VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

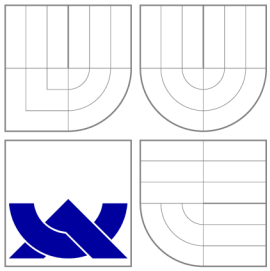
INTELIGENTNÍ ZPRACOVÁNÍ ZÁLOŽEK (BOOKMARKS)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

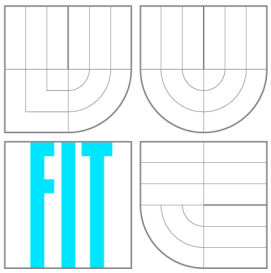
AUTOR PRÁCE
AUTHOR

MIROSLAV BRHEL

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTELIGENTNÍ ZPRACOVÁNÍ ZÁLOŽEK (BOOKMARKS)

INTELLIGENT PROCESSING OF BOOKMARKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MIROSLAV BRHEL

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2013

Abstrakt

Práce se zabývá inteligentním zpracováním záložek, zejména shlukováním internetových stránek podle podobnosti textu. Praktická část představuje návrh a realizaci sytému, který dokáže utřídit záložky a sdružit příbuzné odkazy do skupin.

Abstract

This thesis deals with intelligent bookmarks processing mainly with web pages clustering according to text similarity. As a practical part of the thesis a system which is capable of bookmarks sorting and clustering was designed.

Klíčová slova

zpracování záložek, shlukování dokumentů, k-means, podobnost textů

Keywords

processing of bookmarks, document clustering, k-means, text similarity

Citace

Miroslav Brhel: Inteligentní zpracování záložek (bookmarks), bakalářská práce, Brno, FIT VUT v Brně, 2013

Inteligentní zpracování záložek (bookmarks)

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod odborným vedením doc. RNDr. Pavla Smrže, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Brhel
15. května 2013

Poděkování

Děkuji vedoucímu práce doc. RNDr. Pavlu Smržovi, Ph.D. za vedení, užitečné rady a pomoc při řešení.

Obsah

Kapitola 1

Úvod

Lidé po celém světě mají od nepaměti potřebu získávat nové informace. S nástupem počítačů a internetu se výrazně změnily zdroje, ze kterých jsou informace čerpány. Tištěné noviny, knihy nebo časopisy jsou převáděny do digitální podoby. S tímto trendem je distribuce dat levnější, rychlejší a objemnější. Zpravodajské portály, různé instituce i samotní uživatelé publikují na celosvětové síti téměř nepřetržitě nové informace. Textová, obrazová nebo třeba hudební data neustále nabývají na objemu, a tak se uživatel vyhledávající konkrétní zprávu může někdy cítit pohlcen tokem informací.

Webové prohlížeče, které jsou používány dnes a denně, se v průběhu času vyvíjí a přidávají nové nástroje pro zobrazování dat. Uživatelé si velmi rychle oblíbili prohlížení webových stránek ve více panelech, procházení historií navštívených adres nebo ukládání těch oblíbených a zajímavých. Je velmi pravděpodobné, že počet založených stránek bude u uživatele pořád narůstat a může se mu stát, že nebude v jeho silách manuálně třídit všechny záložky do kategorií. V oblíbených položkách tak může rychle nastat chaos a vyhledávání informací se tak stane nedostupné, nebo přinejmenším nepohodlné. Tomuto problému se věnuje odvětví počítačové lingvistiky, které se zabývá zpracováním přirozeného jazyka. Pomocí výpočetní síly strojů se snaží ve velkém objemu informací nalézt určité vzory či podobnost a uživateli tak srozumitelně prezentovat obsáhlý korpus dat, například rozdělením na menší kategorie.

V práci je ukázáno, jak by se mohlo pracovat na problému třídění webových stránek a jejich shlukování do skupin s podobnou tématikou. Nejprve je popsána teorie analýzy textových dokumentů a metody užívané k shlukování bez zásahu uživatele. V teoriteckém rozboru je možno také nalézt informace o systémech vah, výpočtu podobnosti dokumentů nebo o shlukovacích algoritmech.

Další kapitola je zaměřena na návrh a realizaci systému, který dokáže sdružovat blízké odkazy do skupin. Předposlední kapitola se zabývá testováním vytvořeného programu a také výsledky, které systém dokáže produkovat. V závěrečné kapitole se nachází ohlednutí a zhodnocení celé práce.

Kapitola 2

Rozbor tématu

Před samotným návrhem programu, který bude schopný kategorizovat oblíbené webové odkazy, je potřeba porozumět problematice hledání podobnosti mezi textovými dokumenty[1]. Jelikož stroj není schopen vnímat psaný text a intuitivně poznat, zda si jsou dva dokumenty podobné, je nutné převést text do matematické podoby. S touto reprezentací dat už může počítač pracovat a určit, do jaké míry se dokumenty podobají.

2.1 Zpracování textů

Textové soubory a speciálně webové stránky zpravidla neobsahují pouze prostý text, naopak velmi často budou formátovány. Pro reprezentaci v matematické podobě však je potřeba texty zpracovat a získat základní jednotky, ze kterých jsou tvořeny. Mezi základní kroky patří odstranění interpunkce, tokenizace, stemming nebo lemmatizace. *Tokenizace* textu znamená jeho rozdělení na menší části (většinou slova). Pod anglickým termínem *stemming* je schována operace získání kořene slova a *lemmatizace* představuje vytvoření základního tvaru slova.

Při analýze kolekce textových souborů je možné pozorovat, že výskyt některých slov je nesrovnatelně vyšší v porovnání s jinými. Mezi tato slova patří také *funkční* slova, která se objevují v textu s vysokou frekvencí a díky jejich malému lexikálnímu významu jsou pro porovnání dokumentů nepodstatná. [12] Z tohoto důvodu se tato slova při zpracování textů odstraňují. Dalším adeptem k ignorování jsou slova, která se objevují v minimálním procentu dokumentů z celkové kolekce. Podobným případem jsou také slova s vysokým procentem výskytu. Ve formátovaných dokumentech se však mohou objevovat také slova s větší důležitostí. Tato slova jsou většinou zapsána v podobě nadpisů nebo jsou odlišena od obyčejných slov podtržením či jinou proporcí. Tento fakt by se měl v předzpracování textu také projevit. Vynechávání slov ovlivňuje výkon a rychlost zpracování, avšak každým zahozením slova se ztrácí informace, proto technika zpracování textu vyžaduje správné posouzení a pečlivé zvážení. Pokud je přístup k reprezentaci dokumentů špatný nebo zanedbaný, výsledky budou negativně ovlivněny a sebelepší algoritmus nepomůže k jejich zlepšení.

2.2 Reprezentace dokumentů

Aby dokumenty mohly být porovnávány strojem musí být matematicky popsány. Jeden z nejpoužívanějších modelů je *Vector space model*, který můžeme do češtiny přeložit jako

[1] Webové stránky jsou textové dokumenty, ve kterých se objevují speciální formátovací značky.

vektorový model (dále v textu pod zkratkou VSM). Tento algebraický model se díky své jednoduchosti stal velmi populární v oblasti zpracování přirozeného jazyka, a to i přes svá omezení. Popisovaný přístup ignoruje závislost mezi slovy, pořadí slov nebo syntaktickou strukturu, což vede k podstatné ztrátě informací o daném textu.

Tato technika reprezentuje jednotlivé dokumenty jako n-rozměrné vektory, které jsou tvořeny podle vstupní kolekce textových souborů, které jsou předzpracovány podle kapitoly 2.1. Podle počtu unikatních slov, které zůstaly ve vstupních datech, je určena dimenze vektoru. Způsob, jakým jsou vektory vytvořeny, je jeden ze stěžejních úkolů v celé problematice shlukování. Jak bylo zmíněno výše, některá slova v textu mají větší význam než jiná. Z tohoto důvodu byl pro VSM vytvořen systém vah.

2.3 Systémy vah

Mějme množinu dokumentů $D = \{d_1, d_2, \ldots, d_n\}$ a množinu $T = \{t_1, \ldots, t_m\}$, která obsahuje unikátní slova objevující se v D. Pro každý dokument d lze nyní vytvořit vektor

$$\vec{v_d} = (w_1, w_2, \ldots, w_m)$$

Prvky vektoru představují hodnoty w_i, které jsou vypočítány podle následujícího systému vah, jenž představil v roce 1989 ve své práci Gerard Salton. [10] Tento systém vah popisuje 3 složky, podle kterých je vypočten význam slova. Jsou to

1. *Term Frequency Component*

   Tato složka popisuje důležitost slova v daném dokumentu. Příklad nejpoužívanějších technik výpočtu:

b	1.0	binární, pokud se slovo t vyskytuje v dokumentu d váha je nastavena na 1, jinak je 0
t	tf	počet výskytů slova t v dokumentu d
n	$0.5 + 0.5\frac{tf}{\max tf}$	rozšířená (angl. augmented) váha, normalizovaná do intervalu $< 0.5, 1 >$

2. *Collection Frequency Component*

   Reprezentuje význam slova v celé kolekci dokumentů. Dva nejznámější výpočty jsou:

x	1.0	beze změny, je použita pouze první složka
f	$\log \frac{N}{n}$	převrácená četnost výskytu v kolekci dokumentů, N je počet dokumentů, n je počet dokumentů, ve kterých se vyskytuje slovo t

3. *Normalization Component*

   Poslední složka, která normalizuje vektor. Například:

x	1.0	beze změny, jsou použity první dvě složky
c	$\frac{1}{\sqrt{\sum_{vector} w_i^2}}$	kosínova normalizace, každá hodnota w je dělena Euklidovskou normou

Výsledná váha pro dané slovo je získána vynásobením všech tří složek, které byly právě popsány. Mezi nejvíce oblíbená váhová schémata patří *binární*, které je složeno ze složek „bxx“. Další je tzv. *bag-of-words* skládající se z „txx“ a také nenormalizované *tf-idf*, které je zapsáno podle Saltona jako „tfx“. Všechny tři systémy budou podrobněji popsány v následujících podkapitolách.

2.3.1 Binární vektory

V rámci tohoto přístupu jsou vektory vytvářeny ze vstupní kolekce velmi jednoduchým postupem. Podle systému vah, který publikoval Salton [10], je váha vypočtena jen podle první složky. Význam slova je tedy určen jen z faktoru důležitosti v daném dokumentu a to metodou, která je označena písmenem „b“. Ostatní dvě složky jsou rovny 1.0, při vynásobení tedy výslednou hodnotu neovlivní. Prvky výsledného vektoru $\vec{v_d} = (w_1, \ldots, w_m)$ budou nabývat pouze binárních hodnot. Pokud se bude i-té slovo vyskytovat v dokumentu d, váha w_i bude mít hodnotu 1, v opačném případě bude rovna 0.

Mezi výhody této techniky vytváření vektorů patří rychlost a jednoduchá implementace, za hlavní nevýhodu je považováno rovnocené hodnocení slov. [6]

2.3.2 Bag of words

Další metoda, podle popsané teorie v kapitole 2.3, vypočítává váhu slov ve vektorech následujícím způsobem. Pro každé slovo je spočtena jeho četnost v daném dokumentu d. Tyto hodnoty jsou potom uloženy ve vektoru $\vec{v_d}$. Tento způsob je označen písmenem „t“, zbylé dvě složky jsou označeny jako „xx“, tudíž nemají na výslednou váhu w_i žádný vliv.

Na rozdíl od *binárních vektorů* význam jednotlivých slov v dokumentu je rozlišen, ikdyž hodně naivním způsobem. Tento model je použit jako vstup pro *LDA*, což je popsáno v kapitole 2.4.1.

2.3.3 TF-IDF

Jedno z nejpoužívanějších schémat vážení slov v dokumentech se ukrývá pod anglickou zkratkou *tf-idf*. V práci [10] je váha spočtena ze všech třech složek „tfx“, pro normalizovanou verzi je to potom „tfc“. První složka je získána z frekvence výskytu slova t v dokumentu d. Druhá složka, narozdíl od předchozích dvou metod, není zanedbána, ale je počítána jako

$$idf_t = \log \frac{N}{n}$$

kde N je celkový počet dokumentů. Tato hodnota může být také vyjádřena jako $|D|$, kde D je množina všech dokumentů. Hodnota n představuje počet dokumentů, ve kterých se vyskytuje slovo t. Tento faktor se do češtiny překládá jako *převrácená četnost slova ve všech dokumentech*. Třetí složka, pro normalizaci vektoru, může nebo nemusí být použita.

Tento přístup zohledňuje různý význam daného slova. Výrazy, které se objevují ve většině dokumentů, považuje za méně podstatné. Naopak zvyšuje důležitost slov zvláštních a speciálních.

2.4 Jiný pohled na dokument

Výše zmíněná reprezentace dat se často používá, přestože má své nedostatky. Mezi hlavní patří ingorování skryté struktury v textu nebo závislosti mezi slovy. Uveďme si knižní příklad. Mějme dva dokumenty, které chceme porovnat. První bude obsahovat slovo „Barack“ a ve druhém se bude vyskytovat slovo „Obama“. Pokud budeme dané dokumenty reprezentovat pomocí VSM, jak je popsáno v kapitole 2.2, žádný algoritmus nenajde mezi těmito texty podobnost.

2.4.1 Latent Dirichlet allocation

Nastíněný problém řeší tzv. *generativní modely*. Jedním z nich je také *Latent Dirichlet allocation*. Tento model prezentoval poprvé v roce 2003 ve své práci David Blei, Andrew Ng a Michael Jordan. [2] LDA nahlíží na každý dokument jako na směs různých témat. Témata jsou reprezentována jako pravděpodobnostní rozdělení slov. Pokud chceme v množině dokumentů D identifikovat k témat, LDA vrátí pro každý dokument d vektor témat a rozdělení pravděpodobnosti slov pro každé téma. [8]

Mějme množinu dokumentů, které se zabývají zvířecí tématikou. V tomto vstupu požadujeme rozlišit dvě témata. Může například vzniknout první téma, kde slova jako *kočka*, *koťata*, *mléko* budou mít nejvyšší pravděpodobnost. Toto téma označíme jako „kočičí“. Jiné téma bude obsahovat slova s nejvyšší pravděpodobností: *štěkat*, *kost*, *štěně*. Téma bude označeno jako „psí“. LDA vrátí pro každý dokument vektor s pravděpodobnostním rozdělením témat. V uvedeném příkladu by dokument d mohl mít podobu $v_d = (0.9 * kocici, 0.1 * psi)$.

Tento přístup nejenže najde podobnosti mezi dokumenty, které neobsahují na první pohled společná slova, ale také *sníží dimenzi* výsledných vektorů, náročnost při jejich dalšímu zpracování nebude tak příliš velká.

2.5 Výpočet podobnosti dokumentů

Aby bylo možné shlukovat dokumenty patřící k sobě, je potřeba vypočítat podobnost mezi texty. Na základě této hodnoty shlukovací algoritmus rozhodne, kam bude dokument zařazen. V oboru počítačové lingvistiky se používá několik přístupů. Různé postupy se liší v náročnosti výpočtu a také samozřejmě ve výsledcích. V následujících podkapitolách je nastíněna problematika dvou známých přístupů.

2.5.1 Euklidovská vzdálenost

Euklidovská vzdálenost je standartní metrika používaná v geometrice a je definována jako vzdálenost mezi dvěma body. Její výpočet je velmi jednoduchý a lehce implementovatelný. Euklidovská vzdálenost se běžně používá v problematice shlukování, včetně klastrování textových dokumentů. Tato metrika je také standartně používaná v algoritmu *k-means*. [4]

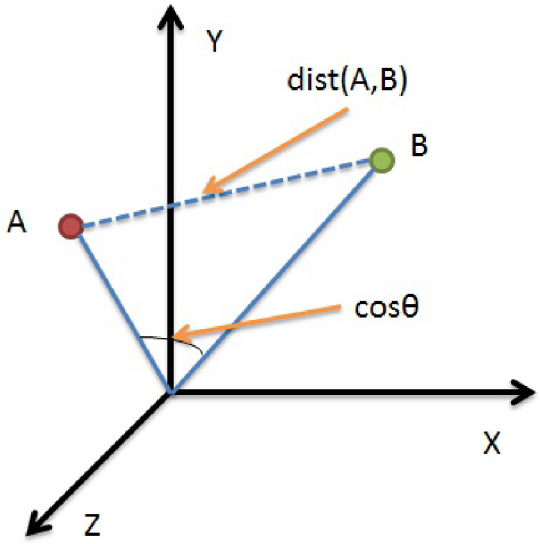
Výpočet vzdálenosti mezi textovými dokumenty d_a a d_b, které jsou reprezentovány vektory $\vec{v_a}$ a $\vec{v_b}$, je definován následovně

$$D_E(\vec{v_a}, \vec{v_b}) = \sqrt{\sum_{t=1}^{m}(w_{t,a} - w_{t,b})^2}$$

kde množina slov je $T = \{t_1, \ldots, t_m\}$. Hodnoty w jsou váhy daného slova, které jsou určeny podle zvoleného systému vah. [4]

2.5.2 Kosínová podobnost

Protože dokumenty jsou popsány pomocí vektorů, jako další metoda výpočtu podobnosti může být uplatněna Kosínová podobnost. Dva vektory mezi sebou svírají úhel a jeho kosinus lze použít pro porovnání podobnosti daných vektorů. Na obrázku 2.1 je zobrazen rozdíl mezi Euklidovskou vzdáleností $dist(A, B)$ a kosínovou podobností $\cos\theta$ v trojrozměrném prostoru.



Obrázek 2.1: Rozdíl mezi Euklidovskou vzdáleností a kosínovou podobností ve trojrozměrném prostoru [9]

Při porovnávání textových dokumentů mají vektory mnohem větší dimenzionalitu, princip výpočtu je však pořád stejný. Jelikož vektory reprezentující texty nemohou obsahovat záporné hodnoty, výsledek kosinu úhlu se nachází na intervalu $< 0, 1 >$. Výsledná hodnota podobnosti dvou shodných dokumentů (vektorů) je rovna 1.

Mějme dva dokumenty a, b a jejich vektory $\vec{v_a}$ a $\vec{v_b}$. Kosínová podobnost dokumentů se vypočítá následujícím způsobem:

$$cos\_sim(a, b) = \frac{\vec{v_a} * \vec{v_b}}{\|v_a\|\|v_b\|}$$

kde vektory mají délku m a jsou tvořeny vahami slov z množiny $T = \{t_1, \ldots, t_m\}$. [7]

2.6 Shlukovací algoritmy

Po uvedení do problému reprezentace dat v podkapitole 2.2 a výpočtu podobnosti 2.5 lze pokročit dále. Dalším stupněm je už samotné shlukování dokumentů. Shlukovací algoritmus porovnává podobnosti dokumentů z dané množiny a vytváří skupiny podobných textů. Existuje velké množství algoritmů, které se používají v oboru shlukování a dělí se do skupin podle různých kritérií. Jeden ze základních způsobů dělení je popsán na následujících podkapitolách.

2.6.1 Hierarchické algoritmy

Jak z názvu vypovídá, metody patřící do této kategorie vytváří shluky, které jsou vzájemně hierarchicky propojené. Tato skupina algoritmů se dále dělí na *aglomerativní* a *divizní*.

Z důvodu vyšší náročnosti výpočtu není tento přístup příliš vhodný pro rozsáhlá vstupní data, proto se jimi dále v práci nebudeme zabývat. Ale na rozdíl od *nehierarchických* metod kvalita výsledných shluků bývá zpravidla lepší. [5]

2.6.2 Nehierarchické algoritmy

Tyto metody vytváří pouze množinu shluků bez hierarchie. Většinou se jedná o iterativní algoritmy, které po prvním průchodu vylepšují shluky vytvořené v minulé iteraci. Implementace algoritmů patřící do této kategorie je poměrně snadná a jejich vypočet není náročný. Ušetření času se však mírně projeví na kvalitě výsledků. [5]

Mezi nejznámější nehierarchické metody patří jednoznačně algoritmus *k-středů* (k-means).

Metoda k-středů (k-means)

Algoritmus *k-means* je použit v praktické části této práce, proto se mu podrobněji věnují následující odstavce.

Tato metoda iterativně shlukuje data, v této práci konkrétně textové dokumenty. Jako vstupní informace algoritmus požaduje číslo k, které znamená počet výsledných shluků, a také množinu vektorů, které mají být roztříděny. V prvním kroku se určí množina shluků $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$, které budou sdružovat podobné vektory. V první iteraci je do každého shluku z množiny $\mathbf{S}$ vybrán náhodně jeden vektor ze vstupní množiny a je považován za střed dané skupiny. V jiné variantě metody *k-means* mohou být středy vypočteny nějakou heuristickou funkcí, aby celý algoritmus konvergoval rychleji.

Mějme n m-rozměrných vektorů reprezentující texty označené jako $x_1, x_2, \ldots, x_n$. Pro každý vektor z této vstupní množiny je vypočítána euklidovská vzdálenost[2] vzhledem k středům shluků. Podle nejmenší vzdálenosti je vstupní vektor přiřazen do dané skupiny S_i. Matematický zápis výpočtu vypadá takto

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

kde μ_i představuje střed shluku S_i. [11]

Po rozřazení všech vstupních vektorů do skupin následuje druhá část iterace, a to výpočet nových středů všech shluků. Nový střed μ_i je získán jako průměr vektorů nacházejících se ve i-té skupině. Po tomto kroku následuje nová iterace, která opět rozřazuje celý vstup do skupin podle výše uvedeného vzorce. Algoritmus je ukončen, když dvě po sobě jdoucí iterace se neliší v rozřazení vektorů do jednotlivých sluků. Některé implementace *k-means* operují s hodnotou maximálního počtu iterací, po kterých je algoritmus ukončen, nedošlo-li k jeho konvergenci. [3]

[2] Výpočet euklidovské vzdálenosti je popsán v podkapitole 2.5.1.

Kapitola 3

Návrh a realizace řešení

Součástí bakalářské práce je také vytvoření programu pro shlukování souvisejících webových stránek. Na následujících řádcích je popsán návrh a implementace této praktické části. V této kapitole jsou neprve zhodnoceny požadavky na výsledný program, dále navazuje popis jeho návrhu a rozčlěnění na jednotlivé moduly. Největší část tohoto oddílu popisuje samotnou realizaci.

3.1 Analýza požadavků na aplikaci

Zadání bakalářské práce přikazuje navrhnout a realizovat systém, který dokáže utřídit záložky a sdružit příbuzné odkazy do skupin.

Protože se jedná o třídení záložek internetových stránek, které se nacházejí na vzdálených úložištích, je potřeba tyto textové dokumenty nejdříve získat. Zadání velikost vstupních dat nijak nelimituje, mohou tedy narůst do velkých rozměrů, a proto realizace stažení stránek musí být provedena pomocí paralelního zpracování.

Webové stránky se zapisují pomocí značkovacího jazyka HTML, který formátuje text pomocí speciálních značek. Tyto elementy však nejsou pro zpracování a sémantickou analýzu dokumnetů důležité, naopak jsou nežádoucí. Druhý dílčí úkol tedy představuje odstranit značky a získat jen relevantní text vztahující se k problematice, kterou se daná webová stránka zajímá. Dokumenty jsou na celosvětové sítí uloženy pod unikátním identifikátorem, který se označuje jako URL. Tento „jednotný lokátor zdrojů“ představuje řetězec znaků s definovanou strukturou. [1] Často se v této adrese objevují klíčová slova, která výstižně charakterizují daný textový soubor. Jejich ignorování by proto nebylo šťastné řešení.

Získaná jednotlivá slova z textů je však potřeba před převodem na vektory opracovat a „pročistit“. V teoretické části 2.1 je nastíněna problematika zpracování textů. Aby porovnání dvou dokumentů bylo přesnější a lepší, je potřeba text tzv. předpřipravit na matematickou reprezentaci. Slova, která se vyskytují příliš často, nebo naopak minimálně, se hodí vynechat. Lidská řeč, narozdíl od strojového kódu, se neskládá pouze z jednoduchých příkazů. Naopak v závislosti na květnatosti daného jazyka exitují slova ve více tvarech a libovolně „ohebná“. Aby si počítač uvědomil, že se jedná o podobná slova, je mu nutné textové výrazy podat v základním tvaru nebo ve tvaru kořene slova, což je jednodušší operace. Pro tuto změnu slova se užívá anglický výraz *stemming*.

Po získání vstupních textových dat z adres následuje krok, který vytvoří shluky dokumentů, které spolu nějakým způsobem souvisí. Shlukovací algoritmy, které jsou potřeba pro řešení podobných problému, vyžadují na vstupu většinou matici vektorů, které reprezen-

tují dokumenty k rozřazení. Vypracování takových vektorů je tedy další podproblém nutný k řešení. Metody vytvářející matematickou reprezentaci dokumentů, které jsou implementované v rámci této práce, jsou teoreticky popsány v kapitole 2.2.

Analýza a návrh celého systému se v průběhu času vyvíjel, zejména tvoření vektorů. Z tohoto důvodu je v práci implementováno více verzí převodu textů na vektory. První způsob pracuje s *binárními* vektory. Ten byl dále vylepšen na model s počítáním vah pomocí schématu *tf-idf*, jak jej popisuje podkapitola 2.3.3. Poslední způsob je realizován pomocí *Latent Dirichlet allocation*.

Všechny implementované přístupy je potřeba porovnat a vyhodnotit, tento úkon završuje celou praktickou část.

3.2 Volba programovacího jazyka

Při řešení úloh pomocí počítače lze využít škálu různých programovacích jazyků a vhodně vybrat právě ten nejvíce se hodící pro daný problém. V oblasti zpracování přirozeného jazyka se využívají zápisy programů podle různých syntaktických pravidel. Mezi nejpoužívanější se řadí iterpretovaný jazyk *Python*, a to zejména díky své jednoduché syntaxi a přirozené manipulaci s textovými řetězci.

Výše zmíněná fakta a také široká škála knihoven, které jsou určeny pro zpracování jazyka, rozhodly při výběru programovacího jazyka pro tuto práci. Pro převážnou část programu je použit konkrétně *Python 2.7.3*, jedna část je napsána ve skriptovacím jazyk *bash*.

3.3 Návrh modularity

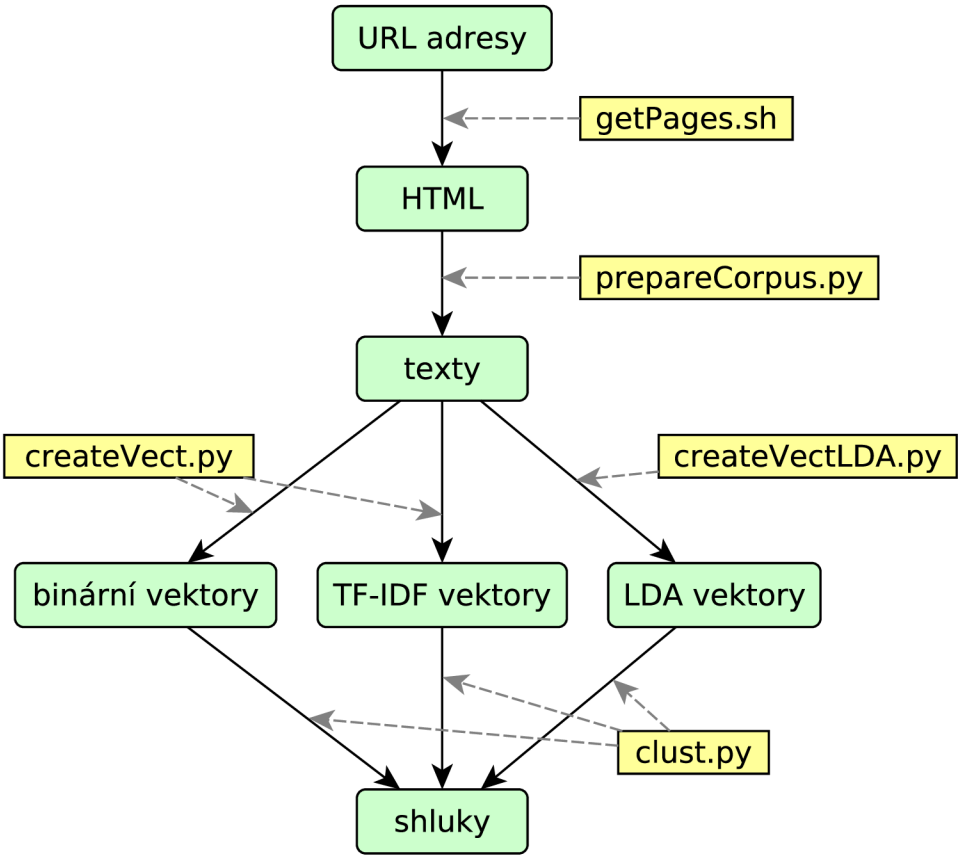
Díky analýze provedené v kapitole 3.1 vyplynulo na povrch několik dílčích úloh, které je potřeba implementovat, aby celý program na shlukování dokumentů fungoval podle představ. Proto jsem se rozhodl systém rozdělit do několika modulů vykonávající část funkcionality. V následujícím odstavci uvedu stručný popis zdrojových souborů seřazený podle pořadí spouštění.

První skript `getPages.sh` se stará o stažení webových stránek na lokální úložiště. Dále následuje část `prepareCorpus.py`, která zpracuje HTML soubory podle teoretického základu popsaného v kapitole 2.1. Tento skript získá z uložených webových stránek pouze relevantní texty, které jsou použity jako vstup pro moduly vytvářející matematický popis dokumentů. Jak bylo zmíněno výše, tvorba vektorů se děje dvěma podobnými způsoby (*binární*, *tf-idf*) a třetím poměrně odlišným (*LDA*). Z tohoto důvodu jsem se rozhodl tuto část implementace rozdělit do dvou skriptů. Program `createVectLDA.py` vytváří vektory za pomocí *LDA*, zbylé dvě metody jsou ukryty v modulu `createVect.py`. Posledním skriptem je `clust.py`, jenž shlukuje vektory vzniklé v předchozím kroku.

Tento způsob jsem zvolil zejména kvůli pohodlnější spustitelnosti jednotlivých částí systému. Při hledání nejlepší konfigurace *LDA* nebo odhalování vad při převodu textu na vektory, tak nebylo potřeba spouštět zbytečně stahování stránek. Celý navržený systém je zobrazen na schématu 3.1, které zobrazuje jednotlivé části společně se vstupy a výstupy.

3.3.1 Skript getPages.sh

Tato část jako jediná není napsaná v programovacím jazyce *Python*, nýbrž je implementována ve skriptovacím jazyce *bash*. Jako vstup modulu je dán seznam záložek, které je



Obrázek 3.1: Návrh modularity řešení se vstupy a výstupy jednotlivých skriptů

potřeba stáhnout na lokální úložiště. Pro formát souboru s adresami oblíbených stránek byl zvolen zápis ve značkovacím jazyce HTML. Volba pro tento přístup byla zcela nasnadě, neboť všechny moderní prohlížeče umožňují vygenerovat oblíbené stránky právě v tomto formátu. Druhý vstupní parametr pro `getPages.sh` představuje název pro výstupní složku, do které budou následně stahovány internetové stránky. Operace stažení obsahu z celosvětové sítě zahrnuje jisté čekání na odpověď od vzdáleného stroje, a tak by sekvenční zpracování bylo při větším počtu vstupních dat časově neúprosné. Záleží na výkonnosti konkrétního počítače, kolik zvládne paralelních procesů, a proto třetí argument programu znamená maximální počet souběžných stahování.

Samotné stažení se provádí pomocí utility `wget`[1], která z dané URL adresy stáhne obsah do specifikovaného souboru. Před tímto krokem se pomocí stejného programu získá odpověď serveru na dotaz o jaký typ dat je uložen pod zadanou adresou. Všechny odkazy, které nemají textovou podobu, jsou přeskočeny a jejich adresa je zapsána do souboru uchovávající informaci o této chybě. Jako textový soubor je vyhodnocen pouze ten, pro který server ve své hlavičce vrátí položku *Content-Type*[2] s obsahem `text/*`. Program `wget` je

[1]Program je k dispozici na adrese http://www.gnu.org/software/wget/manual/wget.html

[2]Možné hodnoty a syntax *Content-Type* je popsán v RFC dokumentu na straně 12. RFC je dostupné z http://www.ietf.org/rfc/rfc2045.txt

nastaven tak, aby několikrát zopakoval požadavek v případě prvotního neúspěchu, čímž se minimalizují případy, kdy je server jen dočasně nedostupný. Pokud se nepodaří z nějakého důvodu stránku získat, je tato situace také zaznamenána. Stránky se ukládají do vytvořené složky, jejíž název byl zadán jako parametr programu. Soubory dostávají název ve formátu `pageX`, kde `X` představuje pořadové číslo dokumentu. Na poslední řádek staženého souboru je dopsána URL adresa, aby bylo možné v budoucnu stránky podle tohoto identifikátoru rozlišit.

V průběhu samotného skriptu je na *standartní výstup* vypisován řetězec `pageX OK` na znamení úspěšného stažení souboru. Na konci skriptu se uskuteční výpis některých statistik.

3.3.2 Skript prepareCorpus.py

Tato část psaná v *Pythonu* plynuje navazuje na činnost, kterou dokončil první skript. Jak bylo předznamenáno v úvodu kapitoly, `prepareCorpus.py` zpracovává stažené stránky a získává z nich jen text, který je podstatný.

Vstupní informace se opět zadávají pomocí parametrů programu. První představuje *název složky*, ve které se nacházejí HTML soubory. Jelikož skript bude vytvářet poměrně hodně souborů, rozhodl jsem zavést druhý argument programu, jenž znamená jednotnou předponu pro všechny výstupní soubory. Tento přístup zpřehledňuje situaci ve stavu, kdy se spouští program několikrát na různá vstupní data. Jak je popsáno v kapitole 2.1, zpracování textu obnáší několik problémů, nad kterými je nutné se zamyslet a následně je implementovat.

Třída Page

Protože při převodu souboru zapsaného v HTML bude potřeba uchovávat informace o dané stránce, rozhodl jsem se vytvořit třídu `Page`, která bude reprezentovat webovou stránku. Tato třída sdružuje *adresu*, *jazyk* textu a také *obsah* dokumentu, jak ve formě řetězce, tak ve formě frekvenčního seznamu.

První volanou metodou je `getUrl()`, která získá internetovou adresu ze vstupního souboru[3]. Při zpracovávání vstupní kolekce souborů se podle adresy rozlišují duplicitní stránky. Jejich adresy jsou zaznamenány a následně přeskočeny. Další volanou metodou je `getContent()`, která je klíčová v tomto skriptu. Pro odstranění HTML značek a získání tak relevantního textu využívá nástroj *jusText*[4], který je zavolán jako funkce *justext()* ze stejnojmenné knihovny. Funkce vrací návratovou hodnotu v podobě retězece textu a k tomu informaci, ze které části stránky je text získán. Nadpisům je přiřazena větší důležitost. Adresy stránek, ze kterých se nepodařilo získat žádný text, jsou zapsány do chybového souboru. Taktéž se evidují adresy stránek, na kterých *justText* „zhavaroval“.

Další krok představuje získat ze souvislého textu, který je také plný interpunkce, pouze slova. Tomuto úkonu se říká *tokenizace* a je provedena pomocí regulárních výrazů, které nabízí programovací jazyk *Python*. Dále následuje získání slov z URL, tato slova jsou také znásobena koeficientem kvůli většímu významu. Tokeny z textové části a z URL jsou spojeny do řetězce a dále se na něj pohlíží jako na text, který reprezentuje danou webovou stránku. Všechna slova převedena pouze na malá písmena, protože například *Počítač* a *počítáč* jsou z hlediska porovnání dokumentů stejná.

Při vývoji programu se ukázalo, že by bylo vhodné, roztřídit texty podle jazyka, ve kterém jsou napsány a dále zpracovávat skupiny odděleně. Získání zkratky jazyka je im-

[3] Jak je uvedeno výše, adresa je uložena na posledním řádku vstupního souboru.

[4] Více informací a možnost stažení se nachází zde http://code.google.com/p/justext/

plementováno v metodě `setLang()`, která používá pro určování jazyka knihovnu *Compact Language Detector*[5]. Tato knihovna určuje jazyk podle zadaného vstupního textu, a to velmi rychle a úspěšně. Aby skupin podle různých jazyků nevznikalo příliš mnoho, omezil jsem se v práci pouze na dvě skupiny jazyků, a to na *českou* a *ostatní* (převážně *anglickou*).

V této fázi zpracování by výsledný vektor dokumentu byl nepoužitelný kvůli své vysoké dimenzionalitě. Proto je potřeba odstranit slova, která se objevují ve většině stránek. Tato slova jsou pro počítání podobnosti dokumentů nevýznamná, naopak dokonce negativně ovlivňují výsledky. Nyní jsou slova uložena v datové struktuře *slovník*, ve které klíčem je právě dané slovo a hodnotou počet výskytů v dokumentu. Z této struktury jsou velmi rychle smazány položky, které se objevují v tzv. *stoplistu*. Krátký popis tohoto souboru je k přečtení v odstavci 3.3.2.

Jak je popsáno v jednom z odstavců kapitoly 3.1 převod slov na jejich kořen je velice prospěšnou operací vzhledem k porovnávání podobnosti textů. Toto je v práci implementováno dvěma různými funkcemi v závislosti na jazyku dokumentu. Pro česká slova je využit pythonovský modul `czech_stemmer.py`[6]. Tento kód byl napsán pro Python verze 3.1, proto jsem jej musel převést do nížší verze pomocí nástroje *3to2*[7]. Obecně by pro češtinu bylo lepší použít lemmatizátor, který by složitá česká slova převedl na základní tvar. To může být objektem dalšího rozšiřování práce, aktuální situace pracuje pouze s kořeny slov. Jiné jazyky jsou oštřeny algoritmem *porter2* z knihovny *stemming*[8].

Jedna část výstupu tohoto modulu představuje chybové soubory. Všechny začínají předponou, která byla zvolena na začátku programu. Jsou to soubory *PRE_errJust.txt*, *PRE_errEmpt.txt* a *PRE_errDup.txt*, které obsahují vždy na novém řádku adresu stránky, u které nastala chyba při zpracování. Statistiku chybovosti při testování je si možné prohlédnout v kapitole 4.2.

Výstupy důležité pro další části systému jsou pojmenovány opět s předponou, a to jako *PRE_text_pages.lang*, *PRE_stemm_pages.lang* a *PRE_urls.lang*, kde koncovka *lang* nabývá hodnoty `cs` nebo `en`. Pro české, respektive anglické soubory. Všechny soubory mají stejný formát. Na jednom řádku se nacházejí data pro danou webovou stránku. V prvním případě se jedná a textová data, v druhém o slova „prohnaná“ přes stemming a třetí soubor uchovává URL adresy stránek.

Stoplist

Tento soubor, který se používá při zpracování textu, obsahuje nejčastější slova a také slova, které nemají téměř žádný lexikální význam. Protože se v práci předpokládají pouze dva druhy jazyků stránek (čeština, angličtina), stačí vytvořit *stoplisty* pouze pro tyto jazyky.

Anglický seznam slov jsem sestavil ze slov objevujících se nejčastěji ve vstupním korpusu a následně doladil pomocí listu, který jsem našel na internetu[9]. Stoplist pro české texty jsem taktéž tvořil podle nejčastějších slov v kolekci souborů, dále jsem ho doupravil podle vlastního uvážení a citu.

Soubory jsou uloženy v adresáři *stoplists* pod výmluvnými názvy `cs.txt` a `en.txt`. Každé slovo je vždy umístněno na vlastním řádku.

[5] Knihovna je dostupná z adresy http://code.google.com/p/chromium-compact-language-detector/

[6] Informace o tomto modulu je možno získat na adrese http://hlt.di.fct.unl.pt/luis/czech_stemmer/

[7] Více zde https://pypi.python.org/pypi/3to2

[8] Dostupná z https://pypi.python.org/pypi/stemming/1.0

[9] Anglický stoplist, ze kterého jsem vycházel, je k dispozice zde http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a11-smart-stop-list/english.stop

3.3.3 Skript createVect.py

Tento skript psaný v *Pythonu* implementuje dva způsoby (*binární* a *tf-idf*) vytváření vektorů z textových dokumentů. Jako první parametr musí být zadána předpona pro vstupní soubory. Druhý parametr potom představuje zkratku jazyka stránek, které budou zpracovány. Zpracovávané texty obsahují příliš velké množství slov, která se vyskytují zcela minimálně. Pro snížení dimenze vektorů se tak použije jen určitý počet slov, která se v kolekci vstupních souborů objevují nejčastěji. Jaký počet to bude, to udává třetí parametr programu.

Třída Page

Pro sdružení informací o zpracovávané stránce (text, URL, frekvenční seznam slov[10]) poslouží třída `Page`. Tato třída implementuje pouze konstruktor, který ji naplní textem a URL dané stránky, a metodu `getFreq()`, která vytvoří datovou strukturu *slovník* představující výše zmíněný frekvenční seznam slov. Pomocná metoda `prin()` pro výpis obsahu třídy byla využívána pouze při testování.

Tvorba vektorů

Pro každou vstupní stránku je vytvořena instance třídy `Page`, která je následně naplněna daty. Na to následuje vytvoření frekvenčního seznamu slov pro celou kolekci stránek a uchování pouze prvních několika nejčastějších slov. Tvorba vektorů není nijak složitá, teorie *binárního* i *tf-idf* přístupu je popsána v kapitole 2.3, která se zabývá systémy vah podle Saltona.

V cyklech se prochází vytvořené frekvenční seznamy pro každou stránku a podle zvoleného přístupu se naplní hodnotou daná složka vektoru. V případě *binárního* způsobu se vektor naplňuje hodnoty 1 nebo 0, podle toho zda se slovo nachází na stránce, či nikoliv. Metoda *tf-idf* nepočítá pouze s výskytem slova v textu, ale pracuje také s hodnotou *kolikrát* se v dokumentu objevilo. Tímto přístupem se vytvoří model *bag-of-words*, na něj je použita transformace do *tf-idf* a to pomocí funkcí z knihovny *scikit-learn*[11].

Vytvořené vektory se nakonec zapisují do výstupních souborů, které jsou pojmenované *PRE_vectors_bin.lang* a *PRE_vectors_tfidf.lang*, kde *PRE* představuje předponu pro soubory a *lang* jazyk zpracovávaných stránek. Obě tyto proměnné jsou získány z parametrů programu, které zadává sám uživatel.

3.3.4 Skript createVectLDA.py

Tato část celého systému implementuje podobnou činnost jako předchozí skript. Pro vytváření vektorů se však používá *Latent Dirichlet allocation* a to konkrétně implementace z knihovny *gensim*[12]. Při výběru této knihovny hrálo roli jednoduché použití a také její náročnost paměti, které je konstantní pro libovolně velký počet vstupních dokumentů.

Vstupní parametry pro `createVectLDA.py` jsou obdobné k těm, které popisuje odstavec 3.3.3. Rozdíl je pouze ve třetím argumentu, jenž u tohoto skriptu představuje počet témat, které algoritmus LDA rozliší v dokumentu. Teoretickému základu problematiky určování témat v dokumentu se věnuje kapitola 2.4.1.

[10] Frekvenční seznam představuje seznam dvojic: slovo – počet výskytů v textu.
[11] Webová stránka této knihovny se nachází na adrese http://scikit-learn.org/stable/
[12] Dokumentace ke knihovně *gensim* je dostupná na http://radimrehurek.com/gensim/

Třída Corpus

Tato třída představuje celou kolekci vstupních textů. Základ pro ni představuje datová struktura *slovník*, který implementuje knihovna *gensim*. Tato struktura vytváří frekvenční seznam slov z dokumentů, které jsou jí předloženy. Pomocí metody `filter_extremes()` jsem se rozhodl filtrovat slova, která se v celém korpusu objeví méně než *4krát* a také slova, která se nachází ve více než *10 %* dokumentů z celé kolekce. Tyto hodnoty byly zjištěny experimentálně.

Vytvoření LDA modelu

Výše popsáná třída se společně s počtem témat použijí jako vstup pro konstruktor `LdaModel()`. Dalším parametrem tohoto konstruktoru je ještě *chunksize*, který upravuje na kolik částí bude rozdělen vstupní korpus. Přes tyto části poté algoritmus iteruje a tzv. *online* vytváří vektory témat pro každý dokument. Vektor má velikost podle počtu témat, který se volí při spuštění skriptu jako třetí parametr.

Vytvořené vektory se ukládají do souboru *PRE_vectors_lda.lang*, pravidlo při tvoření názvu je opět stejné jako výše. Skript také informuje o průběhu zápisu vektorů do souborů pomocí orientačního výpisu procentuálního postupu.

3.3.5 Skript clust.py

Tento skript se stará o shlukování vytvořených vektorů v předchozím kroku. Jako parametry očekává předponu vstupních souborů, jazyk stránek, typ vektorů a počet shluků, které mají být vytvořeny.

Prvním krokem shlukování je načtení vstupních dat. Pro každý vektor je vytvořen objekt podle třídy `Page`, do kterého je uložen vektor a URL adresa identifikující dokument. Konstruktor `KMeans`, který je použit z knihovny *scikit-learn*[13], vytvoří objekt starající se shlukování. Algoritmus na vytvoření shluků je nastaven tak, že prvotní středy jsou vybrány náhodně a maximální počet iterací nepřekročí hranici čísla 100. Konstruktoru je také předán počet shluků, které mají být vytvořeny. Dále jsou vektory sloučeny do matice, ve které jeden řádek reprezentuje jeden dokument. Tato matice je předložena metodě `fit()`, která spouští algoritmus *k-means*. Algoritmus každému dokumentu (vektoru) přiřádí číslo skupiny, do které patří.

Podle těchto identifikátorů jsou v nádledujícím kroku URL adresy roztříděny do patřičných shluků. Výpis celého skriptu se provádí na standartní výstup. Nejprve je vypsán číselný název skupiny, poté následuje 10 nejčastějších slov, které se objevují v daném shluku. K těmto slovům je v závorce vypsán také počet výskytů. Nakonec jsou vypsány všechny URL adresy, která jsou si nějakým způsobem podobné.

[13] Webová stránka této knihovny se nachází na adrese http://scikit-learn.org/stable/

Kapitola 4

Testování a výsledky

Po analýze problému a popisu jeho implementace se dostává na řadu zhodnocení funkčnosti a zobrazení dosažených výsledků. Testování programu probíhalo postupně při vývoji a skripty vykazovaly funkční chování. Jaké výsledky systém produkoval a analyzování do jaké míry jsou správné, to je cílem následujících několika odstavců.

4.1 Spuštění programu

V kapitole 3 je popsán návrh celé aplikace. Protože je systém rozdělen do několika skriptů, spuštění celého programu je určeno přesným pořadím jednotlivých částí. Pro účely závěrečného testování jsem vytvořil ještě jeden skript v *bashi*, který automatizuje a volá postupně ostatní části. Detailní popis spouštění všech skriptů je zapsán v příloze B.

Aplikace očekává od uživatele vstupní informace. Počet výsledných shluků, počet nejčastějších slov, ze kterých budou vytvořeny vektory, a jiné proměnné ovlivňují výsledné rozložení skupin a jejich obsah. Pro všechny testy a výsledky, které jsou popsány níže, byla použita tato vstupní konfigurace. Soubor záložek obsahoval 34077 odkazů na oblíbené webové stránky. Počet výsledných shluků byl nastaven na 50 a délky vektorů popisující dokumenty byly určeny na 1000 prvků u binárního a tf-idf přístupu. U vytváření pomocí LDA měly vektory délku 200.

4.2 Statistiky vstupních dat

Záložky představují uložené oblíbené odkazy. Některé nemusely být navštívené dlouhý čas, proto se může stát, že už neexistují nebo byly přesunuty na novou adresu. Při stahování stránek může nastat několik dalších chyb. Skript `getPages.sh` rozlišuje typ chyb do 3 různých kategorií. První skupina obsahuje adresy serverů, které neodpovídají na žádost o stažení stránky. V této skupině jsou také zahrnuty adresy, které neodkazují na textové stránky, ale například na multimediální nebo jiný obsah. Ve druhé skupině se nacházejí stránky, které jsou zabezpečené a pro přístup požadují uživatelské jméno a heslo. Poslední skupina sdružuje stránky, které na adrese nebyly nalezeny a při jejich zobrazení je většinou vypsána hláška *404 - Not Found*. Tabulka 4.1 zobrazuje přehledně počty neplatných internetových odkazů a podíl z celkového počtu adres (34077). Po odečtení všech chybových odkazů od vstupního počtu adres získáme hodnotu 28732, která představuje množství v pořádku uložených webových stránek.

Chyba	Počet	Podíl z celku
Bez odpovědi	1821	5.3 %
Autorizace	181	0.5 %
Not Found	3343	9.8 %
Celkem	5345	15.6 %

Tabulka 4.1: Tabulka ukazující chybovost vstupních dat

Další statistické údaje poskytuje skript `prepareCorpus.py`. Nejprve informuje o duplicitních odkazech v kolekci souborů. Tyto dokumenty jsou přeskočeny a adresy zaznamenány do speciálního textového souboru. Další dva údaje se týkají převodu HTML zápisu stránek do podoby čistých relevantních textů. Tuto operaci obstarává knihovna *jusText*. Adresy stránek, ze kterých se nepodařilo získat text, nebo které funkce *justext()* nezvládla, jsou zapsány v chybových souborech. Počty jsou opět přehledně zobrazeny v tabulce, tentokrát pod číslem 4.2. V tabulce jsou jako celek brány správně stažené stránky, tedy číslo 28732.

Chyba	Počet	Podíl z celku
Duplicita odkazu	3723	13.0 %
JusText	60	0.2 %
Nezískán text při převodu	3633	12.6 %
Celkem	7416	25.8 %

Tabulka 4.2: Tabulka zobrazující počty chyb při zpracování textu

Po odečtení všech různých chybových odkazů od původního celku (34077) se počet dále zpracovávaných stránek dostane na číslo 21316, z toho českých stránek je pouze 1751. Kolekce cizojazyčných stránek, z drtivé většiny anglických, čítá 19565 internetových adres.

Pozorný čtenář se pozastavil u vysokého procenta stránek, ze kterých se nepodařilo získat žádný text. Funkce *justext()* může být při volání různými argumenty pozměněna a textu by se mohlo podařit získat více. Na druhou stranu s tím se také mohou objevit elementy, které nejsou chtěné. Při použití funkce se řídím doporučením autora, který ji vytvořil.

4.3 Vyhodnocení výsledků

Výsledky shlukování mohou být porovnávány mnoha způsoby. Tato problematika je však velmi citlivá a ne každý algoritmus pro vyhodnocení shluků může být použit v každém případě. Metody hodnocení shlukovacích algoritmů se dělí na dvě základní skupiny. *Interní* míry slouží k měření správnosti struktury shluků bez ohledu na externí informace. Patří sem například suma čtverců odchylek, shluková koheze nebo shluková separace.

Druhou skupinou měr jsou tzv. *externí*, které měří, jak jednotlivé shluky odpovídají třídám objektů. Do této skupiny spadá f-míra, e-míra, entropie nebo čistota. [5]

Pro tuto práci však nebyla použita žádná z výše zmíněných měr, ale vyhodnocení proběhlo porovnáním s očekávanými výsledky pro náhodně vybrané adresy stránek. Celý proces vyhodnocování začíná výběrem náhodných stránek. Velikost náhodného vzorku je velmi důležitá, čím více položek bude vybráno, tím je vyhodnocení věrohodnější. V této práci se

pracuje s 50 náhodnými prvky, ke kterým je uděláno porovnání: očekávaný – vypočtený výsledek. Vybrané stránky je potřeba manuálně projít a rozhodnout, do jaké kategorie patří. Pro porovnání s hodnotami, které vypočítají tři implementované metody, jsem vytvořil 4 kategorie správnosti výsledků. Seřazené od nejlepších k nejhorším vypadají takto: *správně*, *spíše správně*, *spíše špatně* a *špatně*.

Tato metoda srovnání je závislá na citu pozorovatele a hodnotitele. Výsledky, které jsem získal, jsou uvedené v tabulce 4.3. Do tabulky jsem schválně nedal metodu *binárních* vektorů, protože její výsledky nebyly vůbec podobné očekávaným. Hodnoty jsou vypočteny pro 50 anglických, náhodně vybraných stránek.

	Metoda tf-idf	**Metoda LDA**
Správně	54 %	36.7 %
Spíše správně	16 %	20 %
Spíše špatně	14 %	10 %
Špatně	16 %	33.3 %

Tabulka 4.3: Tabulka zobrazující výsledky porovnání mezi očekávaným a vytvořeným shlukováním

Z tabulky je patrné, která metoda dopadla v daném případě lépe. Náhodný vzorek však není příliš rozsáhlý, proto výsledky mohou být trochu zavádějící. U obou metod se objevují výsledky, kdy jsem nebyl schopen „ostře“ rozhodnout, proto vznikly dvě prostřední kategorie. Pokud by se odstranily a rozdělení by se vytvořilo jen pro skupiny *správně* a *špatně*, až 70% zařazení do shluků u metody *tf-idf* by se shodovalo s očekáváním.

Při podrobné analýze výsledků vykazovaly všechny tři metody vytváření vektorů podobné chování v jedné oblasti. A to vytváření shluků stránek, které toho nemají mnoho společného s počítačem, webem nebo daty. Mezi nejlépe vytříděné shluky patří ty, které obsahují velmi specifické webové stránky. Jedná se například o shluky sdružující témata: *umění (art)*, *auta (car)*, *mobil (mobile)* nebo *robot*.

Abych demonstroval výše popsaný jev, uvedu ho na následujícím příkladu. Ve výstupním souboru metod *tf-idf* a *LDA* jsem spočítal adresy, ve kterých se objevuje slovo *robot*. Poté jsem zjistil, kolik těchto adres leží ve shluku zabývající se robotikou a kolik jich leží v ostatních skupinách. Údaje jsou zapsány v tabulce 4.4.

	Metoda tf-idf	**Metoda LDA**
Shluk o robotice	81 %	66.8 %
Ostatní shluky	19 %	33.2 %

Tabulka 4.4: Tabulka zobrazující rozdělení adres obsahující slovo robot

Tento příklad vyhodnocení výsledků je spíše názorný. Na každý pád je už při prohlížení souboru s výsledky zřejmé, že menší počet specifických stránek se daří lépe přiřadit do stejného shluku.

Kapitola 5

Závěr

Cílem práce bylo prostudování metod sémantické podobnosti textů a na základě získaných poznatků navrhnout a realizovat systém, který dokáže utřídit záložky a sdružit příbuzné odkazy do skupin.

Po nastudovaní problematiky sémantické podobnosti a reprezentaci dokumentů jsem byl schopen napsat první kapitolu práce. V této části je popsán teoretický základ zpracování textů, systémů vah a také okrajově se kapitola dotýká metody *Latent Dirichlet allocation*, která se dnes hojně využívá například v internetových vyhledávačích.

Druhá kapitola už je zaměřena na praktickou část práce. Celý systém na třídění záložek je rozdělen do několika modulů, které jsou podrobně popsány v této části písemné zprávy. Program implementuje tři varianty matematické reprezentace dokumentů, a tak umožňuje třemi způsoby shlukovat vstupní kolekci souborů. Jak jednotlivé verze fungují či nefungují popisuje kapitola číslo 4, která se věnuje testování a vyhodnocení výsledků.

Po zamyšlení nad praktickým využitím práce vyvstává řada otázek a nejasností. Nejprve by bylo potřeba zjistit, jakou formou by program byl uživateli nabídnut, protože stávající řešení není komfortní na použití. Podoba rožšíření do prohlížeče se jeví jako jedna z možností. K tomu se ale nabalují další neimplementované části, jako je zadávání nových odkazů. Toto by bylo možné řešit přidáním automatického klasifikátoru, který by nové adresy přidal do již vytvořených shluků. Pokud by program měl být zaměřen i na české texty, bylo by vhodné namísto stemmingu použít lemmatizátor. Uplatnění programu tak vidím jako oporu, podle které by mohl být vytvořen nový nástroj, který by bylo možné reálné použít.

Literatura

[1] Berners-Lee, T.; kol: Uniform Resource Identifier (URI): Generic Syntax [online]. Leden 2005 [cit. 2013-05-10].
Dostupné z URL: http://tools.ietf.org/html/rfc3986#section-1.1.3

[2] Blei, D. M.; Ng, A. Y.; Jordan, M. I.: Latent Dirichlet Allocation. In *Journal of Machine Learning Research 3*, 2003, s. 993–1022.

[3] Franc, V.: Shlukování k-means [online]. 2013-12-05 [cit. 2013-05-10].
Dostupné z URL: http://cmp.felk.cvut.cz/cmp/courses/recognition/Lab_archive/RPZ_02-03l/kmeans/index.html

[4] Huang, A.: Similarity measures for text document clustering. *Proceedings of NZCSRSC*, 2008: s. 49–56.

[5] Kučera, J.: *Textové klastrovací algoritmy [online]*. Diplomová práce, Masarykova univerzita, Fakulta informatiky, 2010 [cit. 2013-05-10].
Dostupné z URL: http://is.muni.cz/th/172767/fi_m/

[6] Lashkari, A. H.; Mahdavi, F.; Ghomi, V.: A Boolean Model in Information Retrieval for Search Engines. In *Proceedings of the 2009 International Conference on Information Management and Engineering*, 2009, ISBN 978-0-7695-3595-1, s. 385–389.
Dostupné z URL: http://dx.doi.org/10.1109/ICIME.2009.101

[7] Manning, C. D.; Raghavan, P.; Schuetze, H.: *Introduction to information retrieval*. Cambridge University Press, 2008.

[8] Materna, J.: Sémantická analýza textů (6) [online]. 2011-12-22 [cit. 2013-05-10].
Dostupné z URL:
http://fulltext.sblog.cz/2011/12/22/semanticka-analyza-textu-6/

[9] Nanyang, H.: Cosine Similarity [online]. 2012-12-18 [cit. 2013-05-10].
Dostupné z URL: http://blog.acmj1991.com/?p=1452

[10] Salton, G.; Buckley, C.: Term-weighting approaches in automatic text retrieval. In *INFORMATION PROCESSING AND MANAGEMENT*, 1988, s. 513–523.

[11] Weisstein, E. W.: K-Means Clustering Algorithm [online]. [cit. 2013-05-10].
Dostupné z URL:
http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html

[12] WWW stránky: function word [online]. [cit. 2013-05-10].
Dostupné z URL: http://grammar.about.com/od/fh/g/functionword.htm

Dodatek A

Obsah CD

Přiložené CD obsahuje následující soubory a adresáře:

- src/ - adresář se zdrojovými soubory praktické části bakalářské práce
- text/ - adresář se zdrojovými soubory technické zprávy
- results/ - adresář s výstupy systému určité konfigurace
- bp-xbrhel02.pdf - odevzdaná technická zpráva ve formátu PDF
- readme.txt - manuál k praktické části

Dodatek B

Manuál

Z důvodů popsaných uvnitř bakalářské práce je praktická část rozdělena do několika programů. Skripty jsou psané v *Pythonu* a *bashi*.

Pro stažení webových stránek se spouští skript `getPages.sh`

```
./getPages.sh inFile outDir maxProc
```

inFile	-	vstupní soubor, ze kterého jsou načteny URL adresy
outDir	-	název výstupního adresáře (nemusí být vytvořen)
maxProc	-	maximální počet procesů stahujících souběžně

Skript pro zpracování HTML stránek a převod na texty

```
./prepareCorpus.py inDir PRE
```

inFile	-	vstupní soubor, ze kterého jsou načteny URL adresy
PRE	-	název předpony, která bude připojena ke všem výstupním souborům

Skript pro vytvoření binárních a tf-idf vektorů

```
./createVect.py PRE lang numTop
```

PRE	-	název předpony vstupních a výstupních souborů
lang	-	zkratka jazyka stránek, které budou zpracovávány (`cs` nebo `en`)
numTop	-	počet top slov, ze kterých bude vytvořen vektor

Skript pro vytvoření vektorů za pomocí LDA

```
./createVectLDA.py PRE lang numTop
```

PRE	-	název předpony vstupních a výstupních souborů
lang	-	zkratka jazyka stránek, které budou zpracovávány (`cs` nebo `en`)
numTop	-	počet témat, které mají být rozlišeny v každém dokumentu

Skript pro vytvoření shluků

```
./clust.py PRE lang type numClust
```

PRE	-	název předpony vstupních a výstupních souborů
lang	-	zkratka jazyka stránek, které budou zpracovávány (`cs` nebo `en`)
type	-	typ vektorů, které budou použity (`bin`, `tfidf` nebo `lda`)
numClust	-	počet výsledných shluků

Skript pro automatizaci všech předešlých kroků

`./cluster.sh inpFile PRE lang numClust`

inFile	-	vstupní soubor, ze kterého jsou načteny URL adresy
PRE	-	název předpony vstupních a výstupních souborů
lang	-	zkratka jazyka stránek, které budou zpracovávány (`cs` nebo `en`)
numClust	-	počet výsledných shluků