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

Diplomová práce

Automatické zodpovídání dotazů založené na sumarizaci textů

Plzeň, 2012 Tárik Saleh Salem

Prohlášení

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

V Plzni dne 17. května 2012

.....

Tárik Saleh Salem

Poděkování

Především bych rád poděkoval své rodině za podporu v průběhu celého studia. Děkuji Doc. Ing. Karlovi Ježkovi, CSc. za odborné vedení mé diplomové práce a poskytnutí volnosti při jejím řešení. Také děkuji panu Ing. Miloslavu Konopíkovi, Ph.D. a panu Ing. Ivanu Habernalovi za přínosné konzultace.

Abstract

Automated Question Answering Based on a Text Summarization

The objective of this diploma thesis was to develop a search engine based on automatic query-based multidocument summarization of texts. The thesis meets this objective and the result of the work is the ASI web application using a latent semantic analysis for summarization of texts from web pages. The ASI application is able to perform summarization of web pages in Czech and English and allows extension of the application by other searching algorithms. Further, the work deals with the text summarization, internet searching, filtration of text from web pages (boilerplate removal) and natural language processing, which play an important role in this work.

Obsah

1 Úvod

Internet je téměř neomezený zdroj informací. Žijeme ale v době informačního přehlcení a hledání požadovaných a v neposlední řadě kvalitních informací se stává čím dál náročnější úlohou. I když standardní internetový vyhledávač vrátí uživateli výsledky vyhledávání, jeho hledání není většinou u konce. Velmi často musí uživatel navštívit nalezené webové stránky, aby posoudil jejich relevantnost nebo našel požadovanou informaci. Webová aplikace ASI, produkt této diplomové práce, se snaží omezit nutnost návštěvy další webové stránky anebo alespoň urychlit výběr relevantní webové stránky.

Klíčovou technologií aplikace ASI je sumarizace textů založená na latentní sémantické analýze, která je podrobně popsána. Kromě sumarizace textů se diplomová práce dotýká dalších oblastí jako jsou internetové vyhledávání, filtrace textů z webových stránek a zpracování přirozeného jazyka, které hrajou nemalou roli v celém procesu vyhledávání a kterým je taktéž věnována pozornost. Při vývoji aplikace ASI byl kladen důraz na rozšiřitelnost. Výsledná práce umožňuje nalezení a sumarizaci webových stránek psaných v češtině a angličtině a dovoluje snadné rozšíření o další algoritmy vyhledávání. Vlastnosti implementovaného systému byly ověřeny.

2 Internetové vyhledávání

Dle zadání má být internetový vyhledávač použit jako zdroj informací. Internet je ideální zdroj dat, který dokáže pokrýt velkou většinu uživatelem kladených požadavků na informace. Vzhledem k rozsáhlosti Internetu a jeho stálého růstu, musíme být ale schopni rychlého vyhledávání požadovaných informací. Tato kapitola pojednává o možnostech vyhledávání dat v prostředí Internetu. Popíšeme si, jak fungují internetové vyhledávače, výhody a nevýhody komerčních a open source vyhledávačů a porovnáme dva v současnosti nejpoužívanější a nejpokročilejší komerční Internetové vyhledávače Bing[1] a Google[2]. Vyhledávání relevantních dat na Internetu je jednou z klíčových součástí aplikace, která je hlavním výstupem této práce.

2.1 Internetové vyhledávače

Internetový vyhledávač je služba umožňující na Internetu nalézt webové stránky s požadovanou informací. Vstupem vyhledávačů je dotaz, který je složen z klíčových slov charakterizujících požadovanou informaci. Na základě zadaného dotazu jsou z databáze vyhledávače vybrány nejrelevantnější webové stránky, které jsou následně nabídnuty uživateli. Aby bylo možné vybírat nejrelevantnější webové stránky, je nutné měřit kvalitu stránek uložených v databázi vyhledávače. Internetové vyhledávače pracují většinou ve třech krocích, které by bylo možné chápat i jako jednotlivé části vyhledávače:

1. **Procházení webových stránek (anglicky *web crawling*)**
   Pro procházení webých stránek mají vyhledávače speciální program, kterému se říká vyhledávací robot (v angličtině se používá pojem *web crawler*). Vyhledávací robot se snaží najít všechny webové stránky na Internetu. Na Internetu se pohybuje sledováním hypertextových odkazů a navštívené stránky včetně jejich adres ukládá pro další zpracování, indexování.

2. **Indexování**
   Vyhledávacím robotem nalezené stránky jsou indexovány a ukládány do databáze (tzv. *indexu*). Úkolem indexování je urychlit vyhledávání a zároveň ohodnotit webové stránky podle jejich relevance. Data jsou ukládány do speciálně navržených datových struktur, které jsou klíčem k rychlému vyhledávání. Nejčastěji používanou strukturou je *invertovaný index*, který mapuje slova na webové stránky resp. odkazy. Webové stránky jsou ale často před vlastním indexováním upraveny. Mezi nejčastější úpravy patří odstranění stopslov[3], stemming[4] či lemmatizace[5], které umožňují

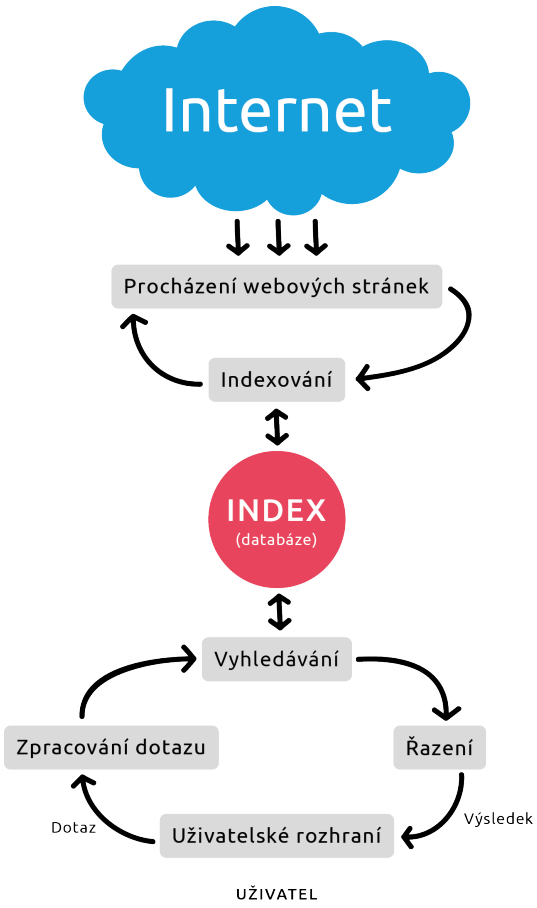
[1] http://www.bing.com
[2] http://www.google.com
[3] Stopslova jsou nevýznamová slova nenesoucí žádnou informační hodnotu nebo slova s příliš častým výskytem v textu, která by mohla zkomplikovat vyhledávání.
[4] Operace, kdy je ke slovu získán jeho kořen. Např. pro „barvě" a „barvou" je kořen „barv".
[5] Operace získání základního tvaru slova (tzv. *lemma*tu). Např. pro „barvě" a „barvou" je lemma „barva".

identifikovat nevýznamová slova a slova různého gramatického tvaru, ale stejného významu. Moderní vyhledávače uvažují kromě ohebnosti slov a stopslov i synonymitu. Míra relevance webové stránky je počítána různými algoritmy. Např. Googlem používaný algoritmus má název PageRank, který hodnotí webové stránky na stupnici od 0 do 10. Tyto algoritmy většinou uvažují váhu slov, atraktivitu a důvěryhodnost stránky, technickou kvalitu stránky a další faktory, o kterých se můžete dočíst např. v [1]. Jaké metriky používal v počátcích Google, se můžete dočíst v [2].

3. **Vyhledávání**
   Máme na mysli fázi, kdy uživatel zadá do uživatelského rozhraní vyhledávače dotaz a jsou mu vráceny výsledky vyhledávání. Na základě uživatelem zadaného dotazu jsou z indexu vybrány odkazy na stránky, které obsahují klíčová slova z dotazu. Nalezené stránky jsou seřazeny podle relevance a vráceny uživateli většinou v podobě seznamu s titulkem webové stránky, krátkým souhrnem obsahující klíčová slova a odkazem.

Vyhledávací robot pro zachování aktuálnosti indexu navštěvuje webové stránky opakovaně v určitých časových intervalech. Ty jsou voleny např. v závislosti na míře relevance dané stránky.

V [5] je podrobněji popsáno fungování internetových vyhledávačů. Vizuálně znázorněné operace internetového vyhledávače můžete vidět na obrázku 1.



Obrázek 1: Diagram operací internetového vyhledávače.

Chceme-li implementovat internetové vyhledávání, nabízí se použití open source vyhledávačů nebo komerčních vyhledávačů resp. vyhledávačů nabízených jako služba. Dále pod pojmem vyhledávač budeme chápat internetový vyhledávač.

2.2 Komerční vyhledávače

Na současném světovém trhu vládne na poli vyhledávačů téměř bezkonkurenčně Google[6]. O další nejvyšší příčky v podílu vyhledávání soupeří Yahoo![7], Baidu[8] a Bing[9]. Přičemž Yahoo! v současnosti využívá Bing pro vyhledávání. Na českém trhu dominují Seznam[10] a Google. Seznam v současnosti využívá služeb vyhledávače Bing pro cizojazyčné vyhledávání.

Nespornou výhodou komerčních vyhledávačů je jejich kvalita a rozsáhlost indexu. Ale podívejme se na výhody a nevýhody, které přináší i v případě jejich použití v programu.

[6] http://www.google.com
[7] http://www.yahoo.com
[8] http://www.baidu.com
[9] http://www.bing.com
[10] http://www.seznam.cz

Výhody:

- Jednoduché řešení.
- Implementují nejmodernější technologie a dosahují vysoké kvality vyhledávání.
- Pokrývají téměř celý Internet.
- Rychlost.

Nevýhody:

- Většinou neposkytují webové API[11].
- Omezující licenční podmínky.
- Nelze přizpůsobit.

2.2.1 Možnosti implementace

Omezíme-li se pouze na vyhledávače, které jsou schopny vyhledávat webové stránky v češtině a angličtině a které mají největší podíl ve vyhledávání v České Republice, získáme následující výčet:

- Bing,
- Google,
- Seznam (využívající Bing pro cizojazyčné vyhledávání).

Bing jako jediný z trojice vyhledávačů dává k dispozici webové API. Výsledky vyhledávání je tak možné získávat ve formátech XML a JSON. Bing poskytoval webové API dlouhou dobu bezplatně, avšak při dokončování této práce jsme byli informování o plánovaném zpoplatnění. Google ani Seznam neposkytují webové API. Dnes pravděpodobně jedinou možností pro získání výsledků vyhledávání je tzv. *web scraping*[12]. To má však také svá omezení jako je frekvence dotazů za jednotku času a relativně častá změna struktury výstupní webové stránky. Dalším omezením mohou být smluvní podmínky daného vyhledávače, které nedovolují jiný přístup než přes poskytovaná rozhraní a mohou zakazovat používání automatizovaných nástrojů.

[11] API je aplikační programové rozhraní.

[12] Web scraping je název počítačových technik pro získávání dat z webových stránek. Software pro web scraping simuluje chování uživatele při přístupu na webové stránky.

2.2.2 Porovnání

Licenční podmínky jsou omezením, které zabraňují použití daného vyhledávače. Ostatní nevýhody zastiňuje fakt, že komerční vyhledávače dosahují kvality, které by bylo obtížné dosáhnout s open source či jiným volně dostupným řešením. Implementace vyhledávačem poskytovaného webového API či web scraping jsou navíc implementačně méně náročné než zavedení a případné přizpůsobení open source vyhledávače.

Samotné měření kvality internetových vyhledávačů je komplexní úkol. Z důvodu variability možných dotazů je velmi obtížné jednoznačně a objektivně měřit kvalitu výsledků vyhledávačů. O tom pojednává článek, který se zabývá měřením kvality vyhledávačů [3]. Obvyklý postup při měření kvality vyhledávačů je:

1. Výběr vzorku dotazů (např. 50 dotazů). Dotazy jsou často vybírány z logů vyhledávačů.
2. Výběr vyhledávačů pro testování.
3. Získání předních (např. 20) výsledků z každého vyhledávače a pro každý dotaz.
4. Předání anonymních výsledků skupině lidí, které je vyhodnotí.

Tak rozsáhlé testování však nebylo možné. Testovány byly vyhledávače Bing a Google. Cílem testování bylo odhalit několik vlastností:

- kvalitu vyhledávačů,
- schopnost zpracování dotazů kladených v přirozeném jazyce (jako otázky),
- nutnost odstraňování stopslov, jelikož současné vyhledávače to již dělají,
- a z výše uvedených bodů by mělo také vyplynout, zda je vhodnější klást dotazy jako souhrn klíčových slov nebo jako otázky.

Testováno bylo provedeno manuálně na sadě 10 dotazů. Dotazy se podobají otázkám kladeným v přirozeném jazyce a tudíž obsahují stopslova. K těmto dotazům byla vytvořena další sada, která vychází z původních dotazů, avšak neobsahují stopslova. Stopslova byla odstraněna dle slovníků, které jsou používány v aplikaci, která je výstupem této práce. Testována byla čeština a angličtina. Testování i hodnocení bylo prováděno manuálně. Uvažovalo se pouze prvních 10 výsledků nabídnutých internetových vyhledávačem. Z těchto 10 výsledků byly hodnoceny pouze HTML[13] stránky a jiné dokumenty (PDF, DOC atd.) byly ignorovány. U dotazů s odstraněnými stopslovy byla hodnocena relevantnost vzhledem k původním dotazům.

[13]HTML (HyperText Markup Language) je značkovací jazyk používaný pro vytváření webových stránek.

- Za **relevantní** byly označeny webové stránky, pokud přímo obsahovaly dotazem požadovanou informaci.
- Za **uspokojivé** byly označeny webové stránky, pokud obsahovaly dotazem požadovanou informaci nepřímo (byla nutná logická indukce) nebo pokud obsahovala požadovanou informaci, ale zároveň obsahovala text, který se nevztahuje k dané tématice (obvykle diskuzní fóra).
- Za **irelevantní** byly označeny webové stránky, které neobsahovaly požadované informace nebo požadované informace byly chybné.

Nemůžeme zapřít fakt, že tento způsob hodnocení je silně individuální, a tudíž ne zcela objektivní.

Vyhledávač	Jazyk	Úprava dotazu	Relevantní	Uspokojivé	Irelevantní
Google	Čeština	Žádná (viz tabulka 7)	**51,6 %**	**12,1 %**	**36,3 %**
Google	Čeština	Bez stopslov (viz tabulka 8)	**67 %**	**10,3 %**	**22,7 %**
Bing	Čeština	Žádná (viz tabulka 9)	**25 %**	**5,2 %**	**69,8 %**
Bing	Čeština	Bez stopslov (viz tabulka 10)	**34,7 %**	**5,3 %**	**60 %**
Google	Angličtina	Žádná (viz tabulka 11)	**79,4%**	**9,3 %**	**11,3 %**
Google	Angličtina	Bez stopslov (viz tabulka 12)	**47,5 %**	**13,1 %**	**39,4 %**
Bing	Angličtina	Žádná (viz tabulka 13)	**64 %**	**10 %**	**26 %**
Bing	Angličtina	Bez stopslov (viz tabulka 14)	**39 %**	**6 %**	**55 %**

Tabulka 1: Porovnání internetových vyhledávačů Bing a Google.

Jak můžete vidět v tabulce 1, Bing i Google dosahují lepších výsledků pro dotazy v angličtině, což se dalo předpokládat, jelikož se jedná o jejich primární jazyk. Překvapivý byl však rozdíl kvality vyhledávačů Bing a Google pro češtinu, kde Bing dosahoval špatných výsledků. Pro češtinu dosahovaly vyhledávače nepatrně lepších výsledků po odstranění stopslov, zatímco pro angličtinu tomu bylo naopak. Testování také ukázalo důležitost rozsáhlosti stopslovníku. Příliš rozsáhlý stopslovník může vést k odstranění pro hledanou informaci důležitých klíčových slov z dotazu. Jak zmiňuje [4], Google v dotazu identifikuje pouze několik málo stopslov. Všechny testované vyhledávače uvažují ohebnost slov. Po shlédnutí výsledků pro jednotlivé dotazy (viz příloha na straně 67), můžeme říci, že odstraněním stopslov jsme vždy nedosáhli lepších výsledků a větší roli pro nalezení požadované informaci hraje volba správných klíčových slov. Navíc vzhledem k tomu, že vyhledávače odstraňují z dotazu některá stopslova, byla tato operace u některých dotazů zbytečná.

2.3 Open source vyhledávače

Alternativním řešením ke komerčním vyhledávačům mohou být open source vyhledávače. O open source vyhledávačích se zde zmíníme jen okrajově, jelikož primárním cílem této

práce bylo využití některého z předních komerčních internetových vyhledávačů. Open source vyhledávače jsou zmíněny především jako vhodné řešení pro experimentální účely. I zde je uveden výčet výhod a nevýhod použití open source vyhledávačů.

Výhody:

- Přístup k programovému kódu (transparentnost).
- Možnost přizpůsobení (může např. předzpracovat potřebná data).
- Zdarma.

Nevýhody:

- Index nebude pravděpodobně pokrývat celý Internet.
- Pravděpodobně nedosáhneme snadno kvality některých předních komerčních vyhledávačů.
- Nasazení vyhledávače, jeho přizpůsobení a vybudování dostatečně rozsáhlého indexu webových stránek je časově náročnější ve srovnání s komerčními vyhledávači, kde není nutné ani možné uvedené kroky provést.
- Větší nároky na hardware.

2.3.1 Příklady open source vyhledávačů

Výběr open source vyhledávačů je široký. Níže uvedený seznam obsahuje pouze vyhledávače, které byly aktualizovány v posledních třech letech. Všechny uvedené vyhledávače kromě jiného uvažují stopslova, používají stemming, hodnotí relevantnost stránek a řadí při vyhledávání stránky podle relevantnosti k dotazu.

Název	Crawler	Jazyk	Poznámka
Lucene[14]	NE	Java	Knihovna pro indexování a vyhledávání.
Solr[15]	NE	Java	Vyhledávací server postavený na Lucene.
Nutch[16]	ANO	Java	Kompletní řešení postavené na Solr.
YaCy[17]	ANO	Java	Plně distribuovaný internetový vyhledávač postavený na principech P2P sítí.
Sphinx[18]	NE	C++	Podporuje i indexování SQL databází.
MG4J[19]	NE	Java	Nevyhledává podobné řetězce (tzv. *fuzzy search*).
Indri[20]	NE	C++	Poskytuje API pro Java, PHP nebo C++.
Terrier[21]	NE	Java	Neumožňuje inkrementální indexování.
Xapian[22]	NE	C++	Knihovna pro indexování a vyhledávání.
Swish-e[23]	ANO	C	Kompletní vyhledávač.
Egothor[24]	ANO	Java	Vyvinuto na Karlově univerzitě v Praze.

Tabulka 2: Přehled open source vyhledávačů.

Vyhledávací robot není většinou součástí open source vyhledávačů. Práce [6] kromě jiného zmiňuje vyhledávací roboty WebSPHINX[25], crawler4j[26], Heritrix[27], vyhledávací robot používaný v Nutch a také popisuje problémy, se kterými se potýkáme při procházení Internetu.

2.3.2 Porovnání

Porovnávání současně dostupných open source vyhledávačů není očividně věnována přílišná pozornost. [5] se věnuje porovnávání volně dostupných vyhledávačů, ale výsledky jsou

[24]http://lucene.apache.org
[24]http://lucene.apache.org/solr
[24]http://nutch.apache.org
[24]http://yacy.net
[24]http://sphinxsearch.com
[24]http://mg4j.dsi.unimi.it
[24]http://www.lemurproject.org/indri
[24]http://terrier.org
[24]http://xapian.org
[24]http://swish-e.org
[24]http://www.egothor.org
[25]http://www.cs.cmu.edu/~rcm/websphinx/
[26]http://code.google.com/p/crawler4j/
[27]https://webarchive.jira.com/wiki/display/Heritrix/Heritrix

již zastaralé. Hodnotí i výkonnosti jednotlivých vyhledávačů, ale nezabývá se možnostmi konfigurace, které mohou značně ovlivnit rychlost indexování a jiné operace. Aktuálnější porovnání vyhledávačů poskytuje C. Middleton v [7]. Kromě open source vyhledávačů uvedených v tabulce 2 zmiňuje i další. [8] je další zdroj, který porovnává některá open source řešení.

3 Filtrování obsahu

Po stažení vyhledávačem nalezených webových stránek je z nich třeba získat textový obsah. Z webových stránek můžeme získávat nejrůznější informace, ale většinou nás zajímá hlavní obsah, kterým bývají články, příspěvky blogů a jiný smysluplný text. Ovšem valná většina současných webových stránek není tvořena pouze hlavním (žádoucím) obsahem, ale i dalšími prvky, které s ním nesouvisí, jako jsou např. hlavičky a patičky stránek, postranní lišty, navigace, nejrůznější seznamy, komentáře, formuláře, reklamy atd. Tento nežádoucí obsah je v angličtině označován jako *boilerplate*. Kvůli tomuto balastu, který často v poměru k hlavnímu obsahu tvoří většinu obsahu celé webové stránky, se stává extrakce hlavního obsahu obtížná. Příklad takové webové stránky můžete vidět na obrázku 2, kde je hlavní obsah stránky zvýrazněn. Zbytek je tzv. boilerplate, jak již bylo uvedeno.

Obrázek 2: Hlavní obsah.

Extrakce hlavního obsahu z webových stránek se využívá např. i pro indexování v internetových vyhledávačích, pro detekci duplikátů nebo pro zobrazení na zařízeních s malým displejem, jako jsou mobilní telefony.

3.1 Metody

Nyní přicházíme k otázce, jak identifikovat a extrahovat hlavní obsah z webových stránek. I když vývoj značkovacích jazyků pomalu směřuje k sémantickému značení (jak ukazuje HTML5[28]), stále neexistuje dostatečně univerzální a rozsáhlý standard, který by popisoval význam jednotlivých částí webové stránky. Nejsnáze identifikujeme hlavní obsah, pokud známe strukturu webové stránky. Ta je nám však většinou neznámá, pokud se nebudeme specializovat pouze na konkrétní webové stránky. Je také nutné zdůraznit, že hlavní obsah může být tvořen několika nesouvislými, ale navzájem závislými částmi.

Jak je popsáno v [9], existují různé přístupy a heuristiky, které využívají pro identifikaci a extrahování hlavního obsahu z webové stránky:

- strojové učení [10, 11],
- strukturu webové stránky [12, 13],
- vizuální model [14],
- n-gramy [10],
- tokenizaci [15],
- tzv. mělké vlastnosti textu (anglicky *shallow text features*) [16, 17, 15], mezi které patří:
  - počet: slov nebo tokenů,
  - průměr: délky slov, tokenů nebo vět,
  - poměr: slov s velkým písmenem nebo interpunkčních znamének,
  - blokové HTML tagy: `<p>`, `<div>`, `<h1>`, `<h2>`, `<h3>` atd.,
  - hustota: odkazů či textu,
- kontext (vlastnosti předchozího a následující bloku, relativní a absolutní pozice bloku, porovnání s jinými dokumenty) [16].

Jednotlivé přístupy bývají často kombinovány.

3.2 Nástroje

I když ve výše uvedeném seznamu jsou uvedeny odkazy na literaturu, popisující různé algoritmy pro odstraňování nežádoucího obsahu, stručně pouze zmíníme několik významných algoritmů, jejichž implementace jsou volně dostupné.

[28] Rozšiřující specifikace standardního HTML. Je stále ve vývoji, ale již hojně používaný.

3.2.1 BTE[29]

BTE (neboli Body Text Extraction) je algoritmus pro identifikaci hlavního obsahu stránky. K identifikaci hlavního obsahu zaznamenává hustotu textu a HTML tagů a vychází z předpokladu, že hlavní obsah je formátován minimálně. Výstupem BTE je nejdelší souvislý blok textu. Zřejmou nevýhodou BTE je, že nedokáže extrahovat hlavní obsah, pokud se skládá z více nesouvislých částí. BTE je implementován v programovacím jazyce Python. Podrobněji je algoritmus popsán v [15].

3.2.2 Victor[30]

Algoritmus je založen na značkování sekvencí s využitím Conditional Random Fields[31] používající vlastnosti extrahovaných sekvencí textu a HTML struktury analyzovaných webových stránek. Algoritmus v počátku vyžaduje (strojové) učení nad ručně anotovanými sekvencemi textu z webových stránek. Oproti BTE neidentifikuje pouze jediný souvislý blok textu z webové stránky. Victor je implementován v programovacím jazyce Perl a je podrobněji popsán v [11].

3.2.3 NCLEANER[32]

NCLEANER nevyužívá pro identifikaci hlavního (žádoucího) nebo nežádoucího obsahu HTML struktury webové stránky a algoritmus je tak možné aplikovat i na prostý text. Pro klasifikaci žádoucího a nežádoucího obsahu používá pravděpodobnostní modely založené na n-gramech. Nevýhodou této metody je její jazyková závislost. Pro použití v jiném jazyce musí být „trénován" nový model. Další nevýhodou je, že současná verze pro angličtinu uvažuje pouze ASCII znaky a pokrytí všech Unicode znaků by vedlo v současném řešení k extrémní datové náročnosti. [10] podrobněji popisuje algoritmus i zmíněné nedostatky. NCLEANER je implementován v programovací jazyce Perl.

3.2.4 Boilerpipe[33]

Boilerpipe je sada algoritmů pro odstraňování nežádoucího obsahu založená na rozhodovacích stromech. Používaný rozhodovací strom používá pouze 6 vlastností, a to počet slov a hustotu odkazů předchozího, aktuálního a následující textového bloku. Rozhodovací strom byl vytvořen na sadě dat uvažující 67 různých vlastností (zahrnující i vlastnosti předchozího a následujícího bloku textu). Následně byl strom zjednodušen (aplikací *reduced-error pruning*) a tím získán menší strom, uvažující zmiňovaných 6 vlastností. Tak identifikuje žádoucí obsah hlavní algoritmus nazývaný DefaultExtractor (v [16] je

[29] http://www.aidanf.net/archive/software/bte-body-text-extraction
[30] http://ufal.mff.cuni.cz/victor/
[31] Conditional Random Fields (zkráceně CRF) je neuspořádaný grafový model používaný pro identifikaci vzorů a ve strojovém učení. Je často používán pro značkování nebo parsování sekvencí dat.
[32] http://webascorpus.sourceforge.net/
[33] http://code.google.com/p/boilerpipe/

referován jako „NumWords/LinkDensity Classifier"). ArticleExtractor (v [16] je referován jako „NumWords/LinkDensity Classifier") je rozšířením algoritmu DefaultExtractor o filtrování (např. komentářů), které podporují extrakci článků např. ze zpravodajských webových stránek. [16] porovnává i použití hustoty textu namísto počtu slov v rozhodovacím stromu a výsledky jsou velice podobné. Další algoritmus CanolaExtractor je založen na algoritmu DefaultExtractor jen s tím rozdílem, že byl trénován na sadě dat Canola[34]. Knihovna implementuje i další primitivní algoritmy pro identifikaci hlavního obsahu, které jsou spíše určeny pro experimentální účely. Knihovna je napsána v programovacím jazyce Java a existuje ve verzi pro Python. Podrobně se algoritmy použitými v Boilerpipe zabývá [16].

3.2.5 jusText[35]

Kromě jiného je výstupem práce [18] J. Pomikálka i nástroj pro extrakci hlavního obsahu z webových stránek nazývaný jusText. Podobně jako u dříve zmíněných algoritmů i jusText převede webovou stránku na text uspořádaný do bloků dle HTML struktury stránky. Poté je každý blok klasifikován na základě hustoty odkazů, počtu slov nebo spíše tokenů a hustoty stopslov. Mohou vzniknou i bloky, které nejsou jednoznačně klasifikovány. Pro jednoznačnou klasifikaci jsou tyto bloky vyhodnoceny v kontextu s okolními bloky. Algoritmus je podrobně popsán v [18]. Je implementován v programovacím jazyce Python.

3.3 Porovnání

J. Pomikálek se v [18] velmi podrobně věnuje problematice identifikace hlavního obsahu webových stránek resp. odstraňování nežádoucího obsahu z nich. Práce porovnává jusText i s výše uvedenými nástroji pro odstraňování nežádoucího obsahu z webových stránek. Testováno bylo na různých sadách dat. Testy ukázaly, že použití BTE se nedoporučuje vzhledem k lepším výsledkům ostatních algoritmů. Ostatní algoritmy jako Victor, NCLEANER, algoritmy Boilerpipe a jusText dosahují velmi dobrých výsledků. Zaměříme-li se na jazykovou nezávislost, tak BTE, Victor a Boilerpipe lze částečně považovat za jazykově nezávislé. jusText je jazykově závislý, ale jeho úprava spočívá pouze v nahrazení slovníku stopslov. Problematická by byla úprava algoritmu NCLEANER, jak již bylo zmíněno v kapitole 3.2.3.

[34] https://krdwrd.org/trac/wiki/Corpora/Canola
[35] http://code.google.com/p/justext/

4 Zpracování přirozeného jazyka

Dalším krokem, který bývá nedílnou součástí procesu sumarizace je zpracování vstupního textu. Typicky mezi takové operace patři rozdělení textu na slova a zbylé znaky (tokenizace), identifikace vět, označení stopslov a stemming nebo lemmatizace. Obecně je tato třída úkolů označována jako zpracování přirozeného jazyka (v angličtině Natural Language Processing, zkráceně NLP) a jak by se mohlo mylně předpokládat, nejedná se o triviální úlohy. Nebude zde nabídnut teoretický základ ani přehled základních technik zpracování přirozeného jazyka, které však můžete získat např. v [20], ale nabídneme přehled nástrojů pro podporu zpracování přirozeného jazyka.

Samozřejmě můžeme potřebné nástroje pro zpracování přirozeného jazyka od počátku implementovat sami, ale v současnosti existuje nespočet kvalitních implementací, i když převážně pro angličtinu a jiné rozšířené jazyky. Poměrně velké množství nástrojů pro zpracování přirozeného jazyka je důkazem toho, že se jedná o disciplínu vysoké důležitosti a je jí věnována značná pozornost.

4.1 Nástroje

Dále zmíněné nástroje pro zpracování přirozeného jazyka musely dovolovat snadné rozšíření o podporu dalších jazyků (kvůli češtině) a musely dovolovat tokenizaci, identifikaci vět a stemmning nebo lemmatizaci alespoň pro angličtinu. Již nyní prozradím, že při výběru nástroje pro zpracování přirozeného jazyka bylo již rozhodnuto o použití programovacího jazyka Java, a tak se výběr nástrojů pro NLP zúžil. Výběr Javy bude odůvodněn později v tomto dokumentu.

Na Wikipedii[36] existuje seznam nástrojů pro zpracování přirozeného jazyka. Seznam obsahuje i implementace v dalších programovacích jazycích.

4.1.1 GATE

GATE neboli General Architecture for Text Engineering je široce využívaná a univerzální open source sada nástrojů pro zpracování přirozeného jazyka. GATE je implementován v programovacím jazyce Java a skládá se z frameworku (GATE Embedded), vývojového prostředí (GATE Developer) a webové aplikace pro anotování v týmech (GATE Teamware).

Mezi hlavní výhody platformy GATE patří její rozšiřitelnost a univerzálnost. Architektura GATE rozděluje prvky systémů pro zpracování přirozeného jazyka na komponenty nazývané *resources* (česky *zdroje*). Dále jen komponenty. Tyto komponenty jsou JavaBeans, implementují požadovanou funkcionalitu, jsou znovupoužitelné a mají definované rozhraní. GATE definuje tři typy těchto komponent:

[36] http://en.wikipedia.org/wiki/Natural_language_processing_toolkits

- Language Resources reprezentující komponenty nesoucí data (slovníky, korpusy, ontologie atd.).
- Processing Resources reprezentující komponenty pro zpracování dat (parsery, lematizátory atd.).
- Visual Resources reprezentující komponenty pro tvorbu grafického rozhraní.

Prostřednictvím pluginů lze systém rozšířit o další komponenty. Komponenty jsou v pluginu umístěny jako JAR soubory a součástí pluginu je XML soubor obsahující popis a základní konfiguraci obsažených komponent.

GATE v sobě implementuje mnoho často používaných algoritmů pro zpracování přirozeného jazyka, jako jsou algoritmy pro identifikaci vět, tokenizaci, identifikaci pojmenovaných entit, strojové učení a další. Tyto do jádra včleněné komponenty jsou nazývány CREOLE (Collection of REusable Objects for Language Engineering). Kromě toho je k dispozici mnoho různých pluginů, a to nejen pro angličtinu.

Zaměříme-li se na námi definované požadavky, jsou pro angličtinu k dispozici pluginy (ANNIE a Tools), které dodávají požadovanou funkcionalitu. Navíc některé komponenty těchto modulů lze využít jako základ, který usnadní vytvoření komponent pro češtinu, která není zatím obsažena v současné distribuci GATE.

Veškeré potřebné informace jsou dostupné na stránkách GATE[37].

4.1.2 Apache OpenNLP a UIMA

Apache OpenNLP[38] je sada nástrojů pro zpracování přirozeného jazyka postavená na strojovém učení. Podporuje tokenizaci, identifikaci vět, part-of-speech tagování[39], identifikace pojmenovaných entit a další. Jednotlivé nástroje používají pro svojí činnosti modely, které jsou jazykově závislé a dostupné pro angličtinu a další jazyky. Sady dat, na kterých byly jednotlivé nástroje učeny, nejsou volně dostupné, což komplikuje vývoj OpenNLP. Pokud bychom chtěli vytvořit modely pro další jazyky nebo zlepšit existující modely, tak je třeba vytvořit sadu dat pro trénování, což není jednoduché. Rozšíření o podporu češtiny by bylo tedy velmi pracné. OpenNLP nenabízí stejnou univerzálnost a rozšiřitelnost jako GATE. Avšak v kombinaci s Apache UIMA[40] může být podobně univerzální a rozšiřitelný jako GATE. Apache UIMA je framework pro zpracování nestrukturovaných dat jako je text, zvuk a obraz. Apache UIMA je tvořen frameworkem, komponentami a infrastrukturou. Framework je implementován v programovacím jazyce Java a také C++. Komponenty mohou být implementovány i v jiných programovacích jazycích. OpenNLP je dostupný pro GATE jako plugin a stejně tako pro Apache UIMA jako komponenta. OpenNLP je implementován v programovacím jazyce Java.

[37] http://gate.ac.uk
[38] http://opennlp.apache.org
[39] Identifikace slovních druhů.
[40] http://uima.apache.org

4.1.3 Tesla

Tesla (neboli Text Engineering Software Laboratory) je virtuální vývojové prostředí pro textové inženýrství, jak je uvedeno na webových stránkách projektu[41]. Tesla je framework pro zpracování textu, který byl vytvořen s cílem usnadnit experimentování a vývoj v oblasti zpracování přirozeného jazyka. Jedná se o klient-server aplikaci, kterou mohou používat i skupiny vývojářů. Další algoritmy pro zpracování přirozeného jazyka lze přidávat ve formě komponent. K dispozici je i celá řada již implementovaných komponent včetně požadovaných.

Tesla je implementována v programovacím jazyce Java. Framework Tesla je vyvíjen na Univerzitě v Kolíně.

4.2 Porovnání

GATE, Apache UIMA i Tesla poskytují prostor pro rozšiřování. Z hlediska zpracování přirozeného jazyka, Apache OpenNLP činí rozšíření podpory pro další jazyky složité, jak již bylo zmíněno. Přičemž u GATE i Tesla je možné pro rozšíření podpory o další jazyky použít existující rozšíření. GATE a Tesla navíc nabízejí nejen pro vývoj velmi užitečné vizuální prostředí. Pokud bychom měli seřadit jednotlivé platformy podle množství volně dostupných rozšíření pro zpracování přirozeného jazyka, GATE by byl na prvním místě, následovala by Tesla a nakonec Apache UIMA.

[41] http://tesla.spinfo.uni-koeln.de

5 Sumarizace

Sumarizace řeší problém prezentace dat v kompaktní formě, a to pokud možno bez ztráty informační hodnoty původního zdroje. Pozornost bude zde věnována sumarizaci textů. Zdroj budeme nazývat dokument. Skupinu dokumentů nazýváme shluk (z anglického *cluster*). Výstup sumarizace budeme nazývat souhrn.

V posledních letech je oblasti sumarizace věnována velká pozornost. I přes neustálý vývoj stále nedosahují automaticky generované souhrny výsledků srovnatelných s intelektuální sumarizací (provedenou člověkem).

V této kapitole si představíme fáze sumarizace a faktory, které je ovlivňují, základní členění sumarizátorů a s tím související pojmy, některé přístupy automatické multidokumentové sumarizace založené na dotazu, způsoby měření kvality sumarizátorů a v neposlední řadě bude podrobně popsána metoda latentní sémantické analýzy, která je v této práci použita pro sumarizaci.

5.1 Fáze a faktory sumarizace

[21] se dívá na sumarizaci jako na proces o třech fázích: interpretace, transformace a generování. Interpretace je krok, kdy vstupní dokument je reprezentován ve formátu, který je použitelný pro výpočet. Při transformaci je ze vstupní reprezentace vygenerována datová reprezentace souhrnu. Generování, poslední krok, převádí reprezentaci souhrnu na sumarizovaný text (extrakt nebo abstrakt). Jak vysvětluje [21] tyto fáze se mohou samozřejmě lišit v závislosti na faktorech jako jsou vstup, účel a výstup sumarizace. [21] tyto faktory podrobně analyzuje.

5.2 Členění sumarizátorů textů

Sumarizátory lze dle několika navzájem si nezávislých charakteristik rozdělit do několika skupin.

Podle způsobu vytvoření souhrnu:

- Kompresivní metody sumarizace vytvářejí **extrakt**, což je souhrn složený ze sekvencí slov získaných z původního dokumentu. Sekvencemi slov mohou být fráze, věty nebo odstavce. Většina současných sumarizátorů generuje extrakt. Je očividné, že takto získaný souhrn trpí chabou souvislostí způsobenou častým opomíjením anaforických vztahů, jak je zdůrazněno v [22]. Generování extraktu je tedy jedním z hlavních důvodů, proč současné sumarizátory nedosahují kvality intelektuální sumarizace.

- Generativní metody vytvářejí **abstrakt**. Abstrakt je souhrn, který na rozdíl od extraktu většinou neobsahuje sekvence slov z původního dokumentu. Snahou je při-

blížit se intelektuální sumarizaci, a proto je pro souhrn generován zcela nový text. Generování abstraktu vyžaduje porozumění textu a je zapotřebí nástrojů pro generování gramaticky správných výstupů. Tato úloha je počítačově stále obtížně řešitelná a generované abstrakty nedosahují vysoké kvality.

Podle rozsahu:

- **Jednodokumentové** sumarizátory vytvářejí souhrny pouze nad jedním dokumentem.
- **Vícedokumentové** sumarizátory vytvářejí souhrny z několika dokumentů většinou stejného tématu. Takové skupiny dokumentů nazýváme shluky. U vícedokumentové sumarizace nastává problém redundance informací, který by neměl být opomenut.

Podle požadovaného výstupu:

- **Obecné** sumarizátory prezentují hlavní téma dokumentu. Souhrn je vytvářen z celého dokumentu.
- **Dotazové** sumarizátory (cílené sumarizátory) vytvářejí souhrny relevantní k položenému dotazu. Na dotazu založená sumarizace je podmnožinou cílené sumarizace, kde je téma vymezeno dotazem. V angličtině jsou používány pojmy *query-based* či *topic-based summarization*.

Podle míry automatizace:

- **Asistovaná** sumarizace je poloautomatická sumarizace, která si vyžaduje zásah člověka např. pro anotování, označení důležitých úryvků textu nebo pro dodatečné zpracování výsledků sumarizace.
- **Automatická** sumarizace nevyžaduje pro vygenerování souhrnu zásah člověka.

Podle jazyka:

- **Jednojazyčné** sumarizátory jsou schopny pracovat pouze s jedním jazykem.
- **Vícejazyčné** sumarizátory jsou schopny práce se shlukem různojazyčných dokumentů nebo s dokumentem, který obsahuje text v různých jazycích. V současnosti je zcela běžné, že nejrůznější zdroje informací obsahují i fráze v cizím jazyce. Základem řešení vícejazyčnosti je provázání termů[42] jednotlivých jazyků [23]. Jiným přístupem může být zachování původního jazyk dokumentu a překládat až výsledný extrakt do výchozího jazyka. Některé systémy pracují na mnohem jednodušším principu a strojově překládají vstupní dokumenty do výchozího jazyka.

[42] Významové slovo obvykle ve formalizované podobě (např. základním tvaru).

5.3 Metody multidokumentové sumarizace založené na dotazu

Nejčastěji je multidokumentová dotazová sumarizace založena na

- grafových metodách [29],
- strojovém učení [27, 24, 25],
- statistických metodách [27],
- algebraických metodách [26, 28],
- lingvistických metodách [26]
- a dalších.

Jednotlivé přístupy jsou často kombinovány. Níže budou stručně popsány přístupy tří sumarizátorů (CIST, CLASSY a JRC), které se zúčastnily Text Analysis Conference (TAC)[43] v roce 2011 a dosahovaly dobrých výsledků (viz [30]). Bude zmíněn i sumarizátor FastSum, který zaujal svou jednoduchostí, a také MUSUTELSA sumarizátor, který byl vyvinut na domovské fakultě. FastSum ani MUSUTELSA se nezúčastnily TAC 2011.

Nebude zde nabízen obecný přehled sumarizačních metod, kterým je věnováno dostatečně času v jiné literatuře, jako např. v [22, 21, 31].

5.3.1 CIST

Sumarizátor CIST je založen na metodě hierarchical Latent Dirichlet Allocation (hLDA) a kompresi vět. hLDA dokáže odkrýt skrytá témata, ale také je hierarchicky seřadit. hLDA vyžaduje učení. Stejně jako u jiných sumarizačních metod je text tokenizován, jsou identifikovány věty, odstraněny stopslova a provedena i filtrace vět. Systém uvažuje i podobnost vět a na věty zařazené do výsledného souhrnu aplikuje kompresi, která odstraňuje nepodstatné nebo redundantní části vět.

Podrobně je sumarizátor CIST popsán v [24].

5.3.2 CLASSY

CLASSY je sumarizátor, který se pravidelně objevoval na DUC a nyní i na Text Analysis Conference (TAC)[44]. V roce 2005 CLASSY vybíral věty do souhrnu podle skrytého Markovského modelu (anglicky Hidden Markov Model, zkráceně HMM). Současná verze je založena na bigramech a naivním Bayesovském modelu. Na zpracovávaný text aplikuje tokenizaci, identifikaci vět, stemming, filtrování a kategorizaci vět. Dále jsou získána klíčová slova z dotazu. Algoritmus pro výpočet skóre vět, podle které jsou pak věty vybírány do souhrnu, přiděluje každému slovu pravděpodobnost výskytu v souhrnu, podle které je

[43] Více informací na http://www.nist.gov/tac/.
[44] Více informací na http://www.nist.gov/tac/.

počítáno výsledné skóre pro větu. Pravděpodobnost výskytu slova ve výsledném souhrnu je počítána na základě naivního Bayesovského modelu, který byl trénován na sadě dat z TAC 2010.

Podrobně sumarizátor CLASSY popisuje [25].

5.3.3 JRC

Sumarizátor JRC je založen na latentní sémantické analýze (anglicky Latent Semantic Analysis, zkráceně LSA), která je popsána v kapitole 5.4. Spoluautorem sumarizátoru JRC je Josef Steinberger. LSA je zde doplněno o další techniky, mezi které patří multijazyková identifikace entit (míst a osob), extrakce událostí pro detekci aspektů, analýza časových informací a komprese vět a jejich rekonstrukce.

Podrobně sumarizátor JRC popisuje [26].

5.3.4 FastSum

FastSum nepoužívá pro sumarizaci žádné časově náročné techniky pro zpracování přirozeného jazyka, ale pouze tokenizaci a identifikaci vět. Na základě detailní analýzy použitím algoritmu Least Angle Regression (LARS) uvažuje FastSum minimální množství vlastností. Vybrané vlastnosti, které jsou používány k výpočtu váhy vět pro výběr do souhrnu, jsou založeny především na frekvenci slov shluku dokumentů, jednotlivých dokumentů a dotazu. Další vlastnosti, které byly uvažovány, jsou i délka či pozice věty. Strojové učení je využito k určení váhy jednotlivých vlastností, aby bylo možné vypočítat výsledné skóre vět, které je důležité při výběru vět do souhrnu. Strojové učení je u tohoto sumarizátoru založeno na metodě Support Vector Machines (SVM). Pro proces učení byly využity sady dat z Document Understanding Conference (DUC)[45].

Podrobně je sumarizátor FastSum popsán v [27].

5.3.5 MUSUTELSA

MUSUTELSA sumarizátor je založen na latentní sémantické analýze. Jeho autorem je M. Křišťan. Dle [32] dosahoval MUSUTELSA sumarizátor na sadě dat z DUC 2005 konkurenceschopných výsledků. LSA sumarizace je popsána v následující kapitole 5.4. MUSUTELSA rozšiřuje LSA především o podporu dotazové a multidokumentové sumarizace, o kterých se píše v kapitole 5.4.5.

Podrobně je sumarizátor MUSUTELSA popsán v [28].

5.4 Sumarizace založená na LSA

Inspirováni latentní sémantickou indexací (anglicky Latent Semantic Indexing, zkráceně LSI) přišli Yihong Gong a Xin Liu v roce 2001 s nápadem využít singulární rozklad matic

[45] Více informací na http://duc.nist.gov.

(anglicky Singular Value Decomposition, zkráceně SVD) pro obecnou sumarizaci textů [33]. Jedná se tedy o poměrně novou algebraicko-statistickou metodu sumarizace.

Proces sumarizace založený na LSA můžeme rozdělit do tří základních kroků:

1. Vytvoření vstupní matice reprezentující vstupní dokument.
2. Singulární rozklad matice, který nám pomůže odhalit vztahy mezi slovy (termy) a větami vstupního dokumentu.
3. Výběr vět na základě výsledků singulárního rozkladu.

Kvalitu výsledného souhrnu lze ovlivnit v krocích 1 a 3, tedy při vytváření matice reprezentující zdrojový dokument a při zpracování výsledků singulárního rozkladu matic. Existuje více způsobů, jak postupovat v krocích 1 a 3, některé z nich budou zde popsány. Teoretický základ bude doprovázet i příklad k lepšímu pochopení principu LSA pro sumarizaci.

5.4.1 Vytvoření vstupní matice

Proces sumarizace začíná vytvořením matice $\mathbf{A} = [a_{ij}]$ reprezentující zdrojový dokument. Dokument je nutné dekomponovat na menší jednotky, kterými mohou být fráze, věty nebo odstavce. Jsou zvoleny věty, které jsou dále rozděleny na slova (resp. termy). Matice $\mathbf{A} = (A_1 A_2 \dots A_n)$ je tvořena sloupcovými vektory $A_j = [a_{1j}, a_{2j}, \dots, a_{mj}]^T$, které reprezentují vážený vektor termů věty. Každý term t_i věty A_j má definovanou váhu a_{ij}. Velikost vektorů A_j je rovna součtu všech různých termů obsažených ve větách dokumentu. Je-li počet všech různých termů ve větách dokumentu m a počet vět n, pak je rozměr matice $m \times n$. Pro matici $\mathbf{A}$ platí, že $1 \leq i \leq m$ a $1 \leq j \leq n$, kde i je index termu a j index věty. Jelikož každý term se běžně nenachází v každé větě, je vytvořená matice $\mathbf{A}$ velmi řídká. Už vzhledem k reprezentaci dat plyne jedna z nevýhod LSA: nezachovává pořadí slov.

5.4.2 Singulární rozklad matice

Singulární rozklad matice $\mathbf{A}$ je definován jako

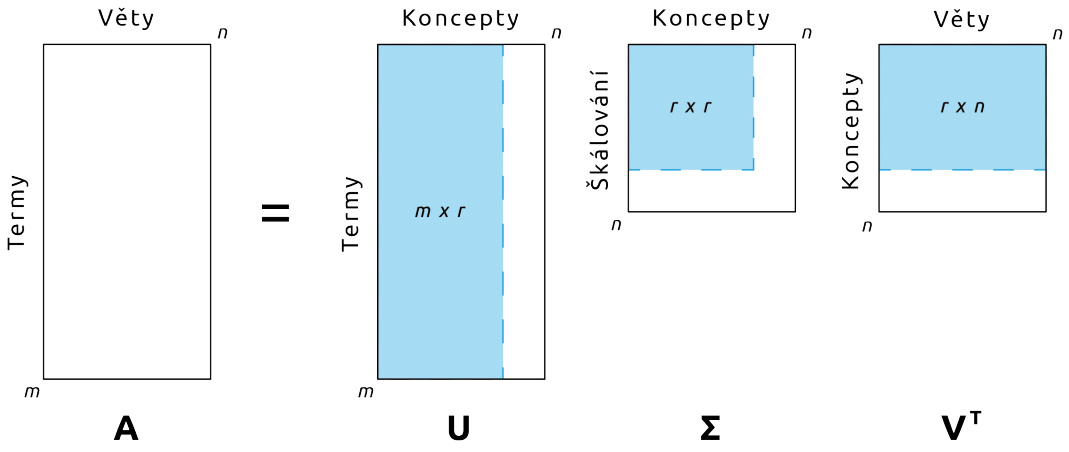
$$\mathbf{A}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

kde matice $\mathbf{U} = [u_{ij}]$ o rozměrech $m \times n$ je ortogonální maticí a její sloupcové vektory jsou nazývány levé singulární vektory. Diagonální matice $\mathbf{\Sigma} = diag(\sigma_1, \sigma_2, ..., \sigma_n)$ má rozměr $n \times n$ a na její diagonále leží singulární čísla seřazená sestupně. Matice $\mathbf{V} = [v_{ij}]$ je také ortogonální s rozměrem $n \times n$ a její sloupcové vektory jsou nazývány pravé singulární vektory. Má-li matice $\mathbf{A}$ hodnost r, tzn. $rank(\mathbf{A}) = r$, platí pro singulární čísla matice $\mathbf{\Sigma}$ následující

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_n = 0.$$

Co nám ale singulární rozklad matice **A** přináší? Ze sémantického hlediska nám singulární rozklad matic umožňuje odhalit latentní (skrytou) sémantickou strukturu dokumentu reprezentovaného maticí **A**. Umožňuje tedy odhalit vazby mezi termy a větami, které nejsou na první pohled patrné. Singulární rozklad matice nám rozloží originální dokument na r lineárně nezávislých bázových vektorů také nazývaných koncepty. Každý term a věta jsou společně indexovány těmito koncepty (viz obrázek 3). Koncepty lze interpretovat jako jednotlivá témata přítomná v dokumentu. Důležitost jednotlivých konceptů vyjadřují singulární čísla matice **Σ**. Tato jedinečná vlastnost nám umožňuje sémanticky shlukovat termy a věty.

Uvažujme slova „doktor", „lékař", „nemocnice", „medicína" a „sestřička". Slova „doktor" a „lékař" jsou synonymy a slova „nemocnice", „medicína" a „sestřička" jsou jejich blízkými koncepty. Synonyma doktor a lékař se často objevují v podobném kontextu, které sdílí související slova jako „nemocnice", „medicína", „sestřička" atd. Protože se slova „doktor" a „lékař" objevují v podobném kontextu s podobným vzorkem jiných slov, budou tyto slova namapována blízko sebe v r-rozměrném singulárním vektorovém prostoru. Navíc častý výskyt určitého vzorku termů v dokumentu bude promítnut do jednoho ze singulárních vektorů. Pak věta, která nejlépe reprezentuje daný vzorek termů, bude mít největší hodnotu indexu s tímto vektorem.



Obrázek 3: Singulární rozklad matice.

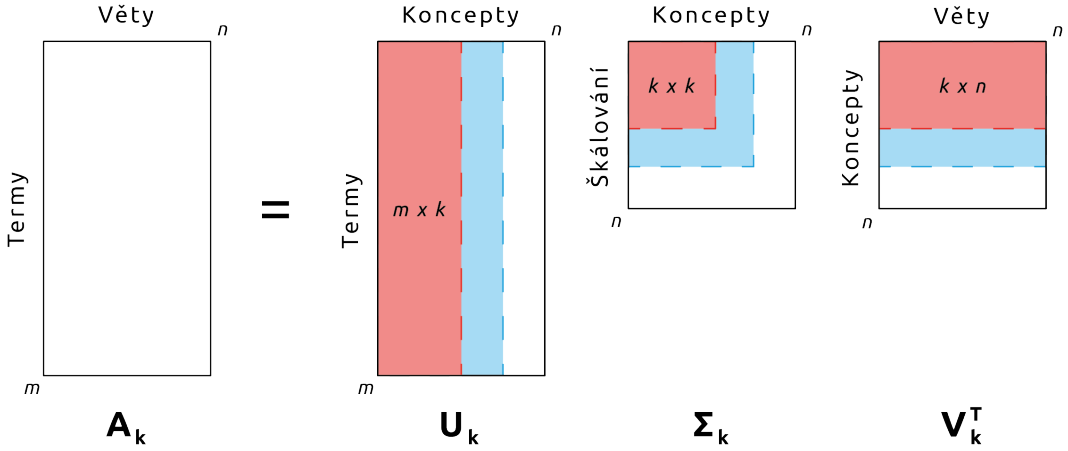
Jendou z hlavních nevýhod singulárního rozkladu matic je její výpočetní náročnost. Navíc, pokud dojde ke změně matice **A**, musí se singulární rozklad provést znovu. Singulární rozklad matic využívá numerických metod, které nejsou obsahem této práce, ale lze je nalézt např. v [34].

5.4.3 Výběr vět

Vzhledem k tomu, že singulární čísla jsou seřazena sestupně dle důležitosti konceptů a pouze k prvních konceptů lze považovat za sémanticky důležité, je možné původní rozklad aproximovat na

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T,$$

kde $\mathbf{U}_k$ je podmaticí matice $\mathbf{U}$, obsahující prvních k sloupců. Matice $\mathbf{\Sigma}_k$ je čtvercová podmatice matice $\mathbf{\Sigma}$ o rozměrech $k \times k$. $\mathbf{V}_k$ je také podmaticí obsahující prvních k sloupců matice $\mathbf{V}$. Názorně k-aproximaci demonstruje obrázek 4.



Obrázek 4: Redukované SVD.

Tato operace je také označována jako k-aproximace nebo redukce prostoru. Právě touto aproximací jsme schopni odkrýt skryté vazby mezi termy a větami. Proto je volba k velmi důležitá. Zde nastává problém, jaké k zvolit, aby byl dosažen optimální výsledek sumarizace. Optimální k však nelze předem určit a může se zároveň lišit pro různá vstupní data. Experimentálním nalezením obecně nejlepší hodnoty k se zabývá [28].

Nyní předpokládejme, že máme zvolené k a lze tedy postoupit k výběru vět do výsledného souhrnu.

Gong a Liu

Popišme si nejdříve metodu výběru vět navrhovanou v již zmiňované práci [33] autorů Yihong Gong a Xin Liu. Pro přehlednost budeme metody výběru vět nazývat příjmeními autorů. Pro výběr vět používají redukovanou matici $\mathbf{V}_k^T = [v_{ji}]$, která mapuje koncepty na věty. Pořadí řádků matice $\mathbf{V}_k^T$ značí sestupně důležitost jednotlivých konceptů, kterých je k v redukované matici. Hodnota v_{ji} matice $\mathbf{V}_k^T$ ukazuje vztah mezi větou a konceptem. Vyšší hodnota (váha) v_{ji} značí, že věta i je více „zpřízněná" s daným konceptem j. Metoda vybírá jednu větu z každého konceptu. k zde figuruje jako maximální počet vět v extraktu. Pro každý koncept j, kde $1 \leq j \leq k$, je vybrána věta i, kde $1 \leq i \leq n$, která má nejvyšší hodnotu v_{ji}. V každé řádce matice $\mathbf{V}_k^T$ je vybrána větu s nejvyšší hodnotou. První věta je vybrána z nejdůležitějšího konceptu, druhá věta z druhého nejdůležitějšího konceptu atd. Tato metoda má však své nevýhody, jak popisuje [35]. Jednou z nevýhod je dvojí význam čísla k: využívá se pro k-aproximaci a pro určení požadovaného počtu vět v souhrnu. Kromě toho s rostoucím k jsou do souhrnu vybírány věty z méně důležitých či zcela

nepodstatných konceptů a jak již bylo řečeno, koncepty reprezentují témata dokumentu. Další nevýhodou je, že do souhrnu nejsou vybírány věty s velkou váhou pro daný koncept, ale pouze věty s nejvyšší váhou, i když by věty s nižší váhou mohli být žádoucí pro výsledný souhrn.

Steinberger a Ježek

[35] popisuje vylepšení, které odstraňují nedostatky předchozí metody. Stejně jako u předchozí metody předpokládáme, že již byl proveden singulární rozklad matice $\mathbf{A}$ a následně k-aproximace. Cílem této metody je vypočítat váhu, někdy také nazývanou index, pro každou větu dokumentu. Pro každý i-tý vektor věty matice $\mathbf{V}_k^T = [v_{ji}]$ je vypočítána jeho váha s_i podle níže uvedeného vzorce.

$$s_i = \sqrt{\sum_{j=1}^{k} v_{ji}^2 \sigma_j^2}$$

Vzorec je vlastně výpočet velikosti vektoru vět po součinu matic $\mathbf{\Sigma}_k$ a $\mathbf{V}_k^T$. Smyslem součinu je zvýhodnit hlavní témata dokumentu. Do souhrnu jsou pak vybrány věty, které mají nejvyšší hodnotu s_i. Metoda však upřednostňuje dlouhé věty. Řešení upřednostňování dlouhých vět nabízí [28], kde je hodnota s_i dělena mocninou délky věty.

Murray, Renals a Carlette

V [36] je navrhován další přístup, který taktéž odstraňuje nevýhody první zmiňované metody. Zde také předpokládejme provedení singulární rozkladu nad maticí $\mathbf{A}$ a následnou k-aproximaci. Pro výpočet využívá matice $\mathbf{V}_k^T = [v_{ji}]$ a $\mathbf{\Sigma}_k$. Metoda vybírá prvních n nejlepších vět pro každý koncept. n je určeno podle procentuálního zastoupení (podílu) příslušného singulárního čísla v matici $\mathbf{\Sigma}_k$.

5.4.4 Příklad

Ukažme si proces sumarizace latentní sémantickou analýzou na příkladu. Pro tento příklad uvažujme pouze obecnou jednodokumentovou sumarizaci. Předpokládejme, že náš dokument obsahuje věty:

1. Doktor operuje pacienta v nemocnici.
2. Pacient čeká na doktora.
3. Pacient leží v nemocnici a čeká na doktora.

Provedeme lemmatizaci slov a odstraníme stopslova. Získáme termy:

1. doktor,
2. operovat,

3. pacient,

4. nemocnice,

5. čekat,

6. ležet.

Nyní vytvoříme vstupní matici **A**, zde o rozměrech 6×3. Výskyt slov ve větě ohodnotíme 1 a v opačném případě 0. Neuvažujeme vícečetnost, i když v tomto příkladu nehraje roli. Pořadí termů a vět odpovídá výše uvedenému pořadí.

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

S definovanou vstupní maticí **A** můžeme pokračovat singulárním rozkladem matic, jehož výstupem jsou matice

$$U = \begin{bmatrix} 0,556 & -0,032 & 0,274 \\ 0,177 & -0,643 & -0,127 \\ 0,556 & -0,032 & -0,274 \\ 0,399 & -0,362 & -0,570 \\ 0,378 & 0,612 & 0,147 \\ 0,222 & 0,282 & -0,698 \end{bmatrix},$$

$$\Sigma = \begin{bmatrix} 3,085 & 0 & 0 \\ 0 & 1,289 & 0 \\ 0 & 0 & 0,907 \end{bmatrix},$$

$$V = \begin{bmatrix} 0,547 & -0,829 & 0,116 \\ 0,483 & 0,425 & 0,766 \\ 0,684 & 0,363 & -0,633 \end{bmatrix}.$$

Abychom mohli vyjádřit závislosti vět a termů v dvourozměrném prostoru, aplikujeme k-aproximaci s $k = 2$. Získáme

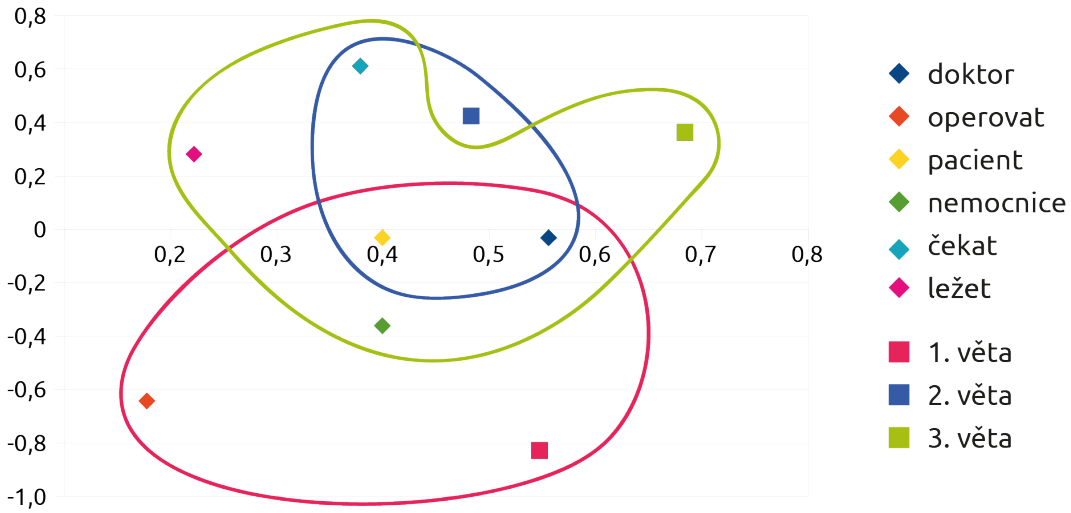
$$U_k = \begin{bmatrix} 0,556 & -0,032 \\ 0,177 & -0,643 \\ 0,556 & -0,032 \\ 0,399 & -0,362 \\ 0,378 & 0,612 \\ 0,222 & 0,282 \end{bmatrix},$$

$$\Sigma_k = \begin{bmatrix} 3,085 & 0 \\ 0 & 1,289 \end{bmatrix},$$

$$V_k = \begin{bmatrix} 0,547 & -0,829 \\ 0,483 & 0,425 \\ 0,684 & 0,363 \end{bmatrix}.$$

Připomeňme si význam těchto matic. Matice $\mathbf{\Sigma}$ (resp. $\mathbf{\Sigma}_k$) obsahuje singulární čísla reprezentující jednotlivé koncepty a jejich důležitost. Matice $\mathbf{U}$ (resp. $\mathbf{U}_k$) mapuje termy na koncepty a řádky zde reprezentují termy a sloupce koncepty. Obdobně matice $\mathbf{V}$ (resp. $\mathbf{V}_k$) mapuje věty na koncepty a stejně tak řádky reprezentují věty a sloupce koncepty. U všech matic jsou koncepty seřazeny podle důležitosti a jak již bylo v této práci zmíněno, koncepty si můžeme představit jako témata dokumentu.

Pro názornost vyjádříme závislosti termů a vět graficky (viz obrázek 5).



Obrázek 5: Graf znázorňující mapování vět a termů na koncepty.

Vodorovná osa značí váhu věty či termu v prvním (nejdůležitějším) konceptu (první sloupec matic $\mathbf{U}$ a $\mathbf{V}$) a svislá osa značí váhu věty či termu v druhém konceptu (druhý sloupce matic $\mathbf{U}$ a $\mathbf{V}$). Z grafu můžeme vidět, že druhá věta souvisí více s třetí než s první větou a termy „doktor" a „pacient" spolu souvisí více než např. termy „doktor" a „ležet". Všimněme si i toho, jak jsou termy umístěny vzhledem k větám.

Zbývá vybrat věty do výsledného souhrnu. Řekněme, že do souhrnu chceme dvě věty. Na každé ze tří zmiňovaných metod výběru vět si ukážeme, jak a jakou větu bychom do souhrnu vybrali.

Gong a Liu

Jak by již mělo být známo, pro výběr vět do souhrnu je potřeba pouze matice $\mathbf{V}_k$, na základě které je pro každý koncept vybrána věta s nejvyšším ohodnocením. Chceme-li pro souhrn dvě věty, musí být $k = 2$. Shodou okolností byla taková aproximace provedena.

	Věta 1	Věta 2	Věta 3
Koncept 1	0,547	0,483	**0,684**
Koncept 2	-0,829	**0,425**	0,363

Tabulka 3: Matice $\mathbf{V}_k$.

První vybranou větou bude věta 3, protože v prvním konceptu má z vět nejvyšší ohodnocení. Stejně stejným způsobem je vybrána druhá věta pro koncept 2. Zde věta 2. Výsledný souhrn bude složen z věty 3 a věty 2 (viz tabulka 3).

Steinberger a Ježek

Váha vět je počítána pro každou větu podle vzorce z kapitoly 5.4.3. Výpočet vah pro jednotlivé věty vypadá následovně

$$s_1 = \sqrt{\sum_{j=0}^{2} v_{j1}^2 \sigma_j^2} = \sqrt{0,547^2 \cdot 3,085^2 + (-0,829)^2 \cdot 1,289^2} \doteq 1,997,$$

$$s_2 = \sqrt{\sum_{j=0}^{2} v_{j2}^2 \sigma_j^2} = \sqrt{0,483^2 \cdot 3,085^2 + 0,425^2 \cdot 1,289^2} \doteq 1,588,$$

$$s_3 = \sqrt{\sum_{j=0}^{2} v_{j3}^2 \sigma_j^2} = \sqrt{0,684^2 \cdot 3,085^2 + 0,363^2 \cdot 1,289^2} \doteq 2,161.$$

Věty 3 a 1 mají nejvyšší váhu, a proto jsou vybrány do souhrnu.

Murray, Renals a Carletta

Metoda autorů Murray, Renals a Carletta vybírá počet vět v každém konceptu podle podílu singulárního čísla daného konceptu. Po k-aproximaci obsahuje matice $\mathbf{\Sigma}_k$ dvě singulární čísla. Jejich procentuální podíl je 70,5 % pro první koncept a 29,5 % pro druhý koncept. Jelikož do souhrnu vybíráme pouze dvě věty, v tomto případě předpokládejme, že 70,5 % je dostatečných pro výběr dvou vět z prvního konceptu. V prvním konceptu tedy vybere věty s nejvyšší váhou, věty 3 a 1 (můžete se přesvědčit např. v tabulce 3). Ale pro názornost uvažujme případ, kdy náš vstupní dokument obsahuje 100 vět a souhrn bude obsahovat pouze 10 vět. Předpokládejme shodná singulární čísla a $k = 2$ jako v tomto příkladu. Pak by bylo do souhrnu vybráno 7 vět z prvního konceptu a 3 věty z druhého konceptu a všechny samozřejmě s nejvyšším ohodnocením v daném konceptu.

5.4.5 Multidokumentová LSA sumarizace založená na dotazu

Podobnost vět

Obecně při multidokumentové sumarizaci vzniká problém redundance vět nebo výskytu velmi podobných vět. To je způsobeno tím, že většinou se multidokumentová sumarizace provádí nad skupinou tématicky podobných dokumentů, které s velkou pravděpodobností

obsahují podobné i shodné věty. Je nežádoucí, aby vygenerovaný souhrn obsahoval velmi podobné nebo dokonce identické věty. Jak popisuje např. [28], nápomocná může být metoda kosinové podobnosti vět. Navíc kosinovou podobnost je možné počítat podle vektorů vět matice **A**.

Budeme-li uvažovat kosinovou podobnost vět, postupuje se při LSA sumarizaci shodně jako u jednodokumentové sumarizace až po výpočet váhy jednotlivých vět. Do extraktu jsou vybírány věty v pořadí dle jejich váhy. První věta s nejvyšší váhou je vybrána do extraktu rovnou. Před vložením další věty do extraktu je vypočítán úhel, který navzájem svírá již vložená věta s aktuálně vkládanou větou. Pokud je úhel menší než nastavený práh, pak je věta shledána za podobnou a není vložena do extraktu. Obsahuje-li extrakt již více vět, je kosinová podobnost vkládané věty počítána s každou větou extraktu.

Výpočet úhlu mezi vektory $\overrightarrow{x}$ a $\overrightarrow{y}$ je dán vzorcem

$$cos\varphi_{xy} = \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{|\overrightarrow{x}| \cdot |\overrightarrow{y}|}.$$

Pro účely této práce je vhodnější vzorec v numerické podobě. Předpokládejme, že $\overrightarrow{x}$ a $\overrightarrow{y}$ jsou vektory vět matice **A**. Vzorec pro výpočet úhlu bude mít tvar

$$cos\varphi_{xy} = \frac{\sum_{i=1}^{m} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{m} x_i^2 \cdot \sum_{i=1}^{m} y_i^2}}.$$

Indexy mají shodný význam jako v kapitole 5.4.1.

Vážení slov

Vážení slov znamená přidělení hodnoty příslušným termům vět v matici **A**. Na příkladu v kapitole 5.4.4 byl použit základní způsob vážení termů, který neuvažuje jejich vícečetný výskyt ve větách. Výskyt každého termu věty byl uvažován pouze jednou a byl ohodnocen číslem 1. Matice tak byla tvořena hodnotami 0 nebo 1. V [28] se neukázalo být přínosné uvažovat vícečetnost. Tento způsob vážení slov je tedy zcela dostačující pro obecnou LSA sumarizaci a není rozdílný pro multidokumentovou sumarizaci.

[28] také navrhuje postup vážení slov při LSA sumarizaci založené na dotazu. Navrhovaná metoda spočívá ve vyšším ohodnocení klíčových slov v matici **A**. Výchozí je matice **A**, která by byla použita pro obecnou sumarizaci. Z dotazu je třeba extrahovat klíčová slova (termy), podle kterých určíme termy vět v matici **A** (řádkové vektory), které získají lepší ohodnocení. Vybrané řádkové vektory matice **A** odpovídající klíčovým slovům jsou vynásobeny předem danou hodnotou. V [28] je navrhována hodnota v rozmezí 2 až 3. Takto je získána vážená matice termů a vět se zvýhodněnými klíčovými slovy dotazu.

Pro názornost si ukažme vážení klíčových slov na příkladu z kapitoly 5.4.4. Matice **A** pro obecnou sumarizaci má tvar

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Řekněme, že uživatelem zadaný dotaz obsahuje klíčová slova „operovat" (2. řádek v matici **A**) a „nemocnice" (4. řádek v matici **A**). Korespondující řádky matice **A** vynásobíme hodnotou např. 3, kterou jsme si zvolili. Výsledná matice bude mít tvar

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 0 & 0 \\ 1 & 1 & 1 \\ 3 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

5.5 Vyhodnocování sumarizovaných textů

Kvalita souhrnů může být vyhodnocována ručně anotátory. Takové hodnocení souhrnů je subjektivní a může se lišit mezi různými anotátory, a tudíž není vhodné např. pro porovnávání různých systémů. Existují i metriky pro automatické (objektivní) vyhodnocování kvality sumarizátory generovaných souhrnů, které nám mohou dát základní představu o výkonnosti sumarizátoru a případných nedostatcích.

U souhrnů můžeme hodnotit lingvistickou kvalitu (gramatickou správnost, neredundantnost, srozumitelnost, souvislost) a informační (obsahovou) kvalitu. Pro hodnocení obsahové kvality souhrnů jsou používány:

- tzv. ko-selekční techniky (pracující s pojmy přesnost, úplnost či f-skóre),
- techniky měřící podobnost (kosinová podobnost, nejdelší společný podřetězec, společné n-gramy, překrývání obsahu a další).

Ko-selekční techniky i techniky měřící podobnost jsou založené na porovnávání s referenčními (ideálními) souhrny.

Ko-selekční techniky umožňují pouze porovnávání extraktů s referenčním extraktem (nikoliv abstraktem) vytvořeným anotátorem. Nejpoužívanějšími metrikami jsou přesnost (*precision*), úplnost (*recall*) a f-skóre (*f-score* nebo *f-measure*). Přesnost (P) je vyjádřena počtem vět vyskytujících se současně v hodnoceném a referenčním extraktu vyděleným celkovým počtem vět v hodnoceném extraktu.

$$P = \frac{|hodnocený \bigcap referenční|}{|hodnocený|}$$

Úplnost (R) je vyjádřena počtem vět vyskytujících se současně v hodnoceném a referenčním extraktu vyděleným celkovým počtem vět v referenčním extraktu.

$$R = \frac{\left|hodnocený \bigcap referenční\right|}{|referenční|}$$

F-skóre je vážený harmonický průměr P a R:

$$F = \frac{2 \cdot P \cdot R}{P + R}.$$

Pro P, R i F platí, že nejlepší hodnoty, které je možné dosáhnout je 1.

Populární metodou pro vyhodnocování kvality souhrnů je ROUGE (Recall-Oriented Understudy for Gisting Evaluation), která automaticky hodnotí podobnost s referenčním souhrnem. ROUGE se používá na TAC (dříve DUC) pro hodnocení soutěžních sumarizátorů. Metoda ROUGE pracuje s n-gramy, což jsou kolekce s n po sobě jdoucími posloupnostmi prvků (zde slova). 1-gramy (unigramy) jsou pak jednotlivá slova a 2-gramy (bigramy) dvojice sousedících slov. Vzorec pro výpočet ROUGE-N má tvar

$$ROUGE - N = \frac{\sum_{S \epsilon \{referenceSumaries\}} \sum_{gram_n \epsilon S} Count_{match}(gram_n)}{\sum_{S \epsilon \{referenceSumaries\}} \sum_{gram_n \epsilon S} Count(gram_n)}.$$

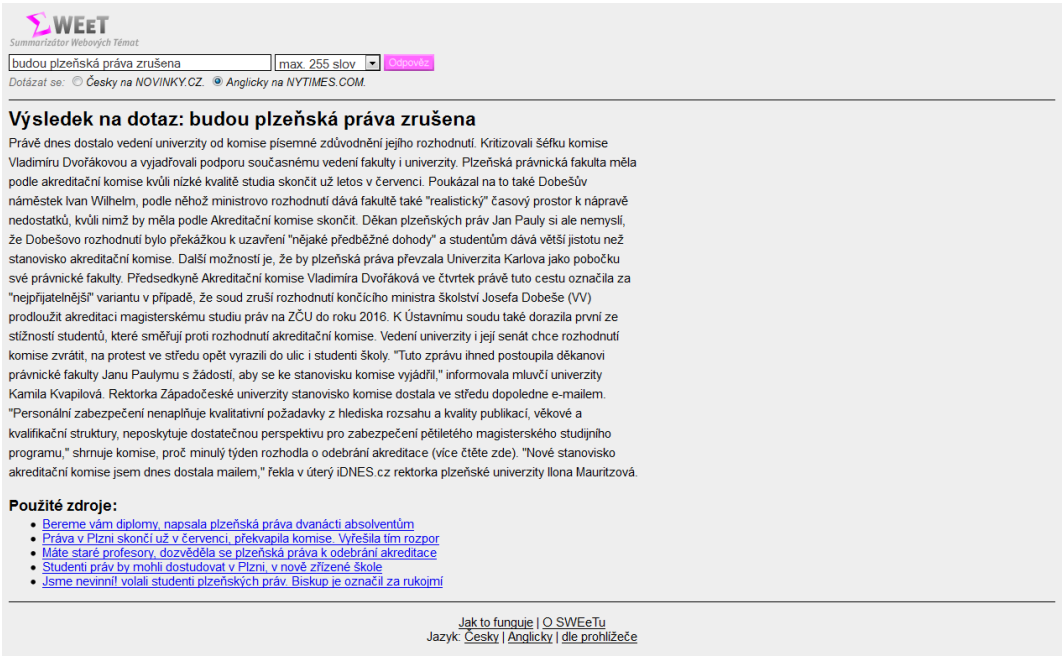
N (resp. n) vyjadřuje délku n-gramu. Čitatel vzorce vyjadřuje počet n-gramů vyskytujících se v referenčních textech ($referenceSumaries$) i v hodnoceném textu. Jmenovatel vyjadřuje počet všech n-gramů vyskytujících se ve všech referenčních textech. Hodnota ROUGE-N logicky nabývá hodnot v rozmezí 0 až 1 a s rostoucím n klesá. Dalšími používanými ROUGE metrikami jsou např. ROUGE-S a ROUGE-SU, které umožňují přeskakování bigramů a unigramů, nebo ROUGE-L a ROUGE-W založené na nejdelších společných sekvencích. Metriky ROUGE jsou podrobně popsány v [37].

Různé způsoby vyhodnocování kvality souhrnů podrobně popisuje [38].

6 Webové sumarizátory

6.1 SWEeT

Zadání této práce bylo inicióváno především z důvodu nedostatků aplikace SWEeT [39], která byla vytvořena v rámci bakalářské práce M. Sloupa. SWEeT je webová aplikace a označuje se jako on-line sumarizátor webů. Pro sumarizaci používá MUSUTELSA sumarizátor. SWEeT používá jako zdroj informací pouze několik zpravodajských serverů. Pro extrakci článků z těchto serverů využívá šablon a musí tedy předem znát strukturu HTML stránek, a tudíž při změně této struktury se stává daný zdroj nepoužitelný. K vyhledávání článků využívá vyhledávačů daných zpravodajských serverů nebo vyhledávače Google. Z vyhledávačů získává výsledky již zmiňovanou technikou web scraping. Další slabinou SWEeTu je primitivní parsování vět a slov, které např. neuvažuje zkratky a data. SWEeT umožňuje sumarizaci anglických a českých textů, ale při jejich parsování neuvažuje rozdíly mezi těmito jazyky. V současnosti není SWEeT funkční, jelikož u všech zdrojových webových stránek došlo ke změně struktury HTML kódu.



Obrázek 6: Sémantický vyhledávač SenseBot.

6.2 SenseBot

SenseBot je sémantický internetový vyhledávač založený na multidokumentové sumarizaci. Generuje textový souhrn z webových stránek, které jsou relevantní k uživatelem zadanému dotazu. Kromě výsledků vyhledání se zobrazuje i tzv. sémantický oblak (*semantic cloud*) konceptů, prostřednictvím kterého lze upřesnit téma, které uživatele v prohledávaných stránkách zajímá. Internetové vyhledávače Google, Bing a Yahoo dodá-

vají k dotazu relevantní webové stránky. SenseBot podporuje vyhledávání v angličtině, francouzštině, němčině a španělštině. Výsledný extrakt se podobá spíše výpisu standardního internetového vyhledávače, ale délka se omezuje na základě počtu vět. Zajímavou funkcí je možnost volby konkrétních webových stránek, které mají být zařazeny do procesu sumarizace. O konkrétní metodě sumarizace se nepíše.

Obrázek 7: Sémantický vyhledávač SenseBot.

7 Realizace

Webová aplikace, která je produktem této diplomové práce dostala název ASI.

7.1 Požadavky

Cílem je vytvořit aplikaci, která se bude snažit na uživatelem zadaný dotaz nalézt relevantní textovou informaci. Aplikace si klade za cíl omezit nutnost návštěvy zdrojové stránky, a tak urychlit proces hledání informace. Jako zdroj informací by měla využívat webové stránky nalezené internetovým vyhledávačem. Aplikace musí být schopna z těchto stránek vyjmout nejdůležitější textový obsah a ten vyhodnotit vybranou sumarizační metodou. Uživateli vrátí nejvýznamnější nalezené věty relevantní k jeho dotazu. Zřetel by měl být brán i na případnou změnu rozhraní použitého internetového vyhledávače. Aplikace bude primárně pro český a anglický jazyk.

Velká většina prací není vyvíjena se zřetelem na znovupoužitelnost, a tak často dochází k opětovné implementaci různých algoritmů. Místo toho, aby se vývojář nebo vědec zabýval vlastním problémem, musí často řešit i věci okolo, jako je např. uživatelské rozhraní. Znovupoužitelnost je tedy dalším důležitým specifikem této práce. Budeme chtít, aby aplikace umožňovala snadné nasazení dalších algoritmů a zároveň, aby jednotlivé její části bylo možné použít i v jiných pracích.

Další specifické požadavky vyplynuly z nedostatků systému SWEeT. Na základě těchto nedostatků by výsledná aplikace měla umožňovat:

- Snadnou změnu používaného internetového vyhledávače.
- Extrahovat text z jakékoliv HTML stránky.
- Jednoduché rozšíření o další sumarizační metody.
- Pokročilejší parsování vět a slov.

Neméně podstatným požadavkem je, aby aplikace byla postavená na volných technologiích, a tím byl i usnadněn případný budoucí vývoj.

7.2 Technologie

Při volbě technologií bylo bráno v úvahu, že výsledkem bude **webová aplikace**, jelikož se jedná o běžnou platformu pro internetové vyhledávače. Aplikace tak bude přístupná z jakéhokoliv zařízení s připojením k Internetu a internetovým prohlížečem. Programovací jazyk a další podpůrné technologie by měly být multiplatformní. To omezuje výběr možných programovacích jazyků, ale zároveň dovoluje větší volnost v dalším vývoji. A jelikož se jedná o experimentální projekt, bude snaha o využití nových technologií. Primárním kódováním aplikace i zdrojových kódů je UTF-8.

Programovací jazyk

Při výběru vhodného programovacího jazyka jsem uvažoval i dostupnost nástroje pro zpracování přirozeného jazyka a v neposlední řadě i rozšířenost v dané oblasti vývoje a také na domovské fakultě. Proto byl vybrán programovací jazyk **Java**, který je hojně využíván v oblasti zpracování přirozeného jazyka a nabízí kvalitní technologie pro vývoj webových aplikací. Jedinou vážnou alternativou k jazyku Java byl Python, který je téže multiplatformní a téže hojně využíván v oblasti zpracování přirozeného jazyka.

Internetový vyhledávač

Pro internetové vyhledávání jsem zvolil vyhledávače Bing i Google. Vzhledem k požadavku vyhledávání v češtině a angličtině nebylo mnoho možností na výběr (viz kapitola 2.2.1). Testování zmíněné v kapitole 1 ukázalo, že nabízejí kvalitní výsledky vyhledávání, i když Bing ukázal slabiny v případě češtiny.

Filtrace obsahu z webových stránek

Kapitola 3 uvádí metody i nástroje pro extrakci hlavního obsahu z webových stránek. Byl vybrána sada algoritmů Boilerpipe, která je implementována v programovacím jazyce Java a dle zdrojů uvedených v kapitole 3.3 dosahuje dobrých výsledků.

Zpracování přirozeného jazyka

GATE verze 7 byl použit jako nástroj pro podporu zpracování přirozeného jazyka. Jak je zmíněno v kapitole 4.1, GATE nabízí oproti jiným nástrojům velké množství pluginů a také popularita a dlouhodobý vývoj GATE v oblasti zpracování přirozeného jazyka vedla k jeho volbě. GATE je navíc používán i v dalších projektech na domovské fakultě a rozšíření o nové pluginy podporující češtinu může být do budoucna užitečné.

Sumarizační metoda

Jako hlavní metodu pro sumarizaci jsem zvolil latentní sémantickou analýzu. Důvodem výběru LSA byla především dostupná kvalitní literatura a původně i možnost porovnání s aplikací SWEeT. Při implementaci byly využity poznatky z vývoje MUSUTELSA sumarizátoru.

Prezentační vrstva

JavaServer Faces (zkráceně JSF) verze 2.1 byl vybrán jako technologie resp. framework pro vývoj webové aplikace. JSF je součástí Java EE (Java Enterprise Edition). Výhodou JSF je kromě oddělení definice uživatelského rozhraní od aplikační logiky i možnost transparentního a rychlejšího vývoje než nabízejí Servlety ve spojení s JavaServer Pages (JSP). Požadavky jsou totiž mapovány přímo na komponenty, zde JavaBeans, které jsou

v případě JSF nazývány ManagedBeans (můžeme se setkat i s označením BackingBeans). Aplikace se tak stává přehlednější. Více informací o JSF může poskytnout např. [40].

Na straně klienta je použito HTML5, CSS3 a JavaScript (resp. jQuery[46] knihovna).

7.3 Vývoj

Aplikace byla vyvíjena a testována na serveru Apache Tomcat 7. Apache Tomcat je certifikován jako plně J2EE[47] kompatibilní.

Vývojové prostředí Eclipse[48] pro Java EE umožnilo urychlení implementace i průběžné nasazování na server za účelem testování. Další nástroje, které také pomáhaly při vývoji, byly pluginy do internetového prohlížeče Mozilla Firefox. Konkrétně Firebug[49], LiveHTTPHeaders[50] a Html Validator[51].

Pro správu verzí byl využíván Git[52] na serverech poskytovatele Assembla[53], který nabízí i další nástroje pro podporu vývoje a řízení projektů.

7.4 Architektura

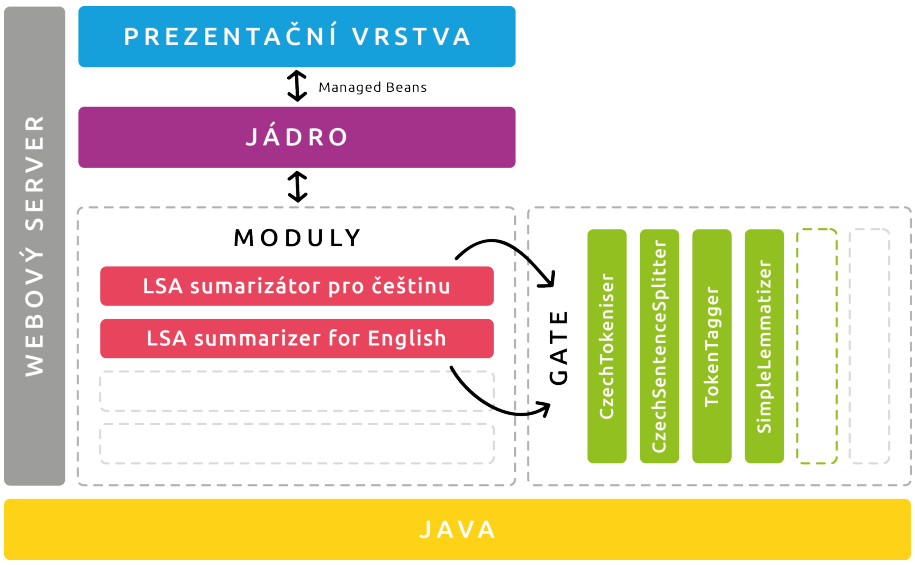
Aplikace je rozdělena do tří hlavních částí: prezentační vrstvy, jádra a modulů. Architekturu popisuje obrázek 8. Prezentační vrstva a jádro využívá pouze prostředků Java EE a nepotřebují ke svému chodu žádné dodatečné knihovny. Moduly jsou nezávislé prvky implementující v sobě celý proces vyhledávání. Modulu jsou předány pouze vstupní data, na jejichž základě se provede zpracování a poté poskytne výstup. Zbylé části jako je prezentační vrstva a jádro pouze umožňují interakci modulů s uživatelem.

Činnost celé aplikace probíhá následovně. Po spuštění serveru načte jádro dostupné moduly a provede jejich inicializaci. Po inicializaci všech modulů je aplikace připravená přijímat uživatelské dotazy. Při zadání dotazu:

1. Prezentační vrstva přijme požadavek a předá jej jádru.
2. Jádro vybere požadovaný modul a předá mu data ke zpracování.
3. Modul provede zpracování a vrátí výstup, který je předán zpět prezentační vrstvě.

Takto navržená architektura umožňuje velmi snadné rozšiřování o další moduly bez nutnosti zásahu do kódu jádra či prezentační vrstvy. Další moduly mohou implementovat odlišné algoritmy a zároveň nejsou nijak omezeny ve využívání dalších knihoven či nástrojů.

[46] http://www.jquery.com
[47] Java Enterprise Edition určená pro serverové aplikace.
[48] http://www.eclipse.org
[49] http://getfirebug.com
[50] http://livehttpheaders.mozdev.org
[51] http://users.skynet.be/mgueury/mozilla/
[52] Distribuovaný systém pro správu verzí.
[53] http://www.assembla.com



Obrázek 8: Architektura aplikace ASI.

V počátečních fázích vývoje aplikace byla snaha o použití technologií pro podporu (vývoje) modulární architektury (např. OSGi[54] a jiné), nakonec bylo shledáno za rozumné, žádnou takovou technologii nepoužít. Důvodem je zachování přehlednosti a jednoduchosti aplikace. Jakákoliv další technologie by zároveň prodloužila i rychlost pochopení aplikace v případě dalšího vývoje.

7.5 Jádro

Jádro tvoří základ aplikační vrstvy a jeho úkolem je interakce s prezentační vrstvou a moduly. Poskytuje prostředky pro manipulaci s moduly. Součástí jádra jsou i ManagedBeans, které řídí interakci mezi prezentační a aplikační vrstvou a tak i chování aplikace. Zdrojové kódy jádra se nachází v balíčku `cz.zcu.asi.core`.

7.6 Moduly

Moduly jsou autonomní prvky implementující v sobě celý proces vyhledávání. Autonomností je míněna nezávislost na jádru. Modul je tak možné použít i v jakékoliv jiné aplikaci. Moduly mohou ke své činnosti používat různé knihovny, frameworky nebo např. GATE atd.

Moduly mají pevně definované rozhraní. Balíček `cz.zcu.asi.module` obsahuje rozhraní `Module`, které musí moduly aplikace ASI implementovat. Kromě vstupu a výstupu (viz níže) hlavní metody definuje rozhraní `Module` i metody pro získání názvu a popisu modulu, které jsou užitečné především pro výběr modulu v uživatelském rozhraní aplikace ASI. Operace pro inicializaci modulů se implementují do konstruktoru třídy implementující rozhraní `Module`.

[54] Specifikace dynamického modulárního systému pro programovací jazyk Java.

Vstupem modulu jsou:

- dotaz,
- počet zdrojů (např. webových stránek),
- délka extraktu (např. počet vět).

Výstupem modulu jsou:

- všechny vstupní data uvedená výše,
- skutečný počet použitých zdrojů,
- doba zpracování,
- seznam vět včetně odkazů na zdroje (věty jsou rozděleny na slova a zbylé znaky).

Moduly jsou do aplikace ASI zaváděny do složky `WEB-INF/modules` jako JAR[55] soubory. Jediný JAR soubor může obsahovat i více modulů. Aby jádro aplikace ASI bylo schopné moduly správně načíst, je důležité, aby moduly implementovaly rozhraní `Module` a v JAR souboru existovala složka `META-INF/services` s textovým souborem `cz.zcu.asi.module.Module` obsahující plně kvalifikované názvy tříd jednotlivých modulů (na každém řádku jeden). Např. v současné verzi aplikace ASI existuje jediný JAR soubor implementující dva moduly. Soubor `cz.zcu.asi.module.Module` ve složce `META-INF/services` daného JAR souboru obsahuje:

```
cz.zcu.asi.module.lsa.czech.CzechLsaSummarizer
cz.zcu.asi.module.lsa.english.EnglishLsaSummarizer
```

Balíček `cz.zcu.asi.module` také obsahuje třídu `ModuleLoader`, která dovoluje zavádění modulů z JAR souborů. Výhodou tohoto přístupu je, že pro zavedení dalšího modulu do aplikace ASI není nutný opětovný překlad ani žádná konfigurace či zásah do kódu aplikace ASI. Pouze je nutné JAR soubor nahrát do složky `WEB-INF/modules` a případně další JAR soubory a jiná data nutná pro chod modulů.

Implementované moduly

V aktuální verzi jsou implementovány dva moduly pro vyhledávání založené na LSA sumarizaci, a to pro češtinu a angličtinu. Činnost obou modulů je shodná a liší se pouze ve zpracování přirozeného jazyka, jelikož tyto operace jsou jazykově závislé. Popišme si činnost těchto modulů od vstupu až po výstup.

1. Modulu jsou předány vstupní data (dotaz, počet zdrojů a délka extraktu).
2. Vyhledání relevantních webových stránek k danému dotazu.

[55] JAR neboli Java Archive je komprimovaný archivní soubor pro programovací jazyk Java.

3. Stažení nalezených webových stránek a filtrace obsahu.

4. Zpracování textů z webových stránek pro vlastní proces sumarizace. Zpracováním textů je na mysli:

   (a) tokenizace,

   (b) identifikace vět,

   (c) lemmatizace,

   (d) označení stopslov,

   (e) označení klíčových slov dotazu.

5. Multidokumentová na dotazu založená LSA sumarizace a vytvoření souhrnu (extraktu).

6. Předání souhrnu a další dat na výstup.

Ke zpracování přirozeného jazyka využívají moduly aplikace ASI framework GATE. Většina z výše uvedených operací byly implementovány jako komponenty (Processing Resources) GATE. Dokumenty jsou pro operace 3 a 4 zpracovávány paralelně.

Příklad hlavní třídy modulu pro češtinu je uveden v příloze na straně 71.

7.7 GATE pluginy

Framework GATE byl již stručně popsán v kapitole 4.1.1. GATE je v modulech aplikace ASI používán především kvůli dostupným pluginům s komponentami pro zpracování přirozeného jazyka, ale také jako nástroj pro podporu modulárnosti samotných modulů aplikace ASI. V modulech aplikace ASI jsou konkrétně používány tři GATE pluginy. Dva základní pluginy ANNIE a Tools pro angličtinu, které jsou součástí distribuce GATE, a nově implementovaný plugin ASI vytvořený pro účely této práce. Pluginy se nachází v ekvivalentně nazvaných složkách ve složce `WEB-INF/plugins`. Ve složce každého pluginu je umístěn soubor `creole.xml`, který deklaruje komponenty nacházející se v daném pluginu a slouží také jako konfigurační soubor komponent.

Použití nástrojů GATE vyžaduje inicializaci frameworku GATE, načtení registrace pluginu, instanciaci komponenty vybrané dle plně kvalifikovaného názvu Java třídy a pak již je možné danou komponentu používat. V příloze na straně 71 uvedený zdrojový kód hlavní třídy modulu pro češtinu to demonstruje.

V následujících kapitolách jsou popsány komponenty pluginů ANNIE, Tools a ASI, které jsou v modulech aplikace ASI používány.

7.8 Vyhledávání

Vyhledávání prostřednictvím internetových vyhledávačů Bing a Google je implementováno jako GATE komponenty implementující rozhraní `WebSearch` v pluginu ASI. Rozhraní `WebSearch` definuje metody pro nastavení počtu výsledků vyhledávání a jazyka, pro předání dotazu a metodu pro získání výsledků vyhledávání.

Vyhledávání prostřednictvím internetového vyhledávače Bing implementuje GATE komponenta s názvem `BingWebSearch` v pluginu ASI. Pro vyhledávání je používána Java knihovna bing-search-java-sdk[56], která využívá Bing API verze 2[57] pro dotazování a získávání výsledků vyhledávání. Knihovna využívá datový formát JSON. Bing API verze 2 poskytuje i data ve formátu XML a SOAP. Bing měl být primárním vyhledávačem obou modulů aplikace ASI, ale při dokončování práce Microsoft oznámil plánované zpoplatnění Bing API.

GATE komponenta implementující vyhledávání Google vyhledávačem se nazývá `GoogleWebSearch` a je také součástí GATE pluginu ASI. Jak bylo uvedeno v kapitole 2.2.1, internetový vyhledávač Google nedává k dispozici žádné API, a tak jediným způsobem využívání jeho služeb je web scraping. K tomu je využívána knihovna jsoup[58], která dovoluje stahování HTML stránek, vyhledáváni a extrakci dat z DOM[59] struktury nebo prostřednictvím CSS[60] selektorů. Nevýhodou tohoto řešení je poměrně častá změna HTML kódu vyhledávače Google, což by způsobilo nefunkčnost modulu. Proto jsou selektory k odkazům s výsledky vyhledávání částečně konfigurovatelné. Pokud by ale změna rozhraní vyhledávače Google byla rozsáhlejší a změna konfigurovatelných selektorů nebyla dostačující, existuje stále možnost úpravy existující nebo implementace zcela nové GATE komponenty.

V případě potřeby zavedení nové komponenty pro vyhledávání do modulu ASI není třeba nijak zasahovat do kódu modulu, jelikož ten používá komponentu `WebSearchWrapper`, která pouze obaluje skutečnou komponentu pro vyhledávání, kterou volí dle konfiguračního souboru pluginu ASI, ve kterém je nastavena plně kvalifikovaná cesta ke GATE komponentě pro vyhledávání. Google je nastaven jako primární vyhledávač.

7.9 Stahování a filtrace

Při stahování webových stránek se ukázalo několik úskalí, které bylo nutné vyřešit:

- Přesměrování na jinou adresu.
- Odezva některých webových serverů byla příliš dlouhá.

[56] http://code.google.com/p/bing-search-java-sdk/
[57] http://msdn.microsoft.com/en-us/library/dd251056.aspx
[58] http://jsoup.org
[59] DOM neboli Document Object Model je objektová reprezentace HTML nebo XML dokumentu.
[60] CSS (Cascading Style Sheets) je jazyk pro stylování především HTML nebo XML stránek.

- Ve výsledcích vyhledávání nejsou pouze HTML stránky, ale také PDF, DOC a jiné datové formáty.

- Správné rozpoznání znakové sady.

Všechny výše uvedené problémy umožnila vyřešit knihovna jsoup, která je používána i pro získávání výsledků vyhledávače Google. Původně byla ale používána vlastní implementace stahování, která filtrovala HTML stránky na základě HTTP[61] hlaviček, nastavovala i maximální dobu odezvy, umožňovala přesměrování a znakovou sadu rozpoznávala na základě HTTP hlavičky nebo HTML meta elementu.

K získání hlavních částí obsahu jako prostý text ze stažených webových stránek je využívána knihovna Boilerpipe a konkrétně algoritmus označovaný jako CanolaExtractor (viz kapitola 3.2.4). Na základě počtu slov a hustoty odkazů identifikuje nežádoucí obsah, který odstraní včetně veškerých HTML značek. Takto získaný prostý text dosahuje velmi dobré kvality, ale stále může obsahovat texty, které nejsou součástí hlavního obsahu, jako např. odkazy pro editaci na Wikipedii, seznamy souvisejících článků, do článku včleněné reklamy atd. Proto bylo implementováno dodatečné filtrování, které vyhodnocuje, jestli daný extrahovaný odstavec obsahuje věty (což je žádoucí), a také kontroluje délku odstavce. Boilerpipe extrahuje textový obsah každého blokového HTML elementu na samostatný řádek. Každý takový řádek je považován za odstavec. Pokud daný odstavec obsahuje text kratší než je definováno (nyní 30 znaků), je odstraněn. Je-li splněno kritérium délky odstavce, pak je na základě poměru znaků vyskytujících se běžně ve větách a znaků, které nejsou běžně ve větě, rozhodnuto, zda bude odstavec ponechán nebo odstraněn.

Jednou z nevýhod latentní sémantické analýzy je výpočetní náročnost singulárního rozkladu matic. S rostoucí dimenzí vstupní matice **A** roste i doba výpočtu. Velmi často je vstupem rozsáhlá stránka s velkým množstvím textu, která značně zvětší dimenze matice, jelikož dimenze matice **A** je závislá na počtu termů a počtu vět. V takovém případě může SVD trvat i několik desítek sekund a v extrémních případech i několik minut. Abychom omezili tento problém, omezuje se délka vstupního textu z každé webové stránky. V aktuální verzi aplikace ASI je extrahováno prvních 1000 tokenů, kde token je textový řetězec oddělený mezerou. To je činěno i na základě poznatku Edmundsona [19]. Edmundson ve své práci představil hypotézu o velké informační hodnotě frází a vět na začátku článků. I průběžné testování aplikace ASI ukázalo, že převážná většina výsledků sumarizace pocházela z první poloviny extrahovaných textů.

Výše popsané stahování webových stránek i filtrace obsahu jsou implementovány jako GATE komponenta `BoilerplateRemover`, která je součástí pluginu ASI.
`BoilerplateRemover` umožňuje prostřednictvím konfiguračního souboru pluginu ASI nastavení: maximální doby čekání na odezvu, počet tokenů zkráceného textu, minimální

[61] HTTP (HyperText Transfer Protocol) je protokol aplikační úrovně používaný pro přenos hypertextových dokumentů.

délky odstavce, znaků vyskytujících se běžně ve větách a také znaků, které nejsou žádané ve větách.

7.10 Příprava pro sumarizaci

Po procesu vyhledávání, stahování a filtrace obsahu webových stránek je získán korpus skládající se z dokumentů obsahujících text z jednotlivých webových stránek. Všechny dokumenty v korpusu projdou před vlastním procesem sumarizace nezbytnými operacemi. Pro přípravu textů pro sumarizaci jsou používány existující, ale i nově implementované nebo upravené komponenty pluginů GATE. Vstupem i výstupem každé komponenty je GATE dokument, který reprezentuje jednu webovou stránku s jejím extrahovaným textem. K dokumentu GATE je možné přidávat libovolné anotace. Cílem následujících úprav je přidání anotací reprezentujících tokenizovaný text, věty, lemmata slov, označující stopslova a klíčová slova. Příklad anotací vyprodukovaných v tomto kroce je uveden v příloze na straně 74.

7.10.1 Tokenizace

Pro tokenizaci textů v angličtině je využívána komponenta `DefaultTokeniser` ze základního GATE pluginu s názvem ANNIE. `DefaultTokeniser` provádí tokenizaci ve dvou fázích a využívá k tomu dvou typů gramatik. V první fázi je text rozdělen na základě gramatiky, která se nachází v souboru s příponou `*.rules`. V tomto textovém souboru je definována gramatika, na základě které je provedena základní tokenizace textu na slova, čísla, mezery, interpunkci a symboly. Takto rozdělený (anotovaný) text však neuvažuje speciální případy, které se v angličtině vyskytují, jako jsou slova „don't", „2nd" apod. Pro tento účel je v druhé fázi zpracování využíváno tzv. JAPE (Java Annotation Patterns Engine)[62], což jsou zjednodušeně řečeno regulární výrazy prováděné nad anotacemi získanými v předchozí fázi. Gramatiky JAPE se nachází v textových souborech s příponou `*.jape`. Gramatiku lze rozdělit do několika fází a každé pravidlo je tvořeno vzorem (regulárním výrazem) a akcí, kterou lze definovat běžným Java kódem. S použitím JAPE gramatik je tak možné ošetřit speciální případy tokenů jako „2nd", který by jinak byl rozdělen na dva tokeny „2" a „nd".

Pro češtinu je používána GATE komponenta, která je postavena na komponentě `DefaultTokeniser` z pluginu ANNIE. Komponenta je nazvána `CzechTokeniser` a je součástí pluginu ASI. Gramatiky pro tokenizaci jsou používány shodné jako v komponentě `DefaultTokeniser` z pluginu ANNIE, jelikož správně tokenizovaly text i pro češtinu. Nebyl použit původní `DefaultTokeniser` z pluginu ANNIE, aby existovala možnost úpravy v případě nutnosti optimalizace pro češtinu.

[62] http://gate.ac.uk/sale/tao/splitch8.html#chap:jape

7.10.2 Identifikace vět

Pro identifikaci vět v textech v angličtině je využívána komponenta `SentenceSplitter` z pluginu ANNIE. `SentenceSplitter` vyžaduje již tokenizovaný text, jehož anotace musí být shodné s těmi, které produkují předchozí zmiňované komponenty. V takto tokenizovaném (resp. anotovaném) vstupním dokumentu je `SentenceSplitter` schopen identifikovat věty. K identifikaci vět využívá také JAPE gramatiky využívající anotací z procesu tokenizace. JAPE gramatiky poskytují dostatečnou volnost, aby bylo možné ošetřit speciální případy, jako jsou zkratky, IP adresy atd. Kromě vzorů (regulárních výrazů) používá k identifikaci zkratek i slovník.

Pro češtinu je používána GATE komponenta, která je postavena na komponentě `SentenceSplitter` z pluginu ANNIE. Komponenta je nazvána `CzechSentenceSplitter` a je součástí pluginu ASI. JAPE gramatiky a slovník zkratek byly upraveny a přizpůsobeny pro češtinu. Oproti angličtině bylo nutné v češtině navíc uvažovat textové řetězce reprezentující datum nebo pořadové číslovky.

7.10.3 Lemmatizace

Pro angličtinu je využíván morfologický analyzátor `Morph` ze základního pluginu Tools. Komponenta `Morph` je založena na jednoduché slovníkové lemmatizaci, ale dovoluje i lemmatizaci uvažující slovní druh slova. Slovníky jsou definovány v souborech s příponou `*.dat`. K lemmatizci vyžaduje již tokenizovaný text stejně jako `SentenceSplitter` pro identifikaci vět. Pravidla uvažující slovní druhy jsou definovány v souboru s příponou `*.rul`. Ty lze využít pouze pokud anotace tokenizovaného textu dokumentu nesou příslušnou informaci o slovním druhu. `POSTagger`, který je součástí pluginu ANNIE, lze využít pro značení slovních druhů, ale vyžaduje jako vstup již tokenizovaný dokument navíc s identifikovanými větami. S uvažováním slovního druhu slova je tak možné rozlišit např. slovo „singing", které může být podstatným jménem i slovesem a v obou případech má jiný základní tvar.

Lemmatizaci textů v češtině zajišťuje nově implementovaná komponenta `SimpleLemmatizer` obsažená v pluginu ASI. Implementuje jednoduchou slovníkovou lemmatizaci bez uvažování slovních druhů. Slovník lemmat pro češtinu byl převzat z již zmíněné aplikace SWEeT a nachází se v souboru `czechLemmas.txt`. `SimpleLemmatizer` respektuje značení v anotacích, které provádí komponenta `Morph`.

7.10.4 Značení stopslov

Pro značení stopslov se používá nově implementovaná GATE komponenta `TokenTagger` z pluginu ASI. `TokenTagger` využívá GATE komponentu `DefaultGazetteer`, která slouží k vyhledávání podřetězců v textu. Vyhledávané řetězce se pro `DefaultGazetteer` definují v textovém souboru většinou s příponou `*.lst`. Ke své činnosti `DefaultGazetteer` vyžadují další soubor (s příponou `*.def`), ve kterém jsou mapovány soubory na název,

který bude použit v anotaci v případě shody s některým z řetězců v daném souboru. Pro názornost předpokládejme existenci souboru `stop.lst` obsahující stopslova a soubor `lists.def`, ve kterém je souboru `stop.lst` přidělena značka „stop". Pokud je v textu nalezeno jakékoliv slovo ze souboru `stop.lst`, pak `DefaultGazetteer` vytvoří novou anotaci, ve které se bude nacházet značka „stop". `DefaultGazetteer` je vhodným nástrojem pro rozpoznávání pojmenovaných entit.

`TokenTagger` upravuje činnost komponenty `DefaultGazetteer` a namísto vytváření nových anotací vkládá značky (jako např. „stop") k anotacím tokenizovaného textu. `TokenTagger` tudíž vyžaduje na vstupu tokenizovaný dokument. `TokenTagger` je možné používat i pro jiné úkoly, ale v aplikaci ASI je používán pro identifikaci anglických stopslov. Slovník stopslov je definován shodně jako pro `DefaultGazetteer`.

Pro češtinu se používá `CzechTokenTagger`, který se nachází také v pluginu ASI. `CzechTokenTagger` je postaven na komponentě `TokenTagger`, ale samozřejmě používá soubory obsahující seznam českých stopslov.

7.10.5 Značení klíčových slov

Klíčová slova z dotazu zadaného uživatelem jsou lemmata, která zároveň nejsou stopslovy. Klíčová slova dotazu lze identifikovat použitím výše uvedených GATE komponent pro tokenizaci, lemmatizaci a značení stopslov. Takto upravený dotaz resp. dokument s dotazem bude nést i anotace, které jsou použity pro určení klíčových slov.

GATE dokument (reprezentující webovou stránku), ve kterém budou označena klíčová slova, musel projít operacemi tokenizace, lemmatizace a značení stopslov. Klíčová slova dotazu jsou porovnána se všemi slovy (přesněji lemmaty) v dokumentu a v případě shody je k anotaci slova přidána značka identifikující klíčové slovo.

Tato operace se v modulech aplikace ASI liší pro angličtinu a češtinu pouze použitými GATE komponentami. Jedná se o poslední krok úpravy dokumentů, než nad nimi bude provedena sumarizace.

7.11 Sumarizace

Vlastní proces sumarizace je implementován jako GATE komponenta `LsaSummarizer` v pluginu ASI. `LsaSummarizer` je jazykově nezávislý, jelikož na vstupu požaduje již předzpracovaný korpus s dokumenty obsahující text z jednotlivých nalezených webých stránek. Nad každým dokumentem v korpusu musela být předem provedena tokenizace, identifikace vět, lemmatizace, označení stopslov a označení klíčových slov. Všechna tato data jsou využívána v procesu LSA sumarizace. Implementovaná LSA sumarizace vychází z velmi přínosných poznatků diplomové práce M. Křišťana [28].

7.11.1 Značení slov a vět

Jak se popisuje v kapitole 5.4.1, vstupní matice **A** LSA sumarizace mapuje termy na věty. K tomu je třeba přidělit každému termu i větě jedinečný identifikátor. Identifikátor pro termy i věty musí být jedinečný v rámci celého korpusu a ne pouze v rámci dokumentu, jelikož je prováděna multidokumentová sumarizaci. Jako termy jsou v implementované LSA sumarizaci používány základní tvary slov (lemmata), které nejsou stopslovem. Díky tomu lze identifikovat i slova různých gramatických tvarů. Každý různý term je tak jednoznačně určen identifikátorem a stejně tak věty. Např. v příkladu z kapitoly 5.4.4 je uveden číslovaný seznam vět a z nich získaných termů. Čísla seznamů lze považovat za jejich identifikátory.

7.11.2 Vážení slov a vytvoření matice A

Implementované vážení slov a tvorba matice **A**, která je vstupem pro singulární rozklad, se shoduje s řešením popisovaném v kapitole 5.4.5. Je vytvořena matice o rozměrech $m \times n$, kde m je počet různých termů nalezených v korpusu a n jsou věty všech dokumentů v korpusu. V předchozím kroku získalo každý term identifikátor a stejně tak věty dostaly svůj identifikátor, aby bylo možné mapování v matici **A** a následně v maticích po SVD. Prvky prázdné matice **A** mají hodnotu 0. Matice je plněna po větách. Z anotace slova nacházejícího se ve větě je získán jeho identifikátor a s použitím identifikátoru věty, ve které se slovo nachází, je určena souřadnice v matici, kam je uložena hodnota 1, pokud není uvažována vícečetnost. Uvažuje-li se vícečetnost, je k původní hodnotě na dané souřadnici matice přičtena hodnota 1. Tak by hodnota v matici odpovídala počtu výskytů slov s daným lemmatem ve větě. V základním nastavení není vícečetnost uvažována, a tak hodnota pouze vyjadřuje výskyt, ale nikoliv četnost slova s daným termem věty.

Nakonec jsou všechny řádkové vektory odpovídající lemmatům, které jsou i klíčovými slovy vynásobeny hodnotou 3, stejně jako je popsáno v kapitole 5.4.5.

Uvažování vícečetnosti i násobitele klíčových slov je možné v pluginu ASI konfigurovat.

7.11.3 Singulární rozklad matice a k-aproximace

V tomto kroku je proveden singulární rozklad matice **A**. Singulární rozklad byl podrobně popsán v kapitole 5.4.2. Pro singulární rozklad matice **A** je používána Java knihovna JAMA[63]. Jsou získány matice **U**,**Σ** a **V**. Dále se provede k-aproximace matic **Σ** a **V**, kde $k = 5$. Tato hodnota je vybrána na základě poznatků (doporučení) z [28]. Je získána redukovaná matice $\mathbf{\Sigma}_5$ a $\mathbf{V}_5$. Matice**U** není v dalších výpočtech používána.

Dimenzi k-aproximace (číslo k) lze v pluginu ASI nastavit.

[63] http://math.nist.gov/javanumerics/jama/

7.11.4 Výběr vět do extraktu

Pro výběr vět do extraktu je používána metoda autorů Steinberger a Ježek, která je popsána v kapitole 5.4.3. Metoda počítá váhu každé věty jako velikost vektoru věty v redukované matici $\mathbf{V}_5$, ale navíc uvažuje důležitost konceptů matice $\mathbf{\Sigma}_5$. Ale jak již bylo zmíněno, nevýhodou této metody je, že upřednostňuje dlouhé věty. Řešení nabídl M. Křišťan ve své práci [28], kde uvádí vzorec

$$s'_i = \frac{s_i}{d_i^p},$$

kde s_i je váha věty vypočítána metodou autorů Steinberger a Ježek, d_i je délka věty s indexem i a p je mocnina, s jejíž rostoucí hodnotou jsou více zohledňovány krátké věty. s'_i je nová váha věty. V [28] je doporučeno p v rozmezí $0,2$ až $0,4$. Při průběžném testování aplikace ASI se ukázalo, že dosahuje velmi dobrých výsledků i při $p = 0$.

Dle váhy uspořádané věty lze vybrat do výsledného souhrnu (extraktu). V kapitole 5.4.5 byl ale zmíněn problém podobnosti vět, který by neměl být především v multidokumentové sumarizaci přehlížen. Během průběžného testování se ukázalo, že výsledné extrakty příliš často obsahovaly krátké věty s klíčovými slovy, které měly odlišné pouze jeden až tři termy a nepřinášely žádnou přidanou informační hodnotu. Po implementaci kosinové kontroly podobnosti a s příslušným nastavením se podařilo částečně eliminovat tento jev. Práh kosinové podobnosti byl volen tak, aby výsledný souhrn příliš nepotlačoval nejvíce relevantní věty, které většinou obsahují nejvíce klíčových slov, ale zároveň souhrn dostatečně zachytil téma zadaného dotazu.

V komponentě `LsaSummarizer` je nyní nastavena hodnota p na $0,1$ a prahový úhel podobnosti vět na 20°. V [28] je doporučený prahový úhel 60°. V pluginu ASI je možné tyto hodnoty přenastavit.

7.11.5 Parametry LSA sumarizace

Výše byly zmíněny možnosti konfigurace komponenty `LsaSummarizer` pluginu ASI. Pro přehlednost je uvedena tabulka 4 s přehledem všech možností konfigurace, jejich výchozích hodnot a stručného popisu.

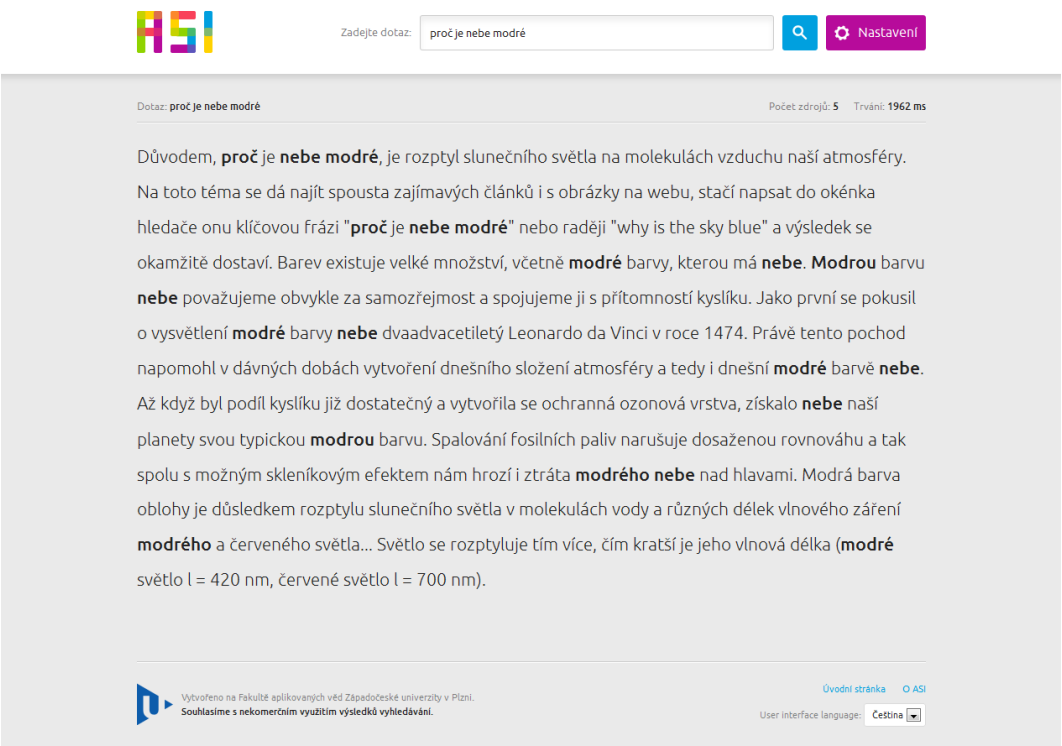
Parametr	Výchozí hodnota	Popis
considerMultipleWords	false	Uvažování vícečetnosti při vážení slov. Základní nastavení neuvažuje vícečetnost.
wordWeight	1	Váha slova při výskytu ve větě. Při povolené vícečetnosti se tato hodnota přičítá, jinak je přiřazena.
keywordMultiplier	3	Hodnota, kterou se násobí váhy klíčových slov.
dimension	5	Dimenze k-aproximace (číslo k).
lengthPower	0,1	Koeficient (p) přepočtu délky věty. Vyšší hodnota zohledňuje více krátké věty.
cosineThreshold	20	Prahový úhel podobnosti vět.

Tabulka 4: Parametry LSA sumarizace.

7.12 Prezentační vrstva

Prezentační vrstva je postavena na technologii JSF verze 2.1. Uživatelské rozhraní je implementováno v souborech s příponou `*.xhtml` a aplikační logiku zajišťují ManagedBeans umístěné v balíčku `cz.zcu.asi.core`. Především použitím JSF bylo docíleno vysoké přehlednosti implementace. Na straně klienta jsou používány technologie HTML5, CSS3 a JavaScript (resp. knihovna jQuery).

Při návrhu uživatelského rozhraní byl kladen velký důraz na přehlednost a jednoduchost užívání aplikace ASI (viz obrázek 9). Oproti běžným internetovým vyhledávačům nabízí aplikace ASI výsledky vyhledávání jako souvislý text se zvýrazněnými klíčovými slovy dotazu. Po najetí kurzoru na text, je věta, na které se nachází kurzor, zvýrazněna, což usnadňuje orientaci v textu. Každá věta odkazuje na webovou stránku, ze které byla extrahována. Uživateli je umožněno kromě nastavení dotazu i volba modulu pro sumarizaci, maximální počet zdrojových webových stránek a délka výsledného extraktu ve větách. Uživatelské rozhraní reaguje na chybové stavy aplikace i na chybně zadané vstupy.



Obrázek 9: Webová stránka s výsledky vyhledávání aplikace ASI.

Také byla provedena internacionalizace uživatelského rozhraní a kromě již implementované češtiny a angličtiny, je možné snadné rozšíření o další jazyky. Přidání dalšího překladu uživatelského rozhraní spočívá pouze v definici nového souboru `locale_L.properties` (místo `L` je zkratka jazyka) s překladem pro daný jazyk a zápisu zkratky `L` identifikující jazyk do souboru `faces-config.xml` ve složce `WEB-INF`.

Vzhledem k použití nových technologií jako jsou HTML5 a CSS3, nebyla prováděna optimalizace pro staré internetové prohlížeče. Dalším důvodem, proč bylo shledáno za zbytečné optimalizovat pro staré prohlížeče, je fakt, že se jedná o experimentální systém a v akademickém prostředí se předpokládá používání moderních internetových prohlížečů.

8 Testování

Nebylo shledáno za nutné měřit kvalitu samotného LSA sumarizátoru některou z metod uvedených v kapitole 5.5 a to z několika důvodů. V modulech implementovaná LSA sumarizace vychází z poznatků získaných z MUSUTELSA sumarizátoru, kde taková měření již byla provedena. Navíc parametry implementovaného LSA sumarizátoru jsou odlišné než optimální doporučené M. Křišťanem v [28]. Sumarizace má zde dát pohled na všechny nalezené webové stránky vzhledem k dotazu, ale zároveň zachovat nejvíce relevantní, i když možná podobné věty, které by mohly být přímou odpovědí k dotazu. Dá se předpokládat, že výsledky např. měření ROUGE by byly horší než u původního MUSUTELSA sumarizátoru, jehož zaměření bylo nepatrně odlišné.

Objektivní testování aplikace ASI jako celku je sama o sobě obtížná úloha, podobně jako objektivně hodnotit kvalitu jiných internetových vyhledávačů. Aplikace ASI byla testována na shodných (nezkrácených) dotazech jako internetové vyhledávače v kapitole 2.2.2. Hodnocené souhrny měly délku 10 vět. Sumarizace byla testována při omezení na prvních 5 a 10 vyhledávačem nalezených webových stránek. Skutečný počet webových stránek v procesu sumarizace mohl být menší, jelikož mezi výsledky vyhledávání primárního vyhledávače Google jsou i PDF nebo DOC soubory, které aplikace ASI neumí zpracovat.

Do testování byly zahrnuty i výsledky vyhledávače Google (pro češtinu) a také systém SenseBot, který se nejvíce podobá aplikaci ASI. V systému SenseBot bylo omezeno množství stránek pro sumarizaci na 10 a nastaven Google jako jediný vyhledávač.

Byly zaznamenávány počty dotazů, na které byly nalezeny požadované informace **přímo** (bez nutnosti návštěvy další webové stránky), **nepřímo** (tedy s nutnou návštěvou na odkazující webové stránky), anebo výsledky vyhledávání neobsahovaly ani odkaz na webovou stránku s žádanou informací (**nenalezeno**). U výsledků, které nebyly zcela jednoznačně rozhodnutelné, bylo přiděleno poloviční „skóre" (např. 0,5 pro přímo a nepřímo nalezené informace). Test je spíše zaměřen na schopnost poskytnout v souhrnu krátkou odpověď, než podat obecný souhrn.

Vyhledávač	**Jazyk**	**Počet zdrojů**	**Nalezeno**			**Průměrná doba vyhledávání**
			Přímo	**Nepřímo**	**Nenalezeno**	
ASI	Čeština	5	7	2	1	5 s
ASI	Čeština	10	7,5	2,5	0	21 s
Google	Čeština	10	7,5	2,5	0	1 s
ASI	Angličtina	5	8	2	0	4 s
ASI	Angličtina	10	5	4	1	11 s
SenseBot	Angličtina	10	7,5	2,5	0	8 s

Tabulka 5: Výsledky vyhledávání.

Výsledky uvedené v tabulce 5 ukazují, že aplikace ASI dosahuje velmi dobrých výsledků i v případě dotazů kladených jako otázka. Nejslabšího výkonu dosáhla při sumarizaci 10 zdrojů v angličtině. Souhrny měly délku pouze 10 vět a pouhým prodloužením extraktu by bylo pravděpodobně dosaženo lepších výsledků (vyšší pravděpodobnosti výskytu požadované informace). Překvapivá byla i kvalita Googlem nabízených souhrnů k nalezeným stránkám. Ty často obsahovaly požadovanou informaci, pokud se jednalo o dotaz, na který bylo možné odpovědět jednou větou. Měřena byla i rychlost vyhledávání. Aplikace ASI (resp. její moduly) dosahuje nejpomalejšího vyhledávání pro češtinu. To je způsobeno charakterem jazyka. České texty jsou obecně bohatší na slovní zásobu než anglické, a proto i slovník lemmat je větší a vstupní matice pro SVD dosahuje většinou větších dimenzí než v případě angličtiny. A jak již bylo uvedeno, SVD je výpočetně

náročná operace.

Dále byla testována schopnost vytváření souhrnů na téma definované dotazem. Jako dotazy posloužily pro češtinu i angličtinu náhodně vybrané titulky ze zpravodajského serveru. Hodnoceno bylo, zda vytvořený souhrn o délce 30 vět pokrývá téma definované dotazem. Jako **relevantní** byly hodnoceny souhrny, které se z větší části týkaly daného tématu. **Uspokojivě** byly označeny souhrny, kde alespoň první věty zachycovaly dané téma. Jako **nevyhovující** byly označeny souhrny, které se z velké části týkaly jiného, i když blízkého tématu.

Vyhledávač	**Jazyk**	**Počet zdrojů**	**Hodnocení souhrnu**			**Průměrná doba vyhledávání**
			Relevantní	**Uspokojivé**	**Nevyhovující**	
ASI	Čeština	5	12	6	2	4 s
ASI	Angličtina	5	15	4	1	5 s

Tabulka 6: Výsledky vyhledávání.

Výsledky testování (viz tabulka na této straně) ukázaly, že aplikace ASI opravdu dokáže z webových stránek extrahovat informace na téma definované dotazem. Testování zároveň odhalilo několik nedostatků. Jedním z nich je uvažování podobnosti. Některé souhrny obsahovaly na první pozicích velmi podobné věty nesoucí téměř shodnou informaci. To je možné částečně vyřešit zvýšením prahového úhlu pro kosinovou podobnost. Pro každý dotaz jsou ale ideální parametry pro LSA sumarizaci odlišné, a ta činí jejich určení obtížné. Další nedostatkem je filtrování nežádoucích textů, které sice funguje velmi dobře, ale stále se v textu v malé míře objevují parazitující texty. Optimalizací stopslovníků by bylo možné dosáhnout dalšího vylepšení.

I když aplikace SWEeT byla pro účely testování zprovozněna pro dva české zpravodajské servery, nebylo nakonec porovnání s aplikací SWEeT realizováno, a to z časových důvodů. Testování by probíhalo na omezené doméně témat, jelikož aplikace SWEeT vyhledává pouze na zpravodajských serverech.

Nutno podotknout, že provedené testování není zcela objektivní a může se lišit v závislosti na anotátorovi. Veškeré výstupy aplikací jsou dostupné na přiloženém CD. Aplikace ASI byla testována na stolním počítači s procesorem Intel Core2Duo 2,1 GHz a 4 GB operační paměti. Průměrná rychlost připojení k Internetu pro download byla 1 860 kB/s a upload 126 kB/s.

9 Navrhovaná vylepšení

Během vývoje vyplynulo nespočet námětů na vylepšení. Některými z nich jsou:

- Implementace dalších sumarizačních metod.
- Optimalizace slovníků stopslov.
- Zlepšení tokenizace (např. pro reálná čísla).
- Zlepšení filtrace vět (stále se v malé míře objevují parazitující texty).
- Zapojení strojového učení pro volbu optimálních parametrů implementované LSA sumarizace.
- Použití vlastního open source vyhledávače.
- Řízení sumarizace podobně jako v SenseBot.
- Podpora dalších typů dokumentů (PDF, DOC atd.).

10 Závěr

Dle mého názoru byly splněny všechny body zadání diplomové práce. Webová aplikace ASI respektive implementované moduly pro vyhledávání předčily mé očekávání. Výsledky testování ukazují, že implementované algoritmy vyhledávání opravdu mohou redukovat nutnost návštěvy nalezených webových stránek a poskytnout požadované informace přímo.

Výsledná aplikace ASI implementuje dva moduly pro vyhledávání na Internetu, a to pro češtinu a angličtinu. Moduly extrahují text z nalezených webových stránek a použitím LSA sumarizace vyhodnotí relevantnost vět k zadanému dotazu. Uživateli jsou vybrané věty prezentovány jako souvislý text. Architektura byla navržena s ohledem na další vývoj a dovoluje snadné rozšíření o další moduly, které mohou skrývat naprosto odlišné algoritmy pro vyhledávání.

Původně byla plánována implementace další sumarizační metody a jejich vzájemné porovnání, ale vývoj současné verze aplikace ASI byl mnohem náročnější, než jsem předpokládal. I když jsem s výsledky spokojený, prostor pro vylepšení vidím v každé části této práce. Jedním z námětů pro vylepšení by mohlo být zavedení open source vyhledávače a odprostit se tak od závislosti na externím internetovém vyhledávači.

Přehled zkratek

API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
CRF	*Conditional Random Fields*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
DUC	*Document Understanding Conferences*
hLDA	*hierarchical Latent Dirichlet Allocation*
HMM	*Hidden Markov Model*
HTML	*HyperText Markup Language*
JAR	*Java ARchive*
JSF	*JavaServer Faces*
JSON	*JavaScript Object Notation*
JSP	*JavaServer Pages*
LARS	*Least Angle Regression*
LSA	*Latent Semantic Analysis*
LSI	*Latent Semantic Indexing*
NLP	*Natural Language Processing*
OSGi	*Open Services Gateway initiative*
P2P	*Peer-to-peer*
PDF	*Portable Document Format*
ROUGE	*Recall-Oriented Understudy for Gisting Evaluation*
SOAP	*Service-Oriented Architecture*
SQL	*Structured Query Language*
SVD	*Singular Value Decomposition*
SVM	*Support Vector Machines*

TAC	*Text Analysis Conference*
WAR	*Web application ARchive*
XML	*Extensible Markup Language*

Reference

[1] Akhilesh Gupta. *Ranking in Search Engines.*
URL: <http://www.stanford.edu/~agupta03/Colloq.pdf> [cit. 2012-05-09].

[2] Amy Langville. *Google's PageRank and Beyond: The Science of Search Engine Rankings.*
URL: <http://langvillea.people.cofc.edu/Citadel.pdf> [cit. 2012-05-09].

[3] David Hawking, Nick Craswell, Peter Bailey, Kathy Griffiths. *Measuring Search Engine Quality.*
URL: <http://es.csiro.au/pubs/hawking_ir01.pdf> [cit. 2012-05-09].

[4] *English Stopwords.*
URL: <http://www.ranks.nl/resources/stopwords.html> [cit. 2012-05-09].

[5] Christian Middleton, Ricardo Baeza-Yates. *A Comparison of Open Source Search Engines.*
URL: <http://wrg.upf.edu/WRG/dctos/Middleton-Baeza.pdf> [cit. 2012-05-09].

[6] Vojtěch Žihla. *Pokročilé metody stahování a filtrování obsahu Internetu.* Plzeň, 2011. Diplomová práce na Fakultě aplikovaných věd Západočeské univerzity v Plzni na Katedře informatiky a výpočetní techniky.

[7] Christian Middleton. *Open Source Search Engines.*
URL: <http://grupoweb.upf.es/WRG/mir2ed/pdf/slides_appendixA.pdf> [cit. 2012-05-09].

[8] Vik Singh. *A Comparison of Open Source Search Engines.*
URL: <http://zooie.wordpress.com/2009/07/06/
a-comparison-of-open-source-search-engines-and-indexing-twitter/> [cit. 2012-05-09].

[9] Christian Kohlschütter, Peter Fankhauser, Wolfgang Nejdl. *Boilerplate Detection using Shallow Text Features.*
URL: <http://www.l3s.de/~kohlschuetter/boilerplate/
WSDM2010-Kohlschuetter-slides.pdf> [cit. 2012-05-09].

[10] Stefan Evert. *A lightweight and efficient tool for cleaning Web pages.* Institute of Cognitive Science, University of Osnabrück, 2008.

[11] Miroslav Spousta, Michal Marek, Pavel Pecina. *Victor: the Web-Page Cleaning Tool.* Praha, 2008. Institute of Formal and Applied Linguistics, Charles University.

[12] David Gibson, Kuna Punera, Andrew Tomkins. *The Volume and Evolution of Web Page Templates.*
URL: <http://research.yahoo.com/files/templates.pdf> [cit. 2012-05-09].

[13] Ziv Bar-Yossef, Sridhar Rajagopalan. *Template Detection via Data Mining and its Applications.*
URL: <http://www2002.org/CDROM/refereed/579/> [cit. 2012-05-09].

[14] Deng Cai, Shipeng Yu, Ji-Rong Wen, Wei-Ying Ma. *Extracting Content Structure for Web Pages based on Visual Representation.*

[15] Aidan Finn, Nicholas Kushmerick, Barry Smyth. *Fact or fiction: Content classication for digital libraries.* Smart Media Institute, Department of Computer Science, University College Dublin.

[16] Christian Kohlschütter, Peter Fankhauser, Wolfgang Nejdl. *Boilerplate Detection using Shallow Text Features.* L3S Research Center; Leibniz Universität Hannover.

[17] Christian Kohlschütter. *A Densitometric Analysis of Web Template Content.* L3S Research Center; Leibniz Universität Hannover.

[18] Jan Pomikálek. *Removing Boilerplate and Duplicate Content from Web Corpora.* Brno, 2011. Faculty of Informatics, Masaryk University.

[19] H. P. Edmundson. *New Method in Automatic Extraction.* University of Maryland, 1968.

[20] Igor A. Bolshakov, Alexander Gelbukh. *COMPUTATIONAL LINGUISTICS: Models, Resources, Applications.* IPN-UNAM-FCE, 2004. ISBN 970-36-0147- 2.

[21] Karen Spärck Jones. *Automatic summarising: the state of the art.* Computer Laboratory, University of Cambridge, 2007.

[22] Karel Ježek, Josef Steinberger. *Sumarizace textů.* Katedra informatiky a výpočetní techniky, FAV ZČU v Plzni.

[23] Josef Steinberger, Michal Toman, Karel Ježek. *Vyhledávání a automatická sumarizace textů v multilinguálním prostředí.* Katedra informatiky a výpočetní techniky, FAV ZČU v Plzni.

[24] Hongyan Liu, Ping'an Liu, Wei Heng, Lei Li. *The CIST Summarization System at TAC 2011.* Center for Intelligence Science and Technology, Beijing University of Posts and Telecommunications.

[25] John M. Conroy, Judith D. Schlesinger, Jeff Kubina, Peter A. Rankel, Dianne P. O'Leary. *CLASSY 2011 at TAC: Guided and Multi-lingual Summaries and Evaluation Metrics.* IDA/Center for Computing Sciences; Department of Defense; University of Maryland

[26] Josef Steinberger, Mijail Kabadjov, Ralf Steinberger, Hristo Tanev, Marco Turchi, Vanni Zavarella. *JRC's Participation at TAC 2011: Guided and Multilingual Summarization Tasks.* Joint Research Centre, European Commission.

[27] Frank Schilder, Ravikumar Kondadadi. *FastSum: Fast and accurate query-based multi-document summarization.* Research & Development, Thomson Corp., 2008.

[28] Martin Křišťan. *Vícedokumentový sumarizátor textů založen na latentní sémantické analýze.* Plzeň, 2007. Diplomová práce na Fakultě aplikovaných věd Západočeské univerzity v Plzni na Katedře informatiky a výpočetní techniky.

[29] Ahmed A. Mohamed, Sanguthevar Rajasekaran. *Query-Based Summarization Based on Document Graphs.* Department of Computer Science & Engineering, University of Connecticut, 2006.

[30] G. Giannakopoulos, M. El-Haj, B. Favre, M. Litvak, J. Steinberger, V. Varma. *TAC 2011 MultiLing Pilot Overview.*
URL: <http://www.nist.gov/tac/publications/2011/additional.papers/Summarization2011_MultiLing_overview.proceedings.pdf> [cit. 2012-05-09].

[31] Dipanjan Das, André F.T. Martins. *A Survey on Automatic Text Summarization.* Language Technologies Institute, Carnegie Mellon University, 2007.

[32] J. Steinberger, M. Křišťan. *LSA-Based Multi-Document Summarization.* Text Mining Group, Dept. of Computer Science and Engineering, University of West Bohemia in Plzeň, Czech Republic.

[33] Yihong Gong, Xin Liu. *Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis.* NEC USA, C & C Research Laboratories.

[34] Golub G., Loan Van C. *Matrix computations.* 3rd edition, Johns Hopkins University Press, Baltimore, 1996.

[35] Josef Steinberger, Karel Ježek. *Using Latent Semantic Analysis in Text Summarization and Summary Evaluation.* Dept. of Computer Science and Engineering, University of West Bohemia in Plzeň, Czech Republic.

[36] Gabriel Murray, Steve Renals, Jean Carletta. *Extractive Summarization of Meeting Recordings.* Centre for Speech Technology Research, University of Edinburgh, 2005.

[37] Chin-Yew Lin. *ROUGE: A Package for Automatic Evaluation of Summaries.* Information Sciences Institute, University of Southern California, 2004.

[38] Josef Steinberger, Karel Ježek. *Evaluation Measures for Text Summarization.* Dept. of Computer Science and Engineering, University of West Bohemia in Plzeň, 2009.

[39] Martin Sloup. *On-line sumarizátor webu.* Plzeň, 2008. Bakalářská práce na Fakultě aplikovaných věd Západočeské univerzity v Plzni na Katedře informatiky a výpočetní techniky.

[40] Kito D. Mann. *JavaServer Faces in Action.* Manning Publications Co, 2004. ISBN 1-932394-12-5.

Přílohy

Instalace

Aplikace ASI je dodána jako komprimovaný WAR balíček a návod popisuje instalaci na webový server Apache Tomcat 7. Předpokládá se základní znalost systému Apache Tomcat. Vyjdeme-li z čisté instalace Apache Tomcat 7, je před zavedením aplikace ASI nutné vyhradit dostatek paměti pro běh a nastavit znakovou sadu UTF-8 pro dekódování URI. Postupujte následovně:

1. Pro Apache Tomcat 7 běžící na operačním systému Windows v souboru `bin/catalina.bat` nastavte:

   ```
   set JAVA_OPTS=-Xms256m -Xmx512m
   ```

   Pro Apache Tomcat 7 běžící na operačním systému Linux v souboru `bin/catalina.sh` nastavte:

   ```
   export CATALINA_OPTS=-Xms256m -Xmx512;
   ```

2. V souboru `conf/server.xml` přidejte parametr `URIEncoding="UTF-8"` ke konektoru s protokolem HTTP.

3. Restartujte server.

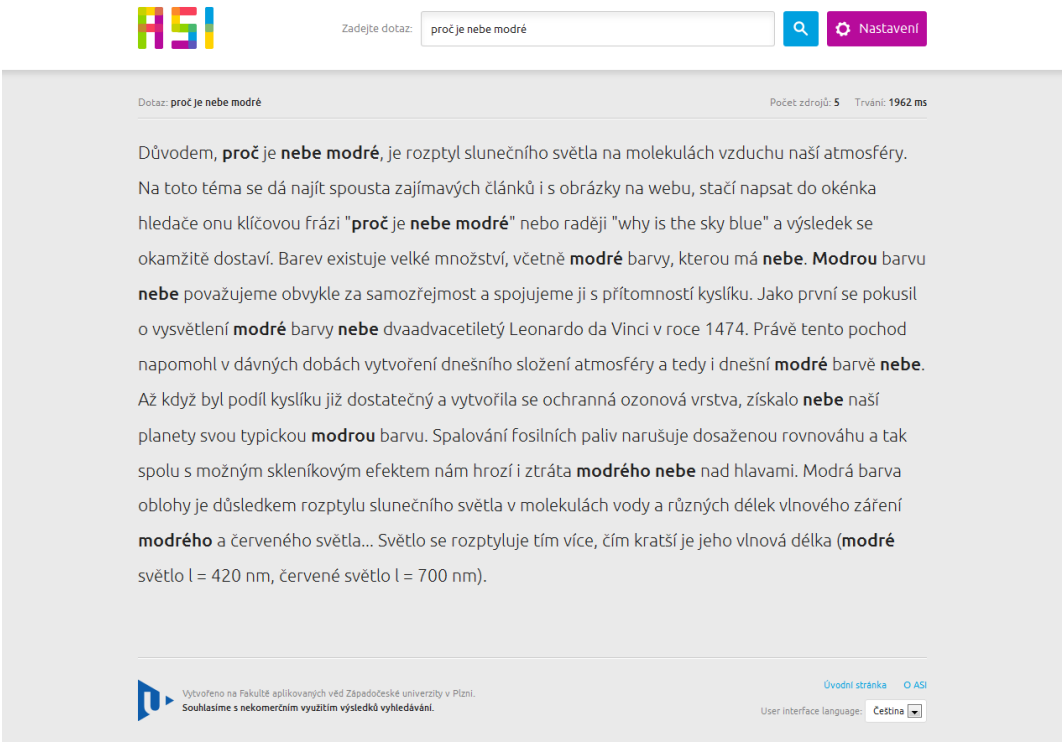
4. Prostřednictvím webového rozhraní Tomcat Manager nahrajeme soubor `asi.war` na server Apacha Tomcat 7.

5. Po zavedení WAR souboru aplikaci ASI v Tomcat Manageru spustíme . Spouštění aplikace ASI může trvat několik sekund.

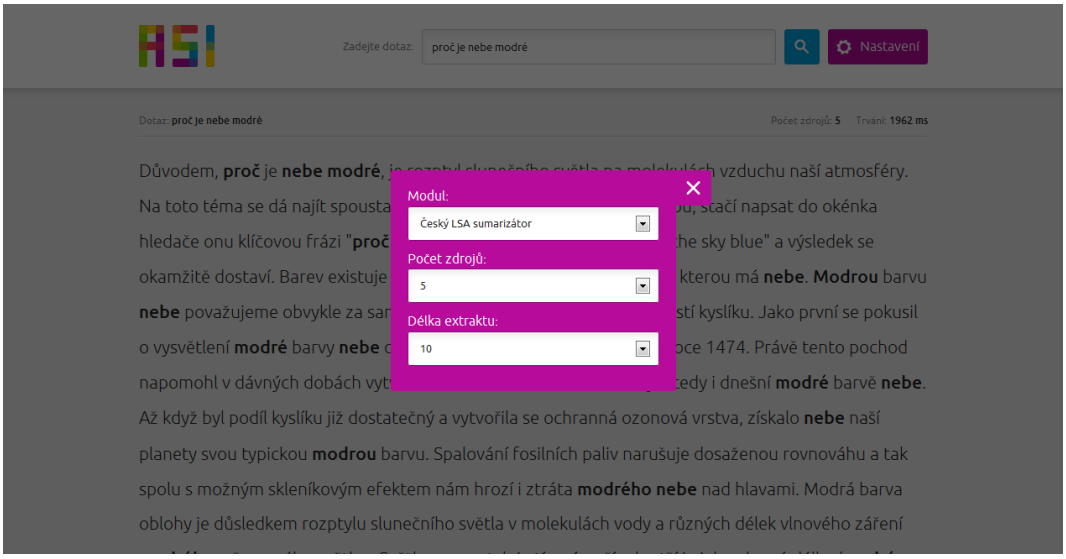
6. Aplikace ASI je nainstalována a připravena k použití.

Uživatelský manuál

Ovládání aplikace ASI je velmi intuitivní (viz obrázek 10). Vždy viditelná horní lišta obsahuje vstupní pole pro zadání dotazu. Vedle vstupního pole se nachází modré tlačítko s lupou pro odeslání dotazu. Možnosti nastavení jsou dostupné po stisknutí fialového tlačítka „Nastavení".



Obrázek 10: Stránka s výsledky vyhledávání.



Obrázek 11: Možnosti nastavení.

Po stisknutí tlačítka „Nastavení" se zobrazí okno (viz obrázek 11) s možností volby modulu, počtu zdrojů (nalezených webových stránek), které má zařadit do sumarizace, a počtu vět ve výsledném extraktu. V současné verzi jsou dostupné moduly „Český LSA sumarizátor" a „English LSA Summarizer", je možné nastavit 1 až 10 zdrojů a generovat extrakt o délce 10 až 100 vět. Na stránce se zobrazenými výsledky vyhledávání je přechod na zdrojovou webovou stránku možný kliknutím na vybranou větu.

V dolní části stránky se nachází navigace a přepínání jazyka uživatelského rozhraní (viz obrázek 10).

Porovnání vyhledávačů Bing a Google

Čeština

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	proč je nebe modré	5	2	3
2	kdo je nejrychlejší muž na světě	4	1	3
3	kdy se konají olympijské hry v Londýně	9	1	0
4	jaká je rychlost světla	2	1	3
5	co dělat při dopravní nehodě	9	0	0
6	kdo byl posledním prezidentem Československa	5	1	3
7	musím mít vízum když cestuji do Číny	2	0	8
8	co se stane když Řecko zbankrotuje	2	3	5
9	při kterých olympijských hrách se vysílalo poprvé živě	3	0	6
10	jak zjistím komu patří doména	6	2	2
	Součty:	**47 (51,6 %)**	**11 (12,1 %)**	**33 (36,3 %)**

Tabulka 7: Kvalita nalezených stránek ve vyhledávači Google (bez odstraněných stopslov z dotazů).

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	proč nebe modré	5	2	3
2	nejrychlejší muž světě	4	2	2
3	konají olympijské hry Londýně	8	2	0
4	rychlost světla	6	0	4
5	dělat dopravní nehodě	9	0	0
6	posledním prezidentem Československa	10	0	0
7	musím vízum cestuji Číny	10	0	0
8	stane Řecko zbankrotuje	4	1	5
9	olympijských hrách vysílalo poprvé živě	4	0	6
10	zjistím patří doména	5	3	2
	Součty:	**65 (67 %)**	**10 (10,3 %)**	**22(22,7 %)**

Tabulka 8: Kvalita nalezených stránek ve vyhledávači Google (s odstraněnými stopslovy z dotazů).

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	proč je nebe modré	1	0	9
2	kdo je nejrychlejší muž na světě	1	0	9
3	kdy se konají olympijské hry v Londýně	3	2	5
4	jaká je rychlost světla	4	1	3
5	co dělat při dopravní nehodě	8	1	1
6	kdo byl posledním prezidentem Československa	4	1	5
7	musím mít vízum když cestuji do Číny	0	0	10
8	co se stane když Řecko zbankrotuje	3	0	7
9	při kterých olympijských hrách se vysílalo poprvé živě	0	0	8
10	jak zjistím komu patří doména	0	0	10
	Součty:	**24 (25 %)**	**5 (5,2 %)**	**67 (69,8 %)**

Tabulka 9: Kvalita nalezených stránek ve vyhledávači Bing (bez odstraněných stopslov z dotazů).

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	proč nebe modré	1	0	9
2	nejrychlejší muž světě	1	0	9
3	konají olympijské hry Londýně	6	2	2
4	rychlost světla	7	1	2
5	dělat dopravní nehodě	7	1	1
6	posledním prezidentem Československa	8	0	2
7	musím vízum cestuji Číny	0	0	9
8	stane Řecko zbankrotuje	3	0	7
9	olympijských hrách vysílalo poprvé živě	0	0	8
10	zjistím patří doména	0	1	8
	Součty:	**33 (34,7 %)**	**5 (5,3 %)**	**57 (60 %)**

Tabulka 10: Kvalita nalezených stránek ve vyhledávači Bing (s odstraněnými stopslovy z dotazů).

Angličtina

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	why is the sky blue	9	0	0
2	who is the fastest man in the world	8	2	0
3	when are the Olympic Games in London	10	0	0
4	what is the speed of light	9	0	1
5	what to do after a car accident	10	0	0
6	who was the last president of Czechoslovakia	7	0	3
7	do I need a visa to travel to China	10	0	0
8	what happens when Greece goes bankrupt	3	4	3
9	which Olympics Games were broadcasted live for the first time	6	1	1
10	how can I find out who owns a domain	5	2	3
	Součty:	**77 (79,4%)**	**9 (9,3 %)**	**11 (11,3 %)**

Tabulka 11: Kvalita nalezených stránek ve vyhledávači Google (bez odstraněných stopslov z dotazů).

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	sky blue	6	0	4
2	fastest man world	8	2	0
3	Olympic Games London	6	0	4
4	speed light	4	0	6
5	car accident	3	0	7
6	president Czechoslovakia	2	6	2
7	need visa travel China	10	0	0
8	Greece bankrupt	1	1	8
9	Olympic Games broadcasted live time	5	1	4
10	find owns domain	2	3	4
	Součty:	**47 (47,5 %)**	**13 (13,1 %)**	**39 (39,4 %)**

Tabulka 12: Kvalita nalezených stránek ve vyhledávači Google (s odstraněnými stopslovy z dotazů).

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	why is the sky blue	9	0	1
2	who is the fastest man in the world	6	1	3
3	when are the Olympic Games in London	4	0	6
4	what is the speed of light	8	0	2
5	what to do after a car accident	10	0	0
6	who was the last president of Czechoslovakia	3	3	4
7	do I need a visa to travel to China	10	0	0
8	what happens when Greece goes bankrupt	3	4	3
9	which Olympics Games were broadcasted live for the first time	5	1	4
10	how can I find out who owns a domain	6	1	3
	Součty:	**64 (64%)**	**10 (10 %)**	**26 (26 %)**

Tabulka 13: Kvalita nalezených stránek ve vyhledávači Bing (bez odstraněných stopslov z dotazů).

Č.	Dotaz	Relevantní	Uspokojivé	Nevyhovující
1	sky blue	3	0	7
2	fastest man world	6	1	3
3	Olympic Games London	5	1	4
4	speed light	3	0	7
5	car accident	0	0	10
6	president Czechoslovakia	3	3	4
7	need visa travel China	10	0	0
8	Greece bankrupt	1	0	9
9	Olympic Games broadcasted live time	1	0	9
10	find owns domain	7	1	2
	Součty:	**39 (39 %)**	**6 (6 %)**	**55 (55 %)**

Tabulka 14: Kvalita nalezených stránek ve vyhledávači Bing (s odstraněnými stopslovy z dotazů).

Hlavní třída modulu pro češtinu

```
package cz.zcu.asi.module.lsa.czech;

import gate.Factory;
import gate.Gate;
import java.io.File;
import cz.zcu.asi.gate.plugin.BoilerplateRemover;
import cz.zcu.asi.gate.plugin.CzechSentenceSplitter;
import cz.zcu.asi.gate.plugin.CzechTokenTagger;
import cz.zcu.asi.gate.plugin.CzechTokeniser;
import cz.zcu.asi.gate.plugin.LsaSummarizer;
import cz.zcu.asi.gate.plugin.SimpleLemmatizer;
import cz.zcu.asi.gate.plugin.WebSearch;
import cz.zcu.asi.module.AbstractModule;
import cz.zcu.asi.module.Input;
import cz.zcu.asi.module.Output;

/**
 * Hlavní třída modulu ASI pro dotazovou sumarizaci webových stránek v češtině.
 * Sumarizace je založená na LSA.
 *
 * @author Tarik S. Salem
 *
 */
public class CzechLsaSummarizerModule extends AbstractModule {

    /**
     * Název modulu.
     */
        private static final String MODULE_NAME = "Český␣LSA␣sumarizátor";

    /**
         * Popis modulu.
     */
    private static final String MODULE_DESCRIPTION =
        "Summarizátor␣českých␣webových␣stránek␣postavený␣na␣LSA.";

    /**
     * GATE komponenta pro internetové vyhledávání relevantních webových
     * stránek.
     */
    private WebSearch webSearch;

    /**
     * GATE komponenta pro stahování webových stránek a extrakci textů.
     */
    private BoilerplateRemover boilerplateRemover;

    /**
     * GATE komponenta tokenizaci textů v češtině.
     */
    private CzechTokeniser czechTokeniser;

    /**
     * GATE komponenta identifikaci vět v češtině.
     */
    private CzechSentenceSplitter czechSentenceSplitter;

    /**
```

```
 * GATE komponenta lemmatizaci českých slov.
 */
private SimpleLemmatizer simpleLemmatizer;

/**
 * GATE komponenta pro značení českých stopslov.
 */
private CzechTokenTagger tokenTagger;

/**
 * GATE komponenta pro LSA sumarizaci.
 */
private LsaSummarizer lsaSummarizer;

/**
 * Konstruktor (inicializace modulu).
 *
  * @throws Exception
 *             Výjimka při inicializaci
 */
public CzechLsaSummarizerModule() throws Exception {
    name = MODULE_NAME;
    description = MODULE_DESCRIPTION;

    // Inicializace GATE
    gateInitialization();
}

/**
 * Metoda provede inicializace GATE, registraci pluginů a instanciaci
 * komponent.
 *
 * @throws Exception
 *             Výjimka při inicializaci
 */
private void gateInitialization() throws Exception {
    // Inicializace frameworku GATE
    Gate.init();

    // Registrace pluginu ANNIE
    Gate.getCreoleRegister().registerDirectories(
            new File(Gate.getPluginsHome(), "ANNIE").toURI().toURL());

    // Registrace pluginu ASI
    Gate.getCreoleRegister().registerDirectories(
            new File(Gate.getPluginsHome(), "ASI").toURI().toURL());

    // Instanciace GATE komponent
    webSearch = (WebSearch) Factory
            .createResource("cz.zcu.asi.gate.plugin.WebSearchWrapper");
    boilerplateRemover = (BoilerplateRemover) Factory
            .createResource("cz.zcu.asi.gate.plugin.BoilerplateRemover");
    czechTokeniser = (CzechTokeniser) Factory
            .createResource("cz.zcu.asi.gate.plugin.CzechTokeniser");
    czechSentenceSplitter = (CzechSentenceSplitter) Factory
            .createResource("cz.zcu.asi.gate.plugin.CzechSentenceSplitter");
    simpleLemmatizer = (SimpleLemmatizer) Factory
            .createResource("cz.zcu.asi.gate.plugin.SimpleLemmatizer");
    tokenTagger = (CzechTokenTagger) Factory
            .createResource("cz.zcu.asi.gate.plugin.CzechTokenTagger");
```

```
            lsaSummarizer = (LsaSummarizer) Factory
                    .createResource("cz.zcu.asi.gate.plugin.LsaSummarizer");
        }

        /**
         * Metoda pro zpracování dotazu.
         *
         * @param input
         *             Vstupní data (dotaz, délka extraktu, počet zdrojů)
         *
         * @return Výsledný souhrn (extrakt), v případě výjimky <code>null</code>
         */
        @Override
        public Output processQuery(Input input) {
            try {
                CzechLsaSummarizer cls = new CzechLsaSummarizer(webSearch,
                        boilerplateRemover, czechTokeniser, czechSentenceSplitter,
                        simpleLemmatizer, tokenTagger, lsaSummarizer);

                return cls.summarize(input);
            } catch (Exception e) {
                e.printStackTrace();
            }

            return null;
        }

    }
```

Příklad anotací v GATE dokumentu

Následující příklad ukazuje anotace v dokumentu GATE nezbytné pro proces sumarizace v `LsaSummarizer`. Anotace jsou pro text „Studuji na Fakultě aplikovaných věd. A tohle je další věta.". Příklad ukazuje i značení klíčové slova „věda". Anotace jsou zkrácené.

```
... type=Token; features={root=studovat, kind=word, orth=upperInitial, length=7, string=Studuji}; ...
... type=Sentence; features={}; start=NodeImpl: id=0; offset=0; end=NodeImpl: id=10; offset=36
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={majorType=stop, root=na, kind=word, orth=lowercase, length=2, string=na}; ...
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={root=fakulta, kind=word, orth=upperInitial, length=7, string=Fakultě}; ...
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={root=aplikovaný, kind=word, orth=lowercase, length=12, string=aplikovaných}; ...
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={keyword=true, root=věda, kind=word, orth=lowercase, length=3, string=věd}; ...
... type=Token; features={string=., length=1, kind=punctuation, root=.}; ...
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={majorType=stop, root=a, kind=word, orth=upperInitial, length=1, string=A}; ...
... type=Sentence; features={}; start=NodeImpl: id=11; offset=37; end=NodeImpl: id=21; offset=59
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={majorType=stop, root=tenhle, kind=word, orth=lowercase, length=5, string=tohle}; ...
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={majorType=stop, root=být, kind=word, orth=lowercase, length=2, string=je}; ...
... type=SpaceToken; features={kind=space, length=1, string= }; ...
... type=Token; features={majorType=stop, root=další, kind=word, orth=lowercase, length=5, string=další}; ...
... type=SpaceToken; features={kind=space, length=1, string= }; start=NodeImpl: id=18; ...
... type=Token; features={root=věta, kind=word, orth=lowercase, length=4, string=věta}; ...
... type=Token; features={string=., length=1, kind=punctuation, root=.}; ...
```