for i ∈ 1...m do
11:         for i ∈ 1...n do
12:             if X(i) = Y(j) then
13:                 k ← w(i − 1, j − 1)
14:                 c(i, j) ← c(i − 1, j − 1) + f(k + 1) − f(k)
15:                 w(i, j) ← k + 1
16:             else
17:                 c(i, j) ← max(c(i, j − 1), c(i − 1, j))
18:                 w(i, j) ← 0
          return c(m, n)
```

kde $f(k)$ je výše zmíněná vážící funkce.

4.2.4 ROUGE-S a ROUGE-SU

Tyto metody jsou zaměřené na tzv. skip-bigramy. Jedná se o páry slov, které se vyskytují ve větě, ale může mezi nimi být libovolné množství jiných slov. Tato metrika se používá pro abstraktní zkracování a v algoritmech pro strojový překlad, proto nebyla v této práci použita.

4.3 Způsob evaluace algoritmů

Jelikož způsob práce obecných algoritmů a algoritmů založených na strojovém učení se liší, jsou tyto dvě varianty testovány odlišně. Kromě způsobu práce se také liší v počtu měřených nastavení a počtu měřitelných výstupů.

4.3.1 TextRank a LSA

Testování algoritmů TextRank a LSA probíhalo naivním způsobem. Pro každý dokument v testovacích datech se provedlo zkrácení cílovým algoritmem a výsledek se porovnal se všemi referencemi daného dokumentu. Každý algoritmus byl testován s 8 nastaveními. Jedná se o kombinaci stemmingu, stop words a jazyka. Pro oba byla nastavena velikost výstupu na 30 % vstupu.

Všechny kombinace nastavení testování byly testovány nad všemi dokumenty v testovací množině, vždy podle příslušného jazyka. U každého testu se měřila doba potřebná ke zkrácení v milisekundách a hodnoty ROUGE-N (s $N = 1$), ROUGE-LCS a ROUGE-WLCS. Hodnoty ROUGE byly po provedení zkrácení všech dokumentů zprůměrovány a tyto průměry brány jako výsledek.

4.3.2 KNN a naivní Bayes

Pro algoritmy KNN a naivní Bayes je potřeba provést nejdříve naučení algoritmů a až poté je možné zkracovat texty. Proto byla použita metoda křížové validace, která pro každý dokument z testovací množiny naučí příslušný algoritmus zbytkem testovací množiny a následně ho zkrátí. Pro naučení je potřeba kromě původního textu ještě referenční zkrácený text a jelikož každý dokument má čtyři reference, byla použita vždy reference od člověka, na jehož označení bylo při parsování naraženo první. V případě ukázky 4.1 by se jednalo o referenci "B".

Algorithm 4 křížová validace

```
1: function XValidation(algorithm, testData)
2:     for document ∈ testData do
3:         trainingData ← testData − document
4:         algorithm.train(trainingData)
5:         summary ← algorithm.summarize(document)
6:         performances.add(Rouge.eval(document.references, summary))
       return performances.averagePerformance()
```

V každém kole křížové validace se měřilo ROUGE-N (s $N = 1$), ROUGE-LCS a ROUGE-WLCS. U naivního Bayese se testovala normalizace a jazyk. Pro KNN se kromě normalizace a jazyka dále testovalo k od 1 do 20 a metrika měření vzdálenosti. Naivní Bayes má celkem 4 testy, zatímco KNN 240 kombinací testů.

Výstupem těchto testů byl čas v milisekundách a průměrná hodnota ROUGE ze všech kol X-Validace.

4.4 Výsledky

Po provedení všech testů byly výsledky interpretovány a zhodnoceny. Jako hlavní porovnávací kritérium se brala hodnota metriky ROUGE-N s $N = 1$. Tato metrika nejlépe vystihuje podobnost dvou textů a je odolná na rozdílné délky. Kromě této metriky se zohledňoval i průměrný čas potřebný ke zkrácení jednoho dokumentu.

4.4.1 Výsledky obecných algoritmů

TextRank CZ				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	průměrný čas [s]
default	0.738	0.635	0.401	0.35
ST	0.746	0.643	0.418	0.6
SW	0.736	0.624	0.404	0.35
ST SW	0.735	0.628	0.406	0.45
TextRank EN				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	průměrný čas [s]
default	0.822	0.672	0.402	0.9
ST	0.845	0.706	0.427	1
SW	0.795	0.627	0.376	0.1
ST SW	0.816	0.654	0.425	0.2
LSA CZ				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	průměrný čas [s]
default	0.55	0.401	0.269	0.1
ST	0.511	0.3644	0.23	0.65
SW	0.546	0.404	0.265	0.1
ST SW	0.504	0.356	0.242	0.6
LSA EN				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	průměrný čas [s]
default	0.712	0.506	0.304	0.1
ST	0.636	0.4	0.238	0.75
SW	0.637	0.414	0.25	0.05
ST SW	0.659	0.431	0.26	0.65
default: bez stemmingu a stop words, ST: stemming, SW: stop words				

Tabulka 4.1: Porovnání obecných algoritmů

Stop words výsledky neovlivňují a v některých případech i zhoršují. Stemming pro TextRank mírně zlepší výsledky, naopak pro LSA zhorší. Důvodem

tohoto fenoménu je, že v algoritmu TextRank stemmingem zlepšíme přesnost porovnávání dvou vět, naopak v LSA zmenšujeme počet řádku matice, čímž dosahujeme menší přesnosti hodnocení vět po provedení SVD.

Angličtina dosahuje v rámci algoritmů lepších výsledků oproti češtině. To je nejspíše způsobeno dvěma důvody. Angličtina skloňuje méně než čeština a zároveň má menší slovní zásobu. Tím je možné přesněji vyjádřit podobnost dvou anglických vět. Tento nedostatek pro češtinu je možné řešit stemmerem, nicméně kvůli komplexnosti češtiny je český stemmer méně přesnější, než anglický.

Co se týče výpočetního času, použití stemmingu zpomaluje výpočetní čas algoritmu LSA cca 6x. Při měření algoritmu TextRank byla relaxovaná podmínka počtu iterací konvergence skóre z původních 30 na 3000. Proto jsou výsledky času algoritmu TextRank nepřesné, neboť některé dokumenty potřebovaly pár desítek iterací, zatímco některé několik tisíc.

4.4.2 Výsledky naivního Bayese

Naive Bayes CZ				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	průměrný čas [s]
default	0.494	0.387	0.246	0.8
norm	0.585	0.47	0.29	0.8
Naive Bayes EN				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	průměrný čas [s]
default	0.895	0.791	0.381	13.45
norm	0.958	0.892	0.4	13.5
default: bez normalizacce, norm: normalizace				

Tabulka 4.2: Porovnání naivního Bayese

Naivní Bayes s normalizací dosahuje pro angličtinu téměř perfektních výsledků. Jelikož se jedná o průměr ze všech kol křížové validace, není výsledek 1, avšak více jak polovina kol křížové validace anglického Bayese s normalizací hodnotu 1 skutečně dosahovala.

Čeština dosahuje, stejně jako u obecných algoritmů, horších přesností než angličtina. Normalizace zlepšuje výsledky cca o 15 %. Jedním z důvodů špatného výsledku češtiny je fakt, že POS tagger pro češtinu obsahuje pouze pár tisíc slov, zatímco pro angličtinu je mnohem rozsáhlejší a přesnější. Jelikož atributy s počtem slovních druhů patří k nejvýznamnějším, dokáže anglický tagger označit více slov.

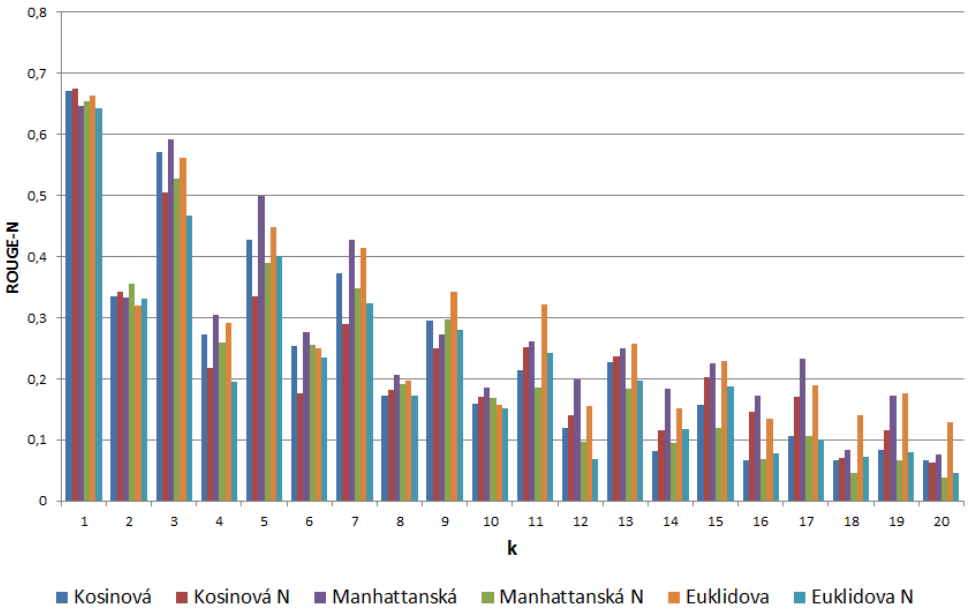
Časy zahrnují i čas potřebný k natrénování klasifikátoru trénovací množinou. Tento čas lze interpretovat jako průměrný čas jednoho kola křížové validace.

Důvod velkého rozdílů ve výpočetním čase mezi jazyky je čas strávený hledáním slovního druhu slova v anglickém slovníku. Jenom pro představu, anglický slovník na disku zabírá přes 15 MB, český 1,5 kB.

4.4.3 Výsledky KNN

Pro KNN bylo měřeno k od 1 do 20 a pro každé k i všechny kombinace normalizace, metriky a jazyka.

4.4.3.1 Angličtina



Obrázek 4.2: Porovnání hodnoty k vůči ROUGE-N pro anglický jazyk.

Jako přesnost byla použita hodnota metriky ROUGE-N s $N = 1$.

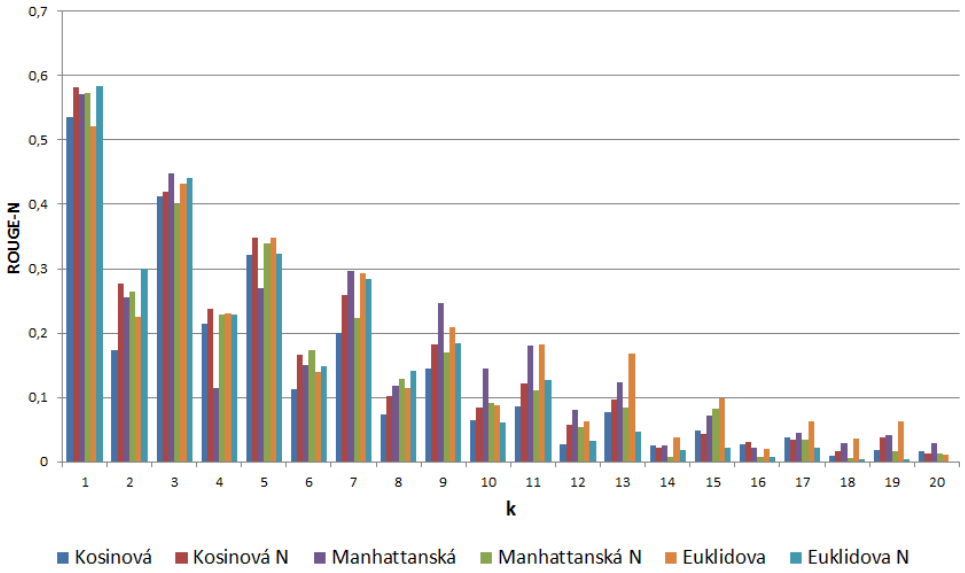
Z grafu lze vyčíst, že optimální k je 1. Je možné pozorovat fenomén sudých k, které dosahují horších výsledků, než liché k. Důvodem je klasifikace do dvou tříd a tedy pro lichá k nemůže nastat situace, kdy máme stejný počet nejbližších sousedů v každé třídě.

Použitá metrika má pro optimální k velmi malý vliv, ale pro většinu lichých k nejlépe vychází Manhattanská nebo Euklidova. Normalizace typicky zhor-

šuje výsledky klasifikace, nicméně v pár konkrétních případech může výsledek vylepšit.

Nejlepších výsledků KNN dosahuje s parametry $k = 1$, metrika = kosinová, normalizace = ano.

4.4.3.2 Čeština



Obrázek 4.3: Porovnání hodnoty k vůči ROUGE-N pro český jazyk

Opět byla použita metrika ROUGE-N pro měření přesnosti.

Čeština pokulhává za angličtinou. Stejně jako u naivního Bayese je důvodem málo rozsáhlá databáze slovních druhů a obecně větší komplexnost češtiny.

Výsledky, co se týče parametrů k, následují stejnou křivku jako pro angličtinu. Liché k suverénně překonávají hodnoty sudých k. Pro $k > 13$ jsou výsledky o dost horší v porovnání s angličtinou.

Naopak nejlepší metrika pro optimální k se ukázala být Euklidova s normalizací. Normalizace dosahuje většinou horších výsledků než klasifikace bez normalizace.

4.4.4 Porovnání a zhodnocení

V následující části jsou porovnávány vždy algoritmy s jejich nejlepším nastavením pro daný jazyk. Jako hlavní porovnávací kritérium je braná metrika ROUGE-N.

4.4.4.1 Čeština

Čeština				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	čas [s]
TextRank	0.746	0.643	0.418	0.6
LSA	0.55	0.401	0.269	0.1
Bayes	0.585	0.47	0.29	0.8
KNN	0.584	0.45	0.315	1.25

Tabulka 4.3: Porovnání nejlepších variant algoritmů pro český jazyk.

Nastavení algoritmů

- TextRank - stemming: ano, stop words: ne, poměr: 0.3
- LSA - stemming: ne, stop words: ne, poměr: 0.3
- Bayes - normalizace: ano
- KNN - k: 1, normalizace: ano, metrika vzdálenosti: Euklidova

4.4.4.2 Angličtina

Angličtina				
	ROUGE-N	ROUGE-LCS	ROUGE-WLCS	čas [s]
TextRank	0.802	0.635	0.389	2.9
LSA	0.712	0.506	0.304	0.1
Bayes	0.958	0.892	0.4	13.5
KNN	0.674	0.46	0.291	11.4

Tabulka 4.4: Porovnání nejlepších variant algoritmů pro anglický jazyk.

Nastavení algoritmů

- TextRank - stemming: ano, stop words: ne, poměr : 0.3
- LSA - stemming: ne, stop words: ne, poměr : 0.3
- Bayes - normalizace: ano
- KNN - k: 1, normalizace: ano, metrika vzdálenosti: Kosinová

4.4.5 Zhodnocení

Dosažené výsledky pro češtinu pokulhávají za angličtinou, ale i přesto dosahuje algoritmus TextRank přesnosti téměř 75 %. Ostatní algoritmy nedopadly tak dobře, ale každý z nich má přesnost alespoň 50 %.

Angličtina, díky klasifikátoru naivního Bayese, dosahuje velice dobrých výsledků. Tento klasifikátor o hodně převyšuje ostatní algoritmy a při použití normalizace s dostatečně velkou trénovací množinou dosahuje přesnosti téměř 96 %. Zbylé algoritmy dosahují horších výsledků, ale v porovnání s češtinou stále lepších. Algoritmus TextRank dosahuje přesnosti pro angličtinu přes 80 %, což je výsledek podobný češtině.

Pro obecné zkracování vychází, a to pro oba jazyky, nejlépe algoritmus TextRank se stemmingem. Dosahuje velkých přesností s poměrně rozumnou časovou náročností. Proto je tento algoritmus doporučován jako nejlepší algoritmus obecného zkracování pro oba jazyky.

Pokud jsou k dispozici trénovací data pro angličtinu, je vhodné použít naivní Bayesův klasifikátor. Tento klasifikátor, se zapnutou normalizací, dosahuje téměř perfektních výsledků. V několika kolech křížové validace tohoto klasifikátoru dosahovalo ROUGE-N skóre 1, tedy výstup byl identický se vstupem.

I kdyby byla trénovací data pro češtinu k dispozici, je stále vhodnější použít algoritmus TextRank.

Závěr

Cílem této práce bylo prozkoumat a implementovat několik algoritmů pro automatizované zkracování textu a následně je porovnat pro angličtinu a češtinu. Byla provedena rešerše algoritmů pro zkracování, včetně existujících řešení. Kromě těchto algoritmů byly také prozkoumány algoritmy a metody pro zpracování přirozeného jazyka počítačem.

Byly implementovány 4 algoritmy, TextRank, latentní sémantická analýza, naivní Bayesův klasifikátor a KNN. Pro tyto algoritmy bylo vytvořeno jednoduché rozhraní v programovacím jazyku Java, jehož cílem bylo snadné použití a případné začlenění do již existující aplikace. Kromě těchto algoritmů byla implementována také evaluační sada metrik ROUGE určená pro měření výsledků a porovnávání jednotlivých algoritmů a jejich nastavení.

V závěru práce proběhlo měření a testování jednotlivých algoritmů. Každý algoritmus byl změřen pro všechny kombinace validního nastavení a byly identifikovány silné a slabé stránky jednotlivých algoritmů. Při testování bylo potvrzeno, že automatizované zkracování textu má smysl, a že pro češtinu a angličtinu lze dosáhnout velice dobrých a přesných výsledků.

Tato práce nabízí možnosti rozšíření, neboť byla programována pomocí SOLID principu softwarového inženýrství, což umožňuje přidávat nové nebo rozšiřovat stávajících algoritmy bez velkých problémů.

Literatura

[1] Mihalcea, R.; Tarau, P.: TextRank. In *Proceedings of EMNLP-04 and the 2004 Conference on Empirical Methods in Natural Language Processing*, July.

[2] Manhattan Distance. 2006, cit. 26.4.2015. Dostupné z: `http://en.wiktionary.org/wiki/Manhattan_distance/media/File:Manhattan_distance.svg`

[3] Nenkova, A.; McKeown, K.: A Survey of Text Summarization Techniques. In *Mining Text Data*, editace C. C. Aggarwal; C. Zhai, Springer US, 2012, ISBN 978-1-4614-3222-7, s. 43–76, doi:10.1007/978-1-4614-3223-4_3. Dostupné z: `http://dx.doi.org/10.1007/978-1-4614-3223-4_3`

[4] Gong, Y.; Liu, X.: Generic text summarization using relevance measure and latent semantic analysis. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2001, s. 19–25.

[5] Radev, D. R.; Jing, H.; Stys, M.; aj.: Centroid-based Summarization of Multiple Documents. *Inf. Process. Manage.*, ročník 40, č. 6, Listopad 2004: s. 919–938, ISSN 0306-4573, doi:10.1016/j.ipm.2003.10.006. Dostupné z: `http://dx.doi.org/10.1016/j.ipm.2003.10.006`

[6] Erkan, G.: LexRank: Graph-based lexical centrality as salience in text summarization.

[7] Alguliev, R.; Aliguliyev, R.; aj.: Evolutionary algorithm for extractive text summarization. *Intelligent Information Management*, ročník 1, č. 02, 2009: str. 128.

[8] Porter, M. F.: An algorithm for suffix stripping. *Program: electronic library and information systems*, ročník 14, č. 3, 1980: s. 130–137.

[9] Porter, M.: Snowball Stemmer. 2001, cit. 21.12.2014. Dostupné z: http://snowball.tartarus.org/

[10] LemmaGen. Cit. 9.4.2015. Dostupné z: http://lemmatise.ijs.si/

[11] Miller, G. A.: WordNet: A Lexical Database for English. *Commun. ACM*, ročník 38, č. 11, Listopad 1995: s. 39–41, ISSN 0001-0782, doi:10.1145/219717.219748. Dostupné z: http://doi.acm.org/10.1145/219717.219748

[12] Belica, M.: sumy. Cit. 22.4.2015. Dostupné z: https://github.com/miso-belica/sumy

[13] Essential Summarizer. Cit. 22.4.2015. Dostupné z: https://essential-mining.com/summarizer/index.jsp?ui.lang=en

[14] Summly. Cit. 22.4.2015. Dostupné z: http://summly.com/index.html

[15] Page, L.; Brin, S.; Motwani, R.; aj.: The PageRank citation ranking: Bringing order to the web. 1999.

[16] Kupiec, J.; Pedersen, J.; Chen, F.: A Trainable Document Summarizer. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, New York, NY, USA: ACM, 1995, ISBN 0-89791-714-6, s. 68–73, doi:10.1145/215206.215333. Dostupné z: http://doi.acm.org/10.1145/215206.215333

[17] Priddy, K. L.; Keller, P. E.: *Artificial neural networks: an introduction*, ročník 68. SPIE Press, 2005.

[18] K-nearest neighbor. Cit. 8.5.2015. Dostupné z: http://www.scholarpedia.org/article/K-nearest_neighbor

[19] Toutanova, K.; Klein, D.; Manning, C. D.; aj.: Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, Association for Computational Linguistics, 2003, s. 173–180.

[20] Global WordNet Grid. Cit. 3.4.2015. Dostupné z: http://deb.fi.muni.cz:9000/

[21] Efficient Java Matrix Library. Cit. 5.4.2015. Dostupné z: http://ejml.org

[22] Hunt, A.; Thomas, D.: *The Pragmatic Programmer: From Journeyman to Master*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999, ISBN 0-201-61622-X.

[23] Lin, C.-Y.: Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, 2004, s. 74–81.

[24] (JRC), E. C. J. R. C.: Multilingual summary evaluation data. 2010, cit. 10.4.2015. Dostupné z: `https://ec.europa.eu/jrc/en/language-technologies`

[25] Cormen, T. H.; Stein, C.; Rivest, R. L.; aj.: *Introduction to Algorithms*. McGraw-Hill Higher Education, druhé vydání, 2001, ISBN 0070131511.

[26] Cit. 8.5.2015. Dostupné z: `http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_subsequence`

[27] Efron, B.; Efron, B.: *The jackknife, the bootstrap and other resampling plans*, ročník 38. SIAM, 1982.

[28] Ranks NL Webmaster Tools. `http://www.ranks.nl/stopwords/`, cit. 3.4.2015.

Příloha A

Seznam použitých zkratek

API Application Programmable Interface

HTML HyperText Markup Language

KNN K Nearest Neighbours

LSA Latent Semantic Analysis

ROUGE Recall-Oriented Understudy for Gisting Evaluation

LCS Longest Commmon Subsequence

WLCS Weighted Common Longest Subsequence

SVD Singular Value Decomposition

XML eXtensible Markup Language.

DRY Don't Repeat Yourself

SoC Separation of Concerns

POS Part Of Speach

JDK Java Development Kit

JVM Java Virtual Machine

POJO Plain Old Java Object

SOLID Single responsibility, Open-closed, Liskov substitution, Interface segregation a Dependency inversion

IEEE Institute of Electrical and Electronics Engineers

Příloha **B**

Hodnoty slovníků atributů klasifikace

České a anglické stop words převzaté z [28].

Seznam českých stop words:

dnes, cz, timto, budes, budem, byli, jses, muj, svym, ta, tomto, tohle, tuto, tyto, jej, zda, proc, mate, tato, kam, tohoto, kdo, kteri, mi, nam, tom, tomuto, mit, nic, proto, kterou, byla, toho, protoze, asi, ho, nasi, napiste, re, coz, tim, takze, svych, jeji, svymi, jste, aj, tu, tedy, teto , bylo, kde, ke, prave, ji, nad, nejsou, ci, pod, tema, mezi, pres, ty, pak, vam, ani, kdyz, vsak, ne, jsem, tento, clanku, clanky, aby, jsme, pred, pta, jejich, byl, jeste, az, bez, take, pouze, prvni, vase, ktera, nas, novy, tipy, pokud, muze, design, strana, jeho, sve, jine, zpravy, nove, neni, vas, jen, podle, zde, clanek, uz, email, byt, vice, bude, jiz, nez, ktery, by, ktere, co, nebo, ten, tak, ma, pri, od, po, jsou, jak, dalsi, ale, si, ve, to, jako, za, zpet, ze, do, pro, je, na

Seznam anglických stop words:

a, about, above, after, again, against, all, am, an, and, any, are, aren't, as, at, be, because, been, before, being, below, between, both, but, by, can't, cannot, could, couldn't, did, didn't, do, does, doesn't, doing, don't, down, during, each, few, for, from, further, had, hadn't, has, hasn't, have, haven't, having, he, he'd, he'll, he's, her, here, here's, hers, herself, him, himself, his, how, how's, i, i'd, i'll, i'm, i've, if, in, into, is, isn't, it, it's, its, itself, let's, me, more, most, mustn't, my, myself, no, nor, not, of, off, on, once, only, or, other, ought, our, ours, ourselves, out, over, own, same, shan't, she, she'd, she'll, she's, should, shouldn't, so, some, such, than, that, that's, the, their, theirs, them, themselves, then, there, there's, these, they, they'd, they'll, they're, they've, this, those, through, to, too, under, until, up, very, was, wasn't, we, we'd, we'll, we're, we've, were, weren't, what, what's, when, when's, where, where's, which, while, who, who's, whom, why, why's, with, won't, would, wouldn't, you, you'd, you'll, you're, you've, your, yours, yourself, yourselves

Následující slovníky byly vytvořeny ručně.

Seznam číslovek:

one, two, three, four, five, six, seven, eight, nine, ten, jeden, dva, tři, čtyři, pět, šest, sedm, osm, devět, deset, milion, bilion, trilion, quadrilion, thousand, tisíc, milarda, miliard, biliarda, bilidard

Seznam písmen řecké abecedy:

alpha, beta, gamma, delta, epsilon, zeta, ete, theta, iota, kappa, lambda, mu, nu, xi, omicron, pi, rho, sigma, tau, upsilon, phi, chi, psi, omega, alfa, gama, zéta, éta, théta, ióta, mí, ný, ksí, omikron, pí, ró, ypsilon, fí, chí, psí

Seznam fyzikálních jednotek:

cm, g, kg, lbs, m, ft, km, miles, l, ml, gallons

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `java` ..... zdrojové kódy implementace
  - `tex` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `IvoSklenarBP.pdf` ..... text práce ve formátu PDF