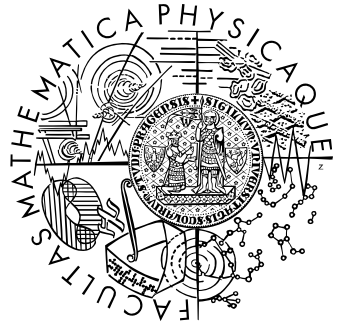
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Karel Vandas

Metody sumarizace textu

Ústav formální a aplikované lingvistiky (ÚFAL)

Vedoucí bakalářské práce: Mgr. Pavel Schlesinger

Studijní program: Informatika, obecná informatika

2009

Děkuji svému vedoucímu za rady a jeho čas, který práci věnoval. Také děkuji Mgr. Pavlu Pecinovi, Ph.D. za poskytnutí velkého množství dat, díky kterým mohla být práce odzkoušena na reálných vstupech, Mgr. Milanu Fučíkovi za ochotu při spouštění programu na jednom ze serverů ÚFALu a v neposlední řadě prof. RNDr. Janu Hajičovi, Dr. za povolení přístupu k jednomu z těchto serverů. Dále děkuji lidem, kteří mě podpořili, Elišce Císařové, která po mně práci nejednou ochotně přečetla, aby mi pomohla s opravami. Taktéž děkuji za konzultaci ohledně citací Báře Bjačkové. Další poděkování věnuji lidem, kteří mi poradili, když jsem byl v programátorských nesnázích. A také děkuji všem, kteří provedli nezbytné anotace pro ověření, jak program funguje. Vám všem díky.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 5.8.2009 Karel Vandas

Obsah

Název práce: Metody sumarizace textu
Autor: Karel Vandas
Katedra (ústav): Ústav formální a aplikované lingvistiky (ÚFAL)
Vedoucí bakalářské práce: Mgr. Pavel Schlesinger
e-mail vedoucího: schlesinger@ufal.mff.cuni.cz

Abstrakt: Bakalářská práce se zabývá automatickou sumarizací textu. Součástí textu je úvod do zvolené problematiky. Práce pak popisuje detailně sumarizační metodu MEAD (Centroid-based summarization of multiple documents). Práce obsahuje rovněž popis shlukové metody CIDR, která je součástí sumarizační metody MEAD. V práci najdeme část, která se věnuje implementaci konkrétních algoritmů včetně uživatelské příručky. Také se věnujeme ukázce rychlosti zpracovávání vstupních dat. V závěru práce prezentujeme obecné představení evaluace. Popisujeme také experimenty provedené nad kolekcí článků. Výsledky experimentů jsou na závěr podrobeny inovativním pokusem o evaluaci.

Klíčová slova: automatická sumarizace, shluková analýza, sumarizace textu, extrakt, evaluace sumarizace textu

Title: Methods of Text Summarization
Author: Karel Vandas
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Pavel Schlesinger
Supervisor's e-mail address: schlesinger@ufal.mff.cuni.cz

Abstract: Bachelor thesis deals with an automatic text summarization. It includes introduction of text summarization. Present work explains in more detail summarization method MEAD (Centroid-based summarization of multiple documents). It includes a description of method of clustering analysis CIDR, which is the component of MEAD method. We consider implementation of algorithms and user guide of program as well. We present demonstration of program's speed. At ending we present general introduction into evaluation. We describe experiments with a collection of articles as well. And the results of those experiments were ended up with an inovative attempt of evaluation.

Keywords: automatic summarization, cluster analysis, text summarization, extract, evaluation of text summarization

Kapitola 1

Podrobnosti o bakalářské práci

1.1 Zadání bakalářské práce

Tématem práce je úloha sumarizace dokumentů pomocí extrakce jejich částí, které co nejvíce charakterizují jejich obsah. Nemá však jít o izolovaná klíčová slova (nebo krátká slovní spojení), nýbrž o celé věty, které tak mohou tvořit jakýsi abstrakt dokumentu. Úspěšné zpracování tohoto tématu spočívá v nastudování problematiky, dále v implementaci a aplikaci vybraných metod, včetně jejich vyhodnocení.

1.2 Řazení kapitol

Cílem bakalářské práce bylo nastudovat a implementovat vybraný sumarizační systém. Za vzor byla zvolena sumarizační metoda MEAD (Centroid-based summarization of multiple documents, Radev a kol., 2004) společně s metodou shlukové analýzy CIDR (Radev a kol., 1999) pro tvorbu shluků.

Druhá kapitola. Na úvod práce se věnujeme zejména představení problematiky sumarizace srozumitelným jazykem.

Třetí kapitola. Následně se podíváme na metodu shlukové analýzy, nejdříve obecně, posléze konkrétně na CIDR, která je nutná pro předzpracování dat pro sumarizační metodu MEAD.

Čtvrtá kapitola. Podrobnější představení sumarizačních metod spolu s konkrétním popisem metody MEAD je umístěno v kapitole čtvrté.

Pátá kapitola. Představení samotného programu, podrobnosti z jeho algoritmů, implementace, zlepšování najdete v kapitole páté.

Šestá kapitola. Věnujeme se také programu z pohledu uživatelského, zmiňujeme nejen možnosti, jak můžeme s programem manipulovat, ale taktéž příklady, jak jej nastavovat pro získání uspokojivých výsledků.

Sedmá kapitola. Protože shluková metoda samotná je proces, který je velmi složitý, přinášíme ukázku, jak se program chová při větším zatížení.

Osmá kapitola. Sumarizační technika by bez vyhodnocení úspěšnosti neměla význam, proto v osmé kapitole představujeme techniky, které se k vyhodnocení používají. Dále zde představujeme techniku vyhodnocení, kterou

jsme v práci použili a samozřejmě výsledky, které jsme získali při tvorbě extraktu s námi zadanými parametry.

Devátá kapitola. V poslední z kapitol představujeme závěr práce.

Literatura. Součástí práce je samozřejmě odkaz na použitou literaturu, kde můžete problematiku nalézt rozvedenou v mnohem větší šíři.

Apendix A. Konkrétní datové výstupy včetně jejich ukázek jsou k nalezení v apendixu A.

Apendix B. Pro snadnější reprezentaci dat byl vytvořen jednoduchý applet v jazyce Java. Zrovna tak byla vytvořena aplikace pro vyhodnocení práce, která sbírala data od anotátorů, v jazycích PHP, SQL, XHTML a CSS.

Kapitola 2

Co je to sumarizace

2.1 Úvod

Sumarizace, česky **shrnutí**, je metoda získávání informací, během níž dochází k záměrnému zkrácení vstupních dat s co nejmenší ztrátou jejich obsahové hodnoty.

Tvorba shrnutí je v dnešní době velmi důležitá. Pro jednoho člověka je zhola nemožné sledovat celý obsah všech informačních kanálů dnešní doby najednou. Automatická sumarizace je tedy jedním z prostředků, jak získat podstatné zprávy v co nejkratším provedení.

Příklad. Jistým způsobem sumarizace je například pojmenovávání článků (ať už v novinách či na Internetu). Editoři obvykle stojí před nelehkým úkolem. Vystihnout třeba i stránkový text poutavým, ale i tak stále věcným nadpisem bývá nelehký úkol. Do tohoto nadpisu je potřeba obvykle shrnout nejen informace typické pro zaměření článku (jako například „vypukl požár“), ale i nějakou informaci, která z této události udělá unikátnost (jako například nějaká neobvyklá citace z rozhovoru se známou osobností).

Příklad. Sumarizace se přitom prováděla ještě dávno předtím, než byla jakkoliv definována. Vzpomeňme si na nástěnné malby v jeskyních, ve kterých se pravěcí lidé snažili zaznamenat události, které prožívali. Znázorňující prostředky, které k tomu využívaly, byly do jisté míry primitivní. I tak jsme však v několika málo rysech schopni po dlouhých staletích rozpoznat, co se v jejich životech udávalo. A právě tyto rysy můžeme považovat za sumarizaci původní informace, za sumarizaci událostí jejich dní před tisíci let.

2.2 Definice sumarizace

Definice. Mani (2001) uvádí hned na začátku své knihy o automatické sumarizaci následující definici. „The goal of (automatic) summarization is to take an information source, extract content from it, and present the most important content to the user in a condensed form and in a manner sensitive to the user's (or application's) needs.“ Přeloženo do češtiny, *cílem (automatické) sumarizace je ze zdroje extrahovat a prezentovat kondenzovaně nejdůležitější obsah způsobem vnímajícím uživatelovy potřeby* (či potřeby aplikace).

Definice nám tedy říká, že cílem sumarizace je především zachovat obsah a vyjádřit jej jen těmi nejnutnějšími prostředky z původního zdroje. V ideální

sumarizaci tedy opomíjíme nejen informace, které nás nezajímají, ale taktéž informace redundantní, tedy přebytečné. To vše tak, aby byla zachována především obsahová hodnota textu s důrazem na celkovou stručnost.

Ve skutečnosti je kvalitní sumarizace jedním z prostředků, jak šetřit lidský čas. Když si uvědomíme, co to sumarizace je a k čemu slouží, po chvíli přemýšlení nám dojde fakt, že jako články v soukolí pracovního procesu stále pracujeme s informacemi. Těchto informací potřebujeme právě tolik, abychom byli schopni splnit náš úkol efektivně. Na druhou stranu není každý z nás připraven a ani ochoten studovat na každý úkol stohy knih. V našem hledání potřebných informací tedy využíváme zdrojů, které o našem problému vypovídají nejvíce a zároveň netrvá tak dlouho jejich prostudování. A tím se opět vracíme k naší definici. Pokud chceme efektivně pracovat, máme tendence vyhledávat právě kondenzované zdroje informací, které shrnují nejdůležitější obsah.

Příklad. Existuje několik druhů sumarizací. Můžeme si například vzít abstrakt této bakalářské práce, který je taktéž jednou z forem sumarizace. Nemusí být pravda, že abstrakt může vždy dopodrobna pokrýt poselství celé práce. I tak však někteří konzultanti či oponenti práce mohou získat jakousi představu, co jim práce má říci. Abstrakt práce by měl správně obsahovat o této práci přesně ty informace, které v ní můžeme najít. V souvislosti s rozsáhlými pojednáními však bude spíše sloužit abstrakt pouze jako „ochutnávka“ obsahu, či jako rozcestník. Na tomto rozcestníku se má možnost každý zastavit, věnovat mu několik minut pro přečtení. Pokud zájemce o informace zjistí, že rozcestník jej navádí na ty informace, které hledá, nebude váhat práci pročíst. Na druhou stranu, neexistoval-li by tento rozcestník, mohl by zájemce o informace práci po přečtení proklínat, protože by na ní mohl zbytečně ztratit hodiny svého času a nedozvěděl by se to, co chtěl. V jistém smyslu tedy sumarizace může sloužit i jako reference na znalosti.

Lačnost po informacích roste s možnostmi, jak je získat. Díky mnohým médiím, která dnes máme, požadujeme nejen kvalitu informací, ale taktéž jejich efektivní podání. Toto podání požadujeme v délce úměrné nejen naší trpělivosti ze zdroje čerpat, ale stejně tak i našim časovým možnostem. Sumarizace pro nás má být nástroj, po kterém saháme, když hledáme informace, může nám být pomocnou rukou, indexem, v nepřeberné záplavě informací, které se na nás každodenně hrnou. Podle tohoto indexu, pakliže je správně vytvořen, můžeme efektivněji vyhledat konkrétní sdělení, která nás zajímají.

Příklad. Indexem, který jsme zmínili, je třeba i obsah v knize. Ten je právě tou částí knihy, do které se zadíváme nejdříve, hledáme-li konkrétní znalosti. Obsah nám zdroj obvykle rozdělí do několika částí, které spolu logicky souvisejí. Hesla v obsahu jsou potom právě našimi malými sumarizacemi. Jedná se o popisy, mnohdy i celých stránek, které jsou shrnuty v jednom jediném výrazu. Zrovna tak je indexem například seznam jmen a telefonních čísel ve Zlatých stránkách. V tomto konkrétním případě o každém člověku existuje mnoho informací, minimálně datum jeho narození, jméno rodičů či sourozenců. Ty nás však nezajímají. Při používání Zlatých stránek chceme zejména rychle najít kontaktní informace.

Jistě, stejně jako existují sumarizace kvalitní, mohou existovat i sumarizace nekvalitní. Na jednu stranu může sumarizace, například abstrakt práce, obsah

a význam práce mnohokrát přecenit, naopak se však také může stát, že jinak kvalitní práci může právě její nekvalitní sumarizace potopit. Pokud se tedy v abstraktu neobjeví kvalitní souhrn, je práce tak či onak odsouzena k celkem rychlému zániku.

2.3 Přehled typů sumarizací

Při procesu sumarizace existují hlediska, která určují, do jaké míry bude splněn účel shrnutí. Zejména jde o pohled z pozice uživatele. Protože sumarizace vždy slouží lidem, měli bychom se zastavit a zamyslet se nad tím, co pro kterého člověka může sumarizace znamenat. Už jen od pohledu je jasné, že každý člověk je jiný. Stejně tak jako jsme každý unikátní, můžeme každý požadovat jiné vlastnosti u shrnutí.

2.3.1 Rozdělení sumarizací podle techniky jejich vzniku

Nejdříve se zaměříme na to, jaké typy sumarizací nám mohou vzniknout.

2.3.1.1 Abstrakt

Příklad. Kolikrát se člověku stane, že přijde ke svému oblíbenému televiznímu pořadu se zpožděním. Celý dosavadní děj chce získat od diváka, který sleduje obrazovku od začátku. Dochvilný divák, aby mu neunikl další děj, se snaží shrnout dosavadní děj a události, které se dosud v pořadu staly, v několika málo větách. Málokdy přitom však použije původních vět. Z pořadu a jeho děje tedy vzniká **abstrakt**. Mani (2001) pro definici abstraktu používá následující definice.

Definice. „An abstract is a summary at least some of whose material is not present in the input.“ Pokud bychom se snažili o volný překlad, jde o to, že *některé ze sumarizovaných vět nebyly přítomné v původním vstupu*. To se může stát hned několika způsoby. Nejen, že některé z vět vstupu mohou být sloučeny do jedné, ale zrovna tak některé z nich mohou být vysvětleny jinak, než byly podány v původním textu.

K vytvoření abstraktu však nestačí obvykle pouze automatická sumarizace. Mimo roviny zápisu věty totiž pro tvorbu abstraktu potřebujeme textu porozumět, minimálně na úrovni roviny významu. Ba co víc, některé abstrakty mohou obsahovat informace obecně známé z našeho porozumění světa. Takové informace pak nejsou záležitostí pouze samotných vět, ale i souvislostí mezi větami, případně mezi informacemi, které dokonce ani nemusí z textu zcela vyplývat.

Abstrakt je tedy jakési uchopení souvislostí v textu na rovině významu věty a jejich přetvoření v logický celek shrnující celý text vydávající většinu původní sdělení. Příkladem abstraktu je například shrnutí tématu, o kterém bude například bakalářská práce. Zrovna tak může být abstrakt přítomen u některých knih, kde, především u beletrie, nalezneme poutavě shrnutou zápletku v několika větách.

Příklad původního textu. „Na zahradě nám parkovala auta. Mimo jednoho BMW zde bylo i jedno Audi a jeden Volkswagen. Žádná další auta jsem na zahradě neviděl. Všechna měla stejnou barvu. ... Karoserie BMW byla žlutá. ...“

Příklad abstraktu předchozího odstavce. „Na zahradě jsem viděl parkovat tři žlutá auta od německého výrobce.“

K tomu, abychom tuto větu mohli napsat, jsme potřebovali porozumět smyslu celého textu, pochopit, že slovo auta zastupuje právě slova BMW, Audi a Volkswagen. Zrovna tak jsme museli porozumět množství aut, která jsme viděli a také tomu, že mají stejnou barvu. Potom, co jsme si přečetli, že BMW bylo žluté, jsme díky stejné barvě všech aut mohli říct to samé i o ostatních. To, že jsou všechny tři značky původně německé je externí informace, kterou však disponují téměř všichni muži (věřím, že i podstatná část žen) v našich zeměpisných šířkách.

Aniž bychom o tomto přemýšleli, jsme schopni toto provést v podstatě automaticky. Počítače však stojí před nelehkým úkolem porozumění textu. Již v tomto úkolu však zaostávají, proto se prozatím experimentuje pouze s extrakty z původního textu.

Příklad. Co by znamenalo, kdyby počítače uměly takovému textu porozumět a uměly jej shrnout? Představme si nespokojeného zákazníka volajícího na reklamace, řekněme kvůli nedoručenému výpisu z účtu. Dnes musí na druhé straně přístroje, pokud se nejedná o nějakou jednodušší úlohu (například nastavení upozorňovací SMS na zůstatek při pohybu na účtu klienta banky), sedět operátor, který se zákazníkem komunikuje.

Příklad. Operátor má především dvě úlohy. První z nich je porozumění tomu, co od něj zákazník žádá, ať použije jakýchkoliv slovních obratů, či ať použije jakýchkoliv správných pořadí slov ve větě. Druhým úkolem operátora na lince je uspokojení zákazníkových potřeb. V tomto případě zjištění, zda se výpisy zákazníkům již odesílaly, či zjištění, kde jinde by mohl být problém. Operátor pracuje podle předem přesně naučených postupů, které zaručí, že zákazník nakonec dosáhne požadovaných informací.

V případě, že by existovala automatická možnost tvorby abstraktu, bylo by možné (předpokládáme-li, že počítač bude přesně schopen rozpoznat všechna slova sdělení), aby práci operátora zastával počítač. Ten by mohl fungovat způsobem, že by zákazník namluvil obsah svého požadavku a počítač by z něj udělal abstrakt. Obsah abstraktu by porovnal s databází dostupných akcí, které by byly indexovány abstrakty. Po nalezení správné akce by ji mohl vykonat a zákazník by byl uspokojen. To vše zcela automaticky.

Naneštěstí, již první úkol, tedy porozumění významu textu, je pro dnešní úroveň zpracování jazyka téměř nedosažitelný. Ale představíme-li si, že v budoucnu by toto bylo možné, nejen operátoři na zákaznických linkách by se měli právo bát o svoje pracovní pozice.

2.3.1.2 Extrakt

Během studia na vysoké škole se člověk potkává s mnoha přednáškami a s mnoha předměty. Zápisky z nich však již nejsou diktované, jak tomu obvykle bývalo na střední škole, ale každý si musí z vět, které vyslechne, vybrat ty podstatné. Ty, které přijdou studentovi podstatné, běžně přepíše v celém jejich původním znění. Aniž by si to kdokoliv z nás uvědomoval, v tu chvíli provádí sumarizaci a to způsobem **extrakce**. Mani (2001) pojmenovává extrakci tímto způsobem.

Definice. „An extract is a summary consisting entirely of material copied from the input.“ Volně řečeno, *extrakt je původní dokument, z něhož jsme některé části vypustili, ale nové jsme již nepřidali*.

U tvorby extraktů je však o něco lepší možnost, jak je zpracovávat automaticky. Není totiž zcela nutné vstupnímu textu plně rozumět, je možné z něj zkrátka vybrat ty části, v nichž se typicky vyskytují právě ty informace, které nás zajímají. Jak navrhuje Mani (2001), jednou z těchto metod může být třeba vybrání prvního odstavce, či první věty. Díky postupům, jakými píší nejen novináři, ale i tvůrci knih, jsou tyto metody celkem úspěšné.

Extrakty, oproti abstraktům, jsou především útržky z původního textu, oproti abstraktům se vyznačují tím, že jednotlivé věty na sebe nemusejí navazovat a často ani nenavazují (resp. to, jestli navazují, je dílem náhody). Jde tedy o jakési útržkovité informování o obsahu textu stěžejními větami.

2.3.2 Rozdělení sumarizací podle účelu

Sumarizace může být určena hned pro několik aplikací. Mani (2001) odkazuje na práci dvojice Borko, Bernier (1975), která se zabývala rozdílem mezi **informativním** a **svědčícím** (anglicky: informative, indicative) typem sumarizací.

Definice. „An indicative abstract provides a reference function for selecting documents for more in-depth reading. By contrast an informative abstract covers all the salient information in the source at some level of detail.“ Česky přeloženo tzv. *svědčící abstrakt tvoří jakousi referenci, zda si máme dokument dále pročítat, oproti informativnímu abstraktu, který se nám rovnou snaží informace z dokumentu s nějakou mírou podrobnosti sdělit*.

Příklad. Za informativní abstrakt můžeme považovat záznam ze schůze, který, samozřejmě v mnohem kratší formě než jakou dobu schůze trvala, podává o schůzi dostatek informací s detaily. Sumarizace sama tedy vypovídá o obsahu zdroje.

Svědčící abstrakt oproti tomu dává jen jakousi ochutnávku z celé původní informace a tím nás nabádá k tomu, abychom buď vyhledali zbytek těchto informací, pokud nás problematika zajímá, anebo tyto informace přeskočili, pokud je neshledáme v abstraktu užitečnými. Jak jsme uvedli již dříve, jde o jakýsi rozcestník a příkladem je například abstrakt bakalářské práce. Můžeme tedy například podat svědectví o užitečnosti informací, které se ve zdroji nacházejí.

Příklad. V případě svědčícího abstraktu může takovou sumarizací být odborné posouzení práce. Avšak ani zde nemusíme zacházet tak daleko. Některé práce mohou poukazovat na kvalitní zdroje už jenom jejich uvedením v citacích.

Definice. Dalším typem, který se v sumarizaci vyskytuje, je tzv. **„critical evaluative abstract“** (česky: kriticky zhodnocující abstrakt). „A critical abstract evaluates the subject matter of the source, expressing the abstractor's view on the quality of the work of the autor.“ Ve zkratce *mimo jiné je do abstraktu zanesen také pohled jeho tvůrce, který zároveň posuzuje kvalitu práce*. Za zmínku stojí, že takovéhoto stupně sumarizace dle našich informací nejsou a po dlouhá léta ještě nejspíše nebudou stroje schopny. Avšak na druhou stranu nám tento abstrakt, pokud je podán objektivně a správně, může do jisté míry zajistit, že informace, které v původní práci získáme, budou odpovídat kvalitě, kterou nastínil tvůrce abstraktu.

2.3.3 Rozdělení sumarizací podle „zadavatele“

Příklad. Mějme fiktivní článek o Hubbleově vesmírném teleskopu. Řekněme, že článek má několik stran textu a zabývá se teleskopem z mnoha pohledů, nejen z hlediska pořizování snímků vesmíru, ale taktéž z hlediska konstrukce teleskopu. Dáme-li článek nadšenému astronomovi, zejména ho bude zajímat výzkum, kterým se teleskop ve vesmíru zabýval. Dáme-li ten samý článek naopak nadšenému konstruktérovi, bude ho hlavně zajímat, jak samotný teleskop funguje. Jde o to, že pro různé skupiny uživatelů je z článku důležitá různá informace.

Definice. Mani (2001) takový rys sumarizace popisuje následovně. „User-focused summaries are tailored to the requirements of a particular user or group of users.“ čili *uživatelsky zaměřené sumarizace jsou ušity na míru požadavkům konkrétního uživatele či skupiny uživatelů*. Ať jste tedy nadšený astronom či konstruktér, tento typ shrnutí by měl být prostředkem, jak získat pro konkrétního člověka odpovídající sumarizaci. Tedy taková sumarizace zejména využívá samotný profil uživatele a snaží se podat přesně ty informace, které zadavateli vyhovují.

Součástí typu **„user-focused“** sumarizací (česky: sumarizací zaměřených na uživatele), jsou i tzv. **„topic-focused“** (česky: sumarizace zaměřené na téma) či

„query-focused“ (česky: sumarizace zaměřené na získávání odpovědí). Zde vidíme, že typem sumarizací může být i jenom klasické dotazování.

Příklad. Představme si, že máme množství dat, řekněme hlasovacích lístků ve volbách. V každé volební komisi se nashromáždí mnoho statistických informací. Mezi jinými například počet lidí, kteří přišli k volbám z volebního okrsku, poměr počtu volících žen a mužů, kteří se k volbám dostavili, a další. V klasických volbách však vždy jde pouze o jedno číslo, o to, o kolik procent hlasů získala která strana více, či méně. Souhrnem statistik z takové volební místnosti tedy může být pouze odpověď na otázku: „Získala strana A více hlasů než strana B?“.

Naopak typ sumarizací zaměřených na téma (anglicky: **„topic-focused“**), podobně jako u konstruktéra a astronoma, upřednostňuje jistou část informací ze zdroje. Při sumarizaci tedy velice záleží nejen na tom, o čem text je, ale taktéž na tom, kdo a pro jaký účel text sumarizuje.

Ačkoliv lze sumarizaci samotnou popsat definicemi, obvykle platí, že co člověk, to různá představa o tom, jak by měla sumarizace vypadat. Každý upřednostňuje jiný typ informací, dokonce i v jejich podání se člověk od člověka liší. Jako druhý typ tedy Mani (2001) uvádí sumarizace, které by měly všeobecně splňovat charakter shrnutí, aniž by při tom byly upřednostňovány některé typy informací před jinými. Tento typ sumarizací popisuje pojem **„Generic summaries“** (česky: generovaná sumarizace).

Příklad. Pokud si ještě vzpomínáme na příklad s astronomem a konstruktérem, generovaná sumarizace by poskytla jeden výstup vhodný pro oba zúčastněné. To ale znamená, že by ve výstupu byly přítomny jak informace zajímající astronoma, tak informace vzrušující pro konstruktéra. Úskalí tohoto typu sumarizací je v tom, že vede ke značnému zevšeobecňování a častokrát tedy těžko vylíčí zajímavá specifika původního textu.

Přehled typů sumarizací	
Vznik sumarizace	Abstrakt / Extrakt
Účel sumarizace	Informativní / Svědčící / Kriticky zhodnocující abstrakt
Zadavatel sumarizace	Uživatelsky zaměřená sumarizace / Tematicky zaměřená sumarizace / Sumarizace zaměřená na získání odpovědi / Generovaná sumarizace

Tabulka 1: Přehled typů sumarizací podle různých kritérií

Kapitola 3

Shluková analýza metodou CIDR

V této kapitole se zaměříme na představení metody shlukové analýzy CIDR, která je nedílnou součástí metody MEAD. Pro pochopení jednotlivých aspektů metody CIDR je nutné nejdříve krátké obecné seznámení s metodami shlukové analýzy obecně.

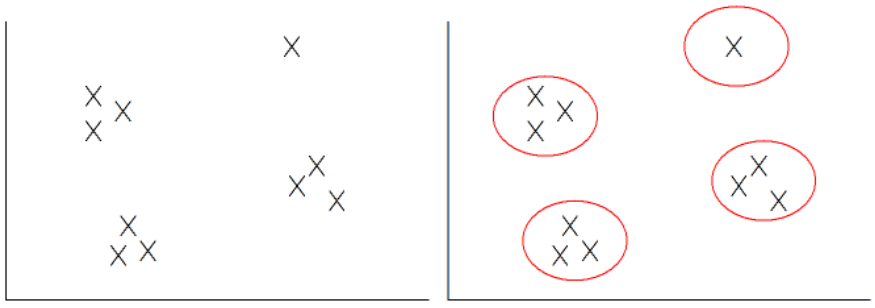
Před samotným užitím sumarizační techniky je nutné získat ohodnocení jednotlivých slov v dokumentech, které chceme sumarizovat. Pro metodu MEAD, kterou jsme v práci implementovali, tato část představuje právě provedení shlukové analýzy metodou CIDR.

3.1 Co je to shluková analýza obecně

Shluková analýza (anglicky: cluster analysis) je proces, při kterém, neformálně řečeno, *porovnáváme jisté elementy za účelem zjištění, zda „patří k sobě“*. Pokaždé je nutné mít elementy zaneseny v n-dimenzionálním prostoru jako body a pro výpočet jejich podobnosti užít jasně definovanou míru podobnosti.

Shluková analýza je tedy proces, při kterém jsou vstupní dokumenty podrobeny postupně testům na (námi zvolenou) podobnost s již existujícími shluky. Výstupem shlukové analýzy jsou tzv. **shluky** či **mraky** (anglicky: clusters) ukazující, které elementy jsou si více podobné a které elementy si jsou méně podobné.

Význačným prvkem ve shluku je tzv. **centroid**. Elementy jsou v n-dimenzionálním prostoru zaneseny jako body. Pro každý shluk existuje jakési těžiště shluku, které reprezentuje vlastnosti všech ostatních bodů ve shluku. Výhodou tohoto bodu je fakt, že v jistém smyslu charakterizuje celý shluk.



Obrázek 1: V levé části ukazujeme jeden z průmětů, kde jsou fiktivní elementy zaneseny do n-dimenzionálního prostoru. V druhé části ukazujeme, jak mohou vypadat shluky po nastavení určitých parametrů.

Příklad. V obrázku ukazujeme, jak by případná tvorba shluků mohla vypadat. Představíme-li si, že jednotlivými elementy jsou dokumenty a ty jsou reprezentovány body v n-dimenzionálním prostoru, kde každá dimenze odpovídá slovu v jeho základním tvaru (tzv. lemmatu) z některého z dokumentů, které jsou v prostoru přítomny.

Příklad. Máme-li slova „les“, „lesům“, „lesích“, stále se jedná o ten samý základní tvar slova les, tudíž jde o tři výskyty stejného lemmatu.

Podobné dokumenty potom mohou být třeba dokumenty, v nichž se vyskytuje slovo „skalpel“ a „nemocnice“. Jiný shluk může naopak sbližovat dokumenty obsahující slova „koncert“ a „zpěvák“. Osamoceně pak může být například dokument, v němž se frekvence jeho slov výrazně liší od jiných dokumentů.

3.2 Shluková analýza v případě metody CIDR

Shluková analýza v našem případě je identická s příkladem uvedeným v předchozí podkapitole. Elementy, které v ní vystupují, jsou jednotlivé dokumenty, které jsou v prostoru, kde jednotlivé dimenze jsou základní tvary slov těchto dokumentů. Tyto dokumenty jsou reprezentovány jako body v již zmíněném n-dimenzionálním prostoru, jejichž pozice jsou dány frekvencí slov, která obsahují. Můžeme říci, že jednotlivé složky vektoru jsou „pojmenovány“ právě základními tvary slov. Žádná dvě jména složky se logicky neopakují.

Centroid každého shluku, který vytváříme, je váženým průměrem složek vektorových reprezentací dokumentů, které jsou již ve shluku přítomny. Při každém přidání nového dokumentu do shluku je tento centroid aktualizován (přepočítáním vážených průměrů).

Míru, do které jsou si dokumenty podobné, označujeme podle vzoru z metody MEAD jako tzv. similarity treshold, tedy mez podobnosti. Dokumenty jsou si podobné, pokud mají naměřenou podobnost vyšší než tuto mez. Naopak, mají-li naměřenou nižší podobnost, považují se za nepodobné. Ty dokumenty, které si jsou dostatečně podobné, se nacházejí v jednom shluku.

Příklad. Vezměme dokument, ve kterém se nalézají pouze 3 slova - „auto, bicykl, motorka“ - jakkoliv je tento příklad bizarní. Vektor lemmat, který tomuto dokumentu odpovídá, uvažujme tak, že *V =(auto, bicykl, motorka)*. Do takového vektoru jsou přitom zahrnuta slova jako „autům“, „bicyklu“, „motorce“, avšak nepatří mezi ně například „motorkářské“, protože základní tvar, tzv. lemma, není motorka, ale motorkářský. Do prostoru tato slova naneseme na příslušné osy s tím, že „vzdálenost“ na ose je ohodnocením tf-idf, které popíšeme dále.

Shlukovou analýzu sumarizovaného dokumentu při sumarizaci provádíme zejména proto, abychom zjistili, jakého „typu“ náš sumarizovaný dokument je. Shluky dokumentů tvoří jakési domény podobných textů, jejichž podobnost je dána

podobnými slovy, která jsou v nich užity. Lze si tedy představit, že se nejspíše texty o vaření budou značně vzdalovat textům například o výpočetní technice. Zařazení dokumentu při sumarizaci do správného shluku je klíčové proto, aby byla při sumarizaci dokumentu užita „ta správná slova“. Tedy aby se například nepovažovalo slovo „plotna“ v sousloví „můj procesor vaří jako plotna“ v dokumentu o výpočetní technice za význačné (a tudíž aby při sumarizaci nemělo význam). A naopak, aby se výraz „počítačová hra“, třeba v nějaké vtipné narážce v textu o vaření, například „musíte brát vaření jako počítačovou hru“, nepovažoval za význačný.

3.3 Parametry shlukové analýzy CIDR

3.2.1 Similarity treshold.

Klíčovým parametrem shlukové analýzy je parametr nazvaný **similarity treshold**, tedy tzv. *mezní podobnost*. Tento parametr udává, jakou podobnost dokumentů považujeme za natolik blízkou, že dokumenty patří k sobě. Pokud například určíme, aby mezní podobností byla hodnota 0.2, říkáme, že na základě spočtené podobnosti dokumentu a shluku považujeme dokument s podobností 0.2 a vyšší (tedy podobností pětinovou a lepší) za náležející do shluku dokumentů. Například hodnota 0.1 při naměření podobnosti dokumentu ke shluku by nás v tomto případě neuspokojila, protože by znamenala pouze desetinovou podobnost dokumentu se shlukem. V praktických aplikacích se ukazuje, že rozumné výsledky získáme při nastavení mezní podobnosti v rozmezí hodnot [0.025, 0.2].

Parametr může mít do značné míry i jiný význam. Například podobnosti mezi dokumenty, které by se pohybovaly v blízkosti hodnoty 1, by znamenaly, že dokumenty jsou identické. Takže parametr mezní podobnosti nám může pomoci i při čištění dat od duplicitních dokumentů, které by naše počítání mohlo ovlivnit.

3.2.2 Keep treshold.

Jedním z dalších parametrů, které bychom mohli při shlukové analýze uvažovat, je tzv. parametr **keep treshold**, čili *mez uchování*. Tato mez nám jednoduše určuje, kolik bychom chtěli, aby centroid měl složek. Tedy samozřejmě těch nejvýznačnějších složek.

Původní implementace zřejmě počítá s tím, že mez uchování je aplikována pokaždé, když do shluku přijde nový dokument. Tvůrcům šlo nejspíše o zrychlení celého procesu. Avšak implementace, kterou jsme v práci zvolili na základě přání vedoucího práce, takto nepracuje a „ořezává“ centroid až v době, kdy jsou již spočteny všechny potřebné podobnosti. Pro takové použití hovoří zejména to, že

při špatných datech pro tvorbu shluků může centroid už ze začátku při přílišném „ořezání“ přestat správně fungovat tím, že jeho pozice bude příliš ovlivněna špatnými daty. Tudíž tento parametr užijeme zejména až při samotné sumarizaci, zřejmě proto, aby byl proces samotné sumarizace (tedy neuvažujeme-li proces shlukování) co nejrychlejší.

3.3 Číselná reprezentace slov, tf*idf

Pro reprezentaci slov jsme v bakalářské práci užili váhu používanou zejména pro získávání informací z dokumentů. Manning a kol. (2009) uvádějí následující definice (str. 117).

3.3.1 Četnost výrazu alias term frequency

Definice. „We would like to compute a score between a query term *t* and a document *d*, based on the weight of *t* in *d*. The simplest approach is to assign the weight to be equal to the number of occurrences of term *t* in document *d*. This weighting scheme is referred to as **term frequency** and is denoted $tf_{t,d}$, with the subscripts denoting the term and the document in order.“

Česky volně přeloženo se v definici říká, že tzv. term frequency*, alias četnost výrazu, udává vztah mezi výrazem samotným a dokumentem, v němž tento výraz zkoumáme*. Nejjednodušším přístupem je potom přiřazení váhy tak, aby se rovnala počtu výskytů tohoto výrazu *t* v dokumentu *d*. Takové váhové schéma odpovídá četnosti výrazu a my jej značíme $tf_{t,d.}$

3.3.2 Inverzní četnost výrazu v dokumentech alias inverse document frequency

Manning a kol. (2009) k tomuto tématu opět dodávají definici, ve které inverzní čestnost výrazu v dokumentech vysvětlují..

Definice. „Denoting as usual the total number of documents in a collection by *N*, we define the **inverse document frequency** (idf) of a term *t* as follows:

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

Obrázek 2: Vztah pro výpočet inverzní četnosti výrazu v dokumentech

where we define df_t to be the number of documents in the collection that contain a term *t*."

Česky toto znamená, že *po označení celkového počtu dokumentů v kolekci písmenkem N, definujeme inverzní četnost v dokumentech výrazu t podle vzorce, ve kterém definujeme df_t jako počet dokumentů v kolekci, které obsahují term t.*

3.3.3 Frekvence tf-idf

Ohodnocení slov samotných dané pouze jejich četností výskytů by nebylo „fér" a nedávalo by nám požadované výsledky. V takovém případě totiž budou slova jako například „být" a „se" mít oproti ostatním slovům mnohem větší váhu a „umístění" dokumentu v prostoru by tak záviselo zejména na nich, protože se vyskytují v textech mnohem častěji než jiná slova. Proto se navíc zavádí inverzní četnost výrazů v dokumentech, kdy se často používané výrazy „perzekuují" tak, aby neměly příliš velkou váhu.

Z výše uvedených definic potom získáváme poslední, která reprezentuje samotné tf-idf, opět Manning a kol. (2009).

„The *tf-idf* weighting scheme assigns to term *t* a weight in document *d* given by $\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$."

Česky toto znamená, že **frekvence tf-idf** vznikne prostým vynásobením četnosti výrazu a inverzní četnosti výrazu v dokumentech.

Manning a kol. (2009) k tomuto ještě dodává, že frekvence tf-idf bude u výrazu tím větší, čím bude četnost tohoto výrazu v malém počtu dokumentů častější. Velikost této váhy pak bude klesat se zvyšujícím se počtem dokumentů, ve kterých se výraz nachází, či se snižující se četností výskytů tohoto výrazu v dokumentech. Velice nízká bude váha v tom případě, pakliže se term vyskytuje ve všech dokumentech.

3.4 Jak počítáme podobnost dokumentů

Podobnost mezi dokumenty vyžaduje, abychom dokumenty převedli na stejný typ struktury. Z dokumentů tedy potřebujeme vytvořit jejich vektorovou reprezentaci. Každá složka vektoru bude věnována jednomu originálnímu základnímu tvaru slova, které se vyskytuje v dokumentu, tak, jak jsme popsali. Jeho reprezentace je založena na váze tf-idf. Mějme tedy v dokumentech d_1, d_2 vektory $V(d_1)$ a $V(d_2)$. Z nich následně spočítáme číslo, které bude reprezentovat jejich vzájemnou podobnost. Zde jsme se mohli rozhodnout pro různé algoritmy na počítání podobnosti, vybrali jsme kosinovou podobnost.

Manning a kol. (2009) uvádějí na straně 121 následující definici, která vysvětluje, jak podobnosti mezi dokumenty počítat.

Definice. „The standard way of quantifying the similarity between two documents $d1$ and $d2$ is to compute the *cosine similarity* of their vector representations $V(d_1)$ and $V(d_2)$

$$\mathrm{sim}(d_{1,} d_2) = \frac{\vec{V}(d_1) * \vec{V}(d_2)}{\left|\vec{V}(d_1)\right| * \left|\vec{V}(d_2)\right|}.$$

Česky řečeno *standardní cestou, jak měřit podobnost mezi dvěma dokumenty, je spočítání kosinové podobnosti jejich vektorových reprezentací*.

Výsledkem spočtené podobnosti je číslo v rozmezí [0,1], které jednoduše označuje, kolik toho mají dokumenty společného. Čím podobnější jsou si dokumenty, tím vyšší číslo jejich podobnosti vychází a naopak. Číslo 0 pak znamená, že dokumenty nemají nic společného.

Upravíme-li definici podobnosti pro počítání podobnosti mezi dokumenty, zjistíme, že reprezentujeme-li stejným způsobem i centroidy shluků či celé shluky, můžeme počítat jejich podobnost stejným způsobem.

Kapitola 4

Sumarizační metoda MEAD

Tato kapitola se zaměřuje na popis sumarizační metody MEAD. Nejdříve se však podíváme na sumarizační metody obecně. Přitom taktéž zmíníme, jaké jiné typy sumarizačních systémů (oproti MEAD) existují.

4.1 Sumarizační metody obecně

Lidé sumarizovali daleko dříve, než byly vynalezeny první stroje připomínající počítače. Zajímavým faktem je, že ačkoliv pro stroje je sumarizace stále velmi obtížný problém, tvoření abstraktů zvládne v podstatě každý člověk, který umí číst. Setkáváme se tedy s problémem, který je, stejně jako rozbor věty, pro absolventa základní školy hračkou.

4.2 Typy sumarizačních systémů

To, co se nám na počítačích líbí, je především jejich schopnost rychle a neomylně zpracovávat obrovské množství dat. Mani (2001) ve své knize uvádí hned několik přístupů k sumarizaci pomocí počítačů. Uvádí, že inspirováni modelem **„Machine Assisted Human Translation“** (česky: Strojem asistovaný lidský překlad) u překladových systémů můžeme pomýšlet na systém, který by podobně mohl fungovat i pro sumarizaci, například systém „**Machine Assisted Human Summarization**“ (MAHS, česky: Strojem asistovaný systém pro lidské sumarizace). Dle předchozích bodů, které jsme uvedli v souvislosti s výhodami, které by přinášel plně automatický systém, bychom u takového modelu mohli uvítat zejména relativní zrychlení práce (ačkoliv by se jednalo o rychlost silně závisející na schopnostech přítomného člověka, dále **sumarizátora**) a z velké části i neomylnost (taktéž závisející na sumarizátorovi). Naopak se u tohoto přístupu můžeme rozloučit s finanční nenáročností, která bude zejména závislá na vyplácení sumarizátora. A v neposlední řadě nemůžeme uvažovat o použitelnosti tohoto systému v nepřetržitém provozu.

Příklad. Ukázkou takového systému může být případ, kdy náš sumarizátor používá software, který co nejintuitivněji a nejsnazším způsobem předkládá data k sumarizaci. Pomáhá člověku při rozhodování, zobrazuje informace, zejména statistického rázu, které by člověk musel náročně sbírat. Takto se snaží zejména oprostit celý cyklus práce sumarizátora od činností, které nesouvisejí s jeho prací, a tím celý proces značně urychlit.

Podobně, můžeme pomýšlet na systém, který bude fungovat strojově pod dohledem člověka, tedy **„Human Assisted Machine Summarization“** (HAMS, česky: člověkem asistovaná strojová sumarizace). U tohoto přístupu získáme oproti předchozímu ještě o něco větší zrychlení práce, avšak opět se nedočkáme finanční nenáročnosti. Částečně však získáme možnost, naučí-li se program, jak člověk reaguje na jeho sumarizace, vytvořit systém, který by se mohl nakonec podobat nejodvážnějšímu přístupu, tedy přístupu „**Fully Automatic Summarization**“ (FAS, česky: Zcela automatická sumarizace), u nějž by platily výše zmíněné body.

Příklad. U člověkem asistované sumarizace si celý proces můžeme představit tak, že program sumarizátorovi předkládá přímo návrhy na sumarizace, nejspíš ve více verzích, ze kterých člověk na základě své nabyté zkušenosti vybere tu nejvýstižněji podanou. Celý systém by se tak dal nazvat systém ověřování sumarizací.

V případě plně automatického systému by byl celý proces schopen zvládnout stroj sám, k tomuto je však zejména nutné určit, jak dobrá je jedna sumarizace oproti jiné. Klíčovou stránkou celé sumarizace je evaluace celého procesu, tedy měření jeho úspěšnosti. Tato část, vzhledem k rozporům o to, jak úspěšnost sumarizace vlastně měřit, je však stále ne zcela prozkoumaná a my se této otázce věnujeme v kapitole věnované evaluaci (viz kapitola 8).

4.3 Automatická sumarizace, její přednosti a úskalí

K čemu by tedy počítače mohly v případě automatické sumarizace pomoci? Podívejme se společně na některé aspekty zpracování počítači.

- **Rychlost**

Ačkoliv jsou člověkem vytvořené abstrakty poměrně kvalitní, musíme lidskému přístupu k sumarizaci vytknout jednu věc a to zdlouhavost celého procesu. Zde tedy počítače vítězí na plné čáře.

- **Homogenita výstupních dat**

Bylo-li by možné sestavit systém, který by byl schopen vytvářet abstrakty, můžeme se spolehnout, že jeho náchylnost k chybám bude menší než u pracovníka, který nad sumarizacemi strávil hodiny práce.

Také se můžeme spolehnout, že počítač podává konstantní výkon. A to nejen vzhledem k rychlosti, ale zrovna tak vzhledem ke zpracování. Na stejných vstupech a stejném způsobu zpracování vždy poskytuje stejné výsledky.

- **Finanční nenáročnost**

Samozřejmě nesmíme zapomenout, že lidská práce vždy něco stojí. V případě automatických sumarizačních systémů by bylo možné nahradit lidské

pracovníky počítači, což by v konečné fázi, v případě, že software na tvorbu sumarizací bude kvalitní, vedlo ke snížení finančních výdajů na práci spojenou se sumarizací

- **Nepřetržitý provoz, oblíbená zkratka 24/7**

Počítače netrpí únavou. Pakliže nejsou přetíženy, podávají neustále konstantní výkon, což se o lidech zdaleka říci nedá.

Kdyby automatická sumarizace byla schopna pracovat srovnatelně s lidmi, uvedené důvody by jistě způsobily, že by sumarizace byla ponechána na strojích. Problém je v tom, že automatická sumarizace prozatím není schopna pracovat tak kvalitně, jak bychom požadovali.

Vzpomeneme-li na extrakt, kde se jedná o několik vět, které dokáží obsáhnout podstatnou část významu původního textu. U extraktu, ačkoliv metody pro jeho výpočet jsou stále spíš intuitivní než takové, aby byly schopny skutečně sledovat význam, máme zajištěn alespoň nějaký úspěch.

Naproti tomu u abstraktu jsme se současnými automatickými metodami sumarizace tak, jak je známe dnes, v podstatě ztraceni. Jak bylo řečeno, abstrakt pracuje na rovině významu a s obecně známými souvislostmi. Abstrakt může obsahovat informace, které nebyly v původním textu, či byly během sumarizace nějak modifikovány.

4.4 Kompresní poměr

Důležitou součástí každé sumarizace je informace o tom, jakou délku extraktu či abstraktu použijeme. Tento tzv. **kompresní poměr** je obvykle označen jako desetinné číslo v rozmezí (0-1], tedy číslo větší než nula a menší než 1 včetně, kde číslo blížící se nule označuje, že bude vybráno co nejméně slov či vět z původního textu a číslo blížící se jedničce označuje, že extraktem bude téměř celý původní text.

V práci se zejména zabýváme extrakty. U těch je kompresní poměr jednoduše vyjádřitelný jako poměr délky nového textu ku délce starého textu. Problém může nastat v reprezentaci této hodnoty. Obvykle se texty mohou značně lišit, proto se může hodnota upravovat třeba na poměr počtu vět v extraktu k počtu vět v původním textu. Či se může uvažovat poměr počtu slov v extraktu k počtu slov v původním dokumentu. Do značné míry požadavek na délku sumarizace souvisí s jejím následným využitím. V bakalářské práci jsme se zabývali zejména takovými kompresními poměry, které označují počet vět. Vychází to z logiky věci, protože základní jednotky textu, alespoň dle našeho pohledu, jsou celistvé věty. Málokdy se nám stane, že pouze část věty dává ucelený význam (pozn.: zejména u neprojektivních vět jazyků s volným slovosledem, například „Soubor se“ je část věty nedávající význam).

4.5 Co je to MEAD

Jak uvádí tvůrci metody Radev a kol. (2004), sumarizační metoda MEAD je jazykově nezávislá a automatická a generuje sumarizace na základě použití většího množství dokumentů pro trénování. Dokumenty užije pro tvorbu tzv. shluků, které jsou výstupem tzv. shlukové analýzy. Pro shlukovou analýzu je možné využít různé druhy algoritmů, což analýze MEAD přidává na síle. Metodu shlukové analýzy, konkrétně CIDR, jsme již rozebrali (viz kapitola 3).

Pro svoji práci metoda MEAD potřebuje pouze základní informace o textu. K její práci postačuje, aby dostala ke každému slovu v textu jeho základní slovní tvar (tzv. lemma) a slovní druh, který jednoznačně identifikuje slovo i v případě stejných tvarů u více slovních druhů. Pro tyto účely tedy postačuje přítomnost tzv. morfologické roviny anotace.

Výstupem shlukové analýzy a následného zpracování je ohodnocení každého ze slov vstupního dokumentu číslem, které udává jeho důležitost vzhledem ke shluku. Metoda MEAD toto ohodnocení slov přebere a na základě algoritmů pracujících právě s těmito daty získá pro každou větu číslo, které reprezentuje důležitost věty v celém textu. Po normalizaci všech získaných ohodnocení vět pro dokument získáváme pro každou větu číslo, které by mělo zastupovat obsahovou hodnotu věty vzhledem ke shluku, podle něhož jsme sumarizovali. Seřazením těchto hodnot získáme pořadí důležitosti vět v sumarizaci a užitím tzv. kompresního poměru vybereme pouze tu část důležitých vět, kterou naše aplikace vyžaduje.

4.6 Představení procesu sumarizace pomocí metody MEAD

Pro zopakování. Před samotnou metodou MEAD musí proběhnout tzv. shluková analýza (viz kapitola 2). Shluková analýza zajistí, že se ze vstupních dokumentů (k tomu určených) vytvoří shluky, tedy jakési množiny podobných dokumentů (které se reprezentují jejich tf-idf). Tyto shluky vznikají postupně. Před každou iterací procesu je na začátku dokument, jehož podobnost se spočítá ke každému z již existujících shluků. Pro podobnost dokumentu ke shluku požadujeme jistou mez tzv. similarity treshold. Pokud tuto mez dokument překročí vzhledem ke shluku, ke kterému je nejpodobnější, přidá se do shluku (načež následuje přepočítání centroidu tohoto shluku). Pokud není dostatečně podobný (není překročena mez), vytvoří se nový shluk pouze s tímto dokumentem.

Příklad. V nově vzniklých shlucích budou obvykle různá význačná slova. Např. shluk se zdravotnickými tématy bude mít nejspíše jiná reprezentující slova než shluk se sportovním tématem.

Pro dokument, který chceme sumarizovat, platí téměř totéž, co pro tvorbu shluků. Tedy jako u shluků spočítáme podobnost sumarizovaného dokumentu ke každému již existujícímu shluku. Nakonec vybereme ten nejpodobnější shluk a z něj přebereme význačná slova s jejich ohodnocením.

Po shlukujícím modulu jsme tedy v situaci, kdy máme všechna slova ve vstupním dokumentu ohodnocena váhami podle tf-idf slov ve shluku, ke kterému má dokument nejvyšší míru podobnosti.

Metoda MEAD samotná je založena na ohodnocování vět čísly. Ke každé větě přiřadíme právě takové číslo, které je tím větší, čím je věta v textu důležitější. K ohodnocování vět se používá několika přístupů, které představíme v následujících odstavcích. Tyto metody ohodnocení jsou opět reprezentovány čísly. Celkovou váhu věty určuje lineární kombinace těchto číselných hodnot.

Vstupními daty pro získání sumarizace jsou věty, jejichž slova mají ohodnocení získané ze shlukové analýzy. Použitím každé z metod získáme z těchto ohodnocení pro každou větu číslo. Lineární kombinací těchto čísel pro každou větu, kdy jednotlivé koeficienty lineární kombinace je možné nastavit v programu, získáme po seřazení pořadí vět v sumarizaci. Aplikací kompresního poměru nakonec zanedbáme věty, které už se do našeho souhrnu nehodí, protože by souhrn příliš prodloužily.

Následující podkapitoly vycházejí zejména z článku, jehož autorem je Radev a kol. (2004). Jde o definice parametrů, které jsme implementovali.

4.7 Atributy metody MEAD

Celá následující podkapitola je věnována popisu uvažovaných aspektů při ohodnocování vět.

4.7.1 Centroid Value

Prvním představovaným atributem je tzv. **Centroid Value**. Jde o nejjednodušší atribut. Přeneseně jde o váhu věty, která je daná prostým součtem všech tf-idf uvedených ve větě. Při uvažování tohoto atributu se vychází z pracovní hypotézy, že delší věty s více význačnými slovy shluku budou mít lepší ohodnocení.

Z každé věty získáme hodnotu atributu Centroid Value sečtením jednotlivých vah slov ve větě. Pokud by věta měla slova uvedená ve tvaru dvojice („základní tvar slova“, ohodnocení slova podle shluku), pak bychom u věty reprezentované vektorem

$$X = ((„\ Ema\ “, 2), (“\ mele\ “, 1), („\ maso\ “, 1.5))$$

získali hodnotu atributu Centroid value pro tuto větu $X.CentroidValue = 4.5$.

4.7.2 Positional Value

Dalším atributem je tzv. **Positional Value**. U některých textů se předpokládá jakási jejich „výstavba“. Vychází se z pracovní hypotézy, že informace, které mohou tvořit shrnutí, budou v textu zařazeny dříve, než informace, které jsou už příliš podrobné a souhrny se z nich již nedělají. Atribut Positional Value představuje pokus o zachycení důležitosti pořadí vět. Výsledkem je tedy hodnota, která určuje jakýsi pokus o zachycení pravděpodobnosti, že věta bude tvořit dobrý souhrn.

Z každé věty získáme hodnotu atributu Positional Value tak, že zjistíme celkový počet vět. Tento počet označme *n*. První věta bude ohodnocena maximální hodnotou Centroid Value z celého dokumentu (viz podkapitola 4.7.1). Tuto hodnotu označme C_{max}. Každá následující věta v pořadí, používáme-li index *i*, bude mít ohodnocení následující:

$$X.PositionalValue = \frac{(n-i+1)}{n} * C_{max.}$$

Věta tedy bude mít tím vyšší ohodnocení, čím bude její pozice blíže začátku. S každou další větou tedy hodnota tohoto ohodnocení klesá.

4.7.3 First Sentence Overlap

Atribut **First Sentence Overlap** nám dává informaci o jakémsi „přesahu“ nebo „počtu společných slov“ každé věty dokumentu z první větou toho samého dokumentu. Vychází se z pracovní hypotézy, že úvodní věta textu v sobě obsahuje nejzásadnější slova (obvykle jde o nadpis či stěžejní větu dokumentu). Zkoumá se tedy, jak moc se každá věta dokumentu obsahem podobá první větě. Uvažujeme však pouze slova se stejným lemmatem, takže i v případě, že by se věta odkazovala například zájmenem *ten* na větu první, nebyla by tato závislost zanesena.

Z každé věty získáme hodnotu atributu First Sentence Overlap vektorovým součinem reprezentace první věty a věty, kterou chceme ohodnotit. V každé větě vytvoříme opět „pojmenovaný“ vektor a jeho složkami ustanovíme lemma a jeho počet výskytů ve větě.

Například máme-li již použitou větu „Ema mele maso.“ a například větu „Míla mele Emu.“, snadno vidíme, že $Y.FirstSentenceOverlap = 2$, protože máme v každé větě lemmata Ema a mlít.

4.8 Další použité atributy.

Oproti článku, který jsme měli k dispozici, jsme ještě navíc implementovali další atributy. Vycházeli jsme z pracovní hypotézy, že by bylo dobré znát počet význačných slov v první větě (a jaksi dlouhé věty s přílišným počtem slov, které

nemají dostatečně velké váhy znevýhodnit) a také, že by bylo dobré zohlednit délku věty (čím kratší, tím lepší).

4.8.1 Atribut pro zvýhodnění kratších vět

Pokud bychom měli dvě věty, ve kterých by byla stejná ohodnocená slova, ale věty by se lišily délkou, bylo potřeba zajistit, aby se jako lepší mohla vzít věta kratší. Tímto nápadem vznikl atribut, který dával větě *tím větší váhu, čím byl její počet slov nižší*.

Například pro větu X, kterou jsme již zmínili („Ema mele maso“), se třemi slovy byl tento atribut roven $X.KratkeVety = 1/3$, pro větu se čtyřmi slovy by byl atribut roven $Z.KratkeVety = ¼$, apod.

4.8.2 Atribut pro uvažování částí dokumentů. Decay Treshold

Článek, který sepsal Radev a kol. (2004), nabádá k použití parametru, který bude načítat pouze části dokumentů, které shlukujeme. Nastavením tohoto číselného parametru na konkrétní číslo, řekněme 30, říkáme, že ze vstupních dokumentů uvažujeme pouze prvních 30 slov.

Tento aspekt vychází z pracovní hypotézy, že pro tvorbu obecných shrnutí se informace obsažené dříve v dokumentu hodí o mnoho více než ty informace, obsažené někde ke konci dokumentu.

4.9 Upravování lineární kombinace atributů

Potom, co program získá váhy všech atributů, znormuje je. Tedy převede atributy na takovou reprezentaci, aby součet každého atributu u všech vět v dokumentu dával součet jedna.

Po znormování tedy každý z atributů má „stejnou váhu“. Pro to, abychom mohli tyto hodnoty lineárně upravovat (a tím měnit důležitosti vah), můžeme reprezentace atributů upravit v poměru, který je možné nastavit při spuštění programu.

Pokud tedy požadujeme, aby program například uvažoval pouze pozici věty v dokumentu, nastavíme koeficient atributu PositionalValue na 1 a ostatní koeficienty atributů na 0.

Kapitola 5

Implementace práce

V této kapitole představíme program, který převádí prezentovanou teorii do praxe. Nahlédneme na něj z mnoha náhledů, nejen z pohledu jeho základních datových struktur, ale taktéž z pohledu členění. Podrobnější popis vstupů a výstupů programu je k nalezení v apendixu A. Nástroje, které byly k práci připojeny pro její jednodušší prezentaci či evaluaci, jsou potom v apendixu B.

5.1 Návrh a specifikace

Návrh aplikace se řídil specifikací, kdy jsme chtěli, aby aplikace byla při práci rozdělena na čtyři moduly. Těmito moduly byly moduly trénující, shlukující, sumarizační a evaluační. Práce měla být vytvořena především v programovacím jazyce C++.

Během práce samozřejmě došlo k mnohým rozšířením, exportu většího množství dat a podobně. V následující kapitole program představujeme v jeho závěrečné formě.

5.2 Hlavní datové struktury

Programovací jazyk C++ umožňuje mnohé. V datových strukturách, které byly navrženy v prvních verzích, vystupovaly některé struktury a třídy z knihovny STL. Avšak během práce s nimi se ukázalo, že jejich přenositelnost mezi operačními systémy je přinejmenším problematická. Proto jsme od zamýšleného využití těchto tříd upustili a práci zcela naprogramovali „od zelené louky“. Použili jsme pouze nejnutnější nástroje pro pohodlnější vývoj.

Postupem doby, po dlouhém uvažování nad strukturami, které by byly pro bakalářskou práci nejvhodnější, jsme dospěli k závěru, že takové struktury existují dvě. A těmi jsou spojový seznam a AVL strom.

Spojový seznam byl zvolen zejména v místech, kde docházelo k opakovanému lineárnímu procházení, ale pouze v těch místech, kde nebylo nutné třídění prvků. AVL strom pak posloužil v ostatních případech.

AVL strom jsme používali v práci zejména tak, aby suploval schopnosti „pojmenovaného“ vektoru. Běžný AVL strom by nestačil, pomohli jsme si tedy stromem s obousměrným provázáním a doplnili jej o možnost lineárně procházet prvky. Ukázalo se, že tato struktura vyhovuje algoritmům, které v práci používáme. Tento vektor byl setříděný vždy podle aktuálních potřeb. K tomuto účelu jsme v mnohém využili možnosti přetěžování operátorů v C++. Mezi AVL stromy pak

byly definovány další operace pro práci s nimi (například při počítání podobnosti dokumentů ke shlukům, apod.).

5.3 Moduly programu

Program byl pro větší přehlednost napsán tak, aby bylo možné používat jenom některé moduly. Logicky vzniklo rozdělení na několik částí. Modul příkazové řádky se samotnou funkcí programu sice úplně nesouvisí, přesto jej zmíním.

5.3.1 Příkazová řádka

Pro příkazovou řádku byly napsány samostatné třídy, procedury a funkce, které jsou uloženy v souborech **PrikazovaRadka.h**, resp. **PrikazovaRadka.cpp**. Rozšiřování příkazové řádky o nové parametry je možné, stačí dopsat podle ostatních nové parametry do funkce DefinujParametry. Přidávat se smí nové moduly (s velkým písmenem zástupného jména) i nové parametry (s malým písmenem zástupného jména).

5.3.2 Načítání vstupů

K dané problematice byl v práci přímo vytvořen soubor **vstup.h**, resp. **vstup.cpp**, v němž jsou uchovány základní třídy pro práce s různými vstupy. V souboru se nacházejí definice základních prvků celého programu – a to třídy Slovo a FSlovo a TfIdfInfo.

Další třídy jsou výlučně věnovány vstupům resp. výstupům programu. Základní šablonovatelné třídy, *Vstup* resp. *Vystup,* jsou potomky třídy *ifstream*, resp. *ofstream*. Další třídy, které zejména načítají z různých zdrojů, jsou potomky těchto tříd *Vstup* resp. *Vystup*. Tím byla zajištěna jistá konzistence a možnost rozšiřovat třídy dále podle nových vstupů. Podmínkou je pouze implementování funkce bool ZiskejSlovo(TSlovo & slovo), která navrací ze vstupu načtené slovo.

Implementováno bylo načítání ze souborů typu PML z projektu PDT (Hajič a kol., 2006) pomocí třídy VstupPDT a načítání ze souborů, které byly zapůjčeny z projektu CLEF 2007. Těmto souborům jsme přiřadili příponu „p“ (třída VstupP).

5.3.3 Trénující modul

Trénující modul má za úkol načíst všechny soubory, spočítat jejich tf-idf spolu s vektorem reprezentace hodnot idf. Také má za úkol očistit data

od nežádoucích vlivů (například vynechat některé slovní druhy, apod.). Během práce zobrazuje aktuálně zpracovávané položky uživateli přímo na obrazovku.

Trénující modul je jedním z prvních, který poskytuje souborový výstup. Výstupem tohoto modulu jsou zejména soubory **trained.tf**, **trained.idf** a **trained.at**, ve kterých se vyskytují natrénovaná data, tedy data, která byla mnoha procedurami získána ze vstupních souborů. Jmenovitě četnost výrazů, inverzní četnost výrazů v dokumentech a také počet dokumentů, ve kterých se jednotlivé výrazy vyskytují. Data jsou řazena podle znakové sady, která je na aktuálním stroji přístupná. Při načítání na pořadí dat nezáleží. Abecední pořadí bylo zvoleno kvůli větší přehlednosti pro uživatele. Společně s informací ze souboru log.txt o počtu souborů a o podobnosti, která byla užita pro výpočet shluků, je možné nejen získaná data přepočítat, ale taktéž je možné data využít k tvorbě většího množství dat.

Trénující modul je místo, kde se ukáže, zda má počítač dostatek paměti pro zpracování. Po načtení všech dat a ukončení modulu je dosaženo maximální najednou používané paměti.

5.3.4 Shlukující modul

Shlukující modul přebírá již načtené dokumenty spolu s vektorem idf a provádí shlukovou analýzu dat. Tato část je nejvíce časově náročná. Postupnou tvorbou shluků, do kterých jsou přidávány příchozí dokumenty, totiž vzniká větší a větší množství dat, které je v každém kroku nutné projít (vzpomeneme-li si na neustálé porovnávání centroidu a dokumentu). V pozdějších fázích tedy může jedna iterace spotřebovat i několikanásobně více času, než iterace v prvotních fázích.

Po této fázi vytvoří program soubor **clusters.txt**, který je z hlediska zpracování pravděpodobně nejdůležitější. Tento soubor obsahuje přehled získaných shluků společně s ohodnocením slov pro každý shluk. V každém shluku je navíc zobrazen každý obsažený dokument a u něj je zobrazeno, kolik obsahoval slov. Z toho se dá udělat celkový obrázek nejen toho, kolik dokumentů je ve shluku obsaženo, ale i toho, kolik slov je ve shluku obsaženo.

Po vytvoření shluků přechází program do fáze, ve které spočítá, jaké jsou mezi-shlukové podobnosti. Výstup směruje do souboru **clustersimilarity.txt**. Pro zjednodušení a zrychlení bylo upuštěno od ukládání matice podobností mezi shluky (protože při nešťastném nastavení parametrů by rozměry matice mohly narůst do gigantických rozměrů) a s tím vznikla i pohnutka ukládat data pouze do jedné poloviny matice. Druhá polovina je samozřejmě symetrická. **Pro případné zobrazení celkových výsledků mezi-shlukových podobností je nutné matici (vyjma hlavičky) ze souboru clustersimilarity.txt „odzrcadlit“ z pravé horní poloviny do levé dolní části matice a následně zpracovat v některém z programů určených pro práci s takovými daty.**

V závislosti na počtu shluků je časově nejnáročnější buď shlukování samotné (při malém počtu shluků) anebo počítání mezi-shlukových podobností (při větším počtu shluků). Po absolvování shlukujícího modulu program zřejmě absolvoval

zhruba 90% celkového času svého běhu (pokud pouze nenačítáte výsledky ze souborů).

5.3.5 Sumarizační modul

Načítání souborů pro sumarizaci v sumarizačním modulu probíhá podobně jako v trénujícím modulu, zjištění nejpodobnějšího shluku je nalezeno podobně jako při proceduře ve shlukujícím modulu.

V sumarizačních algoritmech jde zejména o manipulaci s ohodnocením jednotlivých slov, které bylo zjištěno na základě připodobnění k nejpodobnějšímu shluku.

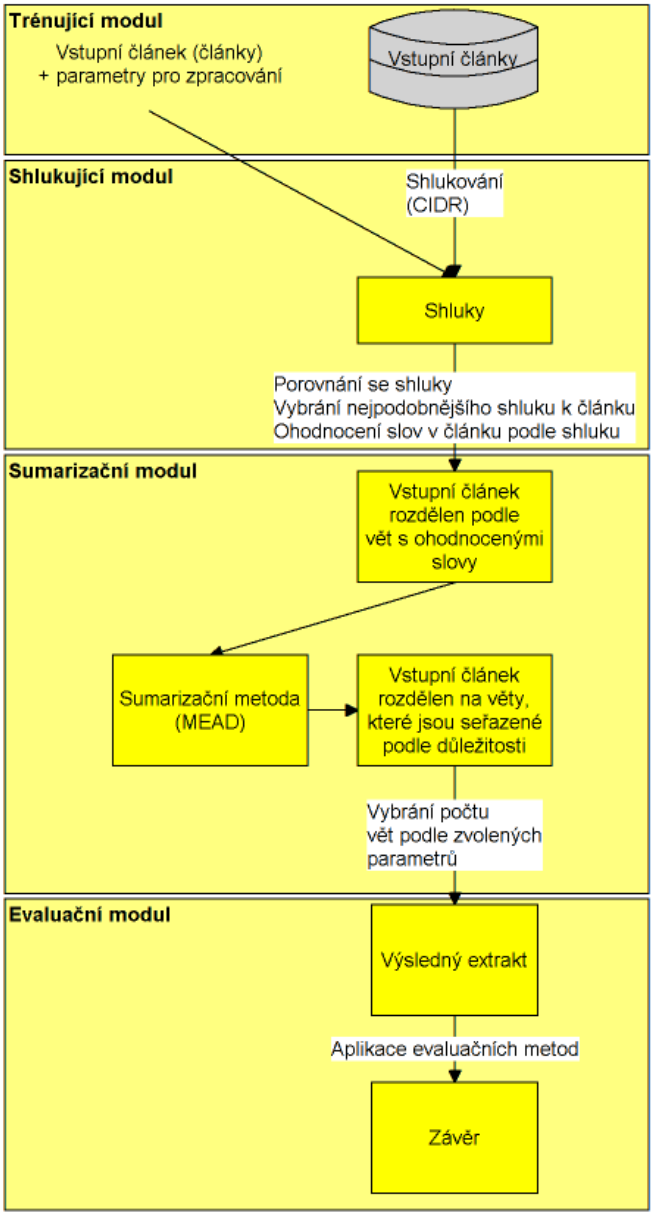
Výstupem sumarizačního modulu jsou pro každý soubor tři soubory. První je označen jako **summarizationmodule**, který uchovává podrobnosti o souboru a nalezení nejpodobnějšího shluku, **summarized**, jenž obsahuje pořadí vět v sumarizaci. Soubor s prefixem **summarized** a sufixem **sum** ukazuje sumarizaci na základě zadaného kompresního poměru.

5.3.6 Evaluační modul

Za poslední z modulů je považována zejména webová aplikace pro anotaci výstupů programu sumarizátory, která poskytne dostatečnou zpětnou vazbu spolu s vytvořením výsledků o úspěšnosti celého projektu. Evaluační modul je tedy spíše „imaginární“ modul, jehož většinu práce zastanou běžní lidé. Nutno podotknout, že evaluace sumarizací samotná není zcela vyřešena a její nynější provedení je jen jedním z pokusů, jak by mohla vypadat. Více viz kapitola 8.

5.3.7 Diagram zpracování dat v bakalářské práci

V následujícím diagramu se snažíme popsat celkově proces zpracování. Jsou zde zachyceny nejen jednotlivé moduly, ale celková návaznost prováděných úkonů na sebe od tvoření shluků až po výstup sumarizace a evaluaci.



Obrázek 3: Diagram zachycuje zpracování dat v práci. Postupně se zapojují moduly trénující, shlukovací, sumarizační i evaluační.

5.4 Generování programátorské dokumentace, problémy

Programátorská dokumentace k programu v C++ byla vygenerována programem Doxygen. Bohužel program není zcela dokonalý, proto, pro doplnění, je dokumentace tříd Uzel a _BASE umístěna v souboru **GlobalDeclarations.h**. K projektu byla použita knihovna DIRENT.H kvůli načítání souborů z disku. Ta obsahuje struktury DIR a dirent, takže v tomto případě není neokomentování funkcí vinou projektu.

Programátorská dokumentace k programu pro vizualizaci výsledků v Javě byla přímo vytvořena v programu NetBeans 6.1.

5.5 Zlepšování vlastností programu

Jednou z klíčových vlastností programu měla být jeho rychlost. V následující kapitolce rozebereme, na jaké problémy jsme při vyhotovení práce narazili.

5.5.1 Zlepšování paměťové náročnosti programu

Při zpracovávání programu jsme do něj zahrnuli i možnost sledování paměti, která je spotřebovávaná. V raných fázích vývoje docházelo ke ztrátám paměti, které však po dotvoření jakési správy paměti zmizely postupným testováním. Tyto ztráty se projevovaly při načítání programu, kdy již neměl dostatek paměti kvůli zapomenutému spuštění jednoho z destruktorů.

5.5.2 Zlepšování časové náročnosti programu

Kamenem úrazu programu byla po dlouhou dobu jeho pomalá práce. Na malých vstupech se kondiční handicap programu neprojevoval, avšak na větších datech (řádově stovky dokumentů) začaly být problémy s načítací fází markantní. Člověk by nevěřil, co dokáže vektor s hodnotami *idf*. Tento vektor byl totiž v prvotních fázích procházen při každém načtení dokumentu. Jelikož se jedná o vektor, který nebyl svou velikostí zcela zanedbatelný, projevovala se do programu stále větší časová náročnost na jeho průchod (tak, jak vektor stále rostl). Řešení bylo nasnadě – při načítání dokumentů se procházejí všechny dokumenty zároveň tak, aby se vytvoření vektoru dělo najednou.

5.5.3 Závěr plynoucí ze zlepšování programu

Díky nastíněným zlepšením se práce programu řádově zrychlila a program je tak nyní schopen na běžných počítačích pracovat až s tisícovkami dokumentů.

Kapitola 6

Uživatelská příručka

V této kapitole představujeme program z uživatelského hlediska. Navíc je příručka doplněna o příklady nastavení parametrů programu pro získání co nejlepších výsledků.

6.1 Úvodní informace

Tato uživatelská příručka dává uživateli přehled o funkcích a možnostech programu pro sumarizaci textu, který byl vytvořen Karlem Vandasem. Dokumentace se vztahuje k programu verze 1.0. Program je schopen ze vstupních souborů daného formátu vytvořit výstup, který bude sumarizací vstupu. Více o informací o programu je uvedeno na webové stránce projektu http://sumarizace.bajecni.cz.

6.2 Instalace

Program je distribuován v archivu zip, který je stažitelný z webové stránky projektu v sekci „Download“. Velikost archivu se pohybuje okolo 1 MB.

Instalace proběhne rozbalením celého zip archivu do složky, ve které má uživatel i program právo zápisu. Preferovaná (a v instalaci přednastavená) je buď složka „c:\textsum\“ anebo její verze v domovském adresáři na OS typu UNIX (cesta do složky ani název složky samotné NESMÍ obsahovat mezery). Základní instalace obsahuje soubory a adresáře, které jsou uvedeny v souboru README.TXT v hlavní složce instalace. Pro OS Windows naleznete v hlavní složce projektu zkompilovanou verzi programu pojmenovanou textsum.exe. Pro ostatní operační systémy je nutné program přeložit kompilátorem pro C++ do hlavní složky projektu pod jménem textsum.exec. V hlavní složce projektu je pro tento účel vytvořen makefile, který stačí spustit příkazem sh ./makefile. Pokud Vám program na operačním systému unixového typu nejde nainstalovat, nebo požadujete jiný typ kompilace pro OS Windows, prostudujte „Instalace krok za krokem“ v README.TXT. Program může při kompilaci generovat varování o nevyužitých proměnných (v důsledku zpětné kompatibility verzí) či používání nebezpečné funkce gmtime(). Jde pouze o informativní hlášky a v žádném případě tyto aspekty programu neovlivňují jeho běh.

6.3 Spuštění programu

Program se spustí z příkazové řádky. Je tedy nutné uvést jméno programu (textsum.exe pro OS Windows, textsum.exec pro OS typu UNIX) a za něj příslušné parametry, které naleznete v dále (viz kapitola 6.6).

6.4 Nároky programu

6.4.1 Nároky programu na technické vybavení počítače

Program vyžaduje dostatečnou výpočetní kapacitu, s níž je možné provádět rozsáhlé výpočty. Lépe program poběží na počítačích s dostatečným množstvím paměti RAM a co nejrychlejším procesorem. Program nevyužívá schopností počítače spouštět více procesů na různých jádrech procesoru, proto vyšší počet jader procesoru rychlosti programu nepomůže. Na počítači se 4 GB RAM je možné zpracovat až několik desítek tisíc souborů. Pro paměťovou náročnost je klíčová část načítání. Pokud ji program úspěšně dokončí, je jisté, že více paměti než v této proceduře již nebude potřeba.

6.4.2 Nároky programu na programové vybavení počítače

Program naprogramovaný v C++ na operačních systémech UNIX vyžaduje přítomnost kompilátoru pro jazyk C++. Zkompilovaná verze pro OS Windows je již součástí základní instalace.

Program pro vizualizaci výstupů programu pro sumarizaci textu je naprogramovaný v Javě a vyžaduje přítomnost interpretu pro Javu, tedy programu, který umožňuje spouštět programy v tomto jazyce, nejméně ve verzi 6. Tento program se dá zdarma stáhnout z internetu. Více na stránkách http://www.java.com/en/download/index.jsp . Není však třeba se hned děsit, programovací jazyk Java používá mnoho aplikací, mimo jiné i prohlížeč webových stránek, proto je možné, že již máte požadované programy nainstalovány.

6.5 Povolené vstupní formáty

Program pracuje hlavně se soubory typu PML. Tento typ souborů byl definován v projektu Pražského závislostního korpusu 2.0 (Hajič a kol., 2006,

Prague Dependency Treebank 2.0). V programu je však možné zpracovávat i soubory typu „p“, které byly zapůjčeny z projektu CLEF.

6.5.1 Umístění zpracovávaných souborů

Soubory, které jsou určené pro sumarizaci buď uvádíme v příkazu, kterým spouštíme program, anebo je vkládáme do složky **toSummarize** v hlavní složce projektu.

Soubory, které jsou určené pouze pro tvorbu shluků můžeme buď vkládat do složky **input**, anebo do jiné složky, kterou následně můžeme přímo uvést v příkazu, kterým spouštíme program.

Výstupní soubory se ukládají do složky **output**.

6.6 Parametry pro práci programu

Program pracuje na základě parametrů, které jsou uvedeny v příkazové řádce. Program vyžaduje, aby byly parametry bez tzv. bílých mezer, tzn., že například jména zpracovávaných souborů či složek včetně jejich relativních či absolutních umístění nesmějí obsahovat mezery, tabulátory či odřádkování. Proto je lepší instalovat program do složek s jednoduchou a krátkou cestou.

Zásady pro vkládání parametrů do příkazové řádky jsou celkem jednoduché. Vůbec nezáleží na tom, jaké zvolíte pořadí parametrů (myšleno, zda nejdříve uvedete parametry načítacího modulu a až pak shlukujícího, či obráceně).

Parametry jsou rozděleny logicky na několik modulů. Každý z těchto modulů může mít jinou definici pro každý jednotlivý parametr. Jména modulů jsou označena velkými písmeny, takže žádný parametr modulu nesmí být pojmenován velkým písmenem.

Například jsme-li v modulu načítacím, v našem případě je tento modul označen písmenkem „L“, tak můžeme uvádět další a další parametry tohoto modulu, aniž bychom potřebovali pokaždé znovu uvádět to, že jsme v modulu „L“. Například tedy můžeme uvést „-L -n -i „dir01“ -p“, což znamená, že nebudeme uvažovat výchozí data ve složce input, ale oproti ní přidáme do souborů ke zpracování složku dir01 a budeme načítat předchozí shluky.

Logické hodnoty se uvedením přepínají z true na false a obráceně.

6.6.1 Načítací a trénující modul

Načítací modul je uvozen parametrem „-L“ a za ním mohou jej následovat parametry uvedené v tabulce.

-L	**LoadingModule, Načítací modul**
-c *soubor* nebo **--PreviousClustersFile** *soubor* *např. -L -c „2009.07.14.21.47.clusters.txt“ označuje, že bude místo počítání shluků načten již vytvořený soubor*	**Parametr nastavuje řetězcovou hodnotu PreviousClustersFile** Výchozí hodnota atributu je prázdný řetězec, což znamená, že se nebude předchozí soubor shlukování uvažovat.
-d *číslo* nebo **--DecayTreshold** *číslo* *např. -L -d 5 označuje, že z každého dokumentu se načte prvních 5 slov*	**Parametr nastavuje číselnou hodnotu DecayTreshold** Výchozí hodnota atributu DecayTreshold je -d -1, což znamená, že nebude omezen počet načítaných slov z dokumentu
-f *soubor* nebo **--ProcessFile** *soubor* *např. -L -f „sample.m“ označuje, že proces sumarizace proběhne na souboru „sample.m“*	**Parametr nastavuje řetězcovou hodnotu ProcessFile** Výchozí hodnota atributu ProcessFile je prázdný řetězec, tedy informace, že zpracovávány budou pouze soubory ze složky toSummarize. **Upozornění. Při každém spuštění se sumarizují soubory ze složky toSummarize. Pokud si je nepřejete sumarizovat, přesuňte je pryč z této složky.**
-i *adresáře* nebo **--AddInputDirectories** *adresáře* *např. -L -i „dir01 dir02 dir03“ označuje, že při procesu shlukování budou oproti výchozímu adresáři uvažovány ještě adresáře dir01, ...*	**Parametr nastavuje řetězcovou hodnotu AddInputDirectories** Výchozí hodnota atributu AddInputDirectories je prázdný řetězec, tedy informace, že žádné další adresáře nejsou uvažovány. **Důležité. Není dovoleno, aby adresáře obsahovaly ve svém názvu či relativní cestě bílý znak (mezeru, tabulátor, odřádkování)! Upozornění. Parametr je povinný při nastavení parametru -L -n.**
-n nebo **--NoDefault** *např. -L -n -i „dir01“ označuje, že nebudou načítána výchozí data ze složky input, místo toho se použijí vstupní soubory ze složky „dir01“*	**Parametr nastavuje logickou hodnotu NoDefault** Výchozí hodnota atributu NoDefault je false, což znamená, že data ze složky input budou načítána. **V případě, že aktivujete tento parametr, je nutné dodat složky k načítání v parametru -L -i!**
-p nebo **--PreviousClusters** *např. -L -p označuje, že bude při shlukové analýze použito dat, která byla jako poslední napočítána ve složce output*	**Parametr nastavuje logickou hodnotu PreviousClusters** Výchozí hodnota atributu PreviousClusters je false, což znamená, že data nebudou načítána z předchozího procesu. Pokud je hodnota nastavena na true, ze souboru .lastClusters.txt ve složce output se načte předpona posledních shluků.
-q nebo **--QuitAfterClustering** *např. -L -q označuje, že po zpracování shluků bude ukončeno provádění programu (bez provedení sumarizací)*	**Parametr nastavuje logickou hodnotu QuitAfterClustering** Výchozí hodnota atributu QuitAfterClustering je false, což znamená, že proces bude po shlukování nadále pokračovat. Pokud je hodnota nastavena na true, program skončí po dokončení tvorby shluků.

6.6.2 Nastavení ořezávání slov

V programu jsme uvažovali i nad tím, jak některá slova oproti ostatním zanedbat. Některé slovní druhy totiž na sumarizaci nemusejí mít úplně vliv. Proto je v programu tato možnost také zavedena. Více v tabulce.

-W	**WorldClassModule, Modul načítání slov**
-n nebo **--Nouns** *např. -W -n označuje, že se budou podstatná jména uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Nouns** Výchozí hodnota atributu je false, což znamená, že se podstatná jména nebudou při procesu uvažovat. Uvedením se atribut nastaví na true.
-a nebo **--Adjectives** *např. -W -a označuje, že se budou přídavná jména uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Adjectives** Výchozí hodnota atributu je false, což znamená, že se přídavná jména nebudou při procesu uvažovat. Uvedením se atribut nastaví na true.
-p nebo **--Pronouns** *např. -W -p označuje, že se budou zájmena uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Pronouns** Výchozí hodnota atributu je false, což znamená, že se zájmena nebudou při procesu uvažovat. Uvedením se atribut nastaví true.
-c nebo **--Counts** *např. -W -c označuje, že se budou číslovky uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Counts** Výchozí hodnota atributu je false, což znamená, že se číslovky nebudou při procesu uvažovat. Uvedením se atribut nastaví true.
-v nebo **--Verbs** *např. -W -v označuje, že se budou slovesa uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Verbs** Výchozí hodnota atributu je false, což znamená, že se slovesa nebudou při procesu uvažovat. Uvedením se atribut nastaví true.
-d nebo **--Adverbs** *např. -W -d označuje, že se budou příslovce uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Adverbs** Výchozí hodnota atributu je false, což znamená, že se příslovce nebudou při procesu uvažovat. Uvedením se atribut nastaví true.
-r nebo **--Prepositions** *např. -W -r označuje, že se budou předložky uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Prepositions** Výchozí hodnota atributu je false, což znamená, že se předložky nebudou při procesu uvažovat. Uvedením se atribut nastaví true.
-j nebo --**Junctions** *např. -W -j označuje, že se budou spojky uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Junctions** Výchozí hodnota atributu je false, což znamená, že se spojky nebudou při procesu uvažovat. Uvedením se atribut nastaví true.
-t nebo --**Particals** *např. -W -t označuje, že se budou částice uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Particals** Výchozí hodnota atributu je false, což znamená, že se částice nebudou při procesu uvažovat. Uvedením se atribut nastaví true.
-i nebo --Interjections *např. -W -i označuje, že se budou citoslovce uvažovat, jinak ne*	**Parametr nastavuje logickou hodnotu Interjections** Výchozí hodnota atributu je false, což znamená, že se citoslovce nebudou při procesu uvažovat. Uvedením se atribut nastaví true.

6.6.3 Nastavení shlukujícího modulu

Důležitá nastavení pro proces shlukování jsou uvedena v parametrech pro shlukující modul. Klíčovým parametrem je hlavně atribut SimilarityTreshold.

-C	**ClusterModule, Modul pro shlukování**
-c nebo **--CountClusterSimilarities** *např. -C -c označuje, že budou po procesu shlukování spočítány mezi-shlukové podobnosti, jinak ne*	**Parametr nastavuje logickou hodnotu CountClusterSimilarities** Výchozí hodnota atributu je false, což znamená, že se mezi-shlukové podobnosti nebudou počítat. Uvedením se atribut nastaví na true. **Upozornění. Počítání mezi-shlukových podobností může být pro velké množství shluků velice časově náročné!**
-k *číslo* nebo **--KeepTreshold** *číslo* *např. -C -k 20 označuje, že v každém shluku bude po ukončení procesu shlukování nejvýše 20 slov*	**Parametr nastavuje číselnou hodnotu KeepTreshold** Výchozí hodnota je -1, což znamená, že počet slov ve shluku nebude tímto atributem omezen. V případě nastavení na kladnou hodnotu ***n*** bude v každém shluku po ukončení procesu shlukování nejvýše ***n*** slov.
-m *číslo* nebo **--MaxClusters** *číslo* *např. -C -m 5 označuje, že při procesu shlukování vznikne nejvýše 5 shluků*	**Parametr nastavuje číselnou hodnotu MaxClusters** Výchozí hodnota je -1, což znamená, že počet shluků nebude tímto atributem omezen. V případě nastavení na kladnou hodnotu ***n*** bude počet shluků po ukončení shlukujícího procesu nejvýše ***n***.
-s *číslo* nebo **--SimilarityTreshold** *číslo* *např. -C -s 0.02 označuje, že požadovaná podobnost u dokumentů v jednom shluku může být nižší, tím pádem vznikne méně větších shluků*	**Parametr nastavuje číselnou hodnotu SimilarityTreshold** Výchozí hodnota je 0.05, což znamená, že při tvorbě shluků bude za dostatečnou podobnost dokumentu ke shluku považována hodnota 0.05 a vyšší. To způsobí přidání dokumentu do shluku. Hodnoty menší než 0.05 budou považovány za nedostatečné, tím pádem nebude dokument do shluku přidán.

6.6.3.1 Jak nastavit parametry u shlukujícího modulu

Jak již bylo popsáno, klíčovým atributem shlukujícího modulu je atribut SimilarityTreshold. Atribut SimilarityTreshold má trochu složitější způsob, kterým se s ním zachází. Jeho nastavení zejména ovlivňuje, co od výsledků shlukové analýzy očekáváme. Pakliže spíše chceme vytvořit více shluků, avšak s dokumenty, které jsou si velmi blízké, volíme hodnotu atributu vyšší, řekněme v rozmezí [0.1,0.2]. Vyšší hodnoty způsobí, že podobnost dokumentu ke shluku bude natolik striktní (pamatujme, že se počítá podobnost dokumentu k celému význačnému vektoru každého shluku, tzv. centroidu, který může mít řádově tisíce slov), že nám takové nastavení vytvoří shluky pravděpodobně pouze s duplicitními soubory v případě, že počítáme podobnost ke shluku, který nebyl příliš ovlivněn ostatními dokumenty.

Pokud bychom chtěli atribut SimilarityTreshold nastavovat v rozmezí hodnot [0,0.025], může se nám zase stát, že podmínky na tvorbu shluků budou natolik lehce splnitelné, že se nám bude vytvářet málo shluků, ale o to větších. Nastavení tohoto parametru tedy není úplně jednoduché a rozhodně vyžaduje jakýsi cit, který může

vycházet z předchozího testování. Doporučenou hodnotou pro nastavení atributu Similarity Treshold je číslo 0.05.

Nastavení každého z dvojice parametrů SimilarityTreshold a CountClusterSimilarities může mít obrovský vliv na celkovou dobu výpočtu programu. Orientační hodnoty parametru Similarity Treshold a dalších, které byly během vývoje programu užívány, jsou uvedeny v závěru práce (viz kapitola 7).

Doplňkem k parametru SimilarityTreshold je parametr MaxClusters, který určitým způsobem omezuje počet vytvářených shluků (pokud by docházelo k tvorbě příliš mnoha shluků). Tento parametr je spíše pojistkou, pokud z nějakého důvodu bude docházet k tvorbě velkého množství shluků i přes nízkou hodnotu atributu SimilarityTreshold.

Parametr KeepTreshold naopak umožňuje, aby proces následné sumarizace probíhal co nejrychleji. Toho je docíleno nízkým počtem slov ve shluku, který KeepTreshold umožňuje omezovat.

6.6.4 Nastavení sumarizačního modulu

Důležitá nastavení pro proces shlukování jsou uvedena v parametrech pro shlukující modul. Klíčovým parametrem je hlavně atribut SimilarityTreshold.

-S	**SummarizationModule** **Modul pro tvorbu sumarizací**
-a nebo **--LogicalAndOfRates** *např. -S -a označuje, že při délce sumarizace bude zároveň uvažován jak počet vět, tak i podíl smyslu textu*	**Parametr nastavuje logickou hodnotu LogicalAndOfRates** Výchozí hodnota atributu je false, což znamená, že se tento atribut nebude uvažovat. Uvedením se atribut nastaví na true. Při zapnutí parametr zajistí, aby se extrakt udržoval v délce takové, aby při jejím překročení zároveň v počtu vět a v podílu jejího „smyslu“, byl extrakt ukončen
-l nebo **--LessOrEqualValuesOfRates** *např. -S -l -r 0.2 označuje, že extrakt bude dlouhý maximálně 20% vět ze vstupního textu*	**Parametr nastavuje logickou hodnotu LessOrEqualValuesOfRates** Výchozí hodnota atributu je false, což znamená, že se tento atribut nebude uvažovat. Uvedením se atribut nastaví na true. Při zapnutí parametr zajistí, aby se extrakt udržoval v délce těsně kratší než požadované.
-m *číslo* nebo **--MaximalNumberOfSentences** *číslo* *např. -S -m 3 označuje, že extrakt bude složen nejvýše ze 3 vět*	**Parametr nastavuje číselnou hodnotu MaximalNumberOfSentences** Výchozí hodnota atributu je -1, což znamená, že se atribut neuvažuje. Pokud nastavíme atribut na kladné číslo ***n***, bude mít extrakt délku nejvýše ***n*** vět.
-r *číslo* nebo **--CompressionRate** *číslo* *např. -S -r 0.5 označuje, že v extraktu bude použito okolo 50% vět ze vstupního textu*	**Parametr nastavuje číselnou hodnotu CompressionRate** Výchozí hodnota atributu je 0.2, což znamená, že se z původního textu použije maximálně zhruba 20% vět. Nastavení má smysl pouze v rozmezí (0,1). Atribut nastavíme na vyšší hodnoty z tohoto rozmezí, pokud chceme delší extrakt, jinak jej nastavíme na nižší hodnoty z rozmezí.
-s *číslo* nebo –**SenceRate** *číslo* *např. -S -s 0.5 označuje, že v extraktu bude použito okolo 50% smyslu z původního textu.*	**Parametr nastavuje číselnou hodnotu SenceRate** Výchozí hodnota atributu je 0.2, podobně jako u CompressionRate, což znamená, že se z původního textu použije maximálně zhruba 20% smyslu. Za tento smysl je považována hodnota, která vyjde pro každou větu po sečtení lineárních kombinací parametrů a znormování. Smysl celého původního textu je roven 1. Nastavení parametru má smysl v rozmezí (0,1).

6.6.4.1 Nastavení jednotlivých sumarizačních metod

Každá sumarizační metoda může mít na extrakt různý vliv, proto je možné mimo toho, že lze některé z těchto metod pro sumarizaci třeba úplně vypnout, je také možné jejich účinnost upravovat v různých poměrech.

Nastavování parametrů u jednotlivých sumarizačních metod	
-c *číslo* nebo **--CentriodValue** *číslo* *např. -S -c 5 označuje, že v lineární kombinaci pro výpočet důležitosti věty bude mít metoda CentroidValue koeficient 5*	**Parametr nastavuje číselnou hodnotu CentroidValue** Výchozí hodnota atributu je 0, což znamená, že se tato metoda nebude při sumarizaci uvažovat. Nastavením na reálnou hodnotu ***r*** říkáme, že v lineární kombinaci pro výpočet důležitosti věty bude mít atribut CentroidValue koeficient ***r***.
-f *číslo* nebo **--FirstSentenceOverlap** *číslo* *např. -S -f 5 označuje, že v lineární kombinaci pro výpočet důležitosti věty bude mít metoda FirstSentenceOverlap koeficient 5*	**Parametr nastavuje číselnou hodnotu FirstSentenceOverlap** Výchozí hodnota atributu je 0, což znamená, že se tato metoda nebude při sumarizaci uvažovat. Nastavením na reálnou hodnotu ***r*** říkáme, že v lineární kombinaci pro výpočet důležitosti věty bude mít atribut FirstSentenceOverlap koeficient ***r***.
-i *číslo* nebo **--InverseWordsNumberValue** *číslo* *např. -S -i 5 označuje, že v lineární kombinaci pro výpočet důležitosti věty bude mít metoda InverseWordsNumberValue koeficient 5*	**Parametr nastavuje číselnou hodnotu InverseWordsNumberValue** Výchozí hodnota atributu je 0, což znamená, že se tato metoda nebude při sumarizaci uvažovat. Nastavením na reálnou hodnotu ***r*** říkáme, že v lineární kombinaci pro výpočet důležitosti věty bude mít atribut InverseWordsNumberValue koeficient ***r***.
-k *číslo* nebo **--KeyWordsValue** *číslo* *např. -S -k 5 označuje, že v lineární kombinaci pro výpočet důležitosti věty bude mít metoda KeyWordsValue koeficient 5*	**Parametr nastavuje číselnou hodnotu KeyWordsValue** Výchozí hodnota atributu je 0, což znamená, že se tato metoda nebude při sumarizaci uvažovat. Nastavením na reálnou hodnotu ***r*** říkáme, že v lineární kombinaci pro výpočet důležitosti věty bude mít atribut KeyWordsValue koeficient ***r***.
-p *číslo* nebo **--PositionalValue** *číslo* *např. -S -p 5 označuje, že v lineární kombinaci pro výpočet důležitosti věty bude mít metoda PositionalValue koeficient 5*	**Parametr nastavuje číselnou hodnotu PositionalValue** Výchozí hodnota atributu je 0, což znamená, že se tato metoda nebude při sumarizaci uvažovat. Nastavením na reálnou hodnotu ***r*** říkáme, že v lineární kombinaci pro výpočet důležitosti věty bude mít atribut PositionalValue koeficient ***r***.

6.6.4.2 Jak nastavit parametry u sumarizačního modulu

Sumarizační modul a jeho parametry vyžadují trochu hravosti uživatele. Program nabízí nepřeberné množství možností, jak parametry nastavit. Při nastavování parametrů je však důležité, aby alespoň jeden z parametrů byl nastavený na nenulovou hodnotu.

Parametr CentroidValue nám může pomoci při upřednostňování vět, v nichž je co nejvíce obsažených slov ze shluku, podle kterého dokument sumarizujeme. Avšak tento parametr „nasává“ všechna slova ze shluku, ať jde o slov hodně významná, či méně významná

Parametr FirstSentenceOverlap nám může pomoci při upřednostňování vět, které mají co nejvíce stejných základních tvarů slov s první větou. Protože z dat, která byla pro projekt určena, je obvykle první věta nadpisem dokumentu, má tento parametr také svůj význam.

Parametr InverseWordsNumberValue nám může pomoci při upřednostňování vět, které mají co nejkratší zápis (myšleno v počtu slov). Parametr je spíš doplňujícího charakteru, aplikace se nepočítá s jeho použitím jako s hlavním parametrem.

Parametr KeyWordsValue nám může pomoci při upřednostňování vět podle počtu klíčových slov, která obsahují. Tento počet není závislý na tf-idf, ale pouze na absolutním počtu výskytů slov. Opět jde pouze o doplňující parametr.

Parametr PositionalValue nám může pomoci při upřednostňování vět, které v původním textu byly dříve než ostatní. Předpokládá se, že text je logicky stavěn od jednoduššího ke složitějšímu. Opět jde spíše o doplňující parametr.

Kapitola 7

Testování programu

V závěru celého projektu došlo k praktickému nasazení programu na výpočetní servery ÚFALu. Předtím však bylo nutné otestovat a odhadnout, jak dlouho program na jistých vstupech poběží. V této kapitole představujeme testy, které tomuto praktickému nasazení předcházely.

7.1 Úvod, použité příkazy k testování

Během testování práce vzniklo několik přehledů toho, jakou zátěž program na běžném počítači snese. Tyto odhady bylo nutné provést zejména kvůli získání informací o tom, jak program poběží dlouho na kterých vstupech. Tyto informace byly potřeba pro nasazení programu na serveru na větší množství dat.

Program byl testován způsobem, kdy byl postupně zatěžován množstvím vstupních souborů pro tvorbu shluků. První vstup, na kterém běžel, byl jeden soubor, následně pět souborů, deset souborů, ... Od pěti souborů se počet souborů vždy dvojnásobil.

Zátěžový test měl celkem 5 částí. Spouštěný příkaz byl stále stejný, lišil se pouze v nastavení shlukujícího modulu. Základní příkaz obsahoval požadavek pro sumarizaci jednoho souboru, přičemž při tvorbě shluků byly užity všechny slovní druhy a v sumarizačním modulu byly užity všechny možné parametry s koeficienty rovnými jedné.

První modifikovaný příkaz (P1) obsahoval požadavek na vytvoření maximálního počtu shluků se spočtením jejich mezi-shlukové podobnosti. Toho bylo docíleno nastavením parametru SimilarityTreshold na 1.

Druhý modifikovaný příkaz (P2) obsahoval ten samý požadavek jako první příkaz, tedy vytvoření maximálního počtu shluků, až na drobný rozdíl a to vynechání počítání jejich podobností.

Třetí modifikovaný příkaz (P3) obsahoval požadavek na vytvoření minimálního počtu shluků bez spočtení jejich mezi-shlukových podobností. Toho bylo docíleno nastavením parametru SimilarityTreshold na 0.000001.

Čtvrtý modifikovaný příkaz (P4) obsahoval požadavek na načtení již zpracovaných shluků z výsledků příkazů P1, či P2. Podobnost již v tomto případě nebyla počítána.

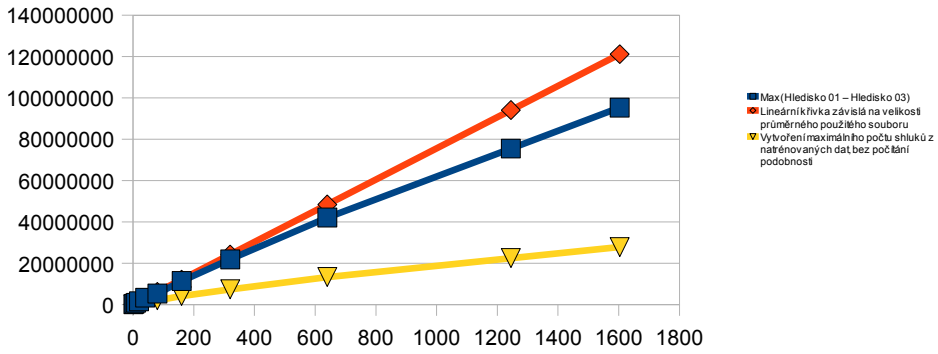
Pátý, poslední modifikovaný příkaz (P5), požadoval vytvoření maximálního počtu shluků, podobně jako první, avšak jejich maximální počet omezil na 40.

7.2 Maximální najednou alokovaná dynamická paměť

První hledisko, které jsme u příkazů zkoumali, bylo, jak moc je program náročný z hlediska paměti užívané najednou. Odhad byl, že by najednou alokovaná paměť měla stoupat lineárně s počtem souborů, které jsou zpracovávány. Přehledy s výstupy testů nalezneme v následujících grafech.

Nejvyšší zatížení oproti počtu vstupních souborů nám dává informaci o tom, kolik program potřebuje paměti RAM pro svůj chod na určitém počtu souborů.

Jak říká popisek podkapitoly, jde pouze o dynamickou paměť. K této paměti musíme tedy ještě připočítat případné struktury, které jsou alokovány staticky (v programu jsou to například řetězce). Odhadem můžeme předpokládat (jak se ukázalo například při sledování programu v běžících procesech), že statická paměť může tento výsledek maximálně zdvojnásobit.



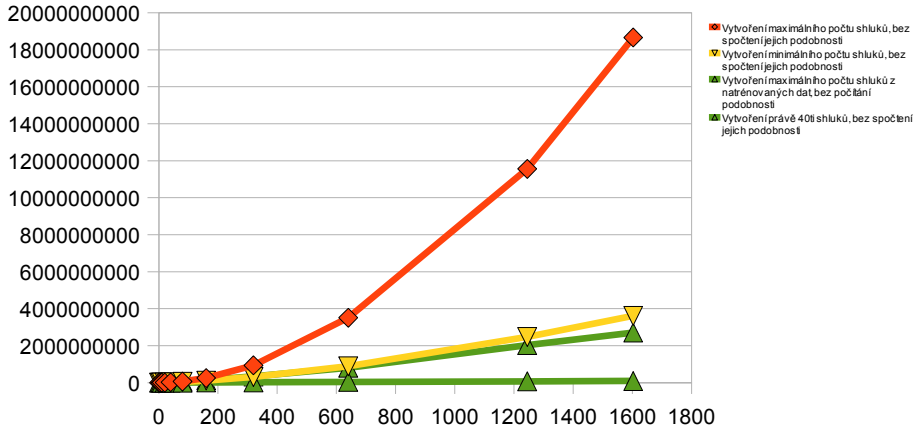
Obrázek 4: Graf znázorňuje vývoj potřebné paměti v bytech (vert. osa) závislosti na počtu vstupních souborů.

Osa X reprezentuje počet zpracovávaných dokumentů a osa Y znázorňuje množství použité paměti v bytech. Graf znázorňuje křivku lineárně závislou (červená), pod ní křivku, která byla vytvořena programem při maximálním zatížení (modrá) a nakonec křivku, která znázorňuje, kolik paměti je potřeba při načítání maximálního množství dat (žlutá).

Z grafu je patrné, že množství paměti závisí lineárně na počtu souborů (resp. na velikosti jejich obsahu). Rozdíl mezi modrou křivkou, kdy načítáme a počítáme nové shluky a žlutou křivkou, je způsoben tím, že při pouhém načítání shluků je maximální množství alokované paměti po ukončení načítání. Při počítání shluků musíme mít v paměti nejen shluky samotné, ale taktéž potřebujeme mít v paměti soubory, které se ke shlukům připodobňují, jejich aktuální centroidy a další informace, které jsou potřebné k výpočtu. Z grafu je patrné, že při načítání je potřeba zhruba třetinová paměť oproti celému počítání.

7.3 Celkově použitá dynamická paměť

Dalším hlediskem, které jsme zkoumali, byla velikost paměti, která byla celkem pro běh programu potřebná. V tomto případě je jasně vidět, že program potřebuje tím více paměti, čím více vzniká shluků (velikost nutné paměti tedy stoupá s počtem vytvořených shluků a nutností jejich porovnávání). Toto se stává až v části při počítání mezi-shlukové podobnosti. V grafu jsou znázorněny příkazy P2 – P5, příkaz P1 všechny ostatní hodnoty několikrát převyšoval.



Obrázek 5: Časová náročnost testů v závislosti na parametrech. Graf zobrazuje závislost celkově alokované paměti v bytech (vertikální osa) na počtu zpracovávaných souborů (horizontální osa) společně s různými nastaveními parametrů.

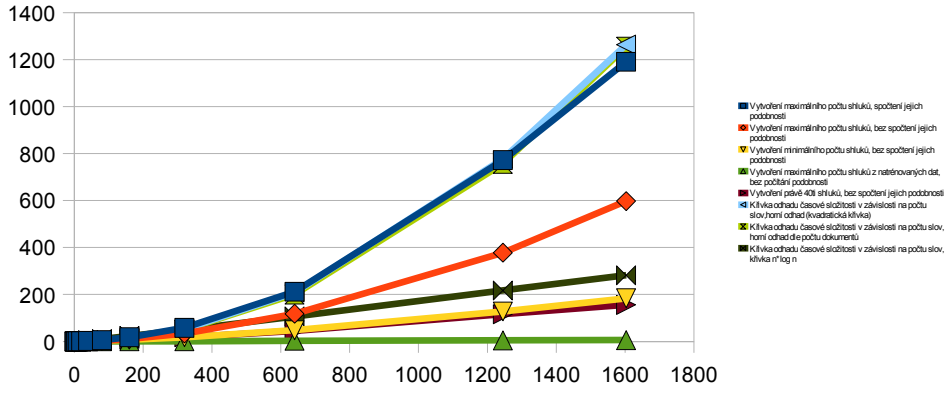
„Vítězem“ v této kategorii se jasně stal příkaz P4, který je znázorněn linkou při ose X. Je zřejmé, že samotné načítání spotřebuje pouze lineární množství paměti vzhledem ke vstupu.

Ostatní případy potřebují nejméně čas O(n*log n) a více (v závislosti na počtu vytvářených shluků a počtu slov v jednotlivých shlucích). Za zmínku stojí, že vytvoření právě čtyřiceti shluků se dokonce dostalo pod vytvoření minimálního počtu shluků. Tato anomálie je zřejmě způsobena tím, že ačkoliv více shluků nad jedním mega-shlukem ve vyhledávání a počítání podobnosti nezvítězí v rychlosti, nakonec zřejmě více shluků zvítězí v přidávání nových prvků do shluku.

Vytvoření maximálního počtu shluků bez spočtení podobnosti, tedy příkaz P2 v této kategorii v grafu celkem nepřekvapivě dominuje. Příkaz P1 jsme v grafu neuvažovali, protože svými hodnotami všechny ostatní příkazy upozadil.

7.4 Časová náročnost

Neméně důležitým hlediskem oproti předchozím je samozřejmě i časová náročnost výpočtů. V tomto ohledu byla předpokládána časová náročnost řádově O(n*log(n)), která mohla, kvůli tvoření mezi-shlukové podobnosti, dorůstat až do O(n*n). V praktických podmínkách by však nemělo shluků vznikat takové množství (jedná se o extrémní případ). Následující tabulka vypovídá vše potřebné.



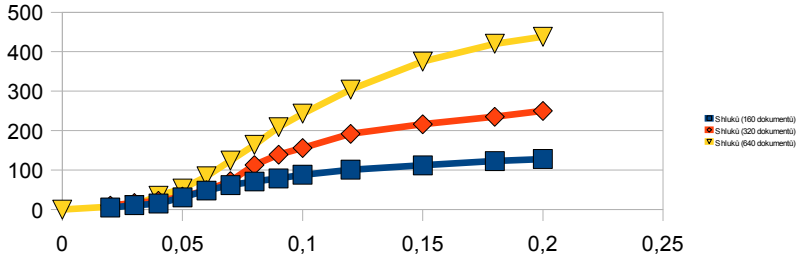
Obrázek 6: Časová náročnost testů v závislosti na parametrech. Graf zobrazuje závislost času v sekundách (vertikální osa) na počtu zpracovávaných souborů (horizontální osa) společně s různými nastaveními parametrů.

Z grafu je patrné, že pouze případy, kdy bylo dovoleno, aby vzniklo přílišné množství shluků, se skutečně blíží kvadratické časové složitosti. Všechny ostatní případy, tedy ty, kde je omezen počet shluků nějakou konstantou, zdá se, nepřerůstají křivku n*log n, která je závislá na počtu zpracovávaných slov.

7.5 Počet vzniklých shluků v závislosti na similarity treshold

Tento aspekt při celkových odhadech nelze zanedbat. Uživatelům testy přinášejí přehled o tom, jakou podobnost nastavit, aby vznikl nějaký počet shluků. Ve srovnání bylo užito tří různých množin, které jsme zpracovali – 160, 320 a 640 dokumentů.

Z výsledků je patrné, že hodnot, při kterých vychází „rozumné“ množství shluků, dosáhneme při nastavení parametru SimilarityTreshold v rozmezí [0.02, 0.1]. Více na obrázku 7.



Obrázek 7: Graf znázorňuje počet vzniklých shluků z množiny 160, 320, 640 vstupních dokumentů v závislosti na nastavení parametru SimilarityTreshold.

7.6 Zátěžové testy

Největším testem bylo pro práci reálné nasazení programu. Ukázalo se, že při nastavení parametru SimilarityTreshold na hodnotu 0.025 při tvorbě shluků z 80 tisíc souborů bylo potřeba přibližně 26 hodin práce programu na jednom ze serverů ÚFALu, celkově bylo použito přibližně 360 GB paměti s maximem při přibližně 4.3 GB najednou alokované paměti.

Taktéž se ukázalo, že při nastavení parametru SimilarityTreshold na hodnotu 0.05 při tvorbě shluků z 80 tisíc souborů bylo potřeba přibližně 41 hodin práce programu na jednom ze serverů ÚFALu, celkově bylo použito přibližně 552 GB paměti s maximem při přibližně 4.3 GB najednou alokované paměti.

Posledním zátěžovým testem bylo nastavení parametru SimilaritTreshold na hodnotu 0.1 taktéž při tvorbě shluků z 80 tisíc souborů. Na tento test bylo potřeba přibližně 56 hodin práce programu na jednom ze serverů ÚFALu, celkově bylo použito přibližně 2.25 TB paměti s maximem při přibližně 4.3 GB alokované paměti.

Tato čísla samozřejmě platí pro tvorbu shluků. Při práci s již natrénovanými daty jsou časové a paměťové nároky mnohem nižší. Pokud chceme sumarizovat řádově tisíc dokumentů, v tom případě se maximální najednou používaná paměť pohybuje řádově ve stovkách MB a čas, který potřebujeme k vykonání, v řádu minut.

7.7 Závěr testů

Testy nám pomohly odhadnout, jak se program bude chovat na obrovských datech. Při osmdesáti tisících souborech, které jsme měli k dispozici, vycházelo, že v nejhorším případě program může běžet až 20 dní. Reálné použití a samozřejmě rozumné nastavení parametrů však prokázalo, že reálně je tato hodnota daleko nižší a pohybuje se pro toto množství dat přibližně v řádu dnů. Pro běžné použití na počítačích se tedy může pracovat až s tisícovkami souborů pro úplné vyhodnocení, či s výsledky shlukové analýzy pouze pro sumarizaci. Sumarizace s výsledky již hotové shlukové analýzy by na běžném pc měly být provozuschopné až do padesáti tisíc natrénovaných souborů ve shlucích. Při tom záleží hlavně na velikosti paměti RAM, kterou počítač disponuje.

Kapitola 8

Evaluace

V této části bakalářské práce nejdříve stručně popíšeme specifické problémy evaluace sumarizačních metod. Dále se zaměříme na popis navrženého alternativního postupu evaluace sumarizace.

8.1 Úvod

Proces automatické sumarizace bez vyhodnocení a srovnání výsledků by v podstatě neměl žádný význam. V této kapitole se podíváme na to, jaké jsou možnosti evaluace, tedy možnosti vyhodnocení získaných dat. Podíváme se blíže na provedené experimenty nad získanými daty.

Celá problematika tvorby sumarizací leží v nemožnosti vytvořit ideální sumarizaci. A to hned z několika hledisek. Jak jsme nastínili v kapitole věnující se úvodu do sumarizací, různí lidé očekávají různé sumarizace, zjednodušeně co člověk, to jiná představa o tom, jak by měla ideální sumarizace vypadat. Je těžké se také shodnout na tom, jak by taková sumarizace měla být dlouhá, co by nutně měla obsahovat a co již nikoliv. Přitom metody, které jsme měli k dispozici nastudovat, počítají právě s tvorbou jakési vztažné sumarizace, tedy takové, která by byla nějakým způsobem „ideální“. Na základě informací od anotátorů tedy můžeme tvořit spíše jenom sumarizace ve smyslu „77% anotátorů se shodlo, že první věta by měla být taková...“.

Lze tedy vůbec úspěšnost sumarizací nějakým způsobem zhodnotit? Připomeňme si z úvodních kapitol, že v práci řešíme konkrétní úlohu sumarizace pomocí extraktů. Jako výstup naší metody tedy dostáváme části textu, přesněji jednotlivé věty, které již byly obsaženy ve vstupním dokumentu. Pro zhodnocení bychom potřebovali mimo automaticky generovaného extraktu také srovnávací extrakt, který však vzniká „ručně“. Tento srovnávací extrakt by mohl vzniknout například předložením dokumentu statisticky významnému počtu lidí, kteří by subjektivně (na základě pevně daných anotačních pravidel, která jsou vytvořená před začátkem anotací) rozhodli, v jakém pořadí by se v „ideálním“ extraktu měly jednotlivé věty vyskytovat. Srovnávací extrakt by následně vznikl právě stylem „77% uživatelů se shodlo na základě anotačních pravidel na tom, že první věta bude taková...“

Těžkopádnost tohoto přístupu spočívá nejen ve velice obtížném získání dostatečného množství dat pro tvorbu srovnávacího extraktu, ale taktéž v problémech při samotné tvorbě „srovnávacího extraktu“.

Příklad. Mějme dokument o dvou větách. Například „Dnes v Praze hořelo. Zemřeli 3 lidé.“ Chtějme ilustrativně vybrat pouze jednu. Dejme tento dokument dvěma anotátorům. Jeden z nich vybere informaci, že hořelo, druhý, že zemřeli lidé.

Výše popsaný postup vzniku referenčního extraktu, včetně ilustrativního příkladu, ovšem produkuje nové problémy. Například je dopředu těžké říct, které věty by se v extraktu měly objevit, či kolik by jich mělo být. Jaká by měla být kritéria pro to, aby se věta dostala do extraktu? Aby se na ní shodlo 70% anotátorů? Nebo 50? Co když se anotátoři vůbec neshodnou a jejich výběr vět bude disjunktní?

Příklad. Mějme 3 anotátory, každý vybírá z extraktu některé 2 věty. První vybírá AB, druhý CB a třetí CA. Jaké věty a jaké jejich pořadí potom v extraktu zvolíme?

Výše popsané problémy nás tedy částečně odrazují od pokusů o tvorbu srovnávacích extraktů. Nemůžeme tedy spíše sumarizace rovnou nějakým způsobem pouze srovnávat mezi sebou, aniž bychom takové srovnávací extrakty vůbec tvořili?

8.2 Evaluace v naší práci

Při volbě způsobu vyhodnocení výsledků v této práci jsme si byli vědomi problémů představených v předchozí části. Snaha vyhnout se alespoň některým problémům a zároveň předložit měřitelné výsledky vedla k následující myšlence. Idea představované evaluace je výsledkem několika setkání s Pavlem Pecinou, kterému za přispění do evaluační části tímto děkujeme.

K tvorbě srovnávacího extraktu bychom potřebovali samostatnou, ne zcela jednoduchou, statistiku od anotátorů. Naše verze evaluace nevyžaduje tvorbu takových srovnávacích extraktů, ale pouze tvorbu jistých triviálních extraktů. Tyto triviální extrakty jsou potom srovnávány s automaticky generovanými extrakty. Triviálními extrakty máme na mysli například několik úvodních vět z dokumentu. Měl-li tedy například extrakt, který jsme sumarizačním procesem získali, tři věty, vzali jsme ze vstupního dokumentu, z něhož byl extrakt vytvořen, první tři věty.

Oporu pro takto jednoduchý výběr nám dává předpoklad, že důležité věci jsou obvykle zmíněny na začátku dokumentu (od jednoduššího ke složitějšímu), v jeho několika prvních větách. Alespoň tak jsou psány novinové články. Na vybraných datech (novinových článcích) se nám tento předpoklad jeví jako opodstatněný.

Abychom zabránili situaci, kdy anotátoři raději volí uspořádané věty v porovnání automatického extraktu a triviálního extraktu (počátečních vět dokumentu), věty v triviálním extraktu jsme náhodně zpřeházeli.

Volíme tedy následující myšlenku evaluace:

Krok 1. Při evaluaci v našem případě nejdříve dojde k první anotaci. Tato anotace vypadá tak, že anotátor dostane jasná anotační pravidla (ve zkratce „vyber

lepší extrakt, aniž bys znal článek“). Na základě těchto pravidel vybere ze dvou předložených sumarizací tu, která z jeho pohledu lépe vyhovuje požadovaným pravidlům. Jedna ze sumarizací je automaticky generována naším programem, druhá sumarizace je vytvořena triviálně tak, že je vybrán určitý počet úvodních vět z původního dokumentu. Tento počet vět je závislý na počtu vět přítomných v automatické sumarizaci. Věty v triviálním extraktu jsou navíc náhodně zpřeházeny, aby tento náš přístup k tvorbě triviálních extraktů nebyl odhalen.

Krok 2. V druhém kroku se opakuje podobný scénář jako v prvním kroku, avšak anotátor má pozměněnou instrukci a k dispozici má dokument, ze kterého byla sumarizace provedena. Instrukce zní ve smyslu „na základě přečtení dokumentu zvol dle svého názoru lepší sumarizaci tohoto dokumentu“.

Z každého dokumentu tedy vzejdou dvě sumarizace – jedna bez znalosti textu a druhá s jeho znalostí.

Příklad. Máme-li anotátora, nejdříve dostane sumarizaci, kde uvidí pouze extrakty, například „Eva mele maso.“ a „Hoří.“. Z nich není sice moc patrné, co by měly extrakty představovat, avšak je zřejmé, že první může být o vaření, druhý o požáru. Řekněme, že uživatel zhodnotí informaci „Eva mele maso“ jako podstatnější. Po vybrání tohoto extraktu se zobrazí celý článek. Mohou nastat dvě situace a to buď taková, že se skutečně objeví článek o vaření a věta „Hoří.“ tam bude zmíněna coby zvolání při zapálení sporáku. Nebo naopak se může objevit článek o rozsáhlém požáru, kde si autor všímá detailu, že hasiči našli na místě požáru knížku se známými slovy „Eva mele maso“. Nebo může nastat třetí situace a to taková, že článek s ani jedním z extraktů vůbec nesouvisí. Po rozhodnutí, který extrakt je článku bližší (či ke kterému extraktu je článek méně vzdálen) se uživatel podruhé rozhodne a anotace je hotova.

8.3 Evaluace experimentů

8.3.1 Data v experimentech

Data pro experimenty byla získána ze dvou různých projektů. Šlo zejména o projekt Pražského závislostního korpusu (Hajič a kol., 2006) a z projektu CLEF (Češka, Pecina, 2007).

Data z projektu Pražského závislostního korpusu jsou ručně anotována, jejich množství je tedy vcelku omezené, celkově jde o necelé dva tisíce souborů. Zato data z projektu CLEF jsou mnohem větší, řádově desetitisíce souborů. K jejich anotaci však došlo automatickou metodou pomocí morfologického taggeru.

8.3.2 Vlastní provedení experimentu

Zcela úvodní otázkou pro jednotlivé experimenty byly parametry, které měl program zpracovat. Těm se budeme věnovat v následující podkapitolce.

První část sumarizačního procesu, provedení shlukové analýzy, je vlastně implementací poznatků z kapitoly třetí. Pro tuto část, tedy shlukování, jsme potřebovali nějaká solidně velká data, která by nám pro pozdější zkoumání poskytla dostatečný „základ“. Tímto základem byla zvolena data z projektu CLEF vzhledem ke své velikosti (na osmdesát tisíc anotovaných dokumentů). Při provádění metody shlukové analýzy však nebyla použita všechna data, ale byla z nich vyčleněna data pro pozdější sumarizační část. Napočítání těchto dat proběhlo na jednom ze serverů ÚFALu.

Nutno podotknout, že nevznikla pouze jedna verze dat, která mohla být použita ke shlukování. Celkem proces běžel „naostro“ třikrát, abychom mohli získaná data porovnat a na základě jejich rozmanitosti se rozhodnout, která data (výstupy programu s vytvořenými shluky) budou pro náš účel nejlepší.

Druhou částí sumarizačního procesu, samotná sumarizace, je implementací poznatků z kapitoly čtvrté. Zde bylo původně použito zbytku dat z projektu CLEF. Avšak při bližším zkoumání výsledků jsme zjistili, že dokumenty z tohoto projektu jsou jistým způsobem poškozené (opakující se věty, které se kvůli téměř shodným výsledkům při ohodnocení sumarizační metodou dostávaly ve více verzích do extraktů). Proto jsme museli přikročit k nahrazení těchto dat právě daty z projektu PDT. Ze srovnání jsme vyřadili dokumenty, které měly stejné věty v sumarizacích (o nichž by nešlo rozhodnout, která je z nabízených možností extraktů je lepší) a celkový počet vět v anotovaném dokumentu jsme omezili na 30 (z důvodu, abychom délkou dokumentů k sumarizaci neodradili příliš sumarizátorů).

8.3.3 Parametry pro experiment

Parametry pro sumarizační proces se volí tak trochu odhadem, je těžké určit, jakou má mít který parametr přesně váhu. Odhady jsme provedli na základě získané zkušenosti a srovnáním několika běhů programu na různých parametrech. Jako konečné jsme zvolili parametry

„-S -c 6 -f 6 -p 1 -i 1.5 -k -2.4 -r 0.4 -s 0.5 -m 5 -W -n -a -p -c -v -d“,

které označují, že budeme sumarizovat podstatná jména, přídavná jména, zájmena, číslovky, slovesa a příslovce (tedy slova, která nesou význam). Pro koeficienty jednotlivých sumarizačních metod jsme zvolili hodnotu 6 pro CentroidValue, hodnotu 6 pro FirstSentenceOverlap, hodnotu 1 pro PositionalValue, hodnotu 1.5 pro InverseWordsNumberValue, hodnotu -2.4 pro hodnotu KeyWordsValue. A nakonec pro parametry extraktu jsme zvolili kompresní poměr 0.4 a vyžadovanou podobnost 0.5 s maximálním počtem vět v extraktu rovnému pěti. Protože jsme neuvedli, že bychom sumarizaci nechtěli v případě nedostatečné podobnosti, bude se

provádět vždy (což je žádoucí). Vše jsme trénovali na již existujících shlucích, které jsme získali z množiny zhruba 80 tisíc dokumentů při nastavení parametru SimilarityTreshold na 0.025. Tímto jsme získali 181 shluků.

8.4 Závěr evaluace

V experimentu jsme podnikli kroky, které jsme popsali v naší verzi evaluace (viz kapitola 8.2). K experimentu jsme využili webovou aplikaci, která je popsána v apendixu B.

Evaluační část byla velmi závislá na vypracovaných anotací anotátory. Webová aplikace sběr anotací usnadnila. Celkem jsme do 3. srpna zpracovali na 161 anotací. Aktuální výsledky statistik jsou ke zhlédnutí na stránce http://sumarizace.bajecni.cz/statistiky.php. Ačkoliv jsme si vědomi faktu, že toto množství nedostačuje pro tvorbu větších závěrů, i tak mohou výsledky napovědět, jak si náš projekt vedl. Více představujeme v tabulce.

	Anotace 2 (po přečtení dokumentu)		
Anotace 1 (bez čtení dokumentu)	**Volba triviálního extraktu (úvodní věty z dokumentu)**	**Volba automaticky generovaného extraktu**	**Součet řádků**
Volba triviálního extraktu (úvodní věty z dokumentu)	**50**	**15**	65
Volba automaticky generovaného extraktu	**10**	**86**	96
Součet sloupců	60	101	161

Tabulka 2: Přehled výsledků anotací k 3.8.2009

Z tabulky je vidět, že mělo smysl tento test uskutečnit, protože často docházelo k volbě automatického extraktu. Ze všech dat jsou zajímavé ty hodnoty, kdy se anotátoři po přečtení článku rozhodli pro volbu odlišné odpovědi. Ačkoliv celkový počet anotací byl 161, k této jejich změně došlo pouze ve 25 případech. To nejspíš vypovídá o tom, že volbu sumarizace ve většině případů už neovlivnilo přečtení článku, což je také zajímavý závěr.

K výběru automatického extraktu se zvolenými parametry tedy docházelo zhruba v 62% případů oproti zhruba 38% případů, které získal triviální extrakt.

8.5 Připomínky od uživatelů

Při vyhodnocování jsme byli několikrát kontaktováni anotátory s připomínkami zejména k poskytnutým datům.

Extrakty neodpovídají dokumentu. Tuto připomínku jsem slyšel nejméně ve třech případech. Některé extrakty totiž ani ve své triviální formě, ani v naší sumarizované formě, téměř neodpovídaly dokumentům, z nichž pocházely. Původně jsme uvažovali o zařazení odpovědi „nevím“, ale obávali jsme se, že bude příliš často využívána hned v několika těžko rozlišitelných případech. Uživatelé v případech špatných extraktů volili ten „méně špatný“, takže výsledek je v souladu s testováním.

Některé extrakty nemají smysl. Jak jsme zmínili, data pro náš test byla získána automatickou metodou z další spousty automaticky získaných dat. Některé dokumenty měly jako samostatné věty uvedeny počáteční „místo vzniku zprávy“, tedy například „Praha -“ apod. Toto nešlo eliminovat, navíc bychom tím měnili charakter dat. Tudíž jsme v tomto případě ponechali původní dokumenty tak, jak byly, a nijak je needitovali (protože to samotné by vydalo na spoustu práce).

Kapitola 9

Závěr

Bakalářská práce měla za úkol vytvořit sumarizační systém, který by pokrýval potřeby automatické sumarizace na základě metody MEAD a metody shlukové analýzy CIDR. Metody jsme podrobně rozebrali (viz kapitola 3, 4) a přidali jsme i jejich vlastní implementaci s podrobným popisem (viz kapitola 5, formát výstupů v apendixu A). Program pro sumarizaci byl napsán bez závislosti na operačním systému, jde tedy o OS independent system.

Oproti původním záměrům byla navíc práce rozšířena o vizualizační applet v programovacím jazyce Java, který usnadňuje reprezentaci výstupů programu. Tento applet umožňuje prohlížení výstupů z programu tak, aby byly tyto výstupy snadněji čitelné.

Taktéž byla vytvořena webová aplikace pro hodnocení sumarizací kombinací jazyků XHTML, PHP a SQL. Podrobnosti o programu samotném i o dalších vytvořených nástrojích jsou k práci také přiloženy (viz kapitola 6 a apendix B).

Za úspěch považujeme spuštění projektu na jednom ze serverů Ústavu formální a aplikované lingvistiky, kde program úspěšně vydal výsledné shluky, díky nimž jsme následně mohli provést evaluaci. Tehdy byla otestována spolehlivost programu, jelikož při složitých výpočtech bylo postupně použito až 2.25 TB paměti. Z tohoto hlediska šlo o implementačně náročnou problematiku. Podrobné analýzy běhu programu byly taktéž zpracovány (viz kapitola 7).

Práce představila odlišný druh evaluace. Tento typ evaluace je založen na porovnávání automatických extraktů a extraktů získaných triviální metodou ze vstupních dokumentů (viz kapitola 8).

Výsledná data, zejména vytvořené shluky, mohou sloužit pro další pokusy nejen se sumarizací textu. Taktéž získané informace od sumarizátorů, kteří byli ochotni věnovat svůj čas vědě, přinášejí další možnosti, jak práci využít, či srovnat s podobnými. Na práci je taktéž možné navázat hned z několika hledisek. Je možné práci využít k vlastnímu zkoumání chování sumarizací v závislosti na různých parametrech, nebo je možné do programu připojit podporu dalších vstupních formátů.

Přínos práce z hlediska osobního rozvoje vidí autor zejména v odzkoušení si moderních přístupů k sumarizaci, vyzkoušení si implementace jedné z metod sumarizace, potýkání se s problémy při programování, odzkoušení si nasazení práce v náročném provozu na velkých reálných datech a konečně v závěrečné prezentaci práce v textové podobě.

Literatura

BORKO, Harold; BERNIER, Charles L. *Abstracting Concepts and Methods*. San Diego, California : Academic Press, 1975. 250 s. ISBN 978-0121186500.

ČEŠKA Pavel; PECINA, Pavel. *Charles University at CLEF 2007 Ad-Hoc Track*. March 31, 2009 [online], Institute of Formal and Applied Linguistics. Charles University, 2007 [cit. 2009-07-31]. Dostupný z WWW: <http://www.clef-campaign.org/2007/working_notes/ceska_adhoc_CLEF2007.pdf >.

HAJIČ, Jan et. al. *Průvodce PDT 2.0 : Kapitola 3: Data* [online]. 2006 [cit. 2009-07-14]. Dostupný z WWW: <http://ufal.mff.cuni.cz/pdt2.0/doc/pdt-guide/cz/html/ch03.html#a-data-formats-pml>.

R Foundation. *The R Project for Statistical Computing* [online]. 2009 [cit. 2009-07-26]. Dostupný z WWW: <www.r-project.org>.

MANI, Inderjeet. *Automatic Summarization*. (MITRE Corporation and Georgetown University) Amsterdam: John Benjamins (Natural language processing series, edited by Ruslan Mitkov, volume 3), 2001. 285 s. ISBN 1-58811-060-5.

MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. *An introduction to information retrieval*. Cambridge [etc.] : Cambridge University Press, 2008. 544 s. Dostupný z WWW: <http://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf>. ISBN 978-0-521-86571-5.

RADEV, Dragomir R.; HATZIVASSILOGLOU, Vasileios; MCKEOWN, Kathleen R. A *Description of the CIDR System as Used for TDT-2*. March 31, 2009 [online], Department of Computer Science, Columbia University, 1999 [cit. 2009-07-13]. Dostupný z WWW: <http://tangra.si.umich.edu/~radev/papers/tdt99.pdf>.

RADEV, Dragomir R., et al. Centroid-based summarization of multiple documents. In CRESTANI, Fabio (ed.). *Information Processing and Management*, ELSEVIER, 2004, s. 919-938. . Dostupný z WWW: <http://tangra.si.umich.edu/~radev/papers/centroid.pdf>.

Apendix A

Vstupy a výstupy

Následující kapitola nás seznámí s tím, kam vkládat vstupní soubory, a kde čekat výstupy. Také nám poví více o jednotlivých složkách, které se v projektu nacházejí, jejich účel a použití.

A.1 Úvod

Výstupy každého běhu programu se ukládají pod unikátním jménem, aby se nestalo, že se budou jednotlivé běhy programů přepisovat. Toto unikátní jméno je zajištěno tzv. prefixem, který uchovává informaci o chvíli, kdy byl program spuštěn. Tento prefix je ve tvaru „rrrr.mm.dd.hh.mm.ss.“, kde „r“ znamená číslici roku, „m“ znamená číslici měsíce, atd. Při dalším povídání budeme občas informaci o prefixu zanedbávat, protože mimo svého účelu, aby bylo poznat, které soubory k sobě patří, a také mimo toho, aby se výstupy stále nepřepisovaly, není její uvádění nutné.

Vstupní soubory, které projekt dostává, se obvykle nacházejí ve složkách „input“, „toSummarize“, případně „output“, jde-li o data již vytvořená v předchozích bězích programu (například soubor s natrénovanými shluky clusters.txt).

Jakékoliv výstupní soubory generované projektem, jsou ukládány do složky „output“ v hlavní složce projektu.

A.2 Vstupní a výstupní soubory shlukové analýzy

Shluková analýza je asi nejdůležitějším procesem, vzhledem ke vznikajícím shlukům, v celém procesu sumarizace. Právě z ní se dozvídáme nejen ohodnocení jednotlivých slov, ale také jejich uplatnění v rámci různých shluků. Následující dvě podkapitolky jsou věnovány vstupům a výstupům shlukové analýzy.

A.2.1 Vstupní soubory shlukové analýzy

Vstupní soubory práce jsou zejména soubory typu PML, které definovali Hajič a kol. (2006) v projektu Pražského závislostního korpusu.

Pro shlukovou analýzu je v základní instalaci určen adresář „input“ v hlavní složce projektu. Do tohoto adresáře smějí být vkládány hlavně soubory typu PML se základní příponou „m“, která značí, že jde o soubory anotované na tzv. morfologické rovině. Tuto morfologickou rovinu si můžeme představovat jako určení slovních druhů a základních tvarů všech slov v projektu.

Dalšími vstupními daty pro shlukovou analýzu, které byly pro projekt implementovány, mohou být soubory typu „p“. Ty byly pro projekt věnovány z projektu CLEF 2007. Tato data byla vytvořena automatickou metodou z dokumentů z Internetu. Pro předpřipravení těchto dat však bylo nutné je rozdělit na jednotlivé části, protože celá data byla věnována vcelku (podobně jako slevy lákající v obchodech, toto bylo přes 80 tisíc v jednom). Soubory tohoto formátu mohou být vkládány do adresáře „input“ s příponou p.

Pro zahrnutí dalších vstupních souborů existuje ještě jedna možnost a to přímé nakopírování složek se soubory určenými ke shlukování do hlavní složky projektu a jejich uvedení v příkazu pro zpracovávané adresáře při načítání. Pokud bychom vkládali adresář „data001“ z hlavní složky, do příkazu při spuštění by pak patřil následující úsek spouštěcího příkazu : „-L -i data001“. V případě, že chceme přidat více vstupních složek, uvedeme za parametr i všechny tyto složky. Je však nutné celý seznam adresářů uzavřít do uvozovek a oddělit každé dva adresáře mezerou, tedy např. „data001 data002“. Pozor, názvy jednotlivých adresářů nesmí obsahovat mezery!

A.2.2 Výstupní soubory shlukové analýzy

Výstupní soubory shlukové analýzy jsou vždy uloženy ve složce „output“ v hlavní složce programu. Příklady se vztahují k jednomu z trénovacích souborů „ln94201_1.m“.

- ***prefix*.clusters.txt**

Tento soubor uchovává klíčové informace o vytvořených shlucích při běhu programu.

Jednotlivé shluky jsou zde uloženy v jednoduchém textovém formátu. Hlavička obsahuje pouze 2 řádky. První řádka obsahuje informace o počtu vytvořených shluků, druhý řádek je pouze informativní. Následuje oddělovací řádek, který označuje, že za ním přichází přehled jednoho ze spočtených shluků.

Samotný shluk má formát takový, že první řádek informací o shluku obsahuje pořadové číslo shluku (číslováno od jedné), druhý obsahuje informaci o počtu dokumentů uložených ve shluku. Další řádky následně obsahují informace o počtu slov v jednotlivých souborech. Za nimi následuje oddělovač (mezera), který je známkou toho, že přijde celkový přehled všech slov ohodnocených ve shluku. Tento soubor slov je seřazen podle jejich tf-idf a nakonec je opět oddělovač (mezera). Pod ním může následuje zbytek shluků ve shodném formátu.

Ukázka ze souboru prefix.clusters.txt

```
clustermodule> IN TOTAL Created 181 clusters.
clustermodule> Cluster review

clustermodule> Cluster no. 1>>
Cluster contains 18301 documents.
Document 01/LN-20020102001.p is composed of 315 words.
Document 01/LN-20020102002.p is composed of 191 words.
Document 01/LN-20020102003.p is composed of 169 words.
…
Document 16/MF-20021220228.p is composed of 230 words.
Document 16/MF-20021220328.p is composed of 658 words.

clustermodule> Keywords of cluster no. 1
0.00654905      V       být
0.00569638      N       firma
0.00544262      N       rok
0.0048743 N     koruna
0.00470766      N       společnost_^(*3ý)
0.00457088      N       procento
0.00451708      N       milión`1000000
…
```

- ***prefix*.clustersimilarity.txt**

Soubor, jehož vytvoření je závislé na přítomnosti parametru pro měření mezi-shlukových podobností, obsahuje v prvním řádku informace o souboru, jehož se mezi-shlukové podobnosti týkají. Podrobnosti o tehdy spouštěném příkazu jsou uloženy v souboru log.txt s odpovídající příponou. Soubor clustersimilarity.txt je lehce čitelný například pro zpracování v některém nástroji pro matematiku, namátkou pro statistický program R. Ten, kdo jej však zpracovává, si musí dávat pozor na to, že informace o podobnostech jsou v horním „pravém“ trojúhelníku celé matice, takže před testy si ji musí „odzrcadlit“.

Ukázka ze souboru *prefix*.clustersimilarity.txt

```
output/2009.07.13.13.08.53.clusters.txt
1       0.326143  0.362833  0.602703  0.480524  0.515043  …
0       1         0.141726  0.272507  0.166157  0.154275
0       0         1         0.269048  0.164425  0.19439
0       0         0         1         0.325381  0.332402
…
```

- ***prefix*.clustersmodule.txt**

Tento soubor ukazuje průběh, jak vznikaly všechny shluky. Soubor je spíš přehledem, než souborem, ze kterého se dá čerpat, proto je generován tak, aby byl čitelný i pro běžné uživatele. Velký pozor však vždy dávejte na jeho velikost. Vzhledem k době, po jakou občas program běží, se může stát, že bude tento soubor narůstat do celkem gigantických rozměrů. Pro další použití není soubor potřeba, takže pokud jej nevyžadujete, můžete soubory s tímto jménem mazat.

Po první řádce, která je informativní, následuje prázdný řádek, který uvozuje vkládání souboru do shluků. Jeho jméno je uvedeno v následujícím řádku. Následné řádky ukazují podobnost tohoto dokumentu ke každému již vytvořenému shluku (jejich počet obvykle po celou dobu procesu stoupá) a poslední řádek označuje nejlepší nalezenou podobnost a informace o tomto shluku (pořadové číslo).

Ukázka ze souboru *prefix*.clustersmodule.txt

```
clustermodule> Cluster similarities. Document similarities to clusters like this (in ascending order).


Inserting a document '01/LN-20020102001.p' into clusters

clustermodule> Because of low similarity, which was 0 is created a new cluster, which contains only file 01/LN-20020102001.p


Inserting a document '01/LN-20020102002.p' into clusters

0.203077

clustermodule> The best similarity of 01/LN-20020102002.p is to cluster no. 1. It equals 0.203077.


Inserting a document '01/LN-20020102003.p' into clusters

0.232426

clustermodule> The best similarity of 01/LN-20020102003.p is to cluster no. 1. It equals 0.232426.


Inserting a document '01/LN-20020102004.p' into clusters

0.0373793

clustermodule> The best similarity of 01/LN-20020102004.p is to cluster no. 1. It equals 0.0373793.


Inserting a document '01/LN-20020102005.p' into clusters

0.0109043

clustermodule> Because of low similarity, which was 0.0109043 is created a new cluster, which contains only file 01/LN-20020102005.p


Inserting a document '01/LN-20020102006.p' into clusters

0.0240552

0

clustermodule> Because of low similarity, which was 0.0240552 is created a new cluster, which contains only file 01/LN-20020102006.p


Inserting a document '01/LN-20020102007.p' into clusters
```

```
0.441934
0.024517
0.0133321
clustermodule> The best similarity of 01/LN-20020102007.p is to cluster no. 1. It equals 0.441934.

Inserting a document '01/LN-20020102008.p' into clusters
0.507013
0.00524911
0.0219999
clustermodule> The best similarity of 01/LN-20020102008.p is to cluster no. 1. It equals 0.507013.

Inserting a document '01/LN-20020102009.p' into clusters
0.295035
0.00482977
0.00328727
clustermodule> The best similarity of 01/LN-20020102009.p is to cluster no. 1. It equals 0.295035.
…
```

- ***prefix*.trained.at, *prefix*.trained.idf, *prefix*.trained.tf**

Tyto soubory uchovávají napočítané hodnoty inverzních četností výrazů, četnosti výrazů a informací o tom, v kolika souborech se jednotlivé výrazy vyskytují. Jednotlivé soubory mají stejný formát, jsou seřazeny podle abecedy, obsahují tytéž výrazy (tudíž všechny tyto tři soubory jsou až na uvedená čísla stejné).

Ukázka souboru *prefix*.trained.idf

```
Idf         Species_of_speech    lemma
11.29       N           &mikro;s-2`mikrosekunda
3.03987     N           0
3.22848     C           00
…
11.29       C           A820
8.80506     C           A9
9.68053     N           AA-1_:B_;K_,t_^(American_Airlines)
10.5968     N           AA-2_:B_;K_,t_^(Athens_News_Agency)
11.29       N           AA-8_:B_^(autorský_arch)
7.48331     N           AAA-1_:B_;K_,t_^(Amateur_Athletic_Association)
10.5968     N           AAA-3_:B_,t_^(anti-aircraft_artillery)
10.1914     N           AAC-2_:B_;K_,t_^(Association_of_American_Colleges)
11.29       N           AAP
11.29       N           AAaB
```

```
11.29    N    ABADDÓN
11.29    N    ABASOVA
9.21053  N    ABB_:B_;K_,t_^(Asea_Brown_Boveri,_evr._společnost)
6.71526  N    ABC-1_:B_;R_^(časopis)
10.5968  N    ABC-1_;R_,h_^(časopis_ABC)
10.5968  N    ABF
11.29    N    ABFFE
9.68053  N    ABM_:B_;K
8.45676  N    ABN
11.29    A    ABS-1_:B_;g_^(Antiblokovací_systém_brzd)
...
```

A.3 Vstupní a výstupní soubory sumarizačního modulu

Sumarizační modul je z hlediska sumarizace posledním článkem v soustavě úprav, které vedou k získání extraktu z dokumentu. V tomto modulu vznikají zejména soubory, které v sobě uchovávají informace o větách. Soubory uchovávají nejen váhy vět, jaké mají v sumarizacích, ale i jejich znění včetně jejich základních tvarů. Více v následujících podkapitolkách.

A.3.1 Vstupní soubory sumarizačního modulu

Vstupními soubory sumarizačního modulu jsou některé výstupní soubory ze shlukové analýzy. Mezi tyto soubory, které jsou pro proces sumarizace potřebné, patří soubory clusters.txt a trained.idf.

První zmiňovaný soubor, clusters.txt, nám zajistí, že budou vytvořeny potřebné shluky a v nich ohodnocena právě ta slova, která jsou v jednotlivých shlucích význačná.

Druhý soubor, trained.idf, nám napomůže vytvořit ze sumarizovaného dokumentu příslušný vektor inverzních četností výrazů, kterým otestujeme podobnost s jednotlivými shluky.

V reálném užití však nad tím, které soubory jsou nutné a které nikoliv, nemusíme vůbec přemýšlet. Proces sumarizace sám z příkazové řádky pozná, zda soubory vytváříme, či načítáme, proto, pokud máte všechny výstupní soubory s příslušného běhu (mezi nimi hlavně tyto dva), můžete si být jisti, že sumarizace správně proběhne.

A.3.2 Výstupní soubory sumarizačního modulu

Sumarizační modul generuje soubory, které uchovávají extrakt, případně informace, které byly potřebné k jeho získání.

- ***prefix*.summarized.*původní jméno souboru*.sum**

Soubor uchovává výsledný extrakt sumarizačního procesu, jde tedy o klíčový výstupní soubor celého programu. Věty, které do extraktu náleží, jsou uchovány společně s informacemi o tom, kolik procent „smyslu“ z původního textu dle nastavených metod obsahují a také o tom, kolik procent z celkového počtu vět je již v extraktu přítomno. Věty jsou ukládány na každém čtvrtém řádku.

Ukázka souboru *prefix*.summarized.*původní jméno souboru*.sum

Konec obchodníků s bílým masem

compression 14.2857% of sentences, 4.71698% in words.

total compression 14.2857% of sentences, 4.71698% in words.

sence 72.3312% of text.

total sence 72.3312% of text.

Bonn - Sedm Vietnamců , kteří obchodovali s dívkami z České republiky , zadržela včera německá policie ve spolkové zemi Porýní - Falcku .

compression 14.2857% of sentences, 22.6415% in words.

total compression 28.5714% of sentences, 27.3585% in words.

sence 11.6804% of text.

total sence 84.0116% of text.

Podle mluvčího policie tak převedli asi 200 lidí a od každého za službu inkasovali 6000 dolarů .

compression 14.2857% of sentences, 16.0377% in words.

total compression 42.8571% of sentences, 43.3962% in words.

sence 10.7309% of text.

total sence 94.7424% of text.

- ***prefix*.summarized.*původní jméno souboru***

Soubor má za úkol uchovat jak původní pořadí vět ve vstupním textu, tak i pořadí vět ve výsledném extraktu. Jde tedy o soubor, který zobrazuje obě pořadí. Mimo to uchovává i výslednou hodnotu „důležitosti“, kterou věta podle zvolených parametrů při ohodnocování v sumarizačním modulu získala. Rozepsání této „důležitosti“ je v souboru *prefix.*weights.*původní jméno souboru*. Na každém řádku se tedy tabulátorem nachází oddělené pořadové číslo věty číslované od nuly, dále její „důležitost“ a nakonec celé znění, které je na dalším řádku doplněno i o jednotlivé základní tvary slov z věty. Pořadí vět v extraktu následuje podobně po oddělovacím prázdném řádku.

Ukázka souboru *prefix*.summarized.*původní jméno souboru*

summarizationmodule> Summarized document toSummarize/ln94201_1.m has 106 original words / 7 sentences.

summarizationmodule> Original sentences order>>

0 0.745479 Konec obchodníků s bílým masem

konec obchodník s-1 bílý maso_^(jídlo_apod.)

1 0.120384 Bonn - Sedm Vietnamců , kteří obchodovali s dívkami z České republiky , zadržela včera německá policie ve spolkové zemi Porýní - Falcku .

Bonn_;G - sedm`7 Vietnamec_;E , který obchodovat_:T s-1 dívka z-1 český republika , zadržet včera německý policie v-1 spolkový země Porýní_;G - Falcko_;G .

2 0.0300018 Na tři hlavní podezřelé byla uvalena vyšetřovací vazba .

na-1 tři`3 hlavní podezřelý být uvalit_:W vyšetřovací_^(*2t) vazba-1_^(obviněného) .

3 0 Zadržení Vietnamci ve věku mezi 22 a 34 roky lákali do zmíněné německé oblasti ženy a nezletilé dívky z České republiky .

zadržený_^(*2t) Vietnamec_;E v-1 věk mezi-1 22 a-1 34 rok lákat_:T do-1 zmíněný_^(*3it) německý oblast žena a-1 zletilý dívka z-1 český republika .

4 0.0241851 Zde je pak nutili k prostituci .

zde on-1_^(oni/ono) pak nutit_:T k-1 prostituce .

5 0 Kromě toho nelegálně převáděli do Německa jiné Vietnamce a Číňany a některé z nich potom pašovali dál do jiných západních zemí .

kromě ten legálně_^(*1í) převádět_:T do-1 Německo_;G jiný Vietnamec_;E a-1 Číňan_;E a-1 některý z-1 on-1 potom pašovat_:T daleko-1_^(dojít_dále_než_...) do-1 jiný západní země .

6 0.110597 Podle mluvčího policie tak převedli asi 200 lidí a od každého za službu inkasovali 6000 dolarů .

podle-2 mluvčí policie tak-3 převést asi 200 člověk a-1 od-1 každý za-1 služba inkasovat_:T_:W 6000 dolar_;b .

summarizationmodule> Sense sentence order>>

0 0.723312 Konec obchodníků s bílým masem

konec obchodník s-1 bílý maso_^(jídlo_apod.)

1 0.116804 Bonn - Sedm Vietnamců , kteří obchodovali s dívkami z České republiky , zadržela včera německá policie ve spolkové zemi Porýní - Falcku .

Bonn_;G - sedm`7 Vietnamec_;E , který obchodovat_:T s-1 dívka z-1 český republika , zadržet včera německý policie v-1 spolkový země Porýní_;G - Falcko_;G .

6 0.107309 Podle mluvčího policie tak převedli asi 200 lidí a od každého za službu inkasovali 6000 dolarů .

podle-2 mluvčí policie tak-3 převést asi 200 člověk a-1 od-1 každý za-1 služba inkasovat_:T_:W 6000 dolar_;b .

2 0.0291097 Na tři hlavní podezřelé byla uvalena vyšetřovací vazba .

na-1 tři`3 hlavní podezřelý být uvalit_:W vyšetřovací_^(*2t) vazba-1_^(obviněného) .

4 0.0234659 Zde je pak nutili k prostituci .

zde on-1_^(oni/ono) pak nutit_:T k-1 prostituce .

5 0 Kromě toho nelegálně převáděli do Německa jiné Vietnamce a Číňany a některé z nich potom pašovali dál do jiných západních zemí .

kromě ten legálně_^(*1í) převádět_:T do-1 Německo_;G jiný Vietnamec_;E a-1 Číňan_;E a-1 některý z-1 on-1 potom pašovat_:T daleko-1_^(dojít_dále_než_...) do-1 jiný západní země .

3 0 Zadržení Vietnamci ve věku mezi 22 a 34 roky lákali do zmíněné německé oblasti ženy a nezletilé dívky z České republiky .

zadržený_^(*2t) Vietnamec_;E v-1 věk mezi-1 22 a-1 34 rok lákat_:T do-1 zmíněný_^(*3it) německý oblast žena a-1 zletilý dívka z-1 český republika .

- ***prefix*.summarizationmodule.*původní jméno souboru***

Soubor uchovává informace, které vedly ke vzniku extraktu společně s některými vlastnostmi textu. Uchovává podobnost souboru ke každému ze shluků, dále uchovává podobnost souboru ke každému ze shluků v závislosti na jednotlivých slovních druzích a v neposlední řadě v sobě ukrývá celý výpis z použitého shluku.

Ukázka souboru *prefix*.summarizationmodule.*původní jméno souboru*

2009.07.13.13.08.53.clusters.txt

1.29194e-08

1.23441e-07

4.47014e-08

2.65745e-08

7.20133e-08

…

0.000247948

0

0

0

0

0

clustermodule> Document has been summarized by a cluster no. 157 because of similarity 0.0291866

0.00265876 similarity of nouns, 1 words included.

0.0226904 similarity of adjectives, 4 words included.

0 similarity of pronouns, 0 words included.

0.00227947 similarity of counts, 3 words included.

3.754e-05 similarity of verbs, 1 words included.

0.00152044 similarity of adverbs, 3 words included.

0 similarity of prepositions, 0 words included.

0 similarity of junctions, 0 words included.

0 similarity of participles, 0 words included.

0 similarity of interjections, 0 words included.

clustermodule> Keywords of document

0.00472149	C	200
0.103149 A	bílý	
0.00322133	V	být
0.00449934	A	jiný
0.00694506	A	každý
0.00672845	C	sedm`7
0.000277882	D	tak-3

0.00106066	C	tři`3
0.0131262	D	včera
0.000646218	D	zde
0.000426715	A	český
0.0354465	N	člověk

- ***prefix*.weights.*původní jméno souboru***

Soubor uchovává váhy, které v součtu dávají výslednou váhu věty oproti nastaveným parametrům. Váhy jsou uchovány v tabulce společně s nadpisy, které je jasně identifikují. Pro každou větu je určen jeden sloupec, přičemž v prvním řádku jsou uvedeny popisky. V prvním sloupci je koeficient, který byl užitý pro metodu, jejíž název je ve druhém sloupci. Poté následují váhy této metody pro všechny věty.

Ukázka souboru *prefix*.weights.*původní jméno souboru*

uzityKoeficient	nazevMetody		V0	V1	V2	V3	V4	V5	V6
6	CentroidValue	3.42544	0.673518	0.142199	0.0141706	0.0214601	0.149417	1.57379	
1	PositionalValue	0.25	0.214286	0.178571	0.142857	0.107143	0.0714286	0.0357143	
6	FirstSentenceOverlap	5	1	0	0	0	0	0	
1.5	InverseWordsNumberValue		0.464851	0.096844	0.258251	0.105648	0.332037	0.105648	0.136721
-2.4	KeyWordsValue	-0.12	-0.528	-0.216	-0.504	-0.168	-0.456	-0.408	

A.4 Ostatní generované soubory

Ostatní soubory, které jsou při procesu generovány, souvisejí s přehledem celého procesu. Jde o soubor *prefix*.log.txt, který zobrazuje přehled, jak proces pracoval, *prefix*.loadingmodule.txt, který ukazuje, jak probíhalo načítání a co bylo nebo nebylo načteno. Soubory .lastSummarization.txt a .lastClusters.txt pouze uchovávají informace o posledních sumarizovaných souborech a posledních vytvořených shlucích (ukládáme pouze jejich prefix).

Apendix B

Další vytvořené nástroje

B.1 Nástroj pro snadnější přehled výsledků programu

Program, který zastřešuje celou sumarizaci, generuje mnoho textových souborů, ve kterých není zcela jednoduché se orientovat. Proto jsme k programu navíc vytvořili malý applet, který výsledky programu reprezentuje.

B.1.1 Instalace a spuštění

Pro běh programu je pouze nutné mít na počítači nainstalovaný interpret pro program Java (program, který umožní spuštění tohoto appletu). Většinou je tento program již nainstalovaný, pokud by však nešel program následujícími instrukcemi spustit, můžete si tento interpret zdarma stáhnout z adresy http://www.java.com/en/download/index.jsp .

Spuštění programu se provádí následujícím příkazem. Pro tento účel byl v hlavní složce projektu vytvořen soubor „applet.bat“, který je na operačních systémech windows spustitelný a na operačních systémech typu unix můžete applet spustit příkazem sh ./applet.sh. Soubor applet.bat obsahuje jediný pokyn pro spuštění a to příkazem

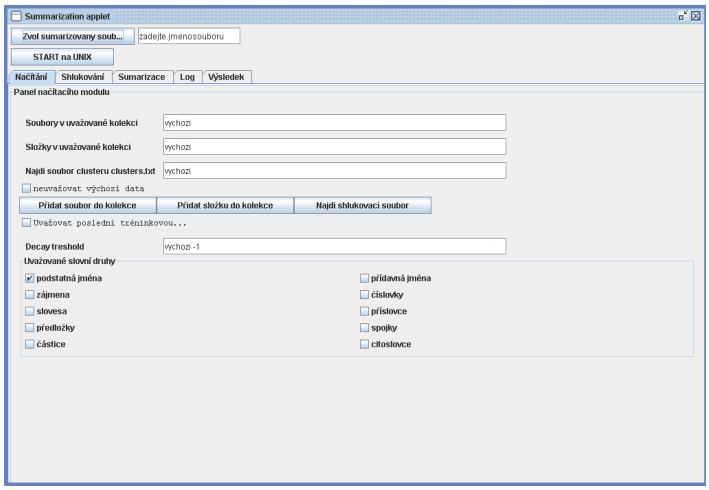
```
java  -Xms256m -Xmx512m -XX:PermSize=64M -XX:MaxPermSize=1000M -jar
TextSummarizationApplet.jar c:\textsum\ windows
```

Předposlední parametr („c:\textsum\“) je cestou do hlavní složky projektu na OS windows, na OS linux by tento adresář měl odpovídat umístění (po spuštění makefile se tento soubor modifikuje). Cesta musí na obou OS končit příslušným lomítkem. Poslední parametr je potom typem operačního systému, na kterém je program spuštěn. Je možné volit mezi „windows“ a „unix“. Pokud program nejde spustit, je možné, že je nutné některý z posledních dvou parametrů změnit tak, aby odpovídal realitě.

B.1.2 Hlavní nabídka

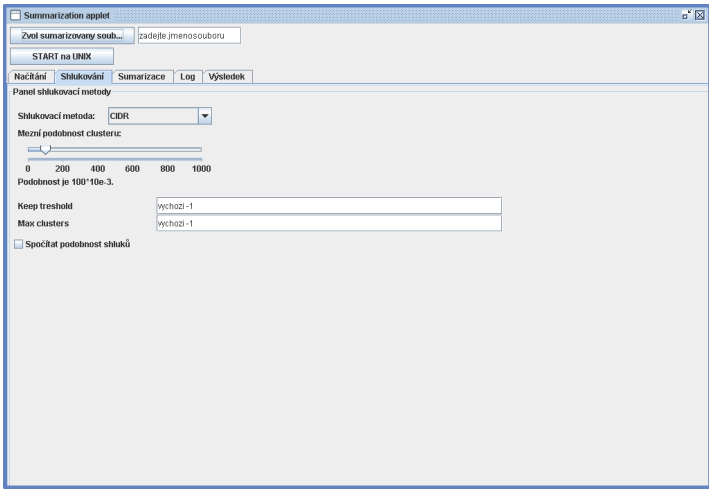
Hlavní nabídka se skládá z pěti panelů, které nám pomáhají zobrazovat výsledky. Pro výsledky běhů programu je určeno okno s výsledky.

První okno nám nabízí možnosti, jak nastavit načítací modul.



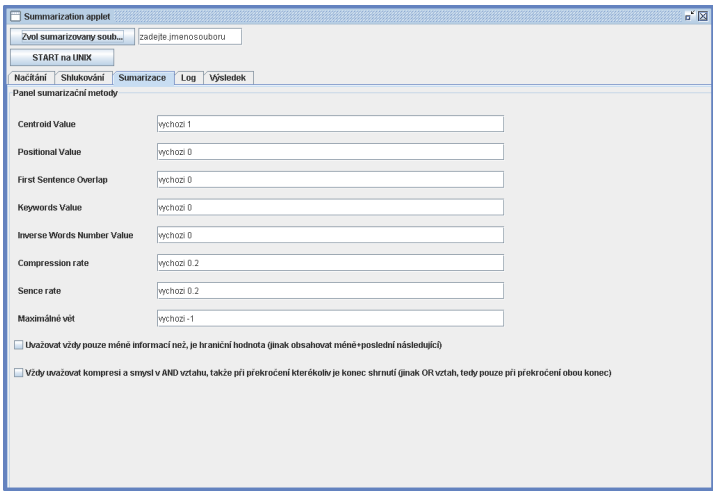
Obrázek 8: Načítací modul sumarizačního appletu

Další okno nám umožňuje nastavovat parametry shlukové metody.



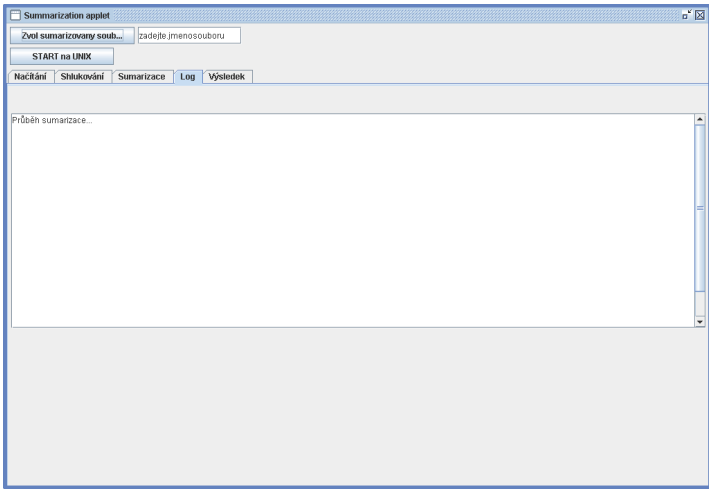
Obrázek 9: Shlukující modul sumarizačního appletu

Následující okno nám umožňuje nastavovat parametry sumarizace.



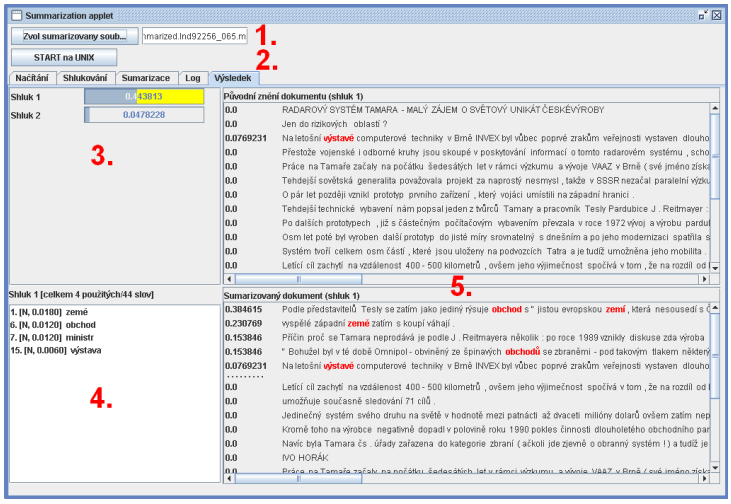
Obrázek 10: Sumarizační modul sumarizačního appletu

Dalším oknem prochází log soubor.



Obrázek 11: Okno s výstupem programu v sumarizačním appletu

Poslední z oken je nejdůležitější, protože zobrazuje přehled výstupních souborů. Toto okno totiž zobrazuje výsledky sumarizovaných souborů. Tyto soubory můžeme načítat pomocí tlačítka „Zvol sumarizovaný soubor“. Tímto tlačítkem můžeme pouze volit soubory s příponou „m“ či „p“, ale pouze ty, které vznikly po sumarizačním procesu a obsahují jak původní pořadí vět, tak pořadí sesumarizované. Tzn. takové soubory, které byly vygenerovány procesem sumarizace, obsahují „summarized“ a jsou zakončeny příponou „p“ či „m“.



Obrázek 12: Výsledkové okno sumarizačního appletu

Některé klíčové prvky výsledkového okna jsou označeny čísly a znamenají následující:

- Načítání sumarizovaného souboru
- Tlačítko START spustí proces sumarizace dle zadaných parametrů v oknech „Načítání“, „Shlukování“ a „Sumarizace“.
- Zde program zobrazuje načtené shluky. Po najetí kurzoru nad každý shluk se ukazují slova, která má dokument a shluk podobná. Žlutě zvýrazněný shluk je ten, podle kterého byl dokument sumarizován. Při kliknutí na jiný shluk navíc oba dokumenty v pravé části okna samy zvýrazní pouze společná slova s nově vybraným shlukem.
- Zde jsou zobrazena slova ze shluku. Pokud před slovem nejsou pomlčky, jde o slovo použité zároveň ve shluku i v dokumentu. Přepínání mezi „slovy pouze společnými“ a „všemi slovy ve shluku“ se provádí kliknutím na „nadpis“ nad tímto seznamem slov.
- Původní a sumarizovaný dokument. Původní dokument je v horní části okna, sumarizovaný je ve spodní části. Slova, která má dokument a shluk stejná, jsou červeně zvýrazněna. Tato podobná slova jsou závislá na shluku, který je aktivní v levé části okna. Shluky se dají měnit klikáním na různé z nich. Pokud máme zvolen nejpodobnější shluk, ukazuje se nám u vět navíc jejich „důležitost“ v sumarizaci. Tečky ve spodní části mezi větami označují, kde podle programu končí náš extrakt.

B.2 Nástroj pro získání evaluací od uživatelů

Pro získávání hodnocení extraktů od uživatelů byla pro snadnou použitelnost vytvořena webová aplikace. Ta měla za úkol umožnit anotátorům jednoduchou možnost, jak s pomocí chvíle svého času strávené u počítače s Internetem pomoci.

B.2.1 Návrh databází, aneb co jsme chtěli uchovat

Základním účelem, pro který byla webová aplikace vytvářena, bylo získání výsledků z porovnání triviálního extraktu, jak jsme jej představili (viz kapitola 8), a naší sumarizace. Chtěli jsme tedy ke každé sumarizaci porovnání s triviální sumarizací z dokumentu, vždy ve dvou exemplářích od dvou různých lidí.

Snažili jsme se o návrh databáze, který by tento účel splnil. Pro tento účel bylo nakonec nutné vytvořit 8 databázových tabulek, které data uchovávaly.

Během programování aplikace byl kladen důraz na vyhovění normě XHTML 1.00 ve verzi Strict podle w3.org.

Tabulka users

Tabulka uchovává data o sumarizátorech. Vyžadovali jsme naprostou svobodu užití a nechtěli jsme odradit případné dobrovolníky nutností vyplňovat svůj e-mail. Proto jsme zvolili pouze tu cestu, kdy jsme k identifikaci vyžadovali pouze jejich smyšlenou přezdívku.

U každého sumarizátora bylo navíc velmi vhodné uchovávat i jeho aktuální zkušenost se sumarizacemi, kdybychom případně chtěli některé první sumarizace zanedbat.

Součástí tabulky bylo samozřejmě unikátní identifikátor uživatele, aby se s ním dalo „pracovat“ jednoduše i v ostatních tabulkách.

Tabulka articles

Zde jsou uchovány základní informace o dokumentech. Mezi nimi unikátní identifikátor dokumentu, jméno souboru, ze kterého pochází, počet vět v dokumentu a datum jeho vložení.

Tabulky sentences00_04, summaries00_04

Pro proces sumarizace jsme vyžadovali, aby všichni sumarizátoři měli stejné podmínky. Proto jsme zvolili, že maximální počet vět v sumarizaci bude 5 a jejich pořadí bude pevné. Ke každému dokumentu se tedy v pořadí uložili věty ze sumarizace i věty z náhodně přeuspořádaného triviálního extraktu. Společně

s unikátním identifikátorem dokumentu jsme uložili v textové podobě jednotlivé věty.

Tabulka toSummarize

Tato tabulka je asi nejdůležitější ze všech, protože uchovává unikátní identifikátory dokumentů, které mají být sesumarizovány, společně s unikátním příkazem, kterým byl extrakt získán. Také uchovává pozici, na které se má sumarizátorům objevit sumarizovaný extrakt oproti triviálnímu, počet již provedených porovnání vykonaných na dokumentu a případně již získaná data od sumarizátora, který jej porovnával.

Při každém provedení porovnání je výsledek tohoto porovnání uložen v tabulce results.

Po provedení dvou porovnání se ke každému ze záznamů v tabulce uloží unikátní číslo uživatele, který jej prováděl, a následně je tento záznam, s vynecháním některých již nepotřebných údajů přesunut do tabulky summarized.

Tabulka summarized

Tabulka pouze uchovává část tabulky toSummarize, která již prošla procesem porovnání sumarizátory. Ke každému unikátnímu identifikátoru dokumentu a příkazu zde existuje právě jeden záznam, který uchovává čísla uživatelů, kteří tento dokument s tímto příkazem již porovnávali.

Tabulka results

Toto je v podstatě klíčová tabulka z hlediska získávání výsledků. Každá anotace zde má právě jednu řádku, která o anotaci vypovídá potřebná data. Nachází se zde kombinace identifikátorů dokument, uživatel, příkaz, které jsou v tabulce unikátní. Tabulka ukazuje také aktuální zkušenost, jakou anotátor má, ale hlavně ukazuje to, zda se uživatel v prvním a v druhém případě sumarizace strefil uživatel do sumarizace vygenerované automaticky naším programem.

Tabulka summarizationCommands

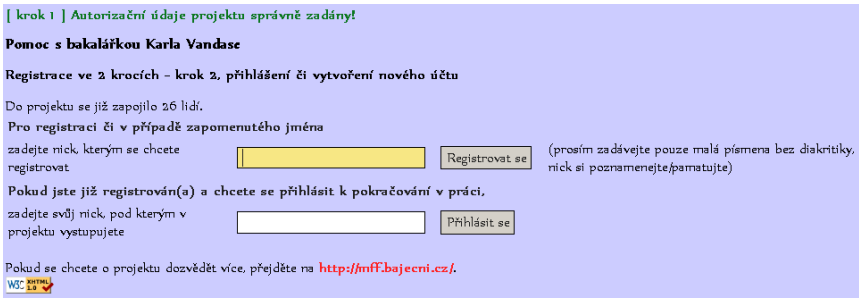
Tato část databáze je spíše pro pozdější použití, aktuálně jsme ji nepotřebovali, jelikož uchovává pro každý identifikátor příkazu soubor, ve kterém se celý příkaz nachází.

B.2.2 Průběh práce s anotačním programem

Na úvod se program načetl ze stránky http://sumarizace.bajecni.cz. Prvotní přihlášení k projektu vyžadovalo vyplnění údajů, které dostal anotátor pomocí některého z informačních kanálů (například emailem).

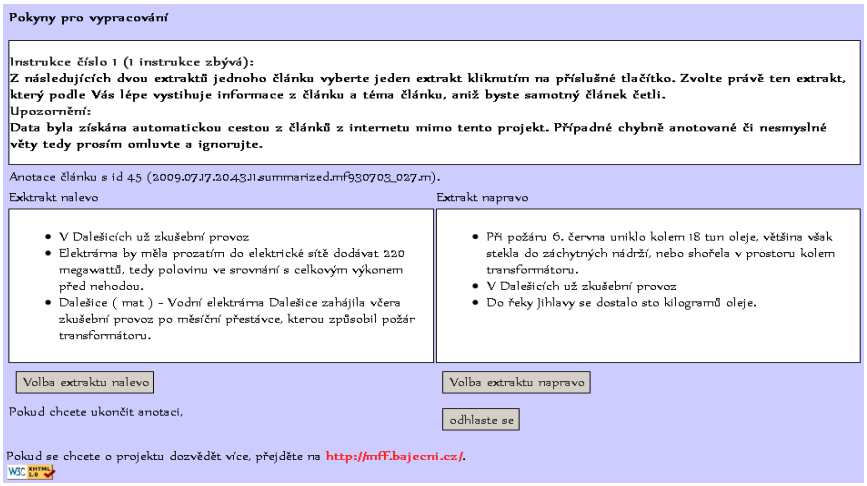
Obrázek 13: Úvodní obrazovka aplikace na http://sumarizace.bajecni.cz

Po úspěšné autorizaci se mohl člověk buď registrovat anebo přihlásit pod svojí již definovanou přezdívkou.



Obrázek 14: Obrazovka vyzývající anotátora k registraci či přihlášení

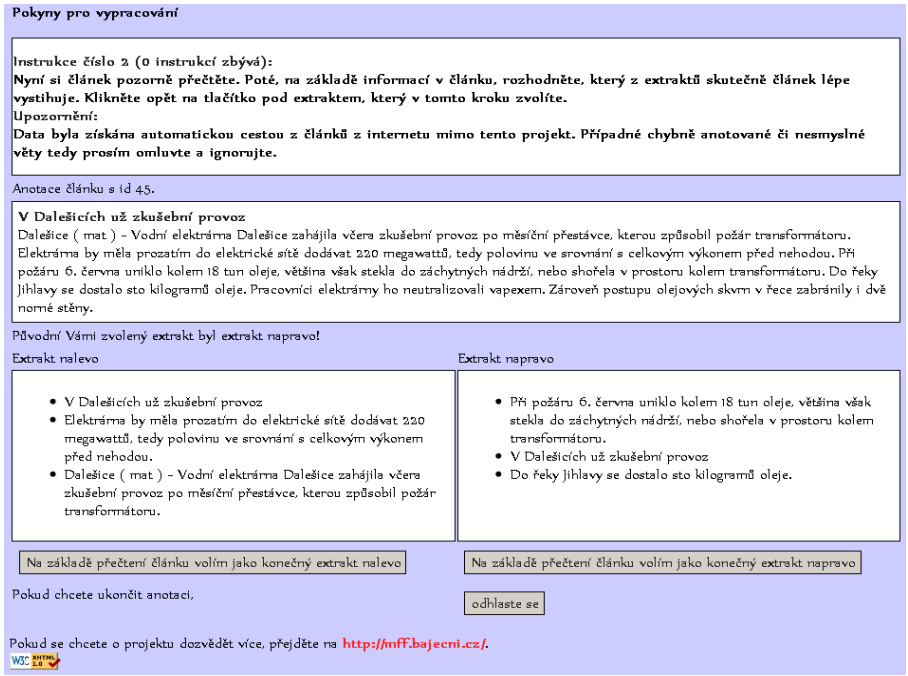
Po přihlášení či registraci se „šlo hned na věc“ a anotátor se přímo objevil u anotace prvního dokumentu. Instrukce jsou přítomné na každé obrazovce, nelze postupovat nesprávně.



Obrázek 15: Anotace, instrukce 1

Po provedení první instrukce se anotátor dostal rovnou na druhou instrukci, po níž došlo k přesměrování na anotaci dalšího dokumentu.

Abychom předešli situacím, kdy by uživatel dostával stejné dokumenty k anotaci, či situacím, kdy více anotátorů dostalo stejný dokument najednou a jejich anotace vyšly vniveč, byly do programu zabudovány pojistky. Například zpoždění umožňovalo, aby každý dokument byl uživatelům ukazován jen jednou za určitý časový úsek. Toho bylo docíleno nastavením „časové pojistky“ u dokumentu v databázi na čas, po němž jej bude možné opět použít.



Obrázek 16: Anotace, instrukce 2

Aplikace je k nalezení na přiloženém DVD.