VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

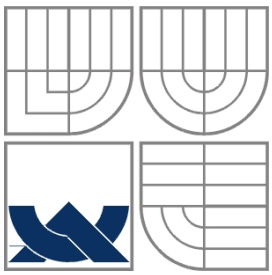
METODY STEMMINGU POUŽÍVANÉ PŘI DOLOVÁNÍ TEXTU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

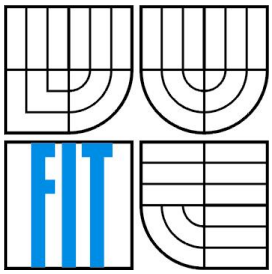
AUTOR PRÁCE
AUTHOR

BC. TOMÁŠ ADÁMEK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

METODY STEMMINGU POUŽÍVANÉ PŘI DOLOVÁNÍ TEXTU

STEMMING METHODS USED IN TEXT MINING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ ADÁMEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2010

Zadání diplomové práce/10574/2009/xadame27

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Adámek Tomáš, Bc.**
Obor: Informační systémy
Téma: **Metody stemmingu používané při dolování textu**
Stemming Methods Used in Text Mining
Kategorie: Databáze

Pokyny:

1. Seznamte se s problematikou dolování dat, podrobněji se zaměřte na dolování v textu.
2. Podrobně prostudujte jednotlivé algoritmy stemmingu pro anglický jazyk. Konzultujte s vedoucím výběr algoritmů, které budou v rámci DP implementovány.
3. Navrhněte aplikaci pro předzpracování textu, v rámci níž budou zvolené algoritmy implementovány.
4. Navrženou aplikaci implementujte a ověřte její funkčnost.
5. Proveďte experimenty, v nichž budou srovnány jednotlivé algoritmy z hlediska vlivu na výsledky klasifikace textu.
6. Zhodnoťte dosažené výsledky a další možné rozšíření projektu.

Literatura:

- Feldman, R., Sanger, J.: The Text Mining Handbook. Cambridge University Press, 2007.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1-3

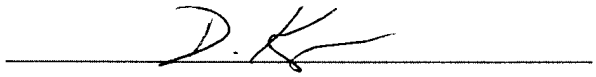
Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2



doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tématem této diplomové práce je problematika jednotlivých metod pro dolování z anglických textových dokumentů. Hlavní část této práce se zabývá analýzou metod pro předzpracování textu, konkrétně stemmingem. Jsou zde rozebrány jednotlivé algoritmy stemmingu (Lovinsův, Porterův a Paice/Husk), které z jednotlivých slov textového dokumentu získávají jejich základní tvar (kořen), za použití speciálních lexikografických pravidel anglického jazyka. Tyto kořeny slov jsou následně uloženy do strukturované podoby pro další zpracování. Další část práce se zabývá návrhem aplikace, která tyto algoritmy využívá pro svoji činnost. Aplikace je postavena na platformě Java s využitím grafické knihovny Swing a architektury MVC. Další kapitola popisuje implementaci navržené aplikace a stemovacích algoritmů v jazyce Java. Poslední kapitola je zaměřena na experimenty s jednotlivými algoritmy a jejich srovnání z hlediska vlivu na výsledky klasifikace textu.

Abstract

The main theme of this master's thesis is a description of text mining. This document is specialized to English texts and their automatic data preprocessing. The main part of this thesis analyses various stemming algorithms (Lovins, Porter and Paice/Husk). Stemming is a procedure for automatic conflating semantically related terms together via the use of rule sets. Next part of this thesis describes design of an application for various types of stemming algorithms. Application is based on the Java platform with using of graphic library Swing and MVC architecture. Next chapter contains description of implementation of the application and stemming algorithms. In the last part of this master's thesis experiments with stemming algorithms and comparing the algorithm from viewpoint to the results of classification the text are described.

Klíčová slova

Stemming, dolování v textu, vyhledávání informací v textu, klasifikace, Java, Swing, předzpracování textu, MVC architektura, RapidMiner, Weka.

Keywords

Stemming, text mining, information retrieval, classification, Java, Swing, text preprocessing, MVC architecture, RapidMiner, Weka.

Citace

Adámek Tomáš: Metody stemmingu používané při dolování textu, diplomová práce, Brno, FIT VUT v Brně, 2010

Metody stemmingu používané při dolování textu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Adámek
1.5.2010

Poděkování

Především bych rád poděkoval mému vedoucímu Ing. Vladimíru Bartíkovi, Ph.D. za poskytnutou pomoc, odborné vedení a čas věnovaný při konzultacích k tématu mé diplomové práce.

Obsah

Úvod

Tématem této diplomové práce je problematika dolování v textu se zaměřením na metody stemmingu, používané pro předzpracování textových dokumentů. Dolování v textu je problematikou, kdy se z velkého množství typicky semistrukturovaných dat snažíme získat informace do takové podoby, aby tyto informace mohly být uloženy ve strukturované podobě, např. v databázi. V dnešní době nalezneme velké množství malých dokumentů, které jsou rozsáhlejší než klasická strukturovaná data. Snažíme se tedy předzpracovat textová data takovým způsobem, abychom vyextrahovali pouze ty informace, které nás zajímají a jsou pro nás důležité. Tyto informace následně není problém uchovat např. v podobě databázových tabulek a s nimi dále efektivně pracovat.

Jednou z metod předzpracování textových dokumentů je metoda zvaná stemming. V textu se často opakují stejná slova, avšak v různých lexikografických tvarech. Toto uchování různých tvarů slov, které mají stejný význam, je v databázích značně nevýhodné a prostorově náročné. Uchovávají se tedy pouze základní tvary těchto slov, tzv. kořeny. Existuje velké množství těchto algoritmů stemmingu, které se využívají pro předzpracování textu a jsou více či méně úspěšné. Při vyhledávání slova např. stemming se nevyhledává v dokumentech pouze toto slovo, ale vyhledávač vyhledává dokumenty, kde se vyskytuje slovo stem, což je kořen hledaného slova.

Úvodní kapitola této práce je zaměřena na architekturu systému a metody využívané pro dolování v textu. Těchto metod specifických pro dolování v textu je typicky několik druhů a nemusí se jednat pouze o předzpracování dat za pomoci stemmingu. Další kapitola je zaměřena na popis jednotlivých algoritmů stemmingu (Lovinsova, Porterova, Paice/Husk) a popis jejich činnosti. Následuje 3. kapitola s popisem návrhu aplikace pro předzpracování anglických textových dokumentů v jazyce Java, která tyto metody dokáže spustit a názorně předvést, jaký bude výsledek po jejich provedení. Aplikace umožňuje výběr některé z implementovaných metod stemmingu a následnou práci s algoritmem. Celá tato aplikace je postavena na platformě Java Swing/GUI na architektuře MVC s graficky přívětivým uživatelským prostředím. Ve 4. kapitole je popsána implementace aplikace. Jsou zde rozebrány jednotlivé třídy aplikace a jejich činnost. Dále jsou zde popsány jednotlivé obrazovky. Následuje popis vstupních dat, které aplikace umožňuje zpracovat a popis výstupních dat, které aplikace produkuje. V této kapitole je také popsána činnost stemmingu dokumentů v aplikaci. Dále jsou zde popsány možnosti kompilace a spouštění programu. V 5. kapitole jsou popsány jednotlivé experimenty s implementovanou aplikací. Tyto experimenty jsou zaměřeny na dobu trvání jednotlivých algoritmů a redukci počtu slov. Dále jsou zde experimenty prováděné v nástrojích RapidMiner a Weka, zaměřené především na vliv stemovacích metod na úspěšnost klasifikace textu. Je zde také provedeno shrnutí jednotlivých experimentů.

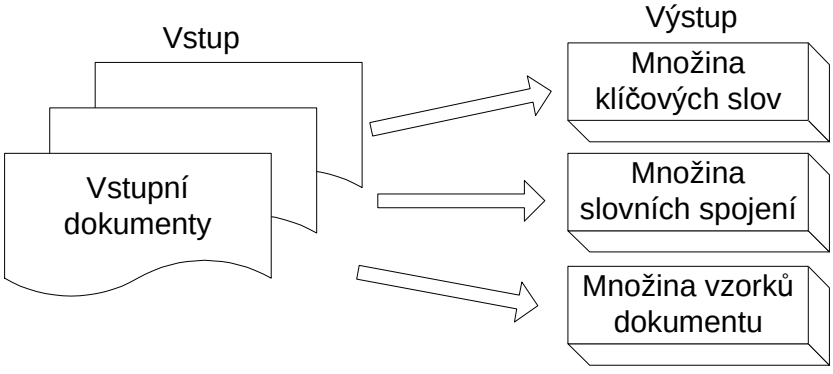
V závěru této práce jsou shrnuty výsledky diplomové práce a možnosti dalšího směru vývoje.

1 Dolování v textu

Dolování v textu (text mining) [1] je speciální metodou pro dolování dat. Při dolování dat máme k dispozici většinou strukturovanou podobu dat, se kterou pracujeme, např. v podobě databázových tabulek. V případě dolování v textu je naším požadavkem především velké množství dokumentů, které jsou typicky v podobě přirozeného jazyka nestrukturované či semistrukturované [2]. Jako semistrukturované dokumenty jsou často označovány textové dokumenty, emaily, html stránky, pdf soubory nebo jiné formáty, které mají pro své vytváření danou pevnou formu či šablonu. Takovéto dokumenty obsahují velké množství informací a klasické metody pro dolování dat se pro jejich zpracování příliš nehodí. Mezi klasické úlohy dolování v textu patří vyhledávání informací, vyhledávání textů, klasifikace dokumentů či sumarizace textů.

Klíčovým elementem pro dolování v textu je kolekce dokumentů. Každý dokument může obsahovat velké množství slov, frází, vět, typografických elementů a úprav, kde každý z těchto elementů může nabývat různého významu. Systémy pro získávání znalostí z textu mají za úkol identifikovat jednoduché podmnožiny vlastností textových dokumentů, které mohou reprezentovat tento dokument jako celek. Tato skupina dokumentů může být statická nebo dynamická. Statická kolekce se v průběhu činnosti nemění, naopak změn doznává dynamická kolekce dokumentů, která v průběhu času pracuje s novými či editovanými dokumenty. Typická dynamická kolekce pro získávání znalostí z textu je např. Národní knihovna medicínských dat, která sdružuje velké množství informací v podobě článků z výzkumných ústavů biomedicínských dat. Snahou této organizace je ručně zmapovat a sladit získaná data ze všech těchto dokumentů tak, aby bylo možné nalézt a identifikovat nové trendy. Automatické metody pro identifikaci a průzkum těchto interních dokumentů výrazně urychlují možnosti nového výzkumu. Tyto metody pro získávání znalostí z textu nejsou užitečné pouze ve výzkumných a vědeckých ústavech, ale také v reálném světě, pro nalezení vzorků uvnitř velkého množství dokumentů s přirozenou řečí.

Výrazný důraz u dolování v textu je kladen především na jeho předzpracování. Předzpracování textu zahrnuje velké množství různých technik, uzpůsobených k získávání relevantních informací z dokumentů a matematicko-lingvistických zkoumání, která transformují nezpracovaná, nestrukturovaná data do podoby propracované struktury. Problémem je však najít takový model vlastností, který bude pro náš jazyk úspěšně pracovat. Jako ukázku lze uvést malou kolekci 15000 dokumentů z agentury Reuters, které obsahují přes 25000 netriviálních kořenů slov. Problém vysoké dimenzionality dat u získávání znalostí z textu je dán především velikostí a velkým množstvím různých kombinací. Tato vysoká dimenzionalita dat přispívá k vývoji automatizovaných systémů pro předzpracování dat v textových dokumentech, zaměřených na tvorbu usměrněné reprezentace modelu. Tyto systémy jsou většinou kvůli efektivitě odděleny od klasických systémů pro zpracování dat. Používají se tedy techniky speciálně upravené pro dolování v textu.



Obrázek 1.1 – Abstraktní architektura systému získávání znalostí z textu.

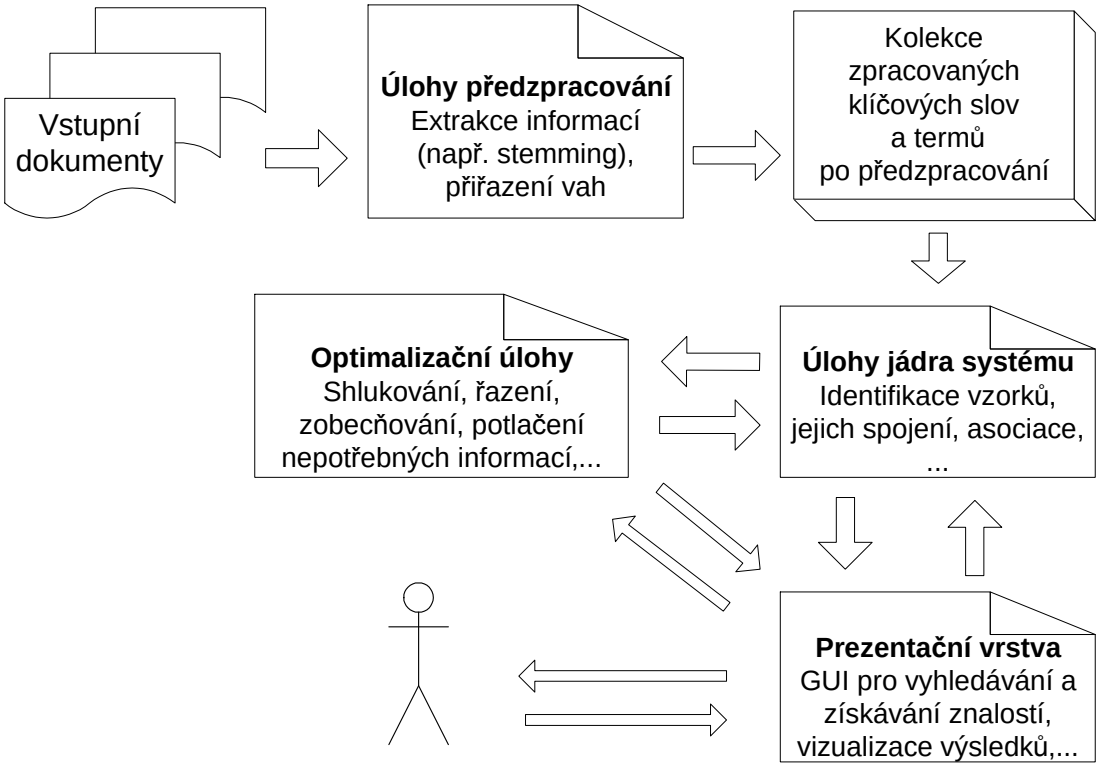
1.1 Architektura systémů

Na nejvyšší úrovni abstrakce lze říci, že systémy získávání znalostí z textu zpracují vstupní dokument a vygenerují z něho množinu slov, vzorků či různých spojení, obsažených ve vstupním dokumentu. Ukázka takovéto abstraktní architektury systému pro předzpracování dokumentů je na obrázku 1.1.

Na funkční úrovni [1] však celý systém můžeme rozčlenit do 4 částí:

1. *Úlohy předzpracování textových dat*: Tato část systému zahrnuje metody pro přípravu dat pro operace jádra systému. Typicky se jedná o úlohy pro extrakci klíčových slov použitím stemmingu, metody pro odstranění stop slov, přiřazení vah jednotlivým termům dokumentu a další. Cílem předzpracování je vytvoření strukturované podoby textových dat.
2. *Operace jádra systému:* Tyto operace mají za úkol nalezení slov či termů, které jsou v dokumentech umístěny často společně a nalézt mezi nimi jejich asociace. Využívá se např. asociačních analýz založených na klíčových slovech. Jádro systému zapouzdřuje získávání nových poznatků o dokumentech a jejich analýzy.
3. *Prezentační vrstva systému:* Tato vrstva zahrnuje GUI systému pro přístup k jednotlivým dokumentům, vyhledávání slov v dokumentech, vizualizace, jednoduché filtry.
4. *Optimalizační úlohy:* Jedná se o metody pro vylepšování výsledků vyhledávání. Typicky používanou dokončovací (optimalizační) úlohou je clustering (shlukování), řazení či zobecňování. Cílem je také odfiltrování (potlačení) nepotřebných informací a uzavření potřebných informací do struktur pro efektivnější spolupráci.

Ukázka funkční úrovně systému je zobrazena na obrázku 1.2. Některé významné metody, využívané pro dolování v textu, které výrazně ovlivňují získávání znalostí, si stručně představíme v dalších kapitolách. Jedná se především o úlohy předzpracování: extrakci informace (získávání kořenů slov dokumentu v kapitole o stemmingu), přiřazení vah termům či klíčovým slovům, stop list. Další z úloh dolování v textu jsou metody pro vyhledávání informací v textu, klasifikaci a shlukování.



Obrázek 1.2 – Funkční úroveň systému pro dolování v textu.

1.2 Váhovací metody

Jednou z technik, která se používá pro předzpracování textových dokumentů, je použití tzv. váhovacích metod. Celý textový dokument je možné popsat množinou slov, která mají v dokumentu různou důležitost. Každému takovému slovu (termu) je následně možné přiřadit jeho numerickou váhu na základě např. frekvence jeho výskytu v dokumentu. Pro přiřazení jednotlivých vah dokumentu se využívají speciální váhovací metody, např. frekvenční matice termů a dokumentů. Váhování slov má za cíl především zlepšit přesnost a úplnost. Váhovací metody typicky využívají tzv. vektorového modelu [15], kde každý dokument D_j z kolekce *n* dokumentů je reprezentován vektorem D_j, kde *1 <= j <= n* v *m*-rozměrném prostoru z celkového počtu slov. Každé slovo na í-tém prvku, kde *1 <= i <= m*, obsahuje váhu (frekvenci) slova *i* v dokumentu D_j.

1.2.1 TF-IDF

Jednou z nejznámějších metod, která se využívá pro váhování termů, je metoda TF-IDF (Term frequency-Inverse document frequency) [13].

Term frequency (TF) určuje počet výskytů slov v daném dokumentu a vypočítáme jej pomocí vzorce: $TF_{i\,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$, kde $n_{i,j}$ je počet výskytů uvažovaného slova v dokumentu D_j a jmenovatel představuje počet slov v dokumentu D_j. TF uvažuje více frekventovaná slova uvnitř dokumentu, což je více prospěšné pro potřeby vyhledávání.

Inverse document frequency (IDF) určuje počet výskytů uvažovaného slova v rámci kolekce dokumentů pomocí vzorce: $IDF_i = \log \frac{|D|}{|d_j : t \in d_j|}$, kde |D| je celkový počet dokumentů v trénovaní množině a $|d_j : t \in d_j|$ je počet dokumentů, ve kterých se vyskytuje term t_i. Váhování lze tedy vypočítat jako $TFIDF_{i\,j} = TF_{i,j} * IDF_i$.

Frekventované slovo v rámci dokumentu nebo kolekce způsobí vysoké TF nebo IDF a to způsobí vysokou váhu. Této váhy se využívá např. v modelech pro vyhledávání informací nebo také pro klasifikaci textových dokumentů, kde je nutné znát počet výskytů jednotlivých slov v dokumentech.

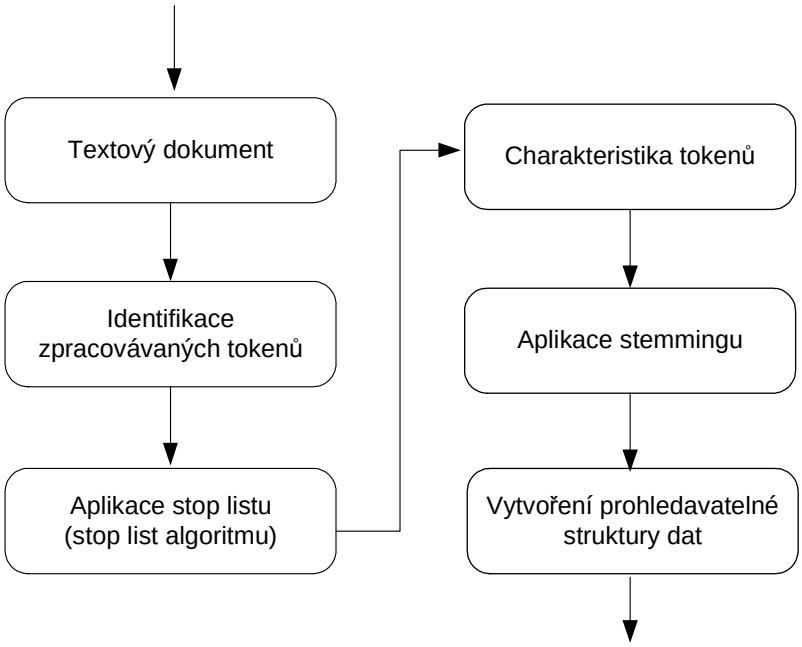
1.3 Stop list

Další technikou využívanou při předzpracování textových dat je použití stop listu, někdy též nazývaného jako seznam stop slov. Úkolem použití stop listu je předzpracování textu před samotným stemovacím procesem. Stop list [29] odstraní z dokumentu slova, která jsou v anglickém jazyce používána nejčastěji a jejich informační hodnota není pro dolování v textu příliš důležitá. Tato vlastnost však významně závisí na typu dokumentu. Stop list zahrnuje seznam slov, jako je např. and, předložky, spojky a jiná slova. Ukázka stop listu, který využívá společnost Google [10] pro anglický jazyk je znázorněna v tabulce 1.1. Další seznam anglických stop slov lze nalézt ve zdroji [5] nebo v příloze D této diplomové práce. Na internetu lze nalézt anglické stop listy v široké škále, obsahující několik málo slov až stovky. Součástí některých programovacích jazyků jsou i knihovny, které v sobě obsahují práci se stop listem. Práci se stop listem obsahují i nástroje využívané pro klasifikaci a předzpracování textových dokumentů popsané v kapitole 5.3.

I	how	where
a	in	who
about	is	will
an	it	with
and	la	und
are	of	the
as	on	www
at	or	
be	that	
by	the	
com	this	
de	to	
en	was	
for	what	
from	when	

Tabulka 1.1 – Stop list anglických slov využívaný společností Google.

Na jednoduché anglické větě si ukážeme, jak stop list při dolování v textu funguje. Pokud budeme chtít vyhledat např. slovní spojení Ronaldo is a player, pak se z vyhledávání nejdříve odstraní slovo „is“ a následně slovo „a“. Tyto dvě slova jsou obsažena ve stop listu v tabulce 1.1. Následně se vyhledají všechny výskyty slova „Ronaldo“ a slova „player“. Aplikace stop list algoritmu je jednoduchá tím, že se hledá celková shoda na slovo v seznamu stop slov. Pokud je nalezeno, odebere se slovo z dokumentu, pokud ne, pak je slovo předáno k dalšímu zpracování tak, jak je uvedeno na obrázku 1.3. Na tomto obrázku je znázorněn postup předzpracování textového dokumentu s využitím stop listu a následnou aplikací stemmingu (podrobněji popsán v 2. kapitole).



Obrázek 1.3 – Předzpracování textového dokumentu s použitím stop listu a stemmingu.

1.4 Vyhledávání informací v textu

Pro měření kvality daného přístupu k vyhledávání informací v textu může být důležitá celá řada faktorů, např. rychlost zpracování dotazů či schopnost poskytnout uživateli informaci o relevantních dokumentech, kterou lze vyjádřit za pomoci dvou koeficientů: přesnosti a úplnosti [2]. Přesnost reprezentuje procento z nalezených dokumentů, které odpovídají zadanému dotazu a úplnost ukazuje procento z relevantních dokumentů, které byly skutečně nalezeny. Relevantní dokumenty jsou pouze takové, které lexikálně obsahují zadaný dotaz. Dotaz uživatele musí být předzpracován stejně jako kolekce dokumentů, musí tedy projít odstraněním stop slov, stemmingem, přiřazením vah atd. Dotaz je tedy opět dokument, pro který hledáme podobné dokumenty, tedy dokumenty relevantní k dotazu uživatele. Mezi nejpoužívanější modely pro vyhledávání informací v textu (Information Retrieval) patří vektorový model a booleovský model.

Vector space model [2] je metoda využívající pro vyhledávání informací tzv. váhovacích metod (popsané v kapitole 1.2) a *m*-dimenzionálních vektorů termů, kde *m* je celkový počet index termů v kolekci dokumentů. Dokumenty a uživatelské dotazy jsou reprezentovány vektorem tvořeným TF-IDF vahami termů. Stupeň podobnosti [30] dokumentu *P* a uživatelského dotazu *Q* je vypočtena jako korelace mezi vektory, které je reprezentují, za pomoci využití výpočtu:

Euklidovské vzdálenosti: $d(P,Q) = \sqrt{\sum_{i=1}^{m} (p_i - q_i)^2}$ nebo

Kosinové vzdálenosti: $s(P,Q) = \frac{\sum_{i=1}^{m} (p_i * q_i)}{\sqrt{\sum_{i=1}^{m} (p_i)^2 * \sum_{i=1}^{m} (q_i)^2}}$, kde *P* je vektor vah jednotlivých termů dokumentu, *Q* je vektor vah jednotlivých termů dotazu a *m* je počet termů ve vektoru.

Booleovský model [2] je další metodou pro vyhledávání informací v textu. Booleovský model určuje pouze relevantnost či irelevantnost dokumentu na základě shody dokumentu s dotazem. Váhy index termů musí být uloženy v binární podobě. Dotazy jsou tvořeny pomocí index termů a logických spojek and, or a not. Nevýhodou tohoto modelu je nalezení pouze relevantních dokumentů na základě zvoleného dotazu, tzn. pouze na základě slov, která jsou v dokumentu lexikálně obsažena. Vyhledávání tedy nerespektuje různá synonyma slov či slova, která mohou mít více významů. Tento model tedy nevystihuje příliš dobře sémantiku jednotlivých slov a dokumentů.

1.5 Klasifikace textu

Klasifikace textu je metoda využívaná při dolování v textu [1]. Někdy je tato technika nazývána též jako kategorizace textů. Cílem klasifikačních algoritmů je zařazení textových dokumentů do předem připravených kategorií (tříd) a přidání strukturované informace k nestrukturovanému textu. Klasifikace je prováděna na základě získaných znalostí naučených z trénovací množiny dat, určující třídu daného nestrukturovaného textu. Na základě výběru několika vhodných klíčových slov dokumentu lze tento dokument zařadit do jednotlivých kategorií. Důležitost této disciplíny je především v obrovském nárůstu nestrukturovaných informací na internetu, ale také v různých oblastech vědy.

Funkci klasifikátorů, které jsou schopny automaticky rozřazovat textové dokumenty do tříd týkajících se určité oblasti, lze rozdělit do následujících třech kroků:

1. Extrakce feature (příznakového) vektoru z textu.
2. Redukce dimenzí.
3. Klasifikace.

Feature vektor (příznakový vektor) je *m*-dimenzionální vektor, reprezentující výskyt slov v dokumentech. Vektor může být buď binární (slovo se v dokumentu vyskytuje nebo ne), nebo může obsahovat četnost výskytu, případně váhy, založené na důležitosti slov v celé kolekci dokumentů, využívající váhovací metody TF-IDF popsané v kapitole 1.2. K extrakci příznakového vektoru se nejčastěji využívá klasického modelu bag-of-words podrobněji popsaného v literatuře [1], který předpokládá nezávislost jednotlivých slov na jejich pozici v textu. Příznakový vektor by následně mohl obsahovat např. jedničku pro slovo nacházející se v textu a nulu pro slovo, které v textu není. Sada dokumentů je pak modelována jako matice termů v dokumentech. Máme tedy k dispozici matici *n* x *m*, kde *n* je počet dokumentů a *m* je počet slov.

Redukce dimenzí je důsledkem vysokého rozměru matice pro příznakový vektor dokumentů a to zejména počtem sloupců. Počet dimenzí v dokumentech se reálně pohybuje ve stovkách tisíc. Pro redukci dimenzí se využívají především metody pro transformaci slov nebo doménové dekompozice. V první metodě pro transformaci slov se využívá tzv. singulárního rozkladu dokumentů SVD = singular value decomposition [15]. Nejoblíbenější metodou využívající SVD je v IR indexování latentní sémantiky (LSI). Tato metoda umožňuje reprezentovat vazby mezi termy a dokumenty, které nejsou na první pohled v dokumentech patrné. Uživatel chce často vyhledávat dokumenty s podobnou sémantikou jako jeho dotaz a ne pouze ty, které obsahují jeho slovo z dotazu. V případě doménové dekompozice jsou data rozdělena do podsouborů.

Klasifikaci lze rozdělit do několika skupin na základě jejich hledání odpovídajících tříd či trénování. Rozlišujeme klasifikaci mono a multi [14], kde je rozdíl především v přiřazování dokumentů do tříd. Při mono-klasifikaci je dokumentu přiřazena ta třída, která dosáhne nejvyššího ohodnocení a každý dokument je možné přiřadit právě do jedné z *n* tříd. Speciálním případem této klasifikace je binární klasifikace, kde existují jen třídy „patří“ a „nepatří“. Při multi-klasifikaci jsou dokumentu přiřazeny všechny třídy, jejichž hodnocení přesáhne určitou mez. Při trénování vyhledáváme taková pravidla, která popisují danou kategorii.

Existuje celá řada klasifikátorů, které je možno využít pro klasifikaci textových dokumentů. Prvním druhem jsou tzv. lineární klasifikátory, popisující kategorie i dokumenty lineárním vektorem. Tyto vektory se nastavují ve shodě s trénovacími dokumenty. Vektor dokumentu se následně porovná se všemi vektory kategorií a vybere se kategorie s nejlepším výsledkem. Mezi lineární klasifikátory patří metody jako jsou SVM (Support Vector Machine), Sleeping expert, NaiveBayes (jednoduchá Bayseovská) klasifikace popsána v kapitole 1.5.2. a další algoritmy podrobněji rozebrány v [14]. Další skupinou jsou metody založené na příkladech. Takovéto klasifikátory nejprve naleznou k dokumentu jemu podobné dokumenty a podle jejich zařazení do kategorií zjistí kategorii, do které patří příslušný dokument. Mezi tyto metody patří např. k-NN algoritmus popsaný v kapitole 1.5.1. Dalšími metodami jsou algoritmy pro strojové učení. Do této kategorie se řadí genetické algoritmy, využívající přirozený vývoj a algoritmy založené na použití fuzzy množin a rozhodovacích pravidel.

1.5.1 K-NN algoritmus

K-NN (k-nearest neighbor) [24] je klasifikační algoritmus založený na hledání nejbližších sousedů. Jedná se o neparametrickou metodu klasifikace, u které není předpoklad znalosti pravděpodobnostních charakteristik tříd. U k-NN algoritmu známe trénovací množinu dokumentů, které mají přiřazenu třídu dokumentu (label). Dokumenty jsou reprezentovány vektorem a každé slovo dokumentu reprezentuje jednu dimenzi. Jednotlivé souřadnice reprezentující dokument jsou tvořeny např. TF-IDF vahami slov dokumentu. Výpočet k-NN algoritmu probíhá setříděním a nalezením *k* nejbližších trénovacích dokumentů k testovanému dokumentu a zařazení tohoto dokumentu do nejvíce podobné třídy. Pro výpočet podobnosti dokumentů je využíváno měření Euklidovy nebo Kosinovy vzdálenosti mezi dvěma vektory dokumentů.

Ve srovnání s jednoduchým bayesovským klasifikátorem, k-NN nespoléhá na předchozí pravděpodobnosti, což je výpočetně efektivnější. Hlavní nevýhodou tohoto algoritmu je volba vhodného *k* pro klasifikaci textových dokumentů.

Výpočet kosinovy vzdálenosti dvou dokumentů v případě k-NN algoritmu lze reprezentovat následovně:

$s(X, D_j) = \frac{\sum_{t_i \in (x \cap D_j)} x_i * d_{i,j}}{\|X\|_2 * \|D_j\|_2}$, kde X je testovaný dokument reprezentovaný vektorem, D_j je *j*-tý trénovací dokument, t_i je slovo společné pro dokument X a D_j, x_i je váha slova t_i v dokumentu X a $d_{i,j}$ je váha slova t_i v dokumentu D_j. $\|X\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + ...}$, $\|Dj\|_2 = \sqrt{d_{1j}^2 + d_{2j}^2 + d_{3j}^2 + ...}$

1.5.2 NaiveBayes algoritmus

NaiveBayes (jednoduchá Bayesovská) klasifikace [14] se řadí mezi pravděpodobnostní klasifikační metody. Tato metoda se snaží pro každý dokument nalézt pravděpodobnost, s kterou náleží do dané třídy. Pokud je tato pravděpodobnost nejvyšší nebo přesáhne jistou hranici, pak je tento dokument do této třídy zařazen.

Podle Bayesova vzorce platí: $P(Y \mid X) = \frac{P(X \mid Y) * P(Y)}{P(X)}$, kde $P(Y \mid X)$ je pravděpodobnost výběru třídy *Y* za předpokladu, že máme vektor dokumentu X. $P(X \mid Y)$ je pravděpodobnost, že dokument vybraný ze třídy *Y* bude popsán vektorem X. $P(Y)$ je pravděpodobnost výskytu třídy *Y* a $P(X)$ je pravděpodobnost výskytu vektoru dokumentu X.

Vektor X je složen z atributu dokumentů $X_1..X_n$, proto lze zapsat $P(X \mid Y) = P(X_1..X_n \mid Y)$.

Atribut X_i typicky udává počet výskytů daného atributu v dokumentu nebo pouze hodnotu 0 pokud se daný atribut v dokumentu nenalézá nebo hodnotu 1 pokud se tam naopak nalézá. Při uvažování nezávislosti jednotlivých atributů lze pravděpodobnost zapsat součinem jako:

$$P(X_1..X_n \mid Y) = P(X_1 \mid Y) * ... * P(X_n \mid Y) = \prod_{j=1}^{n} P(X_j \mid Y)$$

Pravděpodobnost výskytu atributu X_j ve třídě *Y* je lze zapsat jako pravděpodobnost:

$P(X_j \mid Y) = \frac{p_{Yj}}{p_Y}$, kde p_{Yj} je počet výskytů atributů v dokumentu X_j ve třídě *Y* a p_Y je počet výskytů všech atributů ve třídě *Y*.
Z těchto odvození obdržíme výslednou hodnotu podmíněné pravděpodobnosti:

$$P(Y \mid X) = P(Y) * \prod_{j=1}^{n} \frac{P(X_j \mid Y)}{P(X_j)}$$

Důležitou podmínkou tohoto vzorce je nenulová hodnota pravděpodobnosti výskytu každého použitého atributu.

Nevýhodou této klasifikační metody je zpracování velkého množství atributů při výpočtech pravděpodobností. Další nevýhodou je předpokládaná nezávislost slov v dokumentech, které však na sobě mohou ve skutečnosti záviset. Výhodou tohoto algoritmu je rychlost zpracování textových dat při klasifikaci dokumentů, která je srovnatelná s klasifikační metodou k-NN.

1.5.3 SMO algoritmus

SMO (Sequential Minimal Optimazition) [25] je klasifikační algoritmus pro trénování založený na SVM (Support Vector Machines) klasifikátoru. Jedná se o koncepčně jednoduchý, snadno implementovatelný a často rychlejší algoritmus než je tomu u klasických učících algoritmů. Tyto algoritmy spadají do podkategorie tzv. jádrových algoritmů využívajících tzv. jádrových funkcí. Jádrová funkce muže být aplikována na dvojice vstupních dat k vyhodnocení skalárního součinu v nějakém odpovídajícím prostoru. Klasické SVM [26] patří do kategorie lineárních klasifikátorů. Tyto klasifikátory hledají při trénování nadrovinu, která rozděluje data do dvou skupin. Základním principem tohoto algoritmu je rozdělení instancí z *m*-dimenzionálního prostoru do nového vícedimenzionálního prostoru, kde již lze oddělit jednotlivé třídy lineární hranicí. Snahou je nalezení optimálního lineárního oddělovače, který bude mít co nejširší pásmo mezi sebou, pozitivními příklady na straně jedné a negativními příklady na straně druhé. K hledání optimálního oddělovače se v algoritmu SVM využívá tzv. kvadratického programování. SMO spadá navíc do kategorie polynomiálních a RBF. Implementace tohoto algoritmu globálně odstraňuje všechny chybějící hodnoty a transformuje číselné atributy na binární jednotky. Ve výchozím nastavení tohoto algoritmu dochází také k normalizaci všech atributů. Problém multi-klasifikace je v této metodě vyřešen využitím párové klasifikace. Více o tomto klasifikačním algoritmu lze nalézt v literatuře [25].

Výhodou tohoto algoritmu je schopnost zpracování velkého množství atributů. Tento algoritmus je také schopen provádět klasifikaci v kratším časovém úseku než například neuronové sítě.

1.6 Clustering (shlukování)

Další technikou, která se při dolování v textu využívá je clustering (shlukování) [1]. Shlukování je proces, při kterém jsou jednotlivé objekty textových dokumentů děleny do skupin (množin) objektů. Cílem shlukování [2] je vytvořit skupiny dokumentů, které spolu souvisejí na základě jejich obsahu. Tyto skupiny slouží především ke zmenšení velikosti prohledávaných dat a pro výběr skupin, které uživatel považuje za relevantní pro vyhledávání. Klasifikace popsaná v kapitole 1.5 pracovala za podmínky předdefinované trénovací množiny dat. Systémy pro klasifikaci textů měly za úkol dokumenty na základě trénovací množiny dat zařadit do odpovídající třídy. Klasifikace tedy

pracovala jako učení s učitelem. Oproti tomu shlukování má za úkol zařazení objektů do clusteru bez jakékoliv předchozí informace (trénovací množiny). Shlukování pracuje jako učení bez učitele. Nemáme tedy k dispozici informace od učitele.

Shlukovací algoritmy lze rozdělit na hierarchické a modelově založené shlukování. Hierarchické shlukování se provádí na základě podobností dokumentů s využitím shlukovacích algoritmů. V případě shlukování dokumentů je objektem jednotlivý dokument. Dokument je popsán lineárním vektorem pomocí jeho slov. Pro výpočet podobnosti dokumentů lze následně použít některou vzdálenostní funkci popsanou v kapitole 1.4. Hierarchické shlukování lze následně provést pomocí postupného vytváření stromové struktury. Na začátku tvoří každé slovo samostatný shluk a pomocí výběru nejbližších (nejvíce podobných) tyto shluky spojujeme do větších celků. Podobnosti shluků můžeme rozlišit jako min = vzdálenost dvou nejvíce podobných vzorků, max = podobnost dvou nejméně podobných vzorků a group avarage = průměr ze všech možných dvojic. Modelově založené shlukování pracuje především za pomoci neuronových sítí či fuzzy logiky.

Nejznámější metodou, která se využívá pro shlukování je metoda K-Means [16]. Jedná se o nehierarchické shlukování metodou nejbližších středů. Tento algoritmus dělí množinu dokumentů do *k* shluků tak, aby vzdálenosti od středu shluků byly co nejmenší. Využívá se především euklidovského měření vzdálenosti, ale je možné využít některé z dalších metod. Výhodou tohoto algoritmu je především jeho časová složitost a jeho nevýhodou je potřeba uvedení počtu shluků a těžiště ještě před začátkem samotného výpočtu.

Pro zadané *k* pracuje algoritmus K-Means pomocí těchto 4 kroků:

1. krok: Rozděl objekty do *k* neprázdných shluků.

2. krok: Spočítej těžiště (střed, centroid) každého shluku při tomto rozdělení. Těžiště je vektor průměrných hodnot pro každý term vstupující do shlukovací procedury. Slova jsou reprezentována jako 1 a 0 v záležitosti na tom zda v clusteru jsou nebo nejsou. Ke každému slovu jsou přiřazeny frekvence výskytů, určující příslušnost k patřičnému shluku. Ukázka přiřazené frekvence je zobrazena v následující tabulce 1.2.

3. krok: Přiřaď každý objekt do shluku na základě minimální vzdálenosti od středu shluku. Z tabulky 1.2 je vidět, že 1.shluk, neobsahující slovo back, má větší procento požadavků na slovo head a unknown ke svému shluku. Shluk 2, obsahující slovo back, má naopak větší požadavky na slovo train.

4. krok: Pokud bylo změněno přiřazení do shluku nebo střed shluku, opakuj postup od druhého kroku.

Číslo shluku	back	head	train	unknown	acceleration	knee
1	0	0.15	0.05	0.12	0.18	0.12
2	1	0.04	0.40	0.00	0.00	0.10

Tabulka 1.2 – Frekvence termů pro 2 shluky.

Další technikou, která se pro shlukování používá, jsou neuronové sítě [17]. Přijatá slova jsou transformována do vektorů, které jsou následně zakódovány v podobě neuronových sítí. Jednotlivé neurony jsou mezi sebou propojeny spoji ohodnocenými vahami. Schopnost přiřadit jednotlivým uzlům jejich váhy umožňuje učení neuronové sítě na základě množiny trénovacích dat. Postupným učením se vytvářejí jednotlivé shluky. Jednou z používaných metod je Kohonenova samoorganizující neuronová síť podrobněji popsána v dokumentu [17].

2 Stemming

Stemming [11] je technika významná při získávání znalostí z textu. Jedná se o jednu z technik předzpracování textových dat, kdy za použití lexikografických pravidel jazyka získáme z různých tvarů slov jejich kořeny. Jako příklad lze uvést např. anglické slovo search, které se může vyskytovat ve tvarech: search, searches, searching. V tomto textu se zaměřím na algoritmy stemmingu používané v anglickém jazyce. Algoritmy, které zde uvedu, mohou správně pracovat pro několik dalších přirozených jazyků za použití změny v pravidlech, ale tyto jsou optimalizovány především pro anglický jazyk. Pro každý jazyk existují jiná pravidla, která musí daný algoritmus využívat tak, aby výsledek předzpracování byl co nejpřesnější vzhledem k sémantice slov. V případě úpravy pravidel více do hloubky mohou být tyto algoritmy využity v několika jazycích současně, za cenu toho, že výsledek předzpracování nebude tak uspokojivý.

Pro předzpracování textových dokumentů metodou stemmingu existuje pro anglický jazyk několik různých stemovacích algoritmů, které lze využít pro získávání informací z rozsáhlých databází, které uchovávají textové dokumenty. Každý z těchto algoritmů využívá jinou metodiku pro získání kořene slov. V tomto textu se zaměřím na metody, které odstraňují koncovky slov (přípony) a dle gramatických pravidel jazyka tyto koncovky změní na správný tvar slova (kořen) nebo na slovo, které lze za kořen stemovaného slova prohlásit. Na slova, kde by bylo možné odstranit i předponu se zaměřují jiné druhy algoritmů, ale jejich výsledky nejsou zatím takové, aby jejich použití výrazně urychlilo vyhledávání informací a snížilo kapacitu uložených slov ve strukturované podobě. Odstraňování předpon se v současné době využívá pouze v některých oborech, např. v chemii.

Význam těchto metod je především ve vyhledávání v dokumentech, kde vzrůstá počet slov, které mají stejný kořen, ale odlišné koncovky. Za pomoci použití stemmingu tak získáme mnohem menšího množství dat, získaného z textového dokumentu a tato data lze uložit v menší velikosti do databáze či jiné strukturované podoby. Výhodou této strukturované podoby dokumentů je především v úspoře použitého místa a v rychlejším přístupu k informacím. Hlavní nevýhodou metod stemmingu je kvalita vyhledaných výsledků, které jsou od neořezaných slov méně přesné a také doba předzpracování dat. Při vyhledávání pouze podle kořenů slov pak není neobvyklé, že obdržíme výsledky, které vůbec neodpovídají zadanému dotazu. V dnešní době se této metody stemmingu využívá například ve společnosti Google při vyhledávání informací na internetu.

Existují jazykové vývojové skupiny [11], které vyvíjejí velké množství jazykových nástrojů, jenž mohou být použity při dolování v textu. Jejich hlavním úkolem je vytvořit anglickou lexikální databázi slov, která bude obsahovat tvary všech slov a bude poskytovat jejich základní tvar. Díky této metodice lze používat jednotlivé stemovací algoritmy.
Lze uvést několik případů slov z lexikální databáze:

- Jednotný tvar podstatných jmen: children na child
- Infinitiv sloves: understood na understand
- Přídavná jména: best na good
- Zájmena: whom na who

Stemovací algoritmy mohou být jazykovědné, automatické či mixované. Jazykovědné algoritmy využívají jazykových pravidel dle struktury jazyka. Typicky provádí manuální překlad seznamu přípon, které jsou součástí vytvořeného programu. Takovýmto nejznámějším algoritmem je Porterův algoritmus popsaný v kapitole 2.2. Tento algoritmus byl zpočátku vyvíjen primárně pro anglický jazyk. Následně se jeho použití rozšířilo do francouzštiny, italštiny a němčiny. Mezi další algoritmy patří Lovinsův popsaný v kapitole 2.1 a Paice/Husk algoritmus uvedený v kapitole 2.3.

2.1 Lovinsův algoritmus

Jedná se o jeden z prvních stemovacích algoritmů [3], který byl vyvíjen specielně pro aplikace sloužící k získávání znalostí z anglických dokumentů a představuje tedy první myšlenku o stemovacích algoritmech, založených na seznamu přípon anglických slov, např. *ed, *ing, *ses. Tento algoritmus je základem pro novější vyvíjené algoritmy, jako je Porterův, Lennonův a více obecných stemovacích algoritmů, jako jsou Hullův, Krovetzův či Harmanův algoritmus. V případě tohoto algoritmu je využito slovníkového režimu, kdy všechna pravidla jsou uložena v souboru a algoritmus testuje příponu slov na výskyt některého z pravidel či koncovek v souboru. Pokud je koncovka v souboru nalezena, je odebrána v případě, že ji to povolují pravidla, která zachovávají správnou strukturu kořene slov, např. odebrání přípony *s ze slova less by zcela změnilo kontext slova. Přidáním několika zaznamenaných pravidel lze povolit varianty slov, jako jsou ending a end.

2.1.1 Použitá pravidla

Lovinsův algoritmus obsahuje 294 koncovek slov, 29 podmínek a 35 transformačních pravidel. Každá koncovka je spojena s jednou z podmínek. V prvním kroku je nalezena nejdelší koncovka, která vyhovuje přiřazené podmínce a tato koncovka slova je odebrána. V druhém kroku je aplikováno některé z 35 transformačních pravidel na změněnou koncovku slova. Tento krok končí, pokud byla nebo nebyla koncovka odebrána v kroku prvním. Jako příklad lze uvést anglické slovo nationally, ukončené koncovkou ationally s podmínkou spojení B = minimální délka kořene slova jsou 3 znaky. V případě, že by byla odstraněna koncovka ationally, pak by kořen slova obsahoval pouze znak n, což naše podmínka B zavrhuje. Další z kratších koncovek v seznamu je tedy ionally spojené s podmínkou A. Podmínka A neomezuje kořen slova na délku, z čehož plyne, že koncovka ionally je odstraněna. Zůstává tedy pouze kořen slova nat. Základní Lovinsův algoritmus odebírá vždy maximálně jednu koncovku původního slova. Některé modifikace tohoto algoritmu procházejí seznam koncovek a transformačních pravidel v iteracích a zpřesňují tak základní Lovinsův algoritmus.

2.1.2 Seznam koncovek a podmínek

Jak jsem již zde uvedl, Lovinsův algoritmus obsahuje 294 koncovek slov (přípon) a ke každé z těchto koncovek je přiřazena jedna z 29 podmínek. Jednotlivé koncovky, které jsou ve slovech vyhledávány, jsou seřazeny od nejdelší po nejkratší. U každé koncovky je uvedena podmínka, kterou musí daná koncovka splňovat, aby slovo mohlo být transformováno. Seznam koncovek slov je uveden v příloze A této diplomové práce. Jednotlivé podmínky uvedené od A po Z a AA, BB, CC, které platí pro anglické přípony slov, jsou znázorněny v následující tabulce 2.1. Nejčastěji používanou podmínkou je podmínka A. V případě této podmínky se považují za délku kořene slova minimálně dva anglické znaky. Druhou nejčastěji využívanou podmínkou je podmínka B, kde minimální délka kořene slova po odebrání koncovky jsou tři znaky.

Podmínky	Pravidla
A	Bez omezení stemování
B	Minimální délka kořene slova = 3
C	Minimální délka kořene slova = 4
D	Minimální délka kořene slova = 5
E	Zákaz odebrání přípony po znaku e
F	Minimální délka kořene slova = 3 a neodebírat koncovku po znaku e
G	Minimální délka kořene slova = 3 a odebrat koncovku pouze po znaku f
H	Odebrání přípony slova pouze po znaku t nebo ll
I	Zákaz odebrání přípony po znaku e nebo o
J	Zákaz odebrání přípony po znaku e nebo a
K	Minimální délka kořene slova = 3 a pouze po znacích l, i nebo u*e
L	Zákaz odebrání přípony po znaku u, x nebo s, pokud s není po o
M	Zákaz odebrání přípony po znaku a, c, e nebo m
N	Minimální délka kořene slova = 4 po s**, jinde 3
O	Odebrání koncovky pouze po l nebo i
P	Zákaz odebrání přípony po znaku c
Q	Minimální délka kořene slova = 3 a zákaz odebrání přípony po znaku l nebo n
R	Odebrání přípony slova pouze po znaku n nebo r
S	Odebrání přípony slova pouze po znaku dr nebo t, pokud t není po t
T	Odebrání přípony slova pouze po znaku s nebo t, pokud t není po o
U	Odebrání přípony slova pouze po znaku l, m, n nebo r
V	Odebrání přípony slova pouze po znaku c
W	Zákaz odebrání přípony po znaku s nebo u
X	Odebrání přípony slova pouze po znaku l, i, nebo u*e
Y	Odebrání přípony slova pouze po in
Z	Zákaz odebrání přípony po znaku f
AA	Odebrání přípony slova pouze po d, f, ph, th, l, er, or, es nebo t
BB	Minimální délka kořene slova = 3 a zákaz odebrání přípony po met nebo ryst
CC	Odebrání přípony slova pouze po l

Tabulka 2.1 – Seznam podmínek pro Lovinsův algoritmus.

2.1.3 Transformační pravidla

V následující tabulce 2.2 této kapitoly je uvedeno všech 35 anglických transformačních pravidel, používaných v Lovinsově algoritmu, s jednoduchou ukázkou v anglickém jazyce, která pro každé z nich platí. Tabulka má za úkol znázornit, jak jednotlivá pravidla v druhém kroku Lovinsova algoritmu pracují při změně koncovky. Prvním použitým pravidlem v druhém kroku algoritmu je odstranění jednoho znaku ze dvou shodných znaků na konci stemovaného slova. Další pravidla už jsou zaměřena primárně na úpravu stávajících koncovek slova, tak jak je ukázáno v příkladě u aplikovaného pravidla. Znaky v příkladech uvedené ve složených závorkách jsou odstraněny v prvním kroku Lovinsova algoritmu.

Pravidlo	*Koncovka slova před transformací*	→	*Koncovka slova po transformaci*	*Příklad: Před*		*Po*
1	odstranění jednoho ze dvou znaků b, d, g, l, m, n, p, r, s, t			embedd[ed]	→	embed
2	iev	→	ief	believ[e]	→	belief
3	uct	→	uc	induct[ion]	→	induc[e]
4	umpt	→	um	consumpt[ion]	→	consum[e]
5	rpt	→	rb	absorpt[ion]	→	absorb
6	urs	→	ur	recurs[ive]	→	recur
7	istr	→	ister	administr[ate]	→	administ[er]
7a	metr	→	meter	parametr[ic]	→	paramet[er]
8	olv	→	olut	dissolv[ed]	→	dissolut[ion]
9	ul → l kromě následníka znaku a, o, i			angul[ar]	→	angl[e]
10	bex	→	bic	vibex	→	vibic[es]
11	dex	→	dic	index	→	indic[es]
12	pex	→	pic	apex	→	apis[es]
13	tex	→	tic	cortex	→	cortic[al]
14	ax	→	ac	anthrax	→	anthrac[ite]
15	ex	→	ec		→	
16	ix	→	ic	matrix	→	matric[es]
17	lux	→	luc		→	
18	uad	→	uas	persuad[e]	→	persuas[ion]
19	vad	→	vas	evad[e]	→	evas[ion]
20	cid	→	cis	decid[e]	→	decis[ion]
21	lid	→	lis	elid[e]	→	elis[ion]
22	erid	→	eris	derid[e]	→	deris[ion]
23	pand	→	pans	expand	→	expans[ion]
24	end → ens kromě následníka znaku s			defend	→	defens[ive]
25	ond	→	ons	respond	→	respons[ive]
26	lud	→	lus	collud[e]	→	collus[ion]
27	rud	→	rus	obtrud[e]	→	obtrus[ion]
28	her → hes kromě následníka znaku p, t			adher[e]	→	adhes[ion]
29	mit	→	mis	remit	→	remis[s][ion]
30	ent → ens kromě následníka znaku m			extent	→	extens[ion]
31	ert	→	ers	convert[ed]	→	convers[ion]
32	et → es kromě následníka znaku n			parenthet[ic]	→	parenthes[is]
33	yt	→	ys	analyt[ic]	→	analys[is]
34	yz	→	ys	analyz[ed]		analys[ed]

Tabulka 2.2 - Seznam transformačních pravidel Lovinsova algoritmu a ukázka příkladů.

2.2 Porterův algoritmus

Porterův algoritmus [4] se odlišuje od Lovinsova algoritmu ve dvou hlavních věcech. Prvním rozdílem je významná redukce složitosti jednotlivých pravidel, která se využívají pro mazání přípon slov a výrazné snížení časové náročnosti procházení přípon a pravidel. U Lovinsova algoritmu bylo třeba využít 294 přípon, kde každá byla spojena s jedním z 29 pravidel, která určovala, zda daná přípona může být ze slova odstraněna nebo ji odstranit nelze. Mimo jiné tento algoritmus obsahoval také 35 transformačních pravidel. Přestože Lovinsův algoritmus obsahoval velké množství přípon, relativně v malém množství tvarů a zapisovacích pravidel, byl navržen tak, aby dokázal zpracovat především vědecké texty. Druhým rozdílem je využití unifikovaného přístupu při zachování kontextu. Mnoho z Lovinsových kontextově závislých pravidel souviselo především s délkou kořenu slov po aplikaci pravidla. Minimální přijatelná délka slova po aplikaci Lovinsova algoritmu jsou dva znaky. U Porterova algoritmu je využito principu popsaného v následující kapitole 2.2.1, založeného především na zjišťování posloupnosti souhlásek a samohlásek ve stemovaných slovech. Existuje také několik různých verzí Porterova algoritmu, které se od původní verze liší pouze v několika upravených specifikacích, např. Snowball Porterův algoritmus.

2.2.1 Použitá pravidla

Porterův algoritmus pro svoji činnost využívá přibližně 60 přípon slov, dvě přepisovací pravidla a jeden druh kontextově závislého pravidla, které uvádí, zda daná přípona slova může být odstraněna. Kontextově závislá pravidla pracují ve většině případů na principu zjištění počtu kombinace znaků (samohláska – souhláska) v řetězci zbylém po odebrání nalezené přípony slova.

Samohláska (Vowel) je jedna ze znaků A, E, I, O, U a znak Y, kterému musí předcházet souhláska (Consonant), jinak se jedná o souhlásku. Např. slovo TOY má souhlásky T a Y, slovo FYLY má souhlásky F a L. V následující části textu bude souhláska označována znakem C a samohláska znakem V. Seznam souhlásek následujících ve slově ihned za sebou o délce větší než 1 (např. CCC), bude označovat znak C a samohlásky (např. VVV) pouze znak V. Každé slovo může být reprezentováno kombinací [C]VCVC…[V], kde C a V v závorkách je volitelné a VC se může libovolně krát opakovat. Toto opakování VC lze zapsat za použití *m*, kde *m* je počet opakování posloupnosti VC ve slově. Tuto posloupnost lze zapsat také jako [C](VC)*m*… [V], kde posloupnost VC se opakuje *m*-krát.
Uvedu zde několik příkladů měření *m* na různých slovech:

m = 0 : platí např. pro slova B, CC, BR, HA, …

m = 1 : platí např. pro slova DOG, FALSE, …

m = 2 : platí např. pro slova RELATE, VALENCE, FORMAL, …

m > 2 : platí např. pro slova VIETNAMIZE, DIFFERENT, HOPEFUL, …

Kontextově závislá pravidla, která platí pro odebrání přípony slova se zapisují ve tvaru (PODMÍNKA) S1 → S2. Pokud koncovka slova souhlasí s výrazem S1 a je splněna podmínka uvedená v závorce, pak se výraz S1 ve slově zamění za výraz S2. Podmínka je typicky složena z proměnné *m*, tedy počtu opakování VC. Například (m ≥ 2) EMENT → NULL, nahradí koncovku EMENT za prázdný řetězec. REPLACEMENT přepíše na REPLAC, pokud tento výraz splňuje podmínku *m* ≥ 2, což platí ve slově REPLAC pro posloupnost znaků EP a AC = 2 x *m*. Podmínky mohou být vnořeny a spojeny do složitějších za použití booleovských podmínek AND, OR a NOT. Tyto výrazy lze uzávorkovat a zpracovat dle jejich pořadí ve výrazu. Např. výraz (m=1 and (*X or *Z)) znamená, že počet posloupností VC = 1 a zároveň kořen slova nesmí končit znakem X nebo Z.

Pokud více podmínek na straně S1 odpovídá příponě slova, pak je vybírána vždy ta s nejdelší shodou v příponě. Např. Příponu SSES lze podle pravidel algoritmu přepsat na SS nebo S → NULL a podmínky zde nejsou určeny. Pro slovo CARESSES tedy bude platit přípona SSES, která je nejdelší ze všech, která souhlasí, tedy vznikne slovo CARESS. Slovo CARESS se namapuje na CARESS (S1 = SS) a CARES na CARE (S1 = znak S).

Další kontextově závislé podmínky, které se mohou objevit u tohoto algoritmu v závorkách:

a) *C = poslední písmeno kořene slova je znak 'c'.
b) *v* = kořen slova obsahuje samohlásku.
c) *d = kořen slova končí posloupností dvou shodných souhlásek, např. –DD, -EE, …
d) *o = kořen slova končí posloupností CVC, kde druhé C není znak W, X nebo Y, např. –DOG.

2.2.2 Popis algoritmu

Popis jednotlivých kroků Porterova algoritmu:

1a.: Krok, kde participují koncovky jednotlivých slov. Tento krok pracuje s množným číslem slova a minulým časem slov.

SSES → SS	SS → SS
IES → I	S → NULL

1b.: Koncovky, které jsou nutné pro odstranění specifikace podmínek.

(m > 0) EED → EE
(*v*) ED → NULL
(*v*) ING → NULL

Pokud je splněna druhá či třetí podmínka v kroku 1b, pak se aplikují tato další přepisovací pravidla:
AT → ATE, např. conflat(ed) → conflate
BL → BLE, např. troubl(ed) → trouble
IZ → IZE, např. siz(ed) → size

Podmínka, kdy se pravidlo namapuje na jedno písmeno a způsobí odebrání jednoho ze dvou.
(*d and not (*L or *S or *Z)) → např. hopp(ing) → hop, ale fall(ing) → fall

Podmínka pro slova, která mají jednu posloupnost VC a zároveň končí posloupností CVC.
(m = 1 and *o) → E, mapř. fil(ing) → file, ale fail(ing) → fail

1c.: Krok změkčování y na i.

(*v*) Y → I, např. happy → happi, ale sky ponechá

2.: Krok nahrazující konkrétní koncovky za splnění příslušné podmínky. Vyhledávání výrazu S1 ve slově lze efektivně provést vyhledáním předposledního písmene. Jednotlivá pravidla jsou v tomto kroku seřazena podle předposledního písmene výrazu S1. Tento postup lze uplatnit v dalších krocích. Musí zde být také splněna podmínka m > 0, tedy počet posloupností VC musí být větší než 0.

(m > 0) ATIONAL → ATE	(m > 0) IZATION → IZE
(m > 0) TIONAL → TION	(m > 0) ATION → ATE

(m > 0) ENCI → ENCE
(m > 0) ANCI → ANCE
(m > 0) IZER → IZE
(m > 0) ABLI → ABLE
(m > 0) ALLI → AL
(m > 0) ENTLI → ENT
(m > 0) ELI → E
(m > 0) OUSLI → OUS
(m > 0) ATOR → ATE
(m > 0) ALISM → AL
(m > 0) IVENESS → IVE
(m > 0) FULNESS → FUL
(m > 0) OUSNESS → OUS
(m > 0) ALITI → AL
(m > 0) IVITI → IVE
(m > 0) BILITI → BLE

3.:
(m > 0) ICATE → IC
(m > 0) ATIVE → NULL
(m > 0) ALIZE → AL
(m > 0) ICITI → IC
(m > 0) ICAL → IC
(m > 0) FUL → NULL
(m > 0) NESS → NULL

4.: V tomto kroku odebereme poslední koncovky, které lze ze slov odebrat.
(m > 1) AL → NULL
(m > 1) ANCE → NULL
(m > 1) ENCE → NULL
(m > 1) ER → NULL
(m > 1) IC → NULL
(m > 1) ABLE → NULL
(m > 1) IBLE → NULL
(m > 1) ANT → NULL
(m > 1) EMENT → NULL
(m > 1) MENT → NULL
(m > 1) ENT → NULL
(m > 1 and (*S or *T)) ION → NULL
(m > 1) OU → NULL
(m > 1) ISM → NULL
(m > 1) ATE → NULL
(m > 1) ITI → NULL
(m > 1) OUS → NULL
(m > 1) IVE → NULL
(m > 1) IZE → NULL

5.: V tomto kroku se již provádějí pouze úpravy jednotlivých slov.
(m > 1) E → NULL, např. probate na probat
(m = 1 and not *o) E → NULL, např. cease na ceas
(m > 1 and *d and *L) → jedno písmeno ze zdvojeného, např. controll na kontrol

2.3 Paice/Husk algoritmus

Tento algoritmus vznikl v roce 1977. Základní schéma tohoto stemovacího algoritmu bylo nastíněno v dokumentu [6] s přispěním znalostí Garetha Huska při testování a implementaci tohoto algoritmu. Tento stemovací algoritmus využívá a vyvíjí především univerzita v Lancasteru. Tento algoritmus nebyl nikdy formálně specifikován, avšak se ukázalo, že pracuje dostatečně efektivně.

Paice/Husk algoritmus pracuje v iteracích a pro svoji činnost využívá pouze jednu tabulku pravidel. Každé pravidlo může specifikovat jedno smazání či nahrazení koncovky slova. Tento algoritmus pro svoji činnost nepotřebuje žádná transformační pravidla, jak tomu bylo u Lovinsova algoritmu. Pravidla jsou seskupena do skupin, které odpovídají koncovému písmenu přípony slova. Na základě tohoto seskupení lze tato pravidla procházet velice rychle za použití postupného prohledávání tabulky na základě posledního písmene aktuálního slova či jeho zkrácené verze. Některá pravidla jsou však omezena pouze na celá slova, kterým nebyla odebrána či upravena koncovka. Po použití pravidla může být algoritmus přerušen nebo může pokračovat iterativně dál v činnosti.

2.3.1 Použitá pravidla

Ukázka jednotlivých pravidel algoritmu je zobrazena v příloze B tohoto textu. Na každém řádku je znázorněno pravidlo tak, jak pracuje a je zde ke každému pravidlu uveden popis jeho změn. Pravidla mají tvar uzpůsoben tak, aby mohla být jednoduše využita pro stemming. Seznam obsahuje celkem 116 pravidel. Je důležité, aby zůstalo zachováno pořadí pravidel, které je uvedeno v příloze. Každé pravidlo má 5 částí, přičemž 2 z nich jsou volitelné.
Jednotlivé části pravidel jsou tyto:

1. **část** = koncovka slova uvedená v obráceném pořadí písmen od konce slova.
2. **část** = volitelně je zde příznak '*' (slovo nesmí být předem změněno použitím některého z dalších pravidel). V příloze B je toto pravidlo znázorněno políčkem –beze změn.
3. **část** = číslo reprezentující počet odebraných znaků od konce slova (může být nula, pokud se žádný znak nebude ze slova odebírat).
4. **část** = volitelně je zde přípona, která má být nahrazena za odstraněnou. Tato koncovka slova může obsahovat jeden či více znaků.
5. **část** = symbol '>' pro další iteraci či znak '. ' pro ukončení změn slova.

Několik názorných příkladů, jak pravidla uvedená v příloze B pracují.

1. **příklad:** *ai*2.* = pravidlo, které říká, že pokud slovo končí koncovkou –ia a nebylo změněno, pak odebere poslední dva znaky slova a ukončí činnost.
2. **příklad:** *gni3>* = pokud slovo končí koncovkou –ing, pak odebereme poslední 3 znaky, např. slovo skiing na ski a pokračuje další iterací, kdy i nahradí za y, tedy ski na sky (pravidlo i1y).
3. **příklad:** *ju1d.* = pokud slovo končí koncovkou –uj, pak se tato koncovka změní na ud. Odebere se poslední znak a místo něj se do slova vloží znak d.
4. **příklad:** *nee0.* = pokud slovo končí koncovkou –een, pak toto slovo zůstane beze změn.
5. **příklad:** *vis3j>* = pokud slovo končí koncovkou –siv, pak odstraníme 3 znaky z konce slova a přidáme nakonec slova znak j. Po provedení změny pokračujeme od začátku další iterací vyhledávání koncovky.

Dodatečná pravidla pro Paice/Husk stemovací algoritmus zaručují, aby výsledek kořene slova po aplikaci stemmingu nebyl příliš krátký, např. jeden znak. Slova red, rent, river a jiná, by podle pravidel, která jsou uvedena v příloze B tohoto textu, obsahovala po aplikaci pouze znak r. Vznikla tedy dodatečná pravidla, která upravují možnosti použití příslušných pravidel.

1. **pravidlo:** Pokud anglické slovo začíná jednou ze samohlásek, pak po aplikaci pravidla musí mít kořen alespoň 2 znaky, jinak pravidlo není možné použít. Např. na slovo red nelze aplikovat pravidlo odebrání koncovky –ed. Na slovo owed lze toto pravidlo použít, z čehož vznikne kořen ow.
2. **pravidlo:** Pokud anglické slovo začíná jednou ze souhlásek, pak po aplikaci pravidla musí mít kořen nejméně 3 znaky, jinak pravidlo není možné použít a zároveň uvnitř kořene slova musí být nejméně jedna samohláska nebo znak y. Např. slovo crying lze převést na cry, ale nelze převést slovo string na slovo str.

Tyto podmínky si však neumí poradit s krátkými kořeny slov, jako jsou doing, going, being atd. a k úpravě těchto slov by bylo nutné využití např. lexikálního vyhledávání nebo jiné stemovací metody.

2.3.2 Popis algoritmu

Činnost stemovacího algoritmu Paice/Husk lze rozdělit do následujících 4 kroků:

1. **krok:** *Vyhledání příslušné sekce v pravidlech.* Pokud poslednímu znaku ve slově neodpovídá žádná ze sekcí v pravidlech, pak činnost přerušíme. Pokud souhlasí, vybereme první pravidlo z příslušné sekce.
2. **krok:** *Kontrola, zda můžeme pravidlo použít.* Nejprve zkontrolujeme, zda odpovídají znaky z konce slova se znaky pravidla, které jsou uloženy v obráceném pořadí (od konce). Pokud koncovka slova nesouhlasí přejdeme na krok 4. Pokud souhlasí přípona slova s pravidlem a je nastavena maska (příznak) pro slovo beze změn a slovo již bylo změněno, pak přejdeme také na krok 4. Pokud není splněna některá z dodatečných podmínek daného pravidla, pak přejdeme opět na krok 4.
3. **krok:** *Aplikace pravidla.* Nejprve odstraníme z pravé strany stemovaného slova počet znaků, který odpovídá obsahu ve vyhledaném pravidle. Pokud pravidlo obsahuje znaky, které máme přidat, přidáme je. Pokud znakem na konci pravidla je ’.’ , pak přerušíme činnost. Pokud je posledním znakem pravidla znak ’>’, pak přejdeme na krok 1.
4. **krok:** *Hledání dalšího pravidla.* Přejdeme na další pravidlo v seznamu. Pokud se změnilo písmeno sekce, pak přerušíme činnost, jinak přejdeme na krok 2.

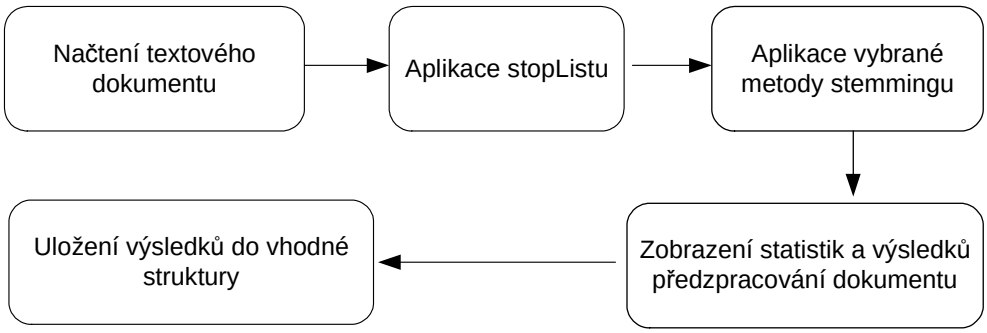
3 Analýza a návrh aplikace

Cílem této kapitoly analýzy a návrhu aplikace je ukázat, jak bude následně celá aplikace implementována a co bude jejím výstupem pro další experimenty. Je zde proto provedena textová specifikace aplikace a navržena architektura. Celá funkční aplikace pro předzpracování textu bude implementovat a využívat stemovací algoritmy popsané v předchozí kapitole. Pro návrh a tvorbu aplikace jsem zvolil architekturu MVC (Model-View-Controller).

3.1 Specifikace aplikace

Základem aplikace bude načtení textového dokumentu či celé kolekce anglických dokumentů. Po načtení obsahu textového dokumentu bude následovat fáze nastavení aplikace. V této fázi bude možné nastavit seznam slov, tzv. stop list. Uživatel bude mít možnost výběru konkrétního stop listu či zakázání použití tohoto seznamu slov. V případě, že nebude aplikován, pak se do výsledku uloží všechna slova z původního dokumentu. Uživatel bude mít dále možnost využití implicitně nastaveného základního seznamu těchto slov bez nutnosti jeho definování přímo v aplikaci. Dalším nastavením bude výběr konkrétního stemovacího algoritmu ze tří implementovaných algoritmů. Jednotlivá pravidla algoritmu budou typicky načtena z odděleného souboru pro přehlednost zdrojového kódu.

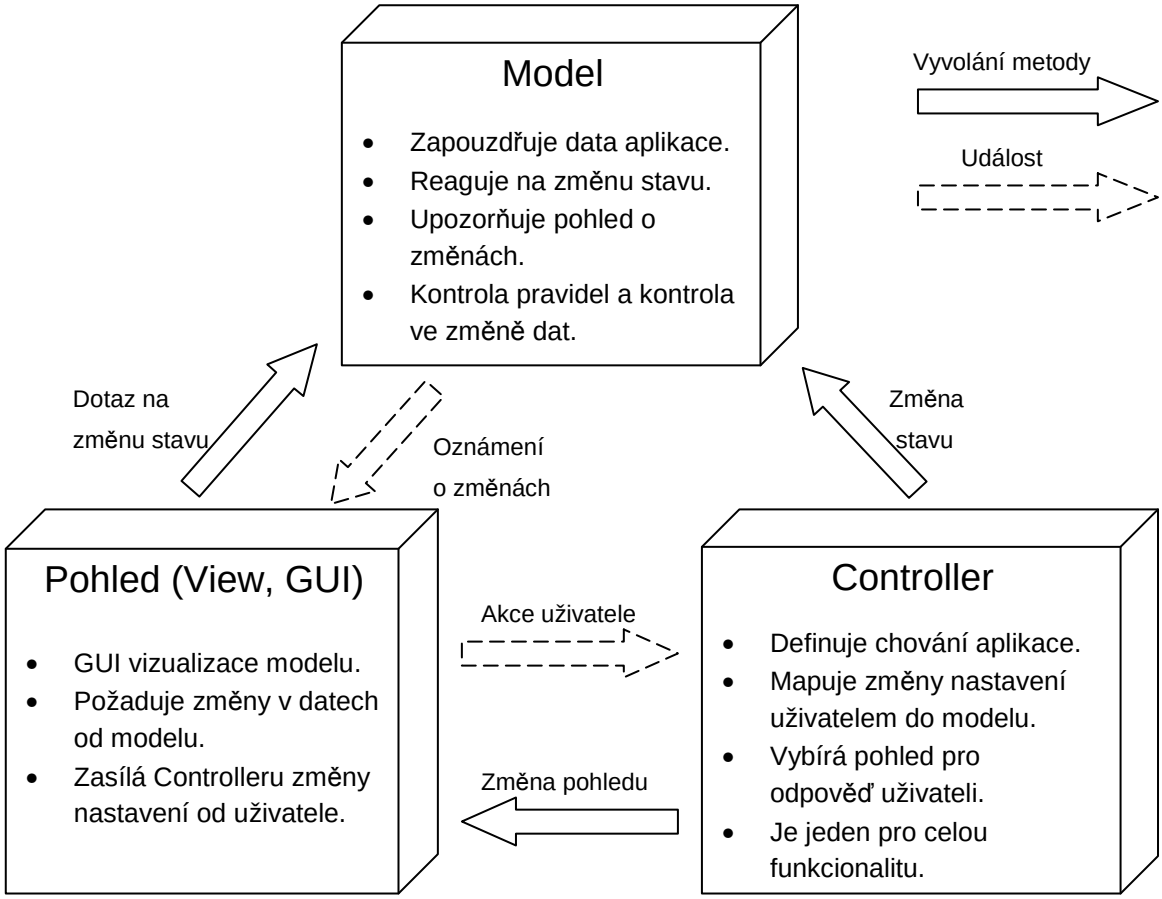
Při stemmingu slova vždy dojde k vyhledání tohoto slova v seznamu stop slov (pokud je nastaven) a pokud nebude slovo nalezeno v tomto seznamu, pak spustí svoji činnost stemovací algoritmus. Algoritmus aplikuje příslušná pravidla a výsledkem bude seznam kořenů slov, vzniklých po stemmingu. Postup činnosti aplikace bude zaznamenán do speciálního logovacího souboru. Součástí aplikace bude také možnost výběru tvaru uložení kořenů slov z dokumentů. Aplikace nabídne dvě možnosti uložení výsledku. Výsledek po provedení stemovacího algoritmu bude možné uložit do nového textového dokumentu. Jednotlivé kořeny slov budou uloženy za sebou v pořadí průchodu textem. Druhou variantou uložení kořenů slov, kterou bude možné v aplikaci vybrat, je možnost uložení výsledku ve speciální datové struktuře pro klasifikaci dokumentů ve specializovaných nástrojích pro text mining. Celý postup uvedený pro jeden anglický textový dokument bude aplikován taktéž na ostatní vybrané textové dokumenty. Ukázka činnosti aplikace je zobrazena na obrázku 3.1.



Obrázek 3.1 – Ukázka analyzované činnosti aplikace.

3.2 MVC architektura aplikace

Implementovaná aplikace bude pracovat na novější (upravené) verzi MVC [9] architektury, která vychází z klasické MVC architektury znázorněné na obrázku 3.2. Tato architektura dostala zkratku ze slov Model-View-Controller. Tato architektura vznikla v roce 1979 při výzkumu programovacího jazyka Smalltalk. Architektura MVC se používá v GUI aplikacích vstup-zpracování-výstup, ale často je využívána také u webových aplikací. MVC se rozšířila především v jazyce Java, ale je možné ji využít v jiných programovacích jazycích. Úkolem tohoto přístupu je oddělení logiky programu (modelu) od řídící logiky (controlleru) a uživatelského rozhraní (view) aplikace.



Obrázek 3.2 – Diagram činnosti architektury MVC.

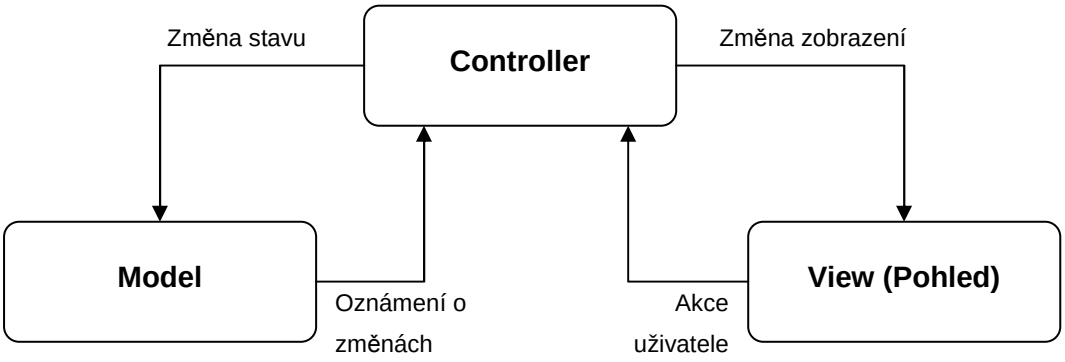
MVC [18] architektura je dělena do tří základních elementů:

1. **Model:** Reprezentuje data, stav aplikace a pravidla. Pravidla v modelu řídí přístup, úpravu, uložení dat a stavů aplikace. Model zajišťuje funkčnost aplikace a může být implementován pomocí objektových tříd nebo také za pomoci klasických funkcí. Navenek je jeho funkčnost zapouzdřena a s pohledem nebo controllerem komunikuje pouze přes své pevně definované rozhraní.
2. **Pohled (view):** Zastřešuje grafické uživatelské rozhraní, přes které dochází ke komunikaci s koncovým uživatelem aplikace. Zahrnuje grafické elementy jako jsou: textová pole, rozbalovací pole, tlačítka, tabulky a jiné grafické prvky. Pohled zachytává vstupní události od uživatele a následně je přeposílá controlleru, který vyvolává odpovídající metody modelu tak, aby byla zajištěna správná funkčnost celé aplikace. Pohled dále reprezentuje model dat pro

jejich zobrazení uživateli. Pokud se model dat změní, pak musí dojít k interakci s pohledem tak, aby na displeji uživatele byla zobrazena korektní data. Pohled zajišťuje průběžné překreslování a aktualizaci jednotlivých oken aplikace. Změny v modelu lze pro jejich zobrazení získat dvěma způsoby. Prvním způsobem je aktualizace pohledu v případě, že si od modelu data sám vyžádá. Druhým způsobem je registrace pohledu u modelu, který následně posílá příslušnému pohledu informace o změnách modelu. Druhého přístupu se v aplikacích využívající MVC architektury používá častěji.

3. **Controller:** Controller odchytává akce uživatele, jako jsou pohyby myší, stisky kláves, výběr z menu a po zpracování je zasílá modelu. Na základě zpracování těchto událostí volá příslušné metody modelu a mění jeho stav. V závislosti na přijaté akci od uživatele může také vybírat příslušný pohled, který bude odchytávat oznámení o změnách modelu a tyto změny následně korektně zobrazovat v uživatelském okně aplikace.

Implementovaná aplikace v programovacím jazyce Java bude mít tedy tři hlavní balíčky nazvané Model, View a Controller. Tyto balíčky budou obsahovat jednotlivé třídy nutné pro správnou činnost celé implementované aplikace. Jednotlivé pohledy (view) aplikace budou implementovány s využitím grafické knihovny Swing. Diagram upravené verze architektury MVC [9], na které bude provedena implementace aplikace pro předzpracování textu je znázorněn na obrázku 3.3. Tato architektura má oproti klasické MVC architektuře zobrazené na obrázku 3.2 několik základních rozdílů. Hlavním rozdílem je umístění controlleru, který od sebe zcela odděluje pohled a model architektury. Controller zabezpečuje detailní správu mezi pohledem a modelem. V této novější verzi architektury MVC není možné, aby model obsahoval přímý odkaz na pohled. Stejně tak naopak pohled nesmí přímo měnit stav modelu. Všechny pohledy a modely použité v aplikaci musí být registrovány pro jediný centrální controller, který má na starost řízení celé aplikace. Controller má na starost odchytávání akcí od uživatele a změnu stavu modelu stejně jako u klasického MVC. Nově controller odchytává oznámení o změnách modelu a zajišťuje překreslení příslušného pohledu, např. zobrazení výsledků v novém pohledu. Hlavní výhodou této modifikované MVC architektury je snadná možnost přidávání nových pohledů a modelů v aplikaci bez nutnosti provádění výrazných změn ve zdrojových souborech. Stačí pouhá registrace pohledu nebo modelu u hlavního controlleru.



Obrázek 3.3 – Diagram činnosti upravené verze MVC architektury.

Mezi hlavní výhody architektury MVC patří:

- Snadná rozšiřitelnost aplikace, znuvupoužitelnost kódu, modularita.
- Možnost změny pohledu, controlleru nebo modelu bez nutnosti nové implementace celé aplikace. Např. pro nový pohled není třeba implementovat celou aplikační logiku.
- Jednotlivé části aplikace lze vyvíjet odděleně.

4 Implementace aplikace

Zvoleným programovacím jazykem pro implementaci stemovacích algoritmů a celé aplikace je programovací jazyk Java. Aby bylo možné vytvořit graficky přívětivé uživatelské rozhraní, využívá tato aplikace knihovny Swing. Výhodou takto implementované aplikace je její multiplatformnost. Rozdělení zdrojových souborů do balíčků a tříd je založeno na navržené MVC architektuře z předchozí kapitoly. Pro testování jednotlivých stemovacích algoritmů jsem využil testovacích dokumentů skupiny Reuters a setříděných anglických textových dokumentů. Jako vstupní data lze tedy využít soubory s koncovkou .txt nebo .sgm nebo celou složku obsahující jeden typ těchto souborů. Celá aplikace byla vyvíjena ve vývojovém prostředí Netbeans. Aby bylo možné provést kompilaci a spouštění programu přes příkazovou řádku nebo ručně přes soubor .jar, byl pro kompilaci programu využit nástroj ANT.

4.1 Java a Swing GUI

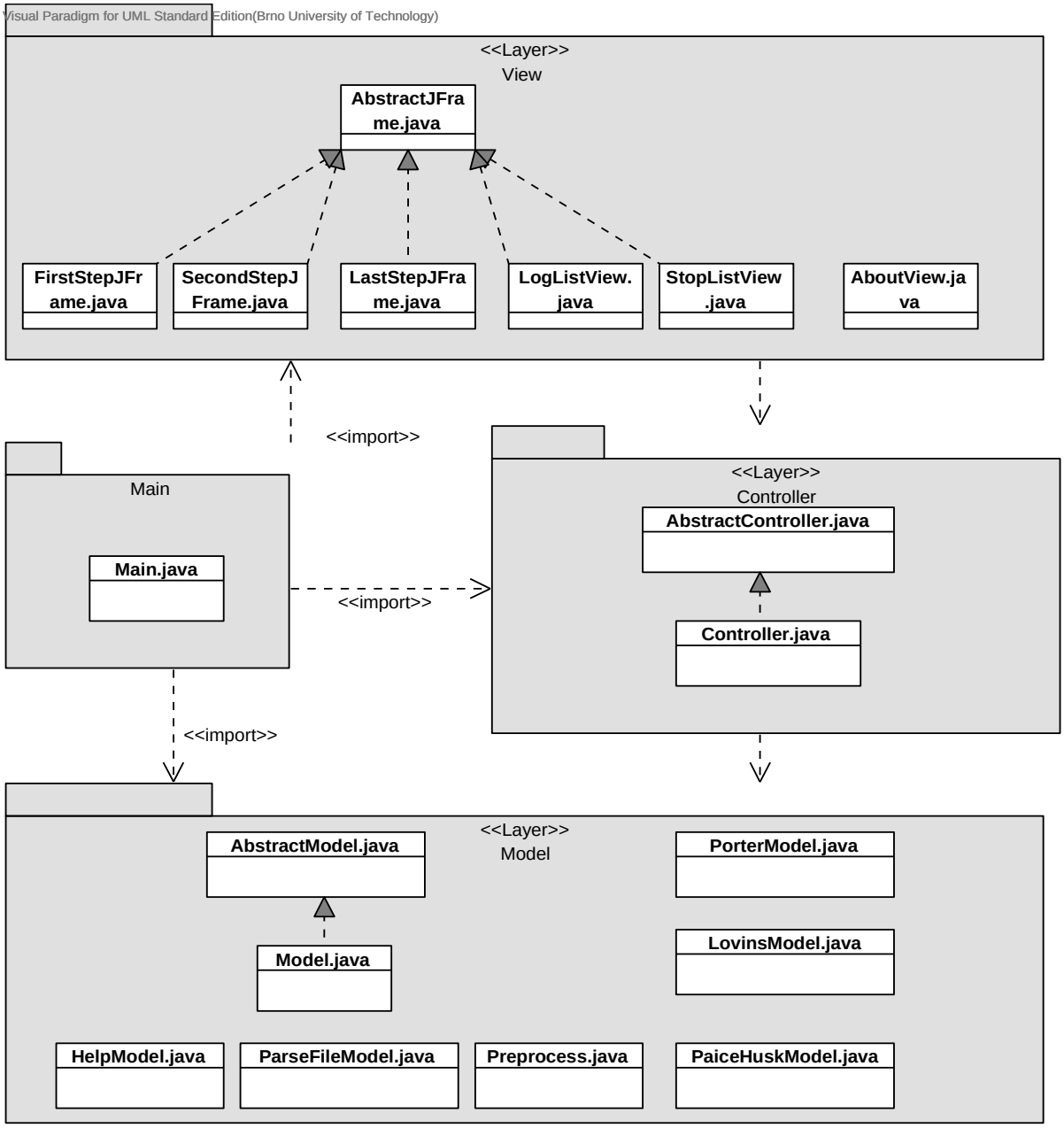
Celá aplikace je implementována na platformě Java. Java [8] je objektově orientovaný programovací jazyk od společnosti Sun Microsystems. Jeho syntaxe je zjednodušenou verzí jazyka C a C++. Jeho hlavní výhodou je jeho platformní nezávislost na architektuře počítače nebo OS. Jedná se o interpretovaný programovací jazyk. Nevytváří se tedy skutečný strojový kód ale pouze tzv. mezikód (bajtkód). Jediné co Java vyžaduje, je existence interpretu Java Virtual Machine (JVM).

Swing GUI [7] je nadstavbou nad klasickou verzí Javy. Jedná se o knihovnu pro zobrazení uživatelských prvků na platformě Java přes grafické rozhraní počítače. Swing nám nabízí aplikační rozhraní pro tvorbu a obsluhu klasického grafického uživatelského rozhraní. Pomocí této knihovny lze vytvářet okna, tlačítka, dialogy, formuláře a jiné grafické prvky. Jeho předchůdcem byl nástroj zvaný Abstrakt Window Toolkit (AWT). Tento nástroj však obsahoval chyby a především byl platformě závislý. Z tohoto důvodu se začalo převážně využívat knihovny Swing. Tato knihovna je rozšířením původní grafické knihovny v Javě.

Knihovna Swingu je založena na hierarchickém uspořádání tříd. Jako každá třída v Javě je i třída grafického uživatelského rozhraní potomkem třídy Object. Základním prvkem knihovny Swing je třída Component, která obsahuje všechny objekty, které mohou být zobrazeny na monitoru a integrovat s uživatelem. Hlavní výhodou této knihovny je opět její platformní nezávislost na použitém OS, což výrazně vylepšuje její přenositelnost. Swing přináší modernější vzhled aplikace, než kterou využívalo AWT a nové grafické komponenty.

4.2 Popis tříd aplikace

Celá aplikace byla implementována a rozdělena do balíčků a tříd tak, aby rozdělení odpovídalo MVC architektuře a aplikace byla snadno rozšiřitelná o další stemovací metody, další uživatelská okna či moduly. Diagram balíčků aplikace, který vychází z MVC architektury je znázorněn na obrázku 4.1. Celá aplikace je rozdělena do vrstev obsahujících jednotlivé třídy aplikace, které spolu mohou vzájemně komunikovat, využívat tříd jiné vrstvy a předávat si data potřebná pro správný chod stemovacích algoritmů a celé aplikace. Z diagramu je zřejmé, že neexistuje přímá vazba mezi vrstvou View (pohled) a vrstvou Model. Komunikace mezi těmito vrstvami je zajištěna pomocí vrstvy Controller. Popis jednotlivých balíčků a tříd aplikace je proveden v následující části textu.



Obrázek 4.1 – Diagram balíčků a tříd implementované aplikace.

4.2.1 Registrace MVC modulů

Pro registraci jednotlivých komponent výsledné MVC aplikace jsem vytvořil balíček dp.mvc.main. Tento balíček obsahuje vstupní bod do programu ve třídě *Main.java*. Po kompilaci a spuštění programu dochází v této třídě k registraci všech pohledů pro controller, který tak má možnost odchytávat jednotlivé události v registrovaných pohledech. Stejně tak dochází k registraci hlavního modelu pro controller tak, aby mohl naopak controller odchytávat změny v modelu a odesílat je zpět příslušnému pohledu, který reaguje na událost překreslením a změnou zobrazení pro uživatele. Tato skutečnost je znázorněna na obrázku 4.1 použitím prvku import. Součástí třídy Main je také inicializace hlavního modelu, tedy dat, která musí být nastavena při spuštění programu. Jedná se o načtení implicitního stop listu ze souboru *stopList.txt* a implementovaných stemovacích metod pro jejich výběr. Implicitní stop list aplikace je zobrazen v kapitole 1.3. Dále je zde vytvořena složka pro dočasné soubory, logovací soubor a následně je provedeno spuštění hlavního okna aplikace.

4.2.2 Model

Model je funkčním a datovým základem aplikace. Balíček dp.mvc.model má za úkol načtení a uložení dat z anglických dokumentů a obsahuje datové struktury pro jednotlivá data potřebná pro aplikaci. Změna dat v pohledu je odchycena controllerem, který vyvolá příslušnou metodu *set* v třídě *Model.java*. Hlavní třída Model je zaregistrována ve třídě Main. Tato třída je zděděna od třídy *AbstractModel.java* a je znázorněna na obrázku 4.1. pomocí šipky (dědičnosti). Třída Model obsahuje především korektní načtení anglických textových dokumentů, spouštění odpovídající nastavené stemovací metody, správu aplikace, dočasné složky tmp, logovacího souboru a také uložení výsledků po provedení stemmingu. Mezi další činnosti modelu patří kontrola zadaných údajů. Model musí tedy např. kontrolovat, zda před zahájením předzpracování dat byl vybrán textový dokument pro stemming a metoda, která bude použita. Dále má na starost vyvolání události pro překreslení pohledu po aktualizaci dat. Tuto událost o změně modelu odchytí controller a vyvolá metody pro překreslení příslušného pohledu aplikace. Tato událost je vyvolána metodou *firePropertyChange()* s příslušnými parametry. Hlavní model pro svoji činnost využívá několika dalších níže popsaných tříd uložených ve stejném balíčku pro rozdělení činnosti do menších logických celků tak, aby bylo zajištěno uchování nastavených dat a nastavení zvolených uživatelem.

Třída *HelpModel.java* je využívána pro uložení načtených dat do pomocné složky tmp, která slouží pro urychlení činnosti všech algoritmů a celé aplikace. Načtená data jsou ukládána do pomocných souborů v této složce a ty jsou následně v aplikaci zpracovány. Další využitím této třídy je pro filtraci článků při ukládání souborů pro klasifikaci. Cílem této třídy je zapouzdřit práci s pomocnými soubory a filtrací dokumentů. Více o těchto funkcích aplikace je popsáno v následující části tohoto textu v kapitole nazvané Činnost aplikace a Výstupní data aplikace.

Třída *ParseFileModel.java* má na starost zpracování načtených souborů pro stemming a uložení do příslušných datových struktur pro následné zpracování aplikací. Z hlavního modelu je rozlišeno zda se jedná o stemming pouze jednoho souboru nebo celé složky. Dále musí být rozlišeno zda se jedná o Reuters dokumenty nebo obyčejné textové dokumenty. Podle tohoto nastavení je provedeno načtení jednotlivých slov do vektoru tak, aby mohla aplikace s těmito slovy následně pracovat. U .sgm souborů je třeba navíc načítat jednotlivé kategorie článků u Reuters dokumentů. Součástí této třídy jsou také metody vracející počet slov v souboru a na základě pozice v tomto souboru také jednotlivá slova.

Třída *Preprocess.java* byla převzata a upravena z diplomové práce pana Uhlíře [27]. Tato třída slouží k načítání jednotlivých článků Reuters dokumentů a kategorií článků. Tyto dokumenty mají speciální strukturu, kterou je třeba zpracovat a vrátit jejich obsah. Více o zpracování Reuters dokumentů je popsáno v kapitole 4.3 nazvané Vstupní data aplikace.

Třída *PaiceHuskModel.java* má za úkol implementaci stemovacího algoritmu Paice/Husk popsaného v kapitole 2.3. Tato třída načítá pravidla uložená v odděleném souboru *PaiceRules.txt* uloženém vždy ve stejném balíčku s pevně definovanou strukturou tohoto souboru.
Pravidla musí být v souboru uložena v následujícím tvaru:

ai	*	2	[]	.	ia
a	*	1	[]	.	a
bb	/	1	[]	.	bb
city	/	3	s	.	ytic
ci	/	2	[]	>	ic
cn	/	1	t	>	nc

Paice/Husk algoritmus prochází jednotlivá pravidla a vyhledává koncovky ve stemovaném slově. První sloupec určuje koncovku v obráceném pořadí písmen od konce. Naopak v posledním sloupci je koncovka otočena ve tvaru normálního procházení textem. Druhý sloupec souboru obsahuje znak '*', který říká, že přípona je odebírána pouze pokud stemované slovo nebylo již změněno nebo znak '/', který neklade na stemované slovo žádné omezení. Třetí sloupec obsahuje počet znaků z konce slova, které mají být odstraněny, pokud je koncovka nalezena. Čtvrtý sloupec obsahuje složené závorky [], jestliže se koncovka nezaměňuje za jinou nebo koncovku, která má být připojena na konec slova po odebrání původní koncovky. V pátém sloupci je znak '.' pro ukončení činnosti stemmingu u zpracovávaného slova nebo znak '>' pro další pokračování procházení pravidel po odebrání příslušné koncovky u stemovaného slova.

Činnost tohoto algoritmu je zajištěna voláním metody *PaiceHushAll( )* z hlavního modelu. Ta nejprve volá metodu *parseRules()* pro zpracování souboru s pravidly a uložení pravidel do pomocných vektorů. Po zpracování souboru s pravidly je spuštěn stemovací algoritmus pomocí metody *firstStep(),* který provádí jednotlivé kroky algoritmu. Ve všech implementovaných stemovacích algoritmech je nejprve prohledán stop list, který určí, zda bude nutné slovo dále stemovat nebo bude ze souboru odstraněno. Toto prohledávání však závisí na nastavení uživatele. Pokud by nezaškrtl použití stop listu, pak se tato operace vůbec neprovádí. Výsledný upravený vektor se stemovanými slovy je následně po provedení stemmingu uložen do pomocného souboru tmp/outPaice+název souboru před stemmingem. Toto uložení slouží pro rychlejší zpracování a načítání při další práci s daty a jejich zobrazení v okně uživatele. Pokud by bylo využito interní struktury např. vektoru, pak by se činnost celé aplikace výrazně zpomalila a data by nebyla uložena i po skončení programu (pro jejich kontrolu či procházení). Tohoto principu je využito také u dalších implementovaných algoritmů a je podrobněji popsán v kapitole 4.4 nazvané Činnost aplikace.

Třída *LovinsModel.java* zapouzdřuje implementaci Lovinsova stemovacího algoritmu popsaného v kapitole 2.1. Spuštění Lovinsova algoritmu z hlavního modelu je zajištěno voláním metody *LovinsAll().* Model musí předat algoritmu slova, která mají být stemována a informace o použitém stop listu a zda se má použít. Dále musí předat název souboru, kam se má uložit výsledek po provedení stemmingu. Pro činnost tohoto algoritmu je využíváno dvou textových dokumentů. Prvním z nich je soubor *LovinsRules.txt* uložený v balíčku dp.mvc.model. Tento soubor obsahuje 34 transformačních pravidel a musí být vždy načten v tomto tvaru:

```
olv     olut    0 0 0
ul      l       a o i
end     ens     s 0 0
her     hes     p t 0
```

V prvním sloupci souboru se nachází koncovka, která je vyhledávána ve stemovaném slově. V druhém sloupci se nachází koncovka, která má být zaměněna. V dalších třech sloupcích souboru se nachází vždy znak, po kterém nesmí být koncovka zaměněna. Znak 0 značí, že takové omezení u příslušné koncovky není. Druhým souborem, který využívá aplikace pro korektní činnost Lovinsova algoritmu je soubor ve shodném balíčku s názvem *LovinsConditions.txt* načítaný metodou *FirstStep().* Tento soubor musí být uložen v tomto tvaru:

```
alistically     B
arizability     A
izationally     B
antialness      A
```

Obsahem tohoto souboru je všech 294 koncovek slov použitých v Lovinsově algoritmu. V druhém sloupci na shodném řádku tohoto souboru je vždy podmínka, kterou musí daná koncovka splňovat. Tato podmínka je kontrolována uvnitř implementace Lovinsova algoritmu v metodě *SecondStep()*. Všechny tyto koncovky musí být v souboru uloženy podle délky (od nejdelší po nejkratší), tak jak to algoritmus vyžaduje. Oba tyto načtené soubory jsou vždy na začátku aplikace Lovinsova algoritmu uloženy do pomocných vektorů pro rychlejší načítání a zpracování dat. Toto vstupní načtení je provedeno v metodě *FirstStep()*. Poslední krok algoritmu je proveden v metodě *ThirdStep()*. Tato metoda má na starost použití transformačních pravidel. Na závěr jsou upravená slova uložena do složky tmp do souboru outLovins + název originálního stemovaného souboru.

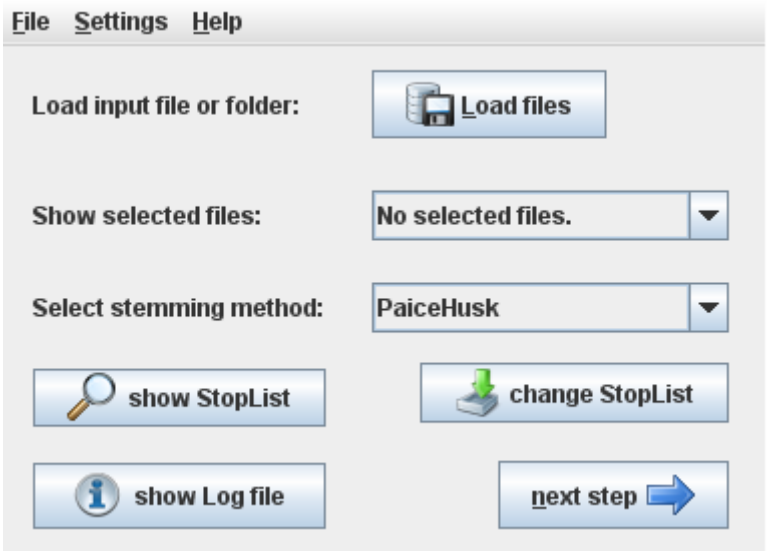
Třída *PorterModel.java* implementuje Porterův algoritmus popsaný v kapitole 2.2. Spouštění jednotlivých kroků Porterova algoritmu má na starost metoda *PorterAll()*. Algoritmus nevyžaduje oproti předchozím dvěma žádný pomocný soubor pro načítání pravidel nebo koncovek. Nejprve se spustí metoda *step1()*, která spustí první krok algoritmu. Po provedení tohoto kroku následují další kroky implementované v metodách *step2()* – *step4()*. Na závěr je spuštěna metoda *step5()* se závěrečnou úpravou stemovaných slov. Po provedení tohoto algoritmu jsou opět slova uložena do složky tmp do souboru outPorter + název stemovaného vstupního souboru. Algoritmus navíc využívá pomocných metod pro zjištění posloupností souhlásek a samohlásek ve slově podle pravidel algoritmu. Pro otestování tohoto algoritmu jsem využil textového souboru *PorterVocabulary.txt*, ke kterému existuje výstupní soubor *PorterOutput.txt* s výsledky po stemmingu jednotlivých slov. Tento výstupní soubor nemá však zcela korektně zpracovány změny znaku y na znak i na konci stemovaných slov podle pravidel Porterova algoritmu.

4.2.3 Pohled

Všechny pohledy (uživatelská okna) použité v aplikaci jsou implementovány v balíčku dp.mvc.view. Pohled má na starost grafické uživatelské rozhraní (GUI) aplikace vytvořené s využitím grafické knihovny Swing. Pohled nám umožňuje reagovat na změny uživatele na vstupu aplikace a tyto změny zasílat controlleru. Ve třídě Main jsou všechna okna aplikace registrována hlavnímu controlleru, který má na starost odchytávání události a přeposílání příslušným akcím v modelu. Pohledy, které jsou zaregistrovány v controlleru jsou odvozeny od třídy *AbstractJFrame.java*.

Třída *FirstStepJFrame.java* obsahuje vzhled hlavního okna aplikace zobrazeného po startu aplikace. Ukázka tohoto hlavního okna aplikace je znázorněna na obrázku 4.2. Hlavní okno aplikace pomocí tlačítka Load files nebo pomocí menu File - Load files umožňuje uživateli vybrat si dokumenty, na které bude aplikován vybraný algoritmus stemmingu. Uživatel aplikace má možnost výběru jednoho textového souboru, Reuters dokumentu nebo celou složku obsahující buď pouze textové soubory nebo Reuters dokumenty v libovolném počtu. Vstupní dokumenty, které lze načítat, jsou podrobněji popsány v kapitole 4.3. Hlavní okno dále umožňuje uživateli vybrat si algoritmus, který se aktuálně pro stemming použije. Tento stemovací algoritmus je možné vybrat také za pomocí menu Settings hlavního okna aplikace. Dalším nastavením je možnost zobrazení načteného stop listu. Tento stop list může být načten již při startu nebo si uživatel může sám vybrat textový soubor, kde má uložen nový stop list. Tento soubor se stop listem musí mít vždy příponu .txt. Implicitní stop list je vždy načítán ze souboru stopList.txt. Pro kontrolu chodu aplikace a zobrazení informací lze na hlavním okně aplikace využít tlačítko pro zobrazení logovacího souboru. Jeho obsah je vždy při novém spuštění aplikace uložen ve složce tmp do souboru log.txt. Součástí hlavního okna je tlačítko pro přechod na další stranu aplikace, které slouží pro zobrazení druhého okna aplikace s obsahem načtených souborů a možností spuštění stemmingu. Toto tlačítko je povoleno pouze v případě korektního načtení vstupních dokumentů aplikace.

Mezi ulehčení nastavení aplikace patří implementace klávesových zkratek, na které jednotlivá tlačítka či položky menu reagují. Typicky jsou znázorněna u popisu tlačítek nebo menu pomocí podtrženého písmena zkratky. Po stisku klávesy ALT + příslušné písmeno na klávesnici dojde k vyvolání příslušné události tlačítka. U každého tlačítka aplikace je navíc implementována krátká pomocná nápověda (tooltip).



Obrázek 4.2 – Hlavní okno aplikace.

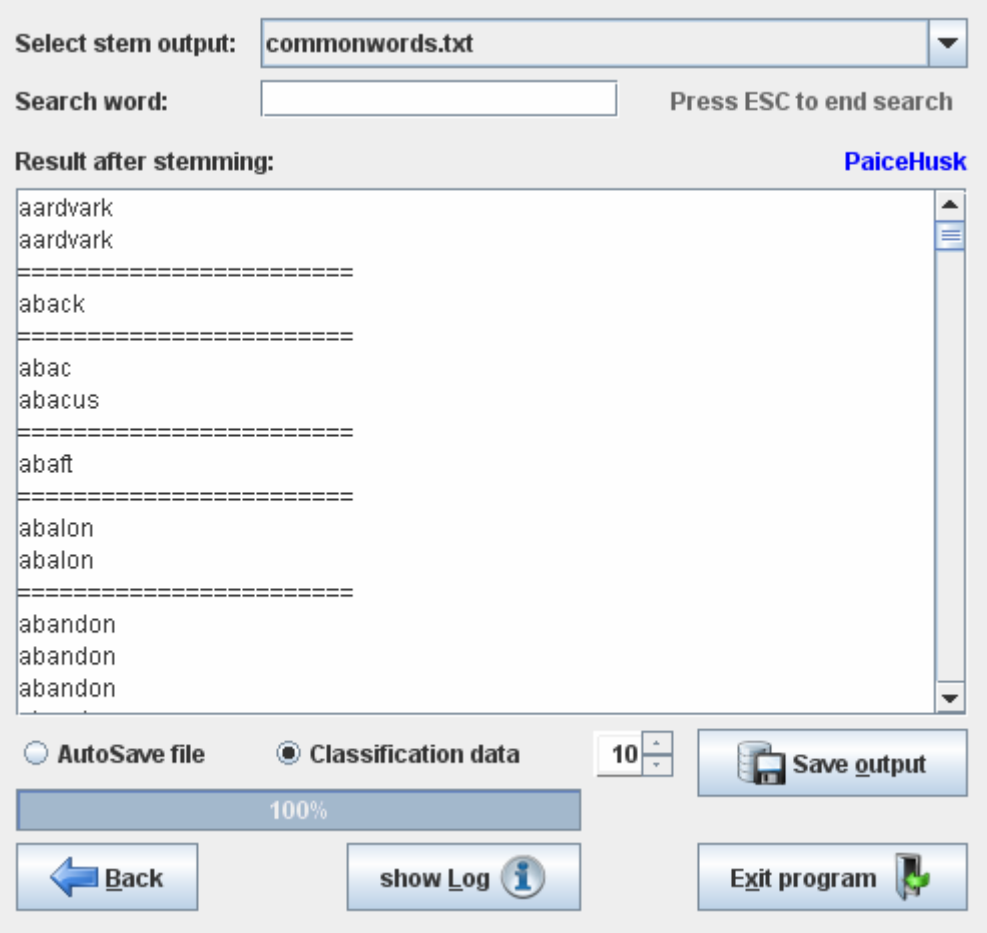
Třída *SecondStepJFrame.java* má na starost vzhled druhého okna aplikace, na které uživatel přejde po korektním načtení vstupních souborů z hlavního okna aplikace a stisku tlačítka next step. Ukázka druhého okna aplikace s načteným textovým souborem commonwords.txt je znázorněna na obrázku 4.3. Součástí tohoto okna je přepínací combo box mezi načtenými vstupními soubory pro stemming (s originálním upraveným textem). Další částí obrazovky je vyhledávání slov ve zvoleném souboru. Toto vyhledávání se ukončuje stiskem klávesy ESC. Stiskem Enter je umožněno uživateli přejít na další vyhledané slovo v okně souboru. Mezi další funkce tohoto okna patří stemming jednotlivých slov souboru a to buď ve směru dopředu nebo dozadu v zobrazeném textu. K této funkci slouží tlačítko Stem prev word a Stem next word. Originální slovo a slovo po provedení stemmingu je zobrazeno na levé straně od tlačítek. Ke stemmingu je vždy využita nastavená stemovací metoda z hlavního okna aplikace. Stop list v tomto případě není aktivován a jednotlivá slova se do pomocných souborů neukládají. Tlačítko Back umožňuje uživateli přejít na hlavní okno aplikace a změnit tak nastavení souborů, stemovacích metod či stop listu a přejít zpět.

Součástí tohoto druhého okna je také dvojice tlačítek pro stemming vybraného souboru nebo celé složky. První tlačítko uživatel nastaví v případě, že chce použít stemovací metodu pouze na předem vybraný soubor. Druhé tlačítko využije pokud chce provést stemming všech načtených souborů. Nad těmito dvěma tlačítky je implementován check box pro aplikaci stop listu. Implicitně je zatržen, což značí, že bude při stemmingu souborů využit předem načtený stop list. Není-li zatržen, pak se stop list na soubory nevyužívá. Průběh stemmingu je procentně znázorněn v progress baru okna aplikace a log souboru. Při stemmingu souborů dochází k dočasnému zakázání zobrazení některých tlačítek tak, aby uživatel nemohl ovlivnit výsledek stemmingu. V průběhu provádění stemmingu má uživatel ve vlastním vlákně možnost vyhledávat slova v předem zvoleném souboru.

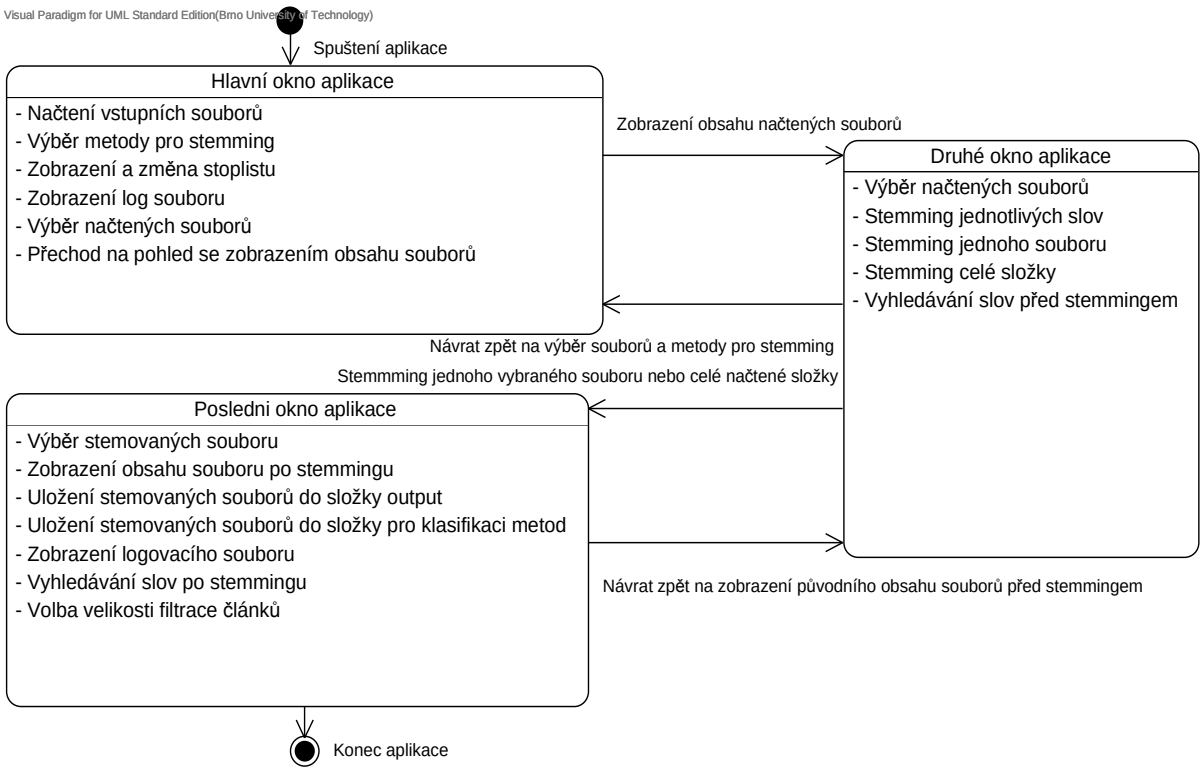
Obrázek 4.3 – Ukázka vzhledu druhého okna aplikace.

Třída *LastStepJFrame.java* implementuje poslední okno aplikace a jeho ukázka po provedení stemmingu je znázorněna na obrázku 4.4. Toto okno obsahuje přepínání mezi výsledky stemovaných souborů. Okno je navázáno na předchozí obrazovku tak, aby si mohl uživatel zobrazit soubor před provedením stemmingu a po jeho provedení. Obě tato okna jsou zobrazena současně. Opět je zde umožněno vyhledávání slov v jednotlivých souborech. Přechod zpět je implementován přes tlačítko Back. Dalším implementovaným prvkem tohoto okna je nastavení uložení výstupu do souborů přes tlačítko Save output. Vedle tohoto tlačítka je navíc implementován grafický prvek pro změnu počtu filtrovaných článků, uložených do výstupního souboru pro klasifikaci při stemmingu Reuters dokumentů. Nastavení uložení je popsáno v kapitole 4.5 o výstupních datech aplikace. Průběh tohoto ukládání je opět zobrazen pomocí progress baru. Informace o průběhu činnosti aplikace je možno zobrazit tlačítkem show Log pro zobrazení logovacího souboru. Posledním tlačítkem tohoto okna je tlačítko Exit program, které ukončí činnost celé aplikace. Aplikaci je dále možno ukončit z hlavního okna aplikace pomocí menu nebo klasickým ukončením programu. U všech oken je implementováno ovládání tlačítek přes klávesové zkratky tak, jak to bylo popsáno u hlavního okna aplikace.

Jednotlivé obrazovky aplikace tak, jak byly implementovány a jsou na sebe navázány, jsou znázorněny na obrázku 4.5. Diagram odpovídá předchozímu popisu o implementovaných třídách v balíčku dp.mvc.view. Mezi další třídy implementované v tomto balíčku patří *AboutView.java*, *LogListView.java* a *StopListView.java*. Tyto třídy již nemají na chod aplikace výrazný vliv a slouží pro uživatele aplikace pro zobrazení některých načtených informací. V první třídě je implementováno zobrazení informací o projektu a aplikaci. Tato obrazovka je přístupná přes menu hlavního okna pomocí položky menu Help – About. Druhá třída slouží pro zobrazení obsahu logovacího souboru aplikace přes tlačítko show Log a třetí pro zobrazení aktuálně načteného stop listu.



Obrázek 4.4 – Ukázka posledního okna aplikace po provedení stemmingu a uložení výstupu.



Obrázek 4.5 – Diagram návaznosti obrazovek aplikace.

4.2.4 Controller

Controller aplikace je implementován v balíčku dp.mvc.controller. Třída *Controller.java* zapouzdřuje chování celé aplikace a je odvozena od třídy *AbstractController.java*. Tato třída obsahuje metody pro registraci jednotlivých pohledů a modelů, které jsou volány při spuštění celé aplikace v metodě Main. Obsahuje metodu *setModelProperty()* pro mapování události odchycené v controlleru na příslušnou metodu *set* v modelu. Třída Controller zajišťuje volání metod pro změnu stavu modelu v závislosti na nastavení jednotlivých prvků GUI aplikace od uživatele. Controller je tedy jakousi ústřední výkonnou jednotkou, která se stará o celkové provázání funkčnosti aplikace. Controller má přímý odkaz na Model, aby mohl upravovat jeho data dle akcí uživatele. Controller dále provádí překreslení všech registrovaných oken aplikace při obdržení události na překreslení od modelu.

4.3 Vstupní data aplikace

Pro korektní otestování funkčnosti implementovaných stemovacích algoritmů umožňuje aplikace načítání dvou typů souborů. Prvním z nich jsou textové dokumenty s příponou .txt a druhým typem jsou Reuters dokumenty popsané v kapitole 4.3.2. Tyto soubory lze načítat vždy jako jeden soubor nebo celou složku s těmito soubory. Tato složka musí obsahovat buď soubory s příponou .txt nebo .sgm a tyto dokumenty nelze společně kombinovat. Složka musí vždy obsahovat maximálně jeden typ souborů a žádné další podsložky tak, aby aplikace rozpoznala, jak vstupní soubor dále zpracovat.

4.3.1 Textové dokumenty

Vstup aplikace umožňuje využít klasických textových dokumentů. Tyto dokumenty mohou být uloženy v libovolném tvaru, ale musí mít příponu souboru .txt. Načítání dat z textových dokumentů má na starost metoda *setIDparse()* v souboru ParseFileModel.java. Tato metoda je volána z hlavního modelu při každém načtení souboru nebo celé složky textových souboru. Pro textové dokumenty následně volá metodu *ParseFolder()*, která provede načtení jednotlivých slov dokumentů do vektorů a uloží cesty k jednotlivým textovým dokumentům pro jejich zobrazení v okně aplikace.

Textové dokumenty jsou po jejich načtení zobrazovány vždy ve formě jednoho slova na jeden řádek. Toto zobrazení umožňuje snadnou kontrolu po provedení příslušného stemovacího algoritmu. Načítání textových dokumentů v aplikaci slouží především pro testovací účely stemovacích algoritmů. Tyto soubory neprochází speciálním předzpracováním, které by z nich odstranilo např. čísla. Čísla mohou být v textu charakteru, bez kterého by celý text nedával smysl a při klasifikaci se o jejich odstranění postará přímo speciální použitý nástroj. Znaky, které by mohly bránit provedení stemmingu slova, jako je např. závorka na konci slova, jsou před zpracováním příslušného načteného slova odstraněny voláním metody *removeNonLiterals()* ze souboru Preprocess.java.

4.3.2 Reuters dokumenty

Dalším vstupním dokumentem, které aplikace umožňuje načítat je sada Reuters dokumentů. Jedná se o sadu dokumentů pod názvem Reuters-21578, které lze nalézt v článku [12] nebo na přiloženém CD. Jde o kolekci textových dat, které se využívají ke kategorizaci textů. Tato sada obsahuje 22 textových souborů s příponou .sgm a speciálním tvarem formátování. Tyto soubory obsahují jednotlivé články, které jsou zařazeny do kategorií (tříd). Každý takový dokument obsahuje 1000 článků (poslední Reuters dokument obsahuje pouze 578 článků), kde některé z těchto článků nemusí být zařazeny do žádné z kategorií nebo jejich obsah může být prázdný. Tato množina dokumentů byla poprvé zveřejněna v roce 1987 a dále upravována panem Lewisem do formátu SGML. Dokumenty jsou rozděleny na trénovací a testovací tak, aby jich bylo možné využít při klasifikaci textů.

Ukázka formátování jednoho článku uvnitř Reuters dokumentu:

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="12650" NEWID="467">
<DATE> 2-MAR-1987 10:50:34.12</DATE>
<TOPICS><D>acq</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN></UNKNOWN>
<TEXT>
        <TITLE>AMERICAN NURSERY BUYS FLORIDA NURSERY</TITLE>
        <DATELINE> TAHLEQUAH, OKLA., March 2 - </DATELINE>
        <BODY>American Nursery Products Inc said it purchased Miami-based Heinl's Nursery Inc, for undisclosed terms.
        Reuter
        </BODY>
</TEXT>
</REUTERS>
```

Z ukázky je vidět, jak jsou jednotlivé články uloženy v souborech. Každý článek je uzavřen do značek <REUTERS>. Jeho textový obsah lze nalézt mezi značkami <BODY>. Další prvek, který nás pro klasifikaci dokumentů zajímá je uzavřen ve značkách <TOPICS>. Jedná se o kategorii (třídu) článku, která je při klasifikaci textu označována jako label. Dle těchto kategorií se v klasifikátoru určuje úspěšnost klasifikace textu a u trénovacích dat dochází k učení na základě této kategorie a jednotlivých slov (atributů) článku. Hlavní nevýhodou těchto článků je, že velké množství z nich neobsahuje žádnou třídu, a proto se musí pro klasifikaci z těchto dokumentů předem odstranit a výrazně se tak snižuje počet článků použitelných pro klasifikaci dokumentů. Toto odstraňování článků implementované v aplikaci bude popsáno v následující části textu v kapitole 4.5.2 o filtraci článků.

Pro extrakci dat z Reuters dokumentů jsem využil třídy Preprocess.java z diplomové práce pana Uhlíře, který se zpracováním SGM souborů již dříve zabýval. Tato třída má na starost načtení všech článků dokumentu mezi značkami <BODY>. Ke každému článku jsem musel upravit třídu tak, aby navíc načítala ke každému článku jeho kategorie uložené mezi značkami <TOPICS>. Zde jsou jednotlivé kategorie navíc uzavřeny do značek <D>. Jeden článek může mít žádnou, jednu nebo více kategorií. Jednotlivé načtené články jsou poté z dokumentu uloženy do složky tmp do souboru outSGM.txt, kde každý řádek začíná písmenem T s číslem článku a následuje na řádku načtený článek. Tímto způsobem jsou také články zobrazeny v okně aplikace. Třídy článků jsou ukládány do odděleného souboru outSGM2.txt. Na každém řádku je opět uložen T s číslem článku a následuje kategorie článku uzavřená do značek <D>, pokud byla u článku nalezena.

Načítání předzpracovaných dat z Reuters dokumentů má na starost metoda *setIDparse()* v souboru ParseFileModel.java, která volá metodu *ParseSGM()*. Tato metoda zpracuje soubory outSGM.txt a outSGM2.txt do interních struktur pro další zpracování dat v aplikaci. Oba tyto soubory jsou při načítání Reuters dokumentů zpracovány a uloženy tak, aby mohly být použity pro stemming. Soubory s obsahy článků jsou vždy uloženy do složky tmp do souboru *outText* + název původního načteného SGM souboru i s příponou .sgm. Soubory s kategoriemi článků jsou uloženy do složky tmp do souboru *outTopic* + název původního načteného SGM souboru. Tyto soubory jsou následně využívány při ukládání souborů do formátu pro klasifikaci a v prostředí RapidMiner ve fázi předzpracování výstupních dat aplikace.

4.4 Činnost aplikace

Při spuštění a za chodu aplikace jsou na disku vytvářeny soubory nutné pro korektní činnost stemovacích algoritmů a zároveň celé aplikace. Prvním vytvářeným typem jsou dočasné soubory popsané v kapitole 4.4.1. Složka *tmp* nutná pro ukládání těchto souborů je vytvořena vždy při prvním spuštění aplikace. Druhým typem je logovací soubor, který slouží pro zaznamenávání nastavení uživatele a zobrazování informací o chodu aplikace. Informace o tomto souboru jsou popsány v kapitole 4.4.2. Teprve po korektním vytvoření těchto souborů dochází ke spuštění hlavního okna aplikace. Následující postup provádění a spouštění algoritmů nad stemovanými soubory v aplikaci byl již naznačen v kapitole 4.2 při popisu jednotlivých implementovaných tříd aplikace. Postup spouštění jednotlivých obrazovek aplikace odpovídá krokům, které musí uživatel učinit, aby obdržel zpracovaná data dle jeho požadavků a možností aplikace. Ukázka činnosti aplikace při stemmingu vybraného dokumentu je zobrazena v příloze C tohoto textu. Tento diagram sekvence znázorňuje provedení stemmingu od druhého okna aplikace a předpokládá, že všechny soubory a nastavení aplikace již uživatel na první obrazovce korektně nastavil.

4.4.1 Dočasné soubory

Implementovaná aplikace, jak již jsem v předchozí části textu zmínil, využívá pro svoji činnost pomocného ukládání souborů do složky *tmp* v adresáři aplikace. Tato složka slouží především k ukládání souborů po jejich načtení v upravené podobě tak, aby se při každé změně nastavení aplikace nemuseli načítat a předzpracovávat znovu. Výhoda této složky vzniká především u zpracování Reuters dokumentů popsaných v předchozí kapitole 4.3.2. Tyto dokumenty není tedy třeba před každým stemmingem předzpracovat a získat obsah článků. Toto předzpracování je provedeno pouze při načtení dokumentu ze složky nebo souboru, což zpomaluje načítání dokumentů, ale urychluje činnost při změně stemovací metody, změně nastavení aplikace nebo samotném stemmingu. Pokud tedy chci provést všechny tři stemovací algoritmy nad stejnou množinou dokumentů, pak jsou tyto dokumenty načteny pouze jednou a je třikrát proveden stemming. Pokud bych dočasné soubory nevyužíval, bylo by nutné při každém spuštění stemovací metody spustit předzpracování Reuters dokumentů a až následně provést stemming, což by výrazně celé zpracování zpomalilo. Soubory poté můžeme v této složce procházet a kontrolovat správnost provedených operací i po ukončení chodu implementované aplikace. Další výhodou této složky je při ukládání výstupních dokumentů. Při ukládání jsou zpracované dokumenty ze složky tmp pouze překopírovány nebo upraveny do odpovídající výstupní podoby. Ukládání výstupu je popsáno v následující kapitole 4.5. Tuto složku je možné před každým spuštěním aplikace odstranit tak, aby po spuštění aplikace obsahovala pouze aktuálně zpracovaná data. Složka by mohla být odstraněna vždy před ukončením aplikace a sloužila by reálně jako dočasná složka, ale pro testovací účely zůstává uložena na disku.

4.4.2 Logovací soubor

Součástí implementované aplikace je také logovací soubor, který zaznamenává především korektní nastavení uživatele v aplikaci. Tento soubor je ukládán do složky tmp do souboru *log.txt* a při každém spuštění aplikace je přepsán novým záznamem. Logovací soubor zaznamenává cesty načtených souborů, nastavené metody stemmingu, zda je nastaven stop list či není (popř. adresářovou cestu nového načteného stop listu), čas provádění stemovacího algoritmu, počet filtrovaných řádků Reuters dokumentů, popřípadě cesty, kam je ukládán výstup aplikace. Dále je zde zobrazena informace zda se jedná o stemming jednoho souboru nebo celé složky. Tento soubor pomůže uživateli zkontrolovat nastavení aplikace a nalézt cesty k uloženým souborům. Dalším prvkem logovacího souboru je zobrazení počtu článků originálního Reuters dokumentu a počtu článků uložených po filtraci.

Ukázka formátování tohoto logovacího souboru zobrazeného po stisku tlačítka show Log, provedení Porterova algoritmu s vypnutým stop listem a nad prvními dvěmi načtenými Reuters dokumenty:

```
.... Set stem method: Porter
Actual stop list: stopList.txt
INIT OK
##### Directory load: C:\Documents and Settings\DP\trainReuters2
File load: C:\DP\trainReuters2\reut2-000.sgm
File load: C:\DP\trainReuters2\reut2-001.sgm

**** Stem all files ****
-> Stop list not set.
Stem file: reut2-000.sgm
Stem file: reut2-001.sgm
Time of stemming: 15.219s

**** Save data ****
File for save: reut2-000.sgm
Number of original lines: 1000
Number of lines after filtration: 379
File for save: reut2-001.sgm
Number of original lines: 1000
Number of lines after filtration: 414
Stemmed data for classification save to file: outClassification/outClassPorter.txt
Original data for classification save to file: outClassification/outTextPorter.txt
Topic data for classification save to file: outClassification/outTopicPorter.txt
```

4.5 Výstup aplikace

Součástí implementované aplikace (posledního okna) je také možnost výběru uložení výsledků po provedení stemmingu. Aplikace nabízí možnost uložení buď v podobě upraveného textového souboru nebo souboru pro klasifikaci. Průběh ukládání souborů je znázorněn v progress baru posledního okna aplikace, který znázorňuje kolik procent souborů již bylo korektně uloženo. V průběhu tohoto ukládání souborů jsou zakázána tlačítka aplikace, která by mohla negativně ovlivnit výsledné uložené soubory a změnit jejich obsah. Ve vlastním vlákně v průběhu ukládání souborů je možnost vyhledávání slov před a po provedení stemmingu. Dále je v průběhu tohoto procesu ukládání možnost provádět stemming jednotlivých slov a zobrazovat průběh ukládání pomocí logovacího souboru.

4.5.1 Uložení výsledků stemmingu

Uložení výsledku stemmingu v podobě textového výpisu zobrazeného v okně aplikace nastává v situaci, kdy uživatel zvolí na posledním okně aplikace AutoSave file a stiskne tlačítko Save output. Dojde k uložení souborů do složky *output*. Jednotlivé stemované soubory jsou uloženy pod názvem *stemmed* + název aktuálně použité metody + název stemovaného souboru s příponou (zůstává stejná). Jeden řádek vždy obsahuje jedno načtené slovo nebo jeden článek a slova (články) jsou uloženy za sebou do konce souboru. Originální soubory jsou v upravené podobě (jedno slovo nebo článek na řádek) uloženy ve stejné složce v souborech začínajících *Original* + název originálního souboru.

Druhý případ, který je v aplikaci implementován, je uložení výstupu aplikace po provedení stemmingu do složky pro klasifikaci. Toto ukládání je zajištěno při volbě Classification data na posledním okně a stisknutí tlačítka Save output. Při načtení a uložení textových dokumentů je výstup ukládán do složky *outTextClassification* a při načtení Reuters dokumentů do složky *outClassification*.

Textové soubory mají pro klasifikační program přidány před každé slovo znaky ‘@@‘ tak, aby bylo možné rozdělit soubor v tomto programu na jednotlivá slova. Stemované soubory jsou ukládány v tomto tvaru pod názvem *outStemmed* + název aktuálně použité metody s příponou .txt. Všechny stemované soubory jsou vždy uloženy do jednoho výsledného souboru za sebe v pořadí načítání vstupních dokumentů. Originální vstupní soubory jsou uloženy v souborech začínajících názvem *outOriginal* opět za sebe do jednoho souboru. Při stemmingu Reuters dokumentů je vždy při uložení souborů pro klasifikaci uložena trojice souborů. Soubor *outClass* + název aktuální metody s příponou .txt obsahující stemované články. Na začátek každého řádku jsou přidány znaky = = = a na konec řádku znaky @@@. Tyto znaky jsou zvoleny tak, aby byla minimální možnost, že se v tomto pořadí budou vyskytovat uvnitř Reuters dokumentů. Druhým ukládaným souborem je soubor *outTopic* + název aktuální metody. V tomto souboru je vždy na každém řádku kategorie článku ukončená znakem ‘;‘. Počet řádků tohoto souboru odpovídá souboru outClass. Každý článek uložený v tomto souboru musí mít přiřazenu odpovídající kategorii získanou ze vstupních dat. Tato skutečnost je zajištěna filtrací článků, která je popsána v následující kapitole. Posledním uloženým souborem při ukládání Reuters dokumentů je soubor *outText* + název aktuální metody. Tento soubor obsahuje za sebou seřazené články před provedením stemmingu. Opět počet řádků musí odpovídat předchozím dvěma uloženým souborům. Znaky přidané na začátek a konec každého řádku odpovídají souboru outClass.

4.5.2 Filtrace článků

Součástí implementované aplikace je také filtrování článků z dokumentů Reuters, které nemají uvnitř značek <TOPICS> v originálním souboru žádnou kategorii nebo jich zde mají naopak více. Články, které nemají žádnou, jsou z ukládání odstraněny. Naopak ty články, které jich mají více, mají vždy vybránu první z kategorií a ta může být s obsahem článků následně uložena, pokud překročí hranici počtu článků uvedenou v následujícím odstavci tohoto textu. Nebyl by problém načítat všechny kategorie článků, ale tato skutečnost by výrazně snižovala úspěšnost klasifikačních algoritmů. Tyto algoritmy by se musely učit jeden typ článků s různým názvem, což by vedlo k jejich naučení na první název a při testování by se procento úspěšnosti zařazení článků do správné kategorie snižovalo.

Dalším typem filtrace článků je minimální počet, ve kterých se musí třída (kategorie) článku ve výstupním dokumentu vyskytovat. Tato kategorie článku musí označovat nejméně *X-1* dalších článků. Tento počet článků *X* je v souboru HelpModel.java implementován proměnnou MIN_POCET, kterou lze dynamicky měnit na posledním okně aplikace. U všech typů kategorií v ukládaných souborech je vypočítán jejich výskyt v metodě *filterInit()*. Samotná filtrace článků je poté implementována v metodě *addTopic()*. Na vstupu této metody jsou jednotlivé ukládané stemované články z Reuters dokumentů. Pokud aktuálně načtená kategorie článku překročí uvedenou hranici počtu článků, pak je obsah načteného článku uložen na konec výstupního souboru. Tento filtr funguje zároveň pro ukládání originálních dat a kategorií článku. Počet článků uložených do výstupních souborů se může navíc snížit v případě, že je načtený článek prázdný (obsahuje třídu článků, ale obsah je prázdný). Tento článek není také do výstupního souboru uložen. Filtrace minimálního počtu článků slouží pro klasifikační metody, které nejsou schopny naučit se klasifikovat články do tříd podle několika málo článků stejných kategorií a snižuje se tak úspěšnost jejich klasifikace (popsáno v kapitole 5.6.2).

4.6 Kompilace a spouštění aplikace

Aplikace byla implementována ve vývojovém prostředí Netbeans. Vzhled a nastavení aplikace bylo optimalizováno pro operační systém Windows XP. Aby bylo možné využít aplikaci i na operačním systému Linux nebo v příkazové řádce bez nutnosti využití prostředí Netbeans, bylo třeba vytvořit kompilační a spouštěcí skript, který zajistí překlad a spuštění aplikace pomocí nástroje Apache Ant.

4.6.1 Apache Ant

Apache Ant [19] je speciálním nástrojem využívaným pro Java aplikace pro kompilace a spouštění programů přímo z příkazové řádky. Tento nástroj se podobá utilitě make, která je využívána pro překlad souborů v programovacím jazyce C. Ant je oproti make celý napsán v jazyce Java, takže umožňuje přenositelnost mezi různými verzemi OS. Pro sestavování Ant skriptů se využívá formátu XML. Celý soubor obsahující překlad, spuštění, tvorbu dokumentace je uložen v souboru *build.xml*. Kořenovým elementem celého souboru je značka <project>. Tato značka je povinná a musí být v každém takovém souboru vždy pouze jedna. Další značkou je <property>. V sestavovacím souboru se využívá jako konstanta, kterou lze nastavit, ale nelze ji dál měnit. Dalším důležitým prvkem tohoto souboru je značka <target>. Tento prvek se využívá pro oddělení logických částí souboru do menších celků. Více o nastavení a atributech jednotlivých elementů lze nalézt v textu [19]. Ant musí být součástí PATH OS. V případě, že uživatel tento nástroj nemá nainstalován, je možné stáhnout instalační soubory ze stránek *ant.apache.org*. Součástí souboru build.xml je také vytvoření programové dokumentace. K vytvoření dokumentace slouží příkaz: ant javadoc, který generuje dokumentaci do složky doc.
Ukázka formátování tohoto souboru je znázorněna v následující části textu:

```
<project>
  <property name="src.dir" value="src" />
  <property name="build.dir" value="build" />
  <property name="jar.dir" value="jar" />
  <target name="run" depends="compile">
        <java jar="${jar.dir}/DP.jar" fork="true" />
  </target>
</project>
```

4.6.2 Kompilace aplikace

Pro překlad všech zdrojových souborů slouží vytvořený soubor *build.xml*, umístěný ve složce s projektem. Pro kompilaci programu stačí v příkazové řádce ve složce s projektem zadat příkaz: ant compile a nástroj Ant se postará o překlad zdrojových souborů a uložení spustitelné aplikace DP.jar do složky jar. Tato kompilace se postará o načtení implicitního stop listu do složky se spustitelnou aplikací DP.jar. Pro odstranění složky jar s DP.jar souborem, přeloženými soubory ve složce build a odstranění složky s dočasnými soubory tmp stačí zadat příkaz: ant clean, který se postará o vymazání přeložených souborů z disku.

Druhou možností je vytvořit nový projekt v prostředí Netbeans a do něho naimportovat zdrojové soubory (složku src). O překlad se zde postará implicitně vytvořený soubor build.xml, který vytvoří automaticky toto prostředí.

4.6.3 Spouštění aplikace

Pro spouštění aplikace existují tři možnosti. První možností je nechat spouštění aplikace na prostředí Netbeans. Druhou lépe využitelnou a přenositelnou možností je spouštění již předem přeložené aplikace a využití příkazu java v příkazové řádce. V případě tohoto spuštění je třeba přepnout se do složky jar. Pro spuštění aplikace slouží příkaz: java –jar DP.jar nebo dvojklik na tento soubor. Třetí možností je využití nástroje Ant a souboru build.xml stejně jako při kompilaci. V tomto případě stačí zadat příkaz: ant run. Ant se postará o kompilaci programu (pokud již nebyl zkompilován) a následné spuštění programu (aplikace je načítána ze souboru DP.jar ze složky jar).

5 Experimenty s algoritmy

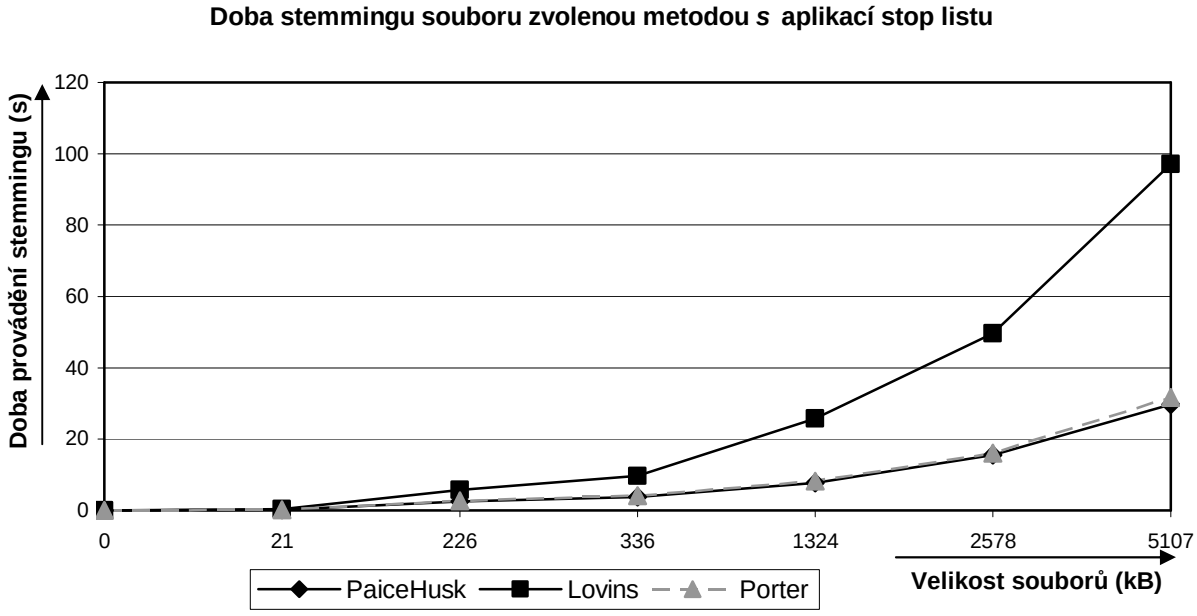
Součástí této diplomové práce jsou experimenty prováděné s jednotlivými algoritmy stemmingu a také porovnání jejich vlivu na klasifikaci textu. Pro základní porovnání stemovacích algoritmů slouží délka běhu algoritmu změřená uvnitř implementované aplikace v průběhu testování jednotlivých metod. Dalším provedeným testem je porovnání redukce počtu slov při použití stemovacích metod oproti originálním textovým dokumentům. Další experimenty s výslednými soubory jsou již prováděny pomocí specializovaných nástrojů pro přípravu vstupních dat a klasifikaci textu. Pro přípravu vstupních dat pro klasifikaci textu byl využit program RapidMiner 5.0, pomocí kterého jsem z výstupních dat, ukládaných do složky outClassification, vytvořil soubory formátu ARFF. Program RapidMiner byl následně využit také pro experimenty s klasifikačními algoritmy a pro porovnání byl použit také program Weka.

5.1 Délka běhu algoritmu

Prvním z provedených experimentů s algoritmy stemmingu (po otestování jejich korektní činnosti) je doba zpracování načtených vstupních dokumentů. Tuto dobu jsem testoval za pomoci textových a Reuters dokumentů přímo uvnitř aplikace. V tabulce 5.1 je znázorněna délka běhu jednotlivých algoritmů v závislosti na velikosti stemovaných souborů a při použití implicitního stop listu z kapitoly 1.3. K tomuto experimentu je uveden také následující graf 5.1.

stopList.txt = true	.txt 21 kB	.txt 226 kB	.txt 336 kB	Reut2-000.sgm 1.32 MB	Reut-000 a 001 2.6 MB	Reuters 4xsgm 5.1 MB
PaiceHusk	0,188 s	2,422 s	3,782 s	7,750 s	15,532 s	29,687 s
Lovins	0,437 s	5,672 s	9,719 s	25,766 s	49,609 s	97,094 s
Porter	0,188 s	2,609 s	4,141 s	8,297 s	16,063 s	31,640 s

Tabulka 5.1 – Doba běhu algoritmů v závislosti na velikosti vstupních dat a stop list je aktivován.

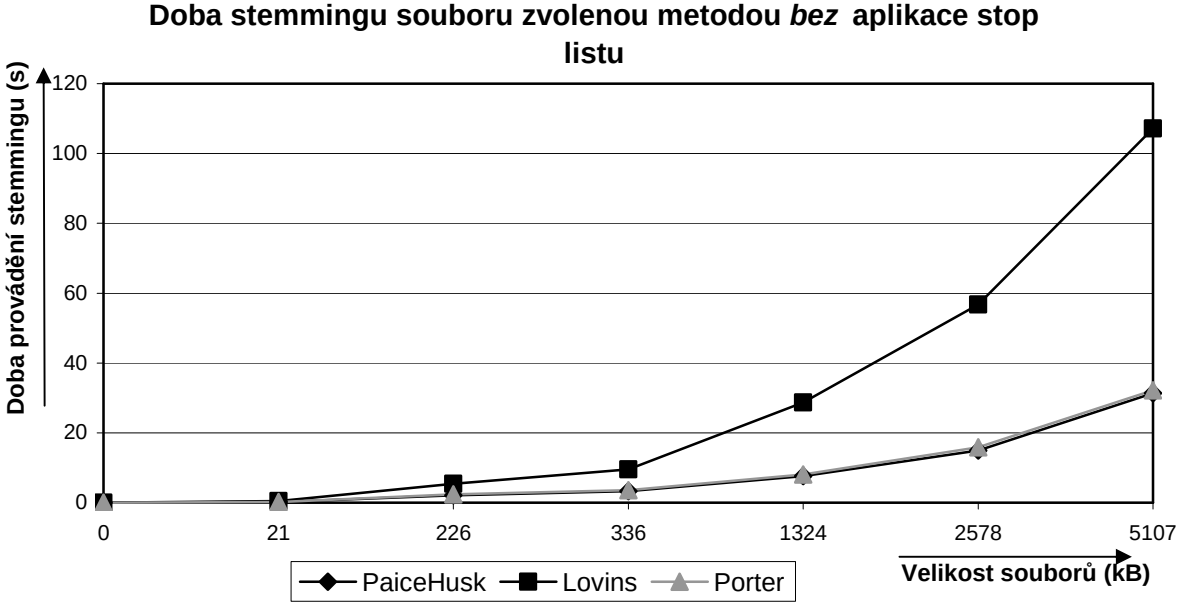


Graf 5.1 - Doba běhu algoritmů v závislosti na velikosti vstupních dat a stop list je aktivován.

Druhý experiment s dobou provádění stemovacích algoritmů je měřen v případě vypnutého stop listu. Tomuto experimentu odpovídá následující tabulka 5.2 a níže uvedený graf.

stopList.txt = false	.txt 21 kB	.txt 226 kB	.txt 336 kB	Reut2-000.sgm 1.32 MB	Reut-000 a 001 2.6 MB	Reuters 4xsgm 5.1 MB
PaiceHusk	0,172 s	2,093 s	3,328 s	7,610 s	14,969 s	31,391 s
Lovins	0,469 s	5,375 s	9,516 s	28,656 s	56,719 s	107,125 s
Porter	0,203 s	2,360 s	3,609 s	8,109 s	15,844 s	32,219 s

Tabulka 5.2 – Doba běhu algoritmů v závislosti na velikosti vstupních dat a stop list není aktivní.



Graf 5.2 - Doba běhu algoritmů v závislosti na velikosti vstupních dat a stop list není aktivní.

Porovnáním těchto dvou experimentů s dobou provádění stemovacích algoritmů je vidět, že časově nejnáročnějším algoritmem je Lovinsův algoritmus. V jeho případě trvá již stemming jednoho textového dokumentu déle o několik desetin. V případě čtyř Reuters dokumentů o velikosti 5.1MB je tato doba provádění u Paice/Husk a Porterova algoritmu okolo 30 sekund a u Lovinsova dosahuje až 107,125 sekund, což je přibližně dvakrát až třikrát delší doba zpracování stejné sady dokumentů. Rozdíl je vidět u Lovinsova algoritmu také při použití stop listu. U Lovinsova algoritmu naroste doba při vypnutí implicitního stop listu a stemmingu čtyř Reuters dokumentů až o 10,031 sekund. Tato narůstající doba je u tohoto algoritmu způsobena velkým množstvím koncovek a pravidel, které musí v tomto případě pro každé stemované slovo projít. Pokud není u Lovinsova algoritmu implicitní stop list využit, pak počet procházení těchto koncovek a pravidel ještě více narůstá. Ostatní algoritmy mají zpracování přípon slov řešeno způsobem, který urychluje jejich zpracování a neobsahují takové množství přípon a pravidel. Z tohoto porovnání je také vidět, že Lovinsův algoritmus byl jedním z prvních, který se pro stemming textových dokumentů využíval a neměl provedeny žádné optimalizace na dobu zpracování dokumentů jako modernější stemovací algoritmy. U Porterova a Paice/Husk algoritmu se doba provádění algoritmu při zapnutém nebo vypnutém implicitním stop listu téměř nemění a obě křivky splývají do sebe. Tato vlastnost je však značně závislá na použitém typu stop listu. V následující tabulce 5.3 jsou naměřeny hodnoty s využitím více specifického anglického stop listu, který již obsahuje 174 anglických slov a je uložen v souboru *otherStopList.txt* nebo v příloze D této diplomové práce. Tato tabulka znázorňuje výrazný nárůst doby předzpracování dokumentů oproti tabulce 5.1, kde bylo využito pouze jednoduchého stop listu obsahujícího 32 slov.

otherStopList = true	.txt 21 kB	.txt 226 kB	.txt 336 kB	Reut2-000.sgm 1.32 MB	Reut-000 a 001 2.6 MB	Reuters 4xsgm 5.1 MB
PaiceHusk	0,297 s	3,359 s	5,328 s	11,829 s	26,578 s	53,063 s
Lovins	0,546 s	7,156 s	11,328 s	30,687 s	59,360 s	118,313 s
Porter	0,328 s	3,875 s	6,000 s	12,453 s	28,203 s	54,532 s

Tabulka 5.3 – Doba běhu algoritmů v závislosti na velikosti vstupních dat a stop list je aktivován.

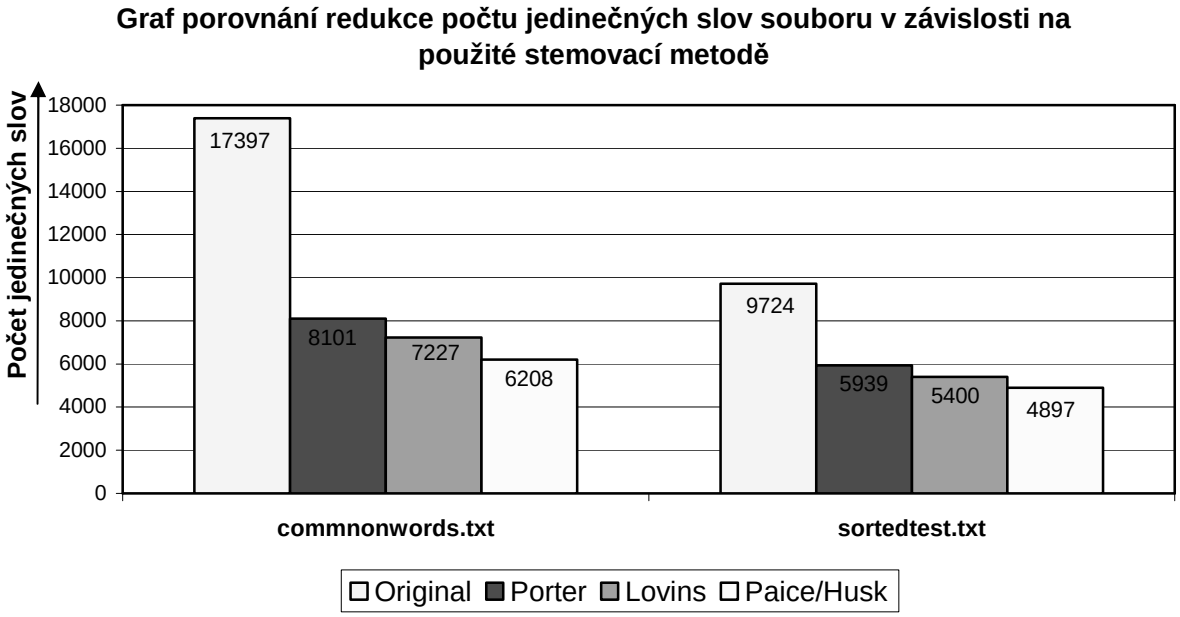
Všechny časové hodnoty, naměřené v tomto experimentu, byly zjištěny pod operačním systémem Windows. Při testování aplikace v operačním systému Linux se všechny algoritmy dostaly s časem pod hodnoty změřené v operačním systému Windows. Pod operačním systémem Linux byla činnost všech stemovacích algoritmů a celé aplikace výrazně rychlejší, ale i přesto Lovinsův algoritmus zůstal nejpomalejším stemovacím algoritmem.

5.2 Porovnání výsledků stemovacích algoritmů

Pro porovnání výsledků implementovaných stemovacích metod jsem využil dvojice souborů *commnonwords.txt* a *sortedtest.txt*, které obsahují setříděná anglická slova vhodná pro testování algoritmů. Tyto soubory lze nalézt ve zdroji [28] nebo na přiloženém CD ve složce DP/test2. Oba tyto soubory obsahují vždy mezi znaky = = = slova, která mají stejný kořen a po stemmingu by všechna slova ve skupině měla být shodná. Při testování jednotlivých stemovacích algoritmů jsem vytvořil tabulku 5.4 a graf 5.3. Tento graf a tabulka zobrazuje počet jedinečných slov v závislosti na testovaném souboru a použitém stemovacím algoritmu bez využití stop listu. Při porovnání těchto výsledků je na výsledných hodnotách zřejmá značná redukce počtu slov po provedení stemmingu. Pro výpočet jedinečných slov dokumentů posloužil vývojářský editor PSPad a jeho vlastnost Seřadit (menu Úpravy - Seřadit se zatrženou možností nazývanou Odstranit duplicity). U originálního souboru commnonwords.txt bylo tímto způsobem vypočítáno 17397 jedinečných slov a u souboru sortedtest.txt to bylo 9724 slov. V případě prvního souboru byla redukce ze 17397 jedinečných slov při použití Paice/Husk stemovacího algoritmu na 6208 jedinečných slov, což je redukce přesahující 64 % oproti originálu. Při použití Lovinsova algoritmu to bylo 7227 slov a Porterův algoritmus vytvořil 8101 jedinečných kořenů slov. V případě druhého souboru je redukce počtu jedinečných slov při využití Paice/Husk stemovacího algoritmu téměř 50 % oproti originálnímu souboru. Nejlépe podle této statistiky vytvářel a mapoval jednotlivé kořeny na sebe Paice/Husk algoritmus. Na druhém místě byl Lovinsův algoritmus a nejhorších výsledků v tomto případě dosáhl Porterův algoritmus. Z tohoto experimentu je zřejmá důležitost předzpracování textových dat. V případě, že budeme chtít tato slova uložit např. do databáze, dochází ke značné úspoře použitého datového prostoru a značnému zrychlení při jejich procházení.

Metoda	Soubor	Počet slov
Original	commnonwords.txt	17397
Lovins	commnonwords.txt	7227
Porter	commnonwords.txt	8101
Paice/Husk	commnonwords.txt	6208
Original	sortedtest.txt	9724
Lovins	sortedtest.txt	5400
Porter	sortedtest.txt	5939
Paice/Husk	sortedtest.txt	4897

Tabulka 5.4 – Počet slov v závislosti na použitém souboru a algoritmu stemmingu.



Graf 5.3 – Graf počtu jedinečných atributů slov získaných z textových dokumentů.

5.3 Použité aplikace pro klasifikaci textu

V dnešní době existuje velké množství systémů, které se využívají pro strojové učení. Mezi tyto nástroje se řadí také Weka a RapidMiner. Bylo by možné využít také jiných nástrojů (např. SAS), ale k experimentům s vlivem stemovacích algoritmů na klasifikaci dokumentů tyto nástroje postačují a jsou volně ke stažení na webových stránkách vývojových skupin. Program RapidMiner 5.0 navíc poskytuje graficky propracovanou možnost předzpracování textových dokumentů s využitím speciálního přídavného textového pluginu.

5.3.1 Weka

Weka celým názvem Waikato Environment for Knowledge Analysis [21] je projektem Nového Zélandu od roku 1993. Jedná se o nástroj pro předzpracování dat s velkým množstvím implementovaných algoritmů strojového učení. Implementačním jazykem tohoto programu je Java a je vyvíjen pod licencí GNU GPL. Výhodou tohoto programu je především snadnost využití jednotlivých algoritmů strojového učení a možnost přidávání nových pluginů. Weka obsahuje nástroje pro předzpracování dat, třídění, vyhledávání, shlukování, klasifikaci a vizualizaci výsledků.

Grafické uživatelské rozhraní obsahuje čtyři základní prostředí aplikace. Prostředí Explorer, které se využívá pro analýzu dat. Toto prostředí jsem využíval také pro klasifikaci textu a pro zobrazení výsledků experimentů. Dalším prostředím, které Weka poskytuje je Experimenter a KnowledgeFlow, které se využívají pro rozsáhlé experimenty a tvorbu nových schémat. Posledním prvkem aplikace Weka je prostředí Simple CLI, které slouží jako jednoduchá příkazová řádka.

Weka podporuje několik druhů vstupních souborů. Podporovány jsou formáty ARFF, CSV, URL, binární formáty a také čtení dat přímo z databáze. Ve své aplikaci využívám pro klasifikaci formátu ARFF, který podporuje jak WEKA, tak také RapidMiner. Tento formát bude popsán v kapitole 5.4.1 o zpracování výstupu aplikace. Instalace a spuštění programu je popsána na stránkách vývojové skupiny [22]. Pro spouštění a klasifikaci větších a rozsáhlejších dat je třeba spustit program z příkazové řádky s větším množstvím nealokované paměti a to např. příkazem:

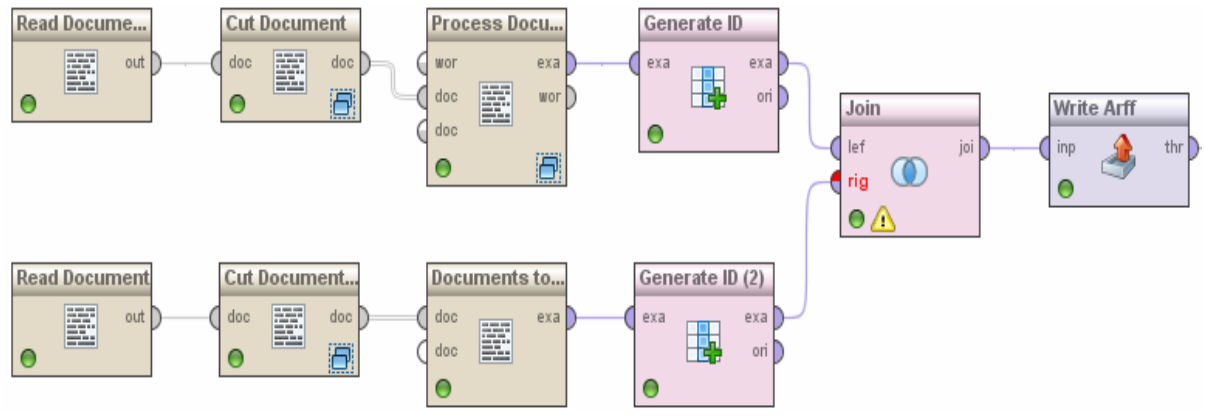
```
java -Xms32m -Xmx1024m -jar weka.jar
```

5.3.2 RapidMiner

RapidMiner (dříve YALE) [20] je flexibilní prostředí implementované v jazyce Java. Podle vývojářů tohoto softwaru se jedná o nástroj s velmi rychlými algoritmy. RapidMiner poskytuje nástroje pro strojové učení, dolování dat, dolování v textu a objevování znalostí v databázích. Hlavní výhodou tohoto programu je rychlost zpracování a vizualizace dat, daná kvalitním graficky provedeným uživatelským rozhraním. Použitím přídavného pluginu pro text mining umožňuje RapidMiner kvalitní možnost zpracování textových dat do strukturované podoby. Program umožňuje export a import dat do/z velkého množství formátů. Jedná se o prostředí založené na formátu XML. Výhodou tohoto nástroje je snadný export všech vytvořených procesů v tomto formátu. Program RapidMiner je nabízen ve třech edicích. První z nich je placená s velmi propracovaným GUI. Druhá edice je nabízena s otevřeným zdrojovým kódem pod licencí GNU GPL a třetí edice je zdarma ke stažení, ale nedává nám k dispozici zdrojový kód. V současné době je nabízena verze RapidMiner 5.0.

5.4 Zpracování výstupu aplikace

Jak již bylo řečeno v kapitole implementace aplikace, výstupem aplikace při uložení Reuters dokumentů pro klasifikaci jsou vždy tři textové dokumenty. Na obrázku 5.1 je znázorněn postup zpracování výstupních textových dokumentů z implementované aplikace v grafickém prostředí programu RapidMiner 5.0. Na horním vstupním prvku Read Document je vždy soubor začínající outClass nebo outText ze složky outClassification. Na dolním vstupním prvku Read Document je vždy soubor s kategoriemi článků outTopic. Pomocí prvku Cut Document jsou vstupní dokumenty rozděleny na jednotlivé řádky. Začátek a konec každého řádku je určen značkami souboru popsanými v kapitole 4.5.1. Pomocí prvku Process Documents jsou dokumenty rozděleny na jednotlivá slova, která odpovídají atributům (sloupcům tabulky) a pro každý atribut je vypočítáno TF-IDF. Pomocí tohoto prvku lze procentně zredukovat počet atributů, které se budou nacházet ve výstupním souboru. Více o této redukci je popsáno v kapitole 5.6.3. Ve spodní větvi jsou prvkem Cut Document kategorie článků rozděleny na jednotlivé řádky (konec řádku určen v souboru znakem: ‘;‘). Prvkem Documents to Data je vygenerována odpovídající množina dat. Dalším prvkem Generate ID je vygenerováno ID pro každý řádek tabulky. Pomocí prvku Join je připojen nový sloupec ke každému řádku s odpovídající kategorií článku. Tento sloupec je při klasifikaci označován atributem label a slouží pro učení a predikci klasifikačních algoritmů. Posledním prvkem je Write Arff, který slouží pro uložení vytvořených atributů do ARFF formátu. Tento formát je podrobněji popsán v následující kapitole o ARFF formátu. Tento soubor musí být zvlášť vytvořen pro originální data a zvlášť pro data po provedení stemmingu tak, aby bylo možné porovnávat jednotlivé výsledky při klasifikaci textu.



Obrázek 5.1 – Zpracování výstupních textových dokumentů aplikace do ARFF formátu.

5.4.1 ARFF formát

Attribute-Relation File Format [23] je formát určený pro klasifikaci, podporovaný nástroji Weka i RapidMiner. Jedná se o textový ASCII soubor obsahující speciální atributy reprezentující tento formát. ARFF soubory jsou složeny ze dvou částí. První částí je hlavička a druhou částí jsou data. Součástí dokumentů mohou být také komentáře s řádkem začínající znakem %. První řádek souboru musí vždy začínat atributem @relation název_relace. Hlavička musí obsahovat název relace a seznam atributů. Atributy jsou vždy uvedeny za atributem @attribute jméno_atributu datový_typ_atributu. Každý atribut má své unikátní jméno. Datová část obsahuje data k příslušným atributům. Tato část je označena atributem @data, za kterým následují jednotlivé hodnoty pro uvedené atributy. Počet hodnot, oddělených na řádku čárkou, musí odpovídat počtu uvedených atributů.

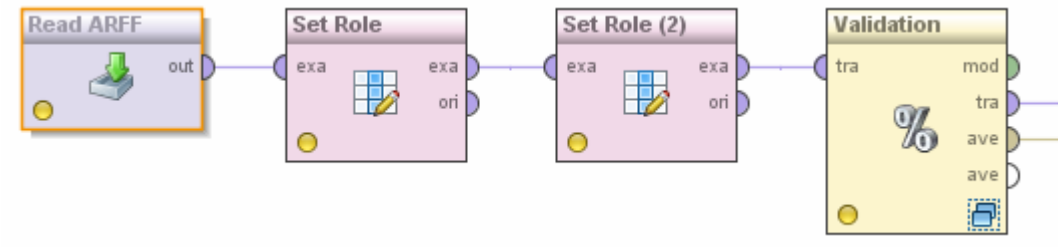
Ve vytvořeném ARFF souboru pro klasifikaci textu, postupem popsaným v úvodu kapitoly 5.4, je vždy prvním atributem odpovídající název kategorie článku a následují vypočítané TF-IDF hodnoty jednotlivých termů (slov). Tyto hodnoty obsahují desetinná čísla (datového typu real) od nuly do jedné. Posledním atributem je vždy hodnota ID daného článku.

Ukázka jednoduchého ARFF souboru:

```
@relation pocasi
        @attribute teplota numeric
        @attribute vlhkost numeric
        @attribute vetrno {ANO,NE}
@data
        15,85,NE
        20,90,ANO
        30,80,ANO
```

5.5 Klasifikace textu

Pro experimenty s klasifikací textu jsem využil programy RapidMiner a Weka. V prostředí Weka jsem pro klasifikaci textu využíval klasifikačního algoritmu SMO a prostředí aplikace Explorer. V programu RapidMiner jsem pro klasifikaci využíval algoritmů NaiveBayes a k-NN. Na obrázku 5.2 je znázorněn postup klasifikace v grafickém prostředí programu RapidMiner 5.0. Na vstupu klasifikace textu je v prvku Read ARFF nastaven předem vytvořený ARFF soubor. Tento soubor je vytvořen podle postupu uvedeného v předchozí kapitole 5.4. Z obrázku 5.2 je také vidět, že po načtení souboru je třeba nastavit role v souboru (atributy ID a label). Atribut label (kategorii článku) je nutné nastavit také při klasifikaci v programu Weka. Posledním operátorem je X-Validation. Pomocí tohoto operátoru se vstupní data v programu rozdělí na testovací a trénovací. Uvnitř tohoto prvku je na straně trénovací části umístěn operátor pro klasifikaci (NaiveBayes nebo k-NN) a na straně testovací části operátor pro aplikaci modelu a výpočet úspěšnosti klasifikace. Celý proces je připojen na výstup aplikace tak, aby mohlo dojít k zobrazení výsledků klasifikace. Výstupem je úspěšnost (chyba) klasifikace textu pomocí zvoleného algoritmu.



Obrázek 5.2 – Klasifikace textu v programu RapidMiner 5.0.

5.6 Porovnání výsledků klasifikace textu

Součástí této kapitoly jsou výsledky experimentů s výstupními daty implementované aplikace prováděných v prostředí nástrojů RapidMiner 5.0 a Weka. Pro zjištění těchto výsledků bylo nutné provádět změny v nastavení implementované aplikace a získat tak nová výstupní data. Dále bylo nutné provádět změny při předzpracování výstupních dat aplikace v RapidMineru a měnit nastavení použitých klasifikačních algoritmů. V této kapitole jsou provedeny experimenty vlivu stemmingu na klasifikaci textu. Tyto experimenty byly prováděny nad různými metodami stemmingu s využitím více typů klasifikačních algoritmů a různým počtem vstupních dokumentů.

5.6.1 Vliv stemmingu na klasifikaci textu

Úkolem tohoto experimentu bylo porovnat jednotlivé implementované stemovací algoritmy s originálními daty z hlediska jejich vlivu na výsledky klasifikace textu. V tabulce 5.5 jsou provedeny experimenty v prostředí RapidMiner s využitím klasifikačního algoritmu k-NN (k-nearest neighbor = k-nejbližších sousedů). Pro experimenty s textovými dokumenty bylo zvoleno 5 nejbližších sousedů. V tabulce 5.6 jsou pro porovnání provedeny experimenty v nástroji Weka s využitím klasifikačního algoritmu SMO. V těchto tabulkách jsou uvedeny výsledky pro všechny implementované stemovací metody (Lovins, Porter, Paice/Husk algoritmus) a pro originální nezměněná data. Při tomto experimentu byly při filtraci v aplikaci odfiltrovány všechny články, které nejsou ve výstupních dokumentech nejméně desetkrát, čímž vznikl uvedený počet článků na daný počet dokumentů. Při zpracování výstupních textových dat a jejich zpracování do tvaru formátu ARFF byly navíc odfiltrovány atributy (slova v textu), které se nevyskytují nejméně v 5 % všech článků nebo naopak přesahují 30% hranici. Touto úpravou byl tedy pro každou metodu vytvořen odpovídající počet atributů uvedených v tabulce. Při zpracování dat v implementované aplikaci bylo navíc využito zapnutého implicitního stop listu. Pro rozdělení textových dat na trénovací a testovací při klasifikaci textu bylo využito operátoru X-validation s rozdělením na 10 složek.

Klasifikační metoda	**k-NN**		**k-NN**		**k-NN**		**k-NN**	
Použitý program	RapidMiner		RapidMiner		RapidMiner		RapidMiner	
Stemovací metoda	Porter		Paice/Husk		Lovins		Žádná	
Redukce atributů %	5		5		5		5	
Filtrace článků	10		10		10		10	
Stop list	ano		ano		ano		ano	
X-validation	10		10		10		10	
Počet dokumentů - článků	**Počet atributů**	**Úspěšnost klasifikace**	**Počet atributů**	**Úspěšnost klasifikace**	**Počet atributů**	**Úspěšnost klasifikace**	**Počet atributů**	**Úspěšnost klasifikace**
4 -> 1879	246	82.06%	264	83.45%	245	82.06%	212	80.94%
8 -> 4065	243	80.93%	265	81.72%	245	81.21%	206	79.11%
16 -> 7694	255	80.17%	273	80.79%	253	80.02%	225	78.72%

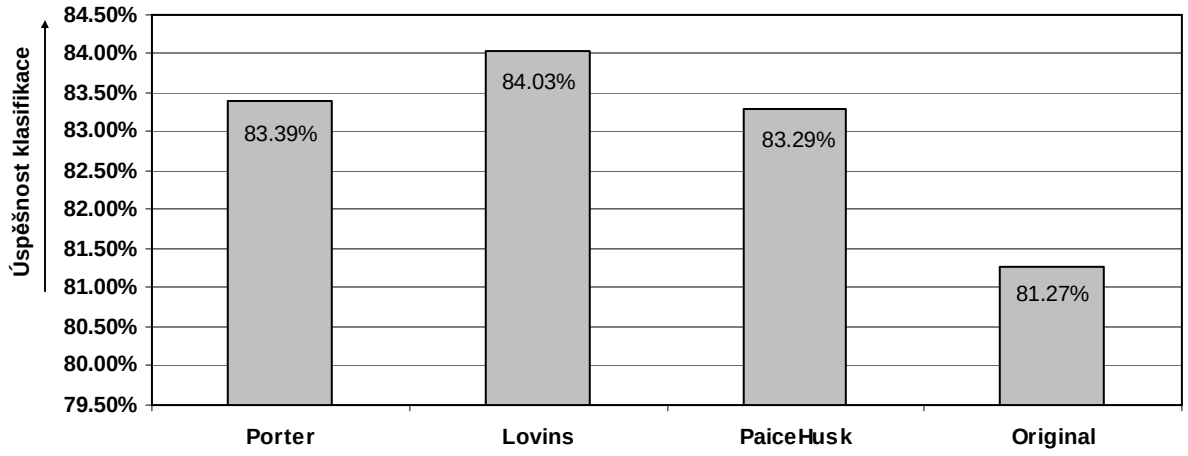
Tabulka 5.5 – Experimenty provedené v programu RapidMiner a klasifikačním algoritmem k-NN.

Klasifikační metoda	**SMO**		**SMO**		**SMO**		**SMO**	
Použitý program	Weka		Weka		Weka		Weka	
Stemovací metoda	Porter		Paice/Husk		Lovins		Žádná	
Redukce atributů %	5		5		5		5	
Filtrace článků	10		10		10		10	
Stop list	ano		ano		ano		ano	
Split-Validation	10		10		10		10	
Počet dokumentů - článků	**Počet atributů**	**Úspěšnost klasifikace**	**Počet atributů**	**Úspěšnost klasifikace**	**Počet atributů**	**Úspěšnost klasifikace**	**Počet atributů**	**Úspěšnost klasifikace**
4 -> 1879	246	84.03%	264	85.25%	245	85.36%	212	82.06%
8 -> 4065	243	85.09%	265	84.92%	245	84.58%	206	82.46%
16 -> 7694	255	83.39%	273	84.03%	253	83.29%	225	81.27%

Tabulka 5.6 – Experimenty provedené v programu Weka a klasifikačním algoritmem SMO.

Při porovnání výsledků obou klasifikačních algoritmů bylo zjištěno, že v tomto případě lepších výsledků dosahuje algoritmus SMO, který má ke klasifikaci textových dat lepší předpoklady a hodnota procentní úspěšnosti klasifikace byla vždy lepší nejméně o 1,12 % než u algoritmu k-NN. Při experimentech s výsledky stemovacích algoritmů a jejich vlivu na klasifikaci textu je zřejmé, že oproti originálním datům je v tomto případě výsledek úspěšnosti klasifikace u stemovaných dat minimálně o 1,12 % a maximálně o 3,3 % lepší. Výsledek klasifikace se u jednotlivých stemovacích metod příliš neliší a zůstává téměř shodný. Z obou tabulek a provedených experimentů, které jsem v průběhu testování prováděl, vyplývá, že u originálních textových dat je snížení úspěšnosti klasifikace dáno menším počtem atributů po jejich redukci. Tento jev bude podrobněji popsán v experimentech v kapitole 5.6.3.

Porovnání výsledků uvedených v předchozích dvou tabulkách je znázorněno v grafu 5.4. Tento graf znázorňuje procentuální změny při provádění klasifikace nad stemovanými nebo originálními daty. V tomto grafu bylo využito hodnot z tabulky 5.6, kde bylo využito SMO klasifikačního algoritmu pro stanovení úspěšnosti klasifikace nad 16 Reuters dokumenty. Z tohoto grafu je zřejmé, že u originálních dat se úspěšnost klasifikace oproti stemovaným datům snižuje a u stemovacích algoritmů se příliš neliší. Nejlepších výsledků úspěšnosti klasifikace textu a nejmenší chyby bylo dosaženo na výstupních datech Paice/Husk algoritmu. Z obou tabulek je také vidět, že mezi úspěšností klasifikace při využití výstupních dat Lovinsova nebo Porterova algoritmu nejsou téměř žádné rozdíly a jejich hodnoty se pohybují na stejném rozmezí přesahujícím 83 %.



Graf 5.4 – Graf závislosti úspěšnosti klasifikace textu na použité metodě stemmingu.

5.6.2 Vliv filtrace počtu článků na klasifikaci textu

Dalším atributem, který při experimentech s výstupními textovými daty vykazoval vliv změn na úspěšnost klasifikace, byla filtrace článků z Reuters dokumentů prováděná přímo uvnitř implementované aplikace. V tabulce 5.7 je znázorněna změna počtu článků ve výstupním dokumentu při změně nastavení hodnoty filtrace počtu článků na posledním okně aplikace (grafickým prvkem vedle tlačítka Save output). Implicitně je v aplikaci tato hodnota nastavena na filtraci článků = 10. Hodnota jedna (minimální možná hodnota) v prvním sloupci tabulky 5.7 udává, že filtrace propouští všechny články, které mají alespoň jeden název článku uveden. Z této tabulky je zřejmé, že téměř polovina všech článků z Reuters dokumentů nemá ve svém těle uvedenu kategorii článku nebo jeho obsah a musí být odfiltrovány z dalšího zpracování. Více o této filtraci článků je popsáno v kapitole 4.5.2, zabývající se filtrací článků.

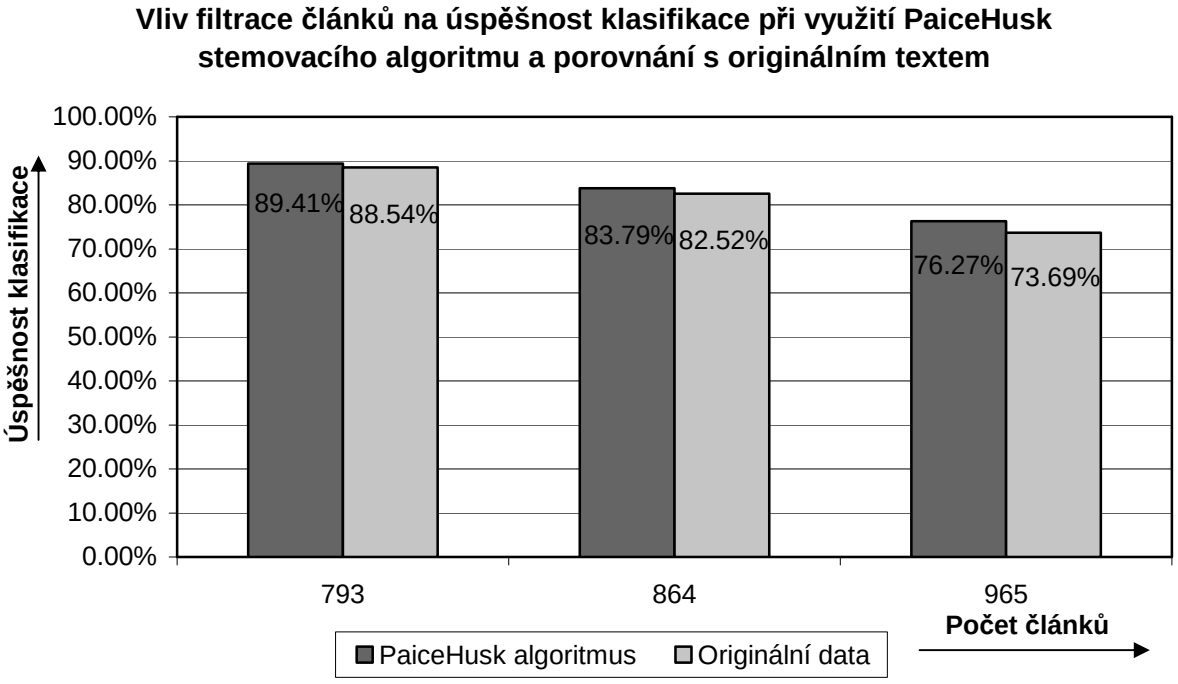
filtrace článků	počet dokumentů	počet článků před filtrací	počet článků po filtraci
1	22	21578	10377
10	22	21578	10286
20	22	21578	10139
1	8	8000	4170
10	8	8000	4065
20	8	8000	3917
1	4	4000	2009
10	4	4000	1879
20	4	4000	1787

Tabulka 5.7 – Experimenty s filtrací článků v implementované aplikaci.

V tabulce 5.8 je znázorněn tento vliv z hlediska úspěšnosti a chyby klasifikace textu. V této tabulce bylo pro experimenty využito Paice/Husk stemovacího algoritmu. Pro klasifikaci byl použit NaiveBayes algoritmus v prostředí RapidMiner. Tento případ je uveden pro první dva Reuters dokumenty s 4% redukcí počtu atributů. Z tabulky je zřejmé, že v případě propuštění článků, které jsou v dokumentech nejméně jednou (filtrace článků = 1), se úspěšnost klasifikace pohybuje okolo 75 %. V případě navýšení počtu filtrovaných článků např. na 20 se tato úspěšnost klasifikace pohybuje již na hranici 90 %. Tato vlastnost plyne z charakteristiky klasifikačních algoritmů, které nejsou schopny naučit se správně klasifikovat text do správné kategorie na základě několika málo článků a potřebují pro trénování větší množství článků se stejným návěštím (kategorií). Tento vliv je znázorněn také v grafu 5.5, který znázorňuje klesající hodnotu úspěšnosti klasifikace při zvětšujícím se počtu článků, které neobsahují větší množství stejných kategorií článků (tzv. TOPIC nebo label). Počet článků, zařazených po filtraci do jednotlivých kategorií, lze znázornit v programu Weka (prostředí Explorer) pod položkou Selected attribute po otevření souboru ve formátu ARFF. Z grafu je také možné pozorovat rozdíl v úspěšnosti klasifikace textu při použití stemovacího algoritmu a klasifikace nad originálními textovými daty.

filtrace článků	počet článků	Úspěšnost klasifikace	Chyba klasifikace	stemovací metoda
20	793	89.41%	10.59%	Paice/Husk
20	793	88.54%	11.46%	Original
10	864	83.79%	16.21%	Paice/Husk
10	864	82.52%	17.48%	Original
1	965	76.27%	23.73%	Paice/Husk
1	965	73.69%	26.31%	Original

Tabulka 5.8 – Experimenty s filtrací článků v aplikaci a vliv této změny na klasifikaci textu.



Graf 5.5 – Graf závislosti úspěšnosti klasifikace textu na vzrůstajícím počtu článků.

5.6.3 Vliv redukce počtu atributů na klasifikaci textu

Dalším prvkem, který jsem při svých experimentech pozoroval a má vliv na klasifikaci textů, je redukce počtu atributů u stemovaných souborů a originálních dat. Ve fázi přípravy ARFF souboru pro klasifikaci textu je možné v programu RapidMiner ovlivnit počet atributů, které budou uloženy do výsledného souboru. K tomuto slouží v programu prvek Process Documents, který počítá TF-IDF a navíc má možnost nastavení minimálního a maximálního počtu procent atributů, ve kterých se musí atribut ve všech článcích vyskytovat. Maximální hodnota je implicitně nastavena na 30 %. Minimální počet procent, se kterým jsem prováděl tento experiment, je uveden ve sloupci redukce atributů v tabulce 5.9 a 5.10. Tato hodnota udává, která slova jsou ignorována v případě, že nepřekračují danou procentní hranici jejich výskytů ve všech dokumentech (článcích). Při zvětšující se hodnotě redukce se snižuje počet atributů uložených ve výsledném ARFF souboru.

První z uvedených tabulek znázorňuje úspěšnost a chybu klasifikace textu při využití Porterova stemovacího algoritmu a změny v úspěšnosti klasifikace při redukci počtu atributů. Druhá tabulka zobrazuje provedení klasifikace nad stejnou množinou dokumentů bez využití stemovacího algoritmu. U obou tabulek je pro klasifikaci využito k-NN algoritmu, implementovaného v prostředí RapidMiner. Při tomto experimentu bylo využito minimálního počtu článků pro filtraci = 20, bez využití seznamu stop slov.

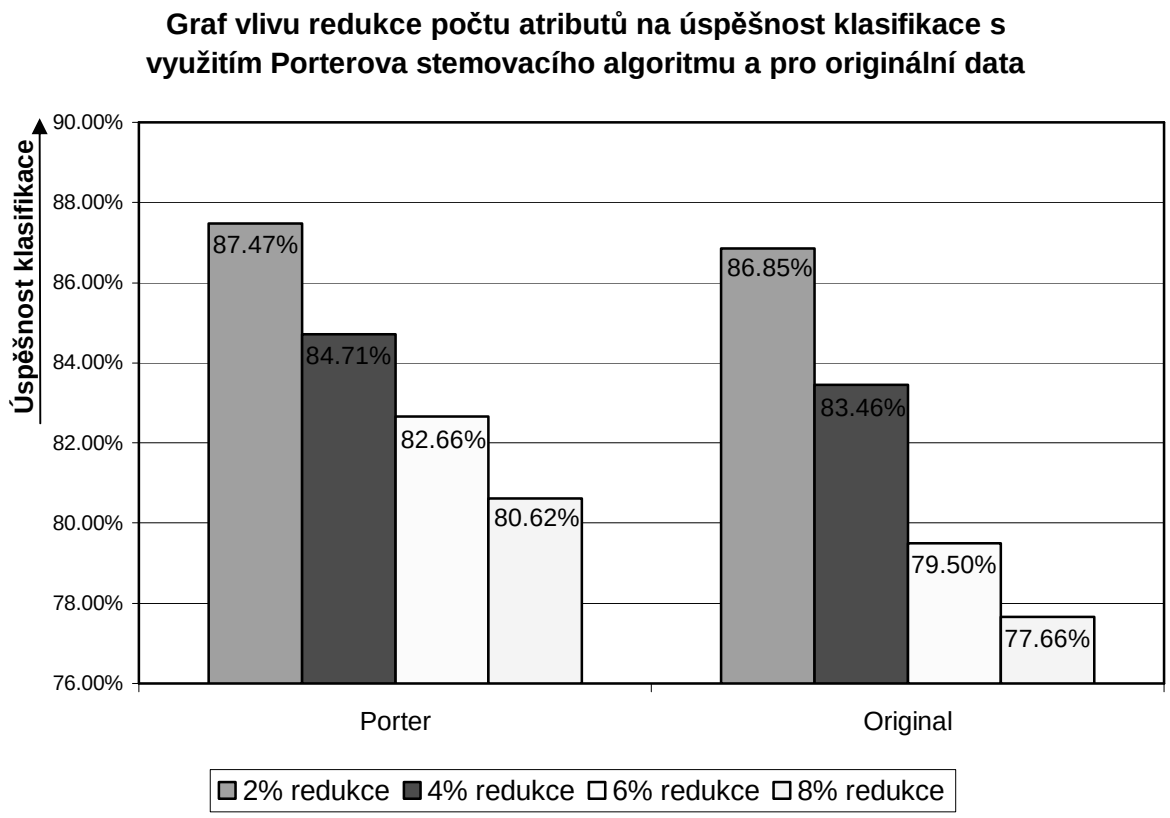
Počet atributů	Redukce atributů	Klasifikace úspěšnost	Chyba klasifikace
615	2%	87.47%	12.53%
321	4%	84.71%	15.29%
193	6%	82.66%	17.34%
130	8%	80.62%	19.38%

Tabulka 5.9 – Klasifikace 8 stemovaných Reuters dokumentů (3917 článků) k-NN algoritmem.

Počet atributů	Redukce atributů	Klasifikace úspěšnost	Chyba klasifikace
593	2%	86.85%	13.15%
283	4%	83.46%	16.54%
159	6%	79.50%	20.50%
101	8%	77.66%	22.34%

Tabulka 5.10 – Klasifikace 8 originálních Reuters dokumentů (3917 článků) k-NN algoritmem.

V porovnání obou tabulek je zřejmé, že při nižších hodnotách procent redukce počtu atributů hodnoty kolísají a pohybují se okolo stejných hodnot. Ke změnám dochází až při zvětšujícím se procentu redukce atributů u originálních dat, kde oproti Porterovu algoritmu výrazně ubývá počtu atributů a tím se snižuje úspěšnost celé klasifikace. K redukci atributů u originálních textových dokumentů dochází z důvodu, že slova nejsou slučována do sebe na stejné kořeny. Stejná slova se vyskytují v různých tvarech, čímž nepřesáhnou požadovanou procentní hranici pro předzpracování a nedostanou se tak do výstupního souboru pro klasifikaci. Proto mají obecně méně sloupců (atributů) originální neupravená data než dokumenty po provedení stemmingu. Při využití 2% redukce počtu atributů je úspěšnost klasifikace u stemovaných dokumentů Porterovým algoritmem 87,47 %, což je pouze o 0,62 % lepší než u originálních nezměněných dokumentů, kde se úspěšnost klasifikace dostala na hodnotu 86,65 %. Při 8% redukci počtu atributů je tento rozdíl v úspěšnosti klasifikace již 2,96 % ve prospěch stemovaných dokumentů. Při této vyšší redukci je však úspěšnost klasifikace v tomto případě pouze 77,66 % u originálních dokumentů a 80,62 % u stemovaných dokumentů. Porovnání obou tabulek je znázorněno také v grafu 5.6. Tento graf znázorňuje vliv redukce počtu atributů u Porterova stemovacího algoritmu a originálních vstupních dat na klasifikaci dokumentů.



Graf 5.6 – Graf závislosti úspěšnosti klasifikace textu na redukci počtu atributů.

5.6.4 Vliv použití seznamu stop slov na klasifikaci textu

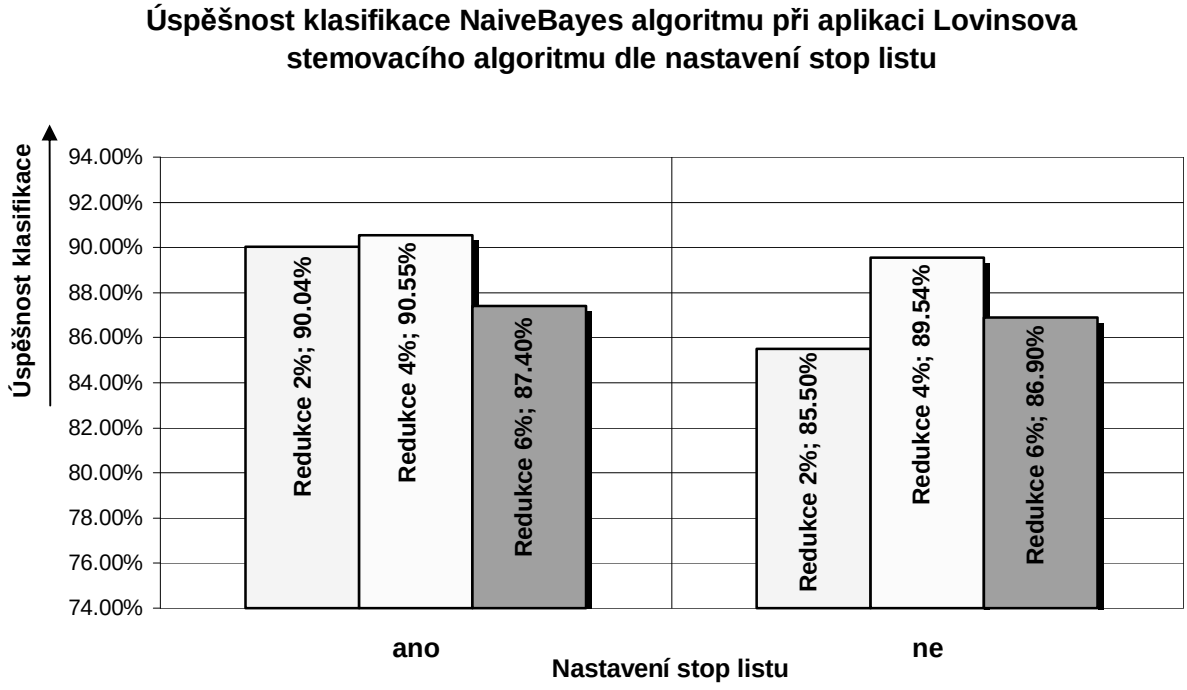
Posledním provedeným experimentem s implementovanou aplikací je vliv použití seznamu stop slov (stop listu) na úspěšnost klasifikace textu. K tomuto experimentu jsem využil Lovinsova stemovacího algoritmu s filtrací článků = 20 a dvojicí Reuters dokumentů na vstupu aplikace. V tomto případě jsem testoval dva různé anglické stop listy. V prvním experimentu jsem využil anglického stop listu, uloženého v souboru *otherStopList.txt* nebo zobrazeného v příloze D tohoto textu, který obsahuje 174 anglických slov. Tento experiment a jeho výsledky se zapnutým stop listem při stemmingu jsou znázorněny v tabulce 5.11 a s vypnutým stop listem v tabulce 5.12. Obě tyto tabulky jsou shrnuty v grafu 5.7. V tomto případě bylo zjištěno, že při zapnutém stop listu dochází k vylepšení výsledku klasifikace až o několik procent. Výsledky úspěšnosti klasifikace jsou při zapnutém stop listu v případě 2% redukce lepší o 4,54 % než je tomu u textu stemovaného bez použitého stop listu. Při zvětšující se redukci počtu atributů se tyto rozdíly smazávají a zapnutí tohoto stop listu přestává mít vliv na vylepšení úspěšnosti klasifikace textu.

Počet atributů	Redukce atributů	Úspěšnost klasifikace	Chyba klasifikace	stopList
525	2%	90.04%	9.96%	Ano
241	4%	90.55%	9.45%	Ano
135	6%	87.40%	12.60%	Ano

Tabulka 5.11 – Vliv zapnutého stop listu u Lovinsova algoritmu na klasifikaci NaiveBayes algoritmem.

Počet atributů	Redukce atributů	Úspěšnost klasifikace	Chyba klasifikace	stopList
598	2%	85.50%	14.50%	Ne
300	4%	89.54%	10.46%	Ne
181	6%	86.90%	13.10%	Ne

Tabulka 5.12 – Vliv vypnutého stop listu u Lovinsova algoritmu na klasifikaci NaiveBayes algoritmem.



Graf 5.7 – Graf závislosti úspěšnosti klasifikace textu na nastavení stop listu.

V druhém experimentu se stop listem jsem při stemovacím procesu využil implicitního stop listu aplikace, uloženém v souboru *stopList.txt*. Tento stop list obsahuje pouze 32 anglických slov, která se zaměřují především na odstraňování anglických spojek a předložek. Naměřené výsledky při použití tohoto stop listu jsou zobrazeny v tabulce 5.13. Z těchto hodnot je zřejmé, že využití tohoto stop listu nemá téměř žádný vliv na úspěšnost klasifikace. V tabulce 5.12 s vypnutým stop listem jsou naměřené hodnoty maximálně o 1,26 % horší než v tabulce 5.13, kde jsou naměřené hodnoty s tímto zapnutým stop listem.

Počet atributů	Redukce atributů	Úspěšnost klasifikace	Chyba klasifikace	stopList
584	2%	86.76%	13.24%	Ano
288	4%	90.05%	9.95%	Ano
168	6%	86.90%	13.10%	Ano

Tabulka 5.13 – Vliv zapnutého implicitního stop listu od společnosti Google na úspěšnost klasifikace.

Z obou těchto provedených experimentů je možné usoudit, že velký vliv na úspěšnost klasifikace má typ použitého stop listu, který lze na internetu nalézt v několika různých variantách. Při využití stemovacího algoritmu s více specializovaným typem stop listu pro anglické texty je možné úspěšnost klasifikace výrazně vylepšit a dosáhnout téměř 5% vylepšení úspěšnosti klasifikace oproti klasifikaci textu bez využití stop listu.

5.7 Zhodnocení výsledků experimentů

V průběhu implementace aplikace a stemovacích metod bylo prováděno několik druhů experimentů s aplikací a následně s využitím specializovaných nástrojů byly prováděny experimenty z hlediska vlivu stemmingu na klasifikaci textu. Nejprve byly přímo uvnitř aplikace provedeny experimenty s dobou běhu jednotlivých stemovacích algoritmů v závislosti na velikosti vstupních dat a použitém stop listu. Tyto experimenty ukázaly výrazný rozdíl v době provádění u Lovinsova algoritmu oproti ostatním stemovacím algoritmům, který jak se ukázalo je zaměřen především na přesnost zpracování než na rychlost zpracování. Při testování implementované aplikace na operačním systému Linux však došlo k urychlení doby zpracování jednotlivých stemovacích algoritmů a celá aplikace tak byla při provádění stemmingu nad větším množství vstupních dokumentů výrazně rychlejší než nad operačním systémem Windows. Rozdíl v době provádění jednotlivých stemovacích algoritmů byl však i zde patrný. Dalším experimentem jsem ukázal redukci počtu slov při použití stemovacích algoritmů. Nejlépe se v tomto směru prezentoval Paice/Husk algoritmus, který při testování nejlépe mapoval slova na jejich kořeny a také byl ve většině případů nejrychlejším algoritmem. Po odstranění duplicitních slov z testovaných souborů vzniklo výrazně méně slov výstupního dokumentu. Více slov se mapovalo na jeden kořen, který ovšem v některých případech viditelně neodpovídal kořenu, na který mělo být slovo podle slovníku upraveno. Některá převedená slova navíc neodpovídala žádnému z kořenů v anglickém slovníku nebo se zcela změnil význam slova. K tomuto dochází v závislosti na použité stemovací metodě a je značně závislé na koncovkách a pravidlech, které algoritmus dokáže zpracovat a také na jaký obor je algoritmus primárně zaměřen.

Další experimenty, které již byly zaměřeny na úspěšnost klasifikace, byly prováděny ve specializovaných nástrojích. Pro tyto experimenty bylo nutné v implementované aplikaci připravit data pro klasifikaci. Některé z těchto vytvořených souborů jsou uloženy na přiloženém CD ve složce DP/outClassification, kde jsou umístěny složky se soubory pod názvem začínajícím vždy počtem stemovaných dokumentů a následovaných hodnotou provedené filtrace uvnitř aplikace.

Tyto experimenty ukázaly několik skutečností, které mají vliv na úspěšnost klasifikace textu. Prvním faktem zjištěným z experimentů s klasifikačními metodami je, že při klasifikaci textových dokumentů je velice důležité vhodné nastavení a výběr klasifikačních algoritmů. Při využití klasifikačního algoritmu SMO v programu Weka bylo vidět zlepšení procentní úspěšnosti klasifikace textu, ale oproti NaiveBayes nebo k-NN algoritmu v programu RapidMiner trvalo učení na větším počtu textových dat výrazně delší dobu a spotřebovalo výrazně větší množství paměti. Důležitým úkolem je také vhodně zvolené odfiltrování článků z výstupního dokumentu pro klasifikaci a vhodné nastavení redukce počtu atributů při zpracování výstupních dokumentů do formátu ARFF nebo jiného použitelného formátu. Při správném nastavení všech těchto parametrů byla při klasifikaci textu překročena 90% hranice úspěšnosti klasifikace. Experimenty v téměř všech testovaných případech ukázaly vylepšení úspěšnosti klasifikace při využití stemovacích algoritmů, ale tyto rozdíly nebyly v řádech desítek procent oproti klasifikaci originálních dokumentů, ale pouze v řádech desetin až jednotek procent.

Dalším experimentem, který jsem provedl, bylo zjištění, zda má zapnutý stop list s využitím stemovacích algoritmů vliv na úspěšnost klasifikace. U stop listu od společnosti Google, který jsem ve své aplikaci využíval bylo zřejmé, že zapnutí tohoto stop listu nemá na úspěšnost klasifikace textu téměř žádný vliv. Při experimentu s více specializovaným typem stop listu pro anglické texty však došlo k zlepšení úspěšnosti klasifikace při jeho použití. Toto zlepšení však bylo závislé na redukci počtu atributů a při zvětšující se redukci přestává mít použití tohoto stop listu vliv na vylepšení úspěšnosti klasifikace.

Závěr

Cílem této diplomové práce bylo podrobné prostudování problematiky dolování z textových dokumentů. Bylo třeba nalézt a nastudovat vhodné algoritmy pro předzpracování textových dokumentů pomocí speciální metody stemmingu. Tyto algoritmy podrobně popsat a ukázat postup jejich činnosti. Na základě dohody s vedoucím diplomové práce jsem zvolil tři algoritmy (Lovinsův, Porterův, Paice/Husk), podrobněji popsané v kapitole o stemmingu. V další části práce jsem se zaměřil na popis činnosti aplikace pro předzpracování textu, která tyto metody pro svoji činnost využívá. Návrh aplikace je založen na architektuře MVC, která se využívá především pro aplikace, kde lze jejich činnost rozdělit do oddělených částí, které spolu mezi sebou komunikují a lze je implementovat odděleně.

Další část této diplomové práce byla zaměřena na implementaci jednotlivých metod stemmingu a také implementaci navržené aplikace pro předzpracování textu v programovacím jazyce Java na platformě Java Swing/GUI. Celá aplikace pracuje na architektuře MVC především z důvodu její snadné rozšiřitelnosti o další moduly předzpracování textu či jednodušší úpravy jednotlivých částí aplikace. Tato aplikace a její popis implementace je podrobněji rozebrána ve 4. kapitole. Součástí této kapitoly je popis vstupních a výstupních dat aplikace a postup činnosti při stemmingu textových dokumentů.

V 5. kapitole jsem provedl experimenty s jednotlivými algoritmy stemmingu z hlediska jejich vlivu na klasifikaci textu, dobu předzpracování a redukci počtu jedinečných slov v textových dokumentech. Při experimentech v klasifikačních nástrojích se objevovalo mírné zlepšení úspěšnosti klasifikace při využití stemovacích algoritmů. Tato skutečnost je zapříčiněna především redukcí počtu slov (atributů), které se mapují na jeden kořen slova a nejsou tak při redukci atributů odstraněny, čímž naroste počet atributů u stemovaných dokumentů, které mají větší vliv na úspěšnost klasifikace a učení klasifikátoru. Vyšší úspěšnost klasifikace u stemovaných dat než je tomu u originálních dat je také ovlivněna vhodným nastavením všech parametrů (filtrací článků, redukcí počtu atributů, nastavením klasifikačních algoritmů) a v některých případech nastavení se tato vyšší úspěšnost klasifikace příliš neprojevovala. Závěrem 5. kapitoly jsem provedl zhodnocení dosažených výsledků při experimentování s algoritmy a vytvořenou aplikací.

Tato diplomová práce ukázala důležitost předzpracování anglických textových dokumentů s využitím stemovacích metod a jejich vliv na úspěšnost klasifikace textu. Všechny tři implementované algoritmy naznačily výraznou redukci počtu slov, které se v dokumentech používají v různých tvarech a po aplikaci jednotlivých algoritmů jsou tato slova mapována na shodné kořeny. Nevýhodou předzpracování textových dokumentů je doba provádění jednotlivých metod, které se pro tuto činnost využívají. Doba zpracování je značně závislá na použité metodě a na typu dokumentu, který má být zpracován. Tato nevýhoda se projevovala také uvnitř implementované aplikace. Další nevýhodou těchto metod je nutnost vybírat jejich typ podle druhu dokumentů, na které jsou aplikovány. Je zřejmé, že stemovací algoritmy nejsou zatím zcela přesné a univerzální tak, aby dokázaly správně upravit všechna načtená anglická slova na jejich slovníkový kořen. Tyto algoritmy však stále procházejí vývojem a v dnešní době, kdy výrazně narůstá počet dokumentů, je předzpracování textu s využitím např. stemmingu jednou z disciplín, na kterou je vhodné se zaměřit.

Mezi rozšíření této diplomové práce je možné zařadit přidání některého z dalších druhů stemovacích algoritmů do vytvořené aplikace. Dalším možným rozšířením je optimalizace chodu implementovaných stemovacích algoritmů tak, aby stemming vstupních dokumentů trval kratší dobu. Do aplikace by bylo dále možné přidat načítání vstupních textových dokumentů z některého z dalších formátů (např. DOC, PDF).

Literatura

[1] Feldman, R., Sanger, J.: *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*, Cambridge University Press, 2007, [cit. 2009-11-04]. ISBN 0521836573.

[2] Bartík, V.: *Dolování z textu a na webu* [online], [cit. 2009-11-20]. Dostupné na URL: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ZZN-IT/lectures/2008/09_TextWebMining.pdf

[3] Lovins, J.B.: *Development of a Stemming Algorithm. Mechanical Translation and Computation Linguistics* [online], aktualizováno březen 1968, [cit. 2009-11-20]. Dostupné na URL: http://www.mt-archive.info/MT-1968-Lovins.pdf

[4] Porter, M.F.: *An Algorithm for Suffix Stripping* [online], aktualizováno červenec 1980, [cit. 2009-11-20]. Dostupné na URL: http://tartarus.org/~martin/PorterStemmer/def.txt

[5] Brahaj, A.: *List of English Stopwords* [online], aktualizace 14.4.2009, [cit.2009-11-20]. Dostupné na URL: http://armandbrahaj.blog.al/2009/04/14/list-of-english- stop-words/

[6] Paice, C.D., 1977 : *Another Stemmer*. Department of computing Lancaster university 1977, GB [online], [cit. 2009-11-01]. Dostupné na URL: http://portal.acm.org/citation.cfm?id=101310

[7] Loy, M., Eckstein, R., Wood, D., Elliott, J., Cole, B.: *Java Swing*[online], 2003, O'Reilly. [cit. 2009-10-28]. Dostupné na URL: http://books.google.cz/books?id=cPxGfk-FZNUC&printsec=frontcover#v=onepage&q=&f=false

[8] Eckel, B. : *Thinking in Java* [online]. Prentice-Hall, 1998 [cit.2009-10-26]. ISBN 0136597238.

[9] Eckstein, R.: *Java SE Application Design With MVC* [online].Aktualizováno březen 2007. [cit. 2009-10-27]. Dostupné na URL: http://java.sun.com/developer/technicalArticles/javase/mvc/

[10] *English Stop words* [online]. [citováno 2009-10-27]. Dostupné na URL: http://www.ranks.nl/resources/stopwords.html

[11] Hull, D.A., Grefenstette, G.: *A Detailed Analysis of English Stemming Algorithms*[online], Xerox Research Center, aktualizováno 31.ledna 1996, [cit. 2009-11-15]. Dostupné na URL: http://www.xrce.xerox.com/content/download/6676/51464/file/DHull-GGrefenstette-Technical-report-MLTT96.pdf

[12] Lewis,D., *Reuters-21578 text categorization test collection* [online]. Dostupné na URL: http://www.daviddlewis.com/resources/testcollections/reuters21578/

[13] Knoth, P., *Kategorizace textu pomocí neuronové sítě LVQ*. FIT VUT, 2007 [online]. [cit. 2009-11-29]. Dostupné na URL: http://www.stud.fit.vutbr.cz/~xknoth00/resources/LVQ.pdf

[14] Sychra, T.: Diplomová práce: *Metody extrakce informace z textových dokumentů*. FIT VUT 2008 [online], [cit. 2009-11-08]. Dostupné na URL: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=4772

[15] Krátký, M.: *Využití SVD pro indexování latentní sémantiky*. [online] VŠB - Technical University of Ostrava 2008, [cit. 2009-11-21]. Dostupné na URL: http://www.cs.vsb.cz/arg/techreports/lsi-svd_ma.pdf

[16] Lorenc, L.: *Shlukování* [online], Získávání znalostí z databází. [cit. 2009-11-22].FIT VUT. Dostupné na URL: http://www.fit.vutbr.cz/study/courses/ZZD/public/seminar0304/Shlukovani2.pdf

[17] Vojáček, A.: *Samoučící se neuronová síť – SOM, Kohonenovy mapy*. aktualizováno 14. května 2006, [cit. 2009-11-22]. Dostupné na URL: http://www.kiv.zcu.cz/studies/predmety/uir/NS/Samouc_NN2.pdf

[18] Tichý, J.*: Programová podpora tvorby webových aplikací*. Diplomová práce[online], aktualizováno 24. srpna 2004, [cit. 2009-11-21]. Dostupné na URL: http://www.jantichy.cz/diplomka

[19] Hynar, M.: *ANT – nebojte se mravence*. 6.2.2003[online], [Cit. 2010-04-14]. Dostupné na URL: http://www.root.cz/clanky/ant-nebojte-se-mravence/

[20] *RapidMiner program* [online], [Cit. 2010-04-14]. Dostupné na URL: http://rapid-i.com/content/view/181/190/

[21] Souhrada, V.*: WEKA software*. Západočeská univerzita v Plzni[online]. [Cit.2010-04-14]. Dostupné na URL: http://www.kiv.zcu.cz/~mautner/Pt/weka_pt_08.pdf

[22] *WEKA software.* [online]. [Cit.2010-04-16]. Dostupné na URL: http://www.cs.waikato.ac.nz/~ml/weka/index.html

[23] *Atribute-Relation File Format(ARFF)*. [online]. [Cit.2010-04-16]. Dostupné na URL: http://www.cs.waikato.ac.nz/~ml/weka/arff.html

[24] Liao, Y., *Using Text Categorization Techniques for Intrusion Detection.* University of California, Aktualizováno 13. května 2002[online].[Cit.2010-04-16]. Dostupné na URL: http://www.usenix.org/events/sec02/full_papers/liao/liao_html/text_ss02.html

[25] Platt, J., C., *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. Microsoft Research. Aktualizováno 14.8.2000 [online].[Cit.2010-04-19]. Dostupné na URL: http://research.microsoft.com/pubs/68391/smo-book.pdf

[26] Sedláček, P.,: *Extrakce informací z textu.*, Diplomová práce Fakulta Informatiky Masarykova Univerzita, 2006. [online].[Cit.2010-04-19]. Dostupné na URL: http://is.muni.cz/th/51819/fi_m/Diplomova_prace.pdf

[27] Uhlíř, M.: Diplomová práce: *Metody pro získávání asociačních pravidel z dat*. FIT VUT 2007 [online], [cit. 2010-04-27]. Dostupné na URL: http://www.fit.vutbr.cz/study/DP/DP.php?id=4771&y=2006&ved=Bart

[28] Hooper, R., Paice, Ch.: *The Lancaster Stemming Algorithm.* Computing Department of Lancaster University[online].[Cit. 2010-05-03]. Dostupné na URL: http://www.comp.lancs.ac.uk/computing/research/stemming/Links/resources.htm

[29] Fox, Ch.: *A Stop List for General Text*, Bell Laboratories [online]. [Cit. 2009-11-13]. Dostupné na URL: http://portal.acm.org/citation.cfm?id=378888

[30] Húsek, J., Pokorný, J., Snášel, V., Řezanková, H., *Metody vyhledávání v rozsáhlých kolekcích dat* [online]. [Cit. 2009-12-05]. Dostupné na URL: http://www.ksi.mff.cuni.cz/~pokorny/papers/Datakon03-tut.pdf

Seznam příloh

Obsah CD

1. README.txt (popis spouštění, kompilace programu a obsahu složky DP).
2. Zdrojové texty (složka DP/src).
3. DP.jar - Spustitelná aplikace (složka DP/jar).
3. Testovací soubory (složka DP/test2).
4. Spouštěcí a kompilační skript (DP/build.xml).
5. Soubory připravené pro klasifikaci textu (složka DP/outClassification).
6. Složka compareStemming se soubory pro porovnání stemovacích algoritmů z kapitoly 5.2.
7. Exportované XML soubory RapidMineru 5.0 pro klasifikaci a předzpracování výstupu:
    - 7.1 - tvorbaArff.xml – Příprava výstupních souborů pro klasifikaci do ARFF formátu.
    - 7.2 - kNNklasifikace.xml - Soubor pro klasifikaci vstupních ARFF dat k-NN algoritmem.
    - 7.3 - NaiveBayesKlasifikace.xml – Soubor pro klasifikaci vstupních ARFF dat NaiveBayes algoritmem.
8. Programová dokumentace (složka DP/doc).
9. Projekt ve vývojovém prostředí NetBeans (celá složka DP).
10. Soubor Weka spusteni.txt obsahující spouštění programu Weka z příkazové řádky pod Windows.
11. Zdrojový soubor textu diplomové práce ve formátu .doc (soubor DP.doc).
12. PDF soubor s textem diplomové práce (soubor DP.pdf).
13. Reuters dokumenty (reuters21578.tar).
14. Implicitní stop list aplikace (soubor stopList.txt ve složce DP).
15. Stop list pro anglické texty (soubor otherStopList.txt ve složce DP).

Příloha A. – Seznam koncovek pro Lovinsův algoritmus.

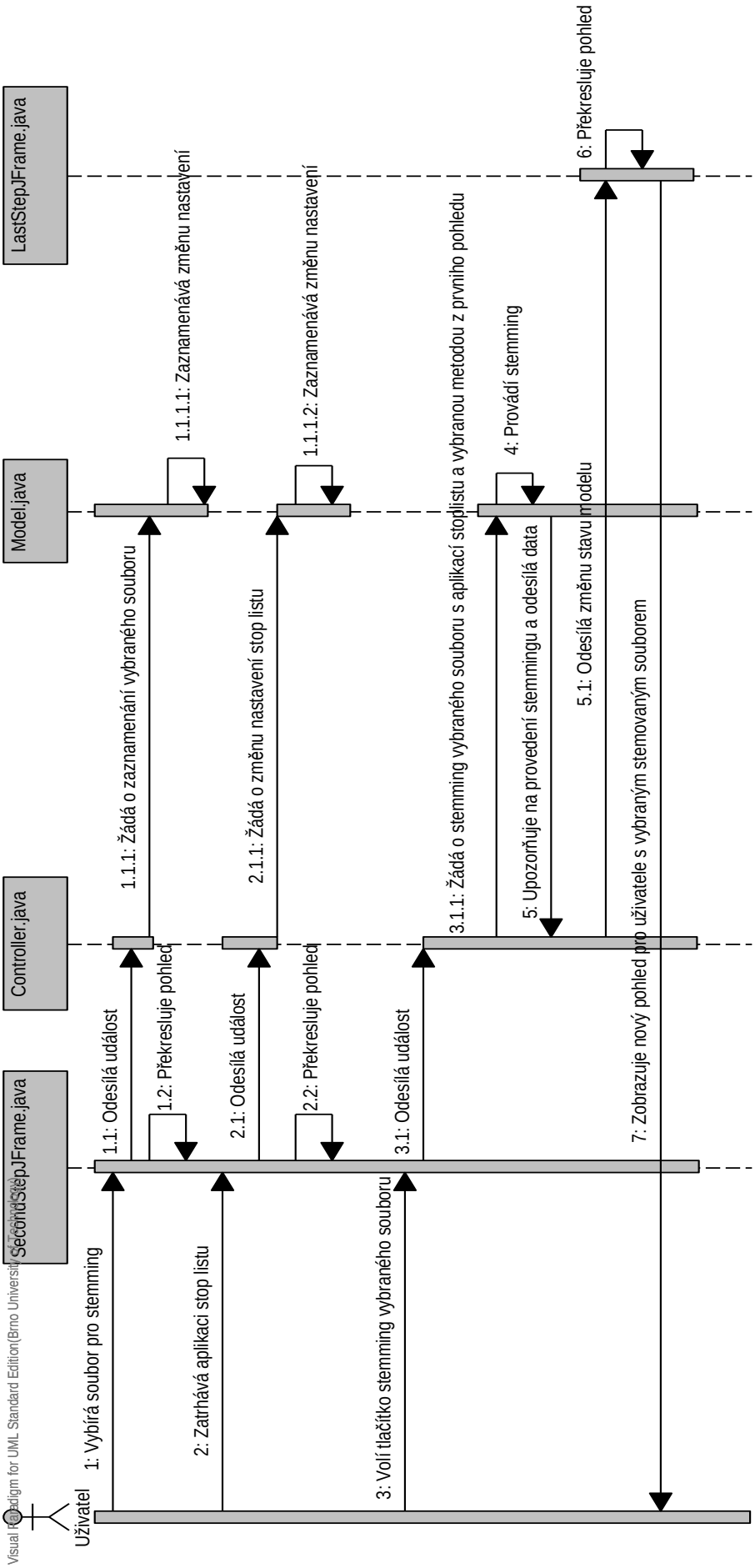
Seznam koncovek								
Délka	***Koncovka***	***Podmínka***	***Koncovka***	***Podmínka***	***Koncovka***	***Podmínka***	***Koncovka***	***Podmínka***
11	alistically	B	arizability	A	izationally	B		
10	antialness	A	arisations	A	arizations	A	entialness	A
9	allically	C	antaneous	A	antiality	A	arisation	A
	arization	A	ationally	B	ativeness	A	eableness	E
	entations	A	entiality	A	entialize	A	entiation	A
	ionalness	A	istically	A	itousness	A	izability	A
	izational	A						
8	ableness	A	arizable	A	entation	A	entially	A
	eousness	A	ibleness	A	icalness	A	ionalism	A
	ionality	A	ionalize	A	iousness	A	izations	A
	lessness	A						
7	ability	A	aically	A	alistic	B	alities	A
	ariness	E	aristic	A	arizing	A	ateness	A
	atingly	A	ational	B	atively	A	ativism	A
	elihood	E	encible	A	entally	A	entials	A
	entiate	A	entness	A	fulness	A	ibility	A
	icalism	A	icalist	A	icality	A	icalize	A
	ication	G	icianry	A	ination	A	ingness	A
	ionally	A	isation	A	ishness	A	istical	A
	iteness	A	iveness	A	ivistic	A	ivities	A
	ization	F	izement	A	oidally	A	ousness	A
6	aceous	A	acious	B	action	G	alness	A
	ancial	A	ancies	A	ancing	B	ariser	A
	arized	A	arizer	A	atable	A	ations	B
	atives	A	eature	Z	efully	A	encies	A
	encing	A	ential	A	enting	C	entist	A
	eously	A	ialist	A	iality	A	ialize	A
	ically	A	icance	A	icians	A	icists	A
	ifully	A	ionals	A	ionate	D	ioning	A
	ionist	A	iously	A	istics	A	izable	E
	lessly	A	nesses	A	oidism	A		
5	acies	A	acity	A	aging	B	aical	A
	alist	A	alism	B	ality	A	alize	A
	allic	BB	anced	B	ances	B	antic	C
	arial	A	aries	A	arily	A	arity	B
	arize	A	aroid	A	ately	A	ating	I
	ation	B	ative	A	ators	A	atory	A
	ature	E	early	Y	ehood	A	eless	A
	elity	A	ement	A	enced	A	ences	A
	eness	E	ening	E	ental	A	ented	C
	ently	A	fully	A	ially	A	icant	A
	ician	A	icide	A	icism	A	icist	A
	icity	A	idine	I	iedly	A	ihood	A
	inate	A	iness	A	ingly	B	inism	J
	inity	CC	ional	A	ioned	A	ished	A
	istic	A	ities	A	itous	A	ively	A
	ivity	A	izers	F	izing	F	oidal	A
	oides	A	otide	A	ously	A		
4	able	A	ably	A	ages	B	ally	B
	ance	B	ancy	B	ants	B	aric	A
	arly	K	ated	I	ates	A	atic	B
	ator	A	ealy	Y	edly	E	eful	A
	eity	A	ence	A	ency	A	ened	E
	enly	E	eous	A	hood	A	ials	A
	ians	A	ible	A	ibly	A	ical	A
	ides	L	iers	A	iful	A	ines	M
	ings	N	ions	B	ious	A	isms	B
	ists	A	itic	H	ized	F	izer	F
	less	A	lily	A	ness	A	ogen	A
	ward	A	wise	A	ying	B	yish	A
3	acy	A	age	B	aic	A	als	BB
	ant	B	ars	O	ary	F	ata	A
	ate	A	eal	Y	ear	Y	ely	E
	ene	E	ent	C	ery	E	ese	A
	ful	A	ial	A	ian	A	ics	A
	ide	L	ied	A	ier	A	ies	P
	ily	A	ine	M	ing	N	ion	Q
	ish	C	ism	B	ist	A	ite	AA
	ity	A	ium	A	ive	A	ize	F
	oid	A	one	R	ous	A		
2	ae	A	al	BB	ar	X	as	B
	ed	E	en	F	es	E	ia	A
	ic	A	is	A	ly	B	on	S
	or	T	um	U	us	V	yl	R
	s'	A	s	A				
1	a	A	e	A	i	A	o	A
	s	W	y	B				

Příloha B. – Seznam Paice/Husk pravidel.

Pravidlo	Komentář změn		
	koncovka		změna
ai*2.	-ia	›	- beze změn
a*1.	-a	›	- beze změn
bb1.	-bb	›	-b
city3s.	-ytic	›	-ys
ci2>	-ic	›	-
cn1t>	-nc	›	-nt
dd1.	-dd	›	-d
dei3y>	-ied	›	-y
deec2ss.	-ceed	›	-cess
dee1.	-eed	›	-ee
de2>	-ed	›	-
dooh4>	-hood	›	-
e1>	-e	›	-
feil1v.	-lief	›	-liev
fi2>	-if	›	-
gni3>	-ing	›	-
gai3y.	-iag	›	-y
ga2>	-ag	›	-
gg1.	-gg	›	-g
ht*2.	-th	›	- beze změn
hsiug5ct.	-guish	›	-ct
hsi3>	-ish	›	-
i*1.	-i	›	- beze změn
i1y>	-i	›	-y
ji1d.	-ij	›	-id
juf1s.	-fuj	›	-fus
ju1d.	-uj	›	-ud
jo1d.	-oj	›	-od
jeh1r.	-hej	›	-her
jrev1t.	-verj	›	-vert
jsim2t.	-misj	›	-mit
jn1d.	-nj	›	-nd
j1s.	-j	›	-s
lbaifi6.	-ifiabl	›	-
lbai4y.	-iabl	›	-y
lba3>	-abl	›	-
lbi3.	-ibl	›	-
lib2l>	-bil	›	-bl
lc1.	-cl	›	-c
lufi4y.	-iful	›	-y
luf3>	-ful	›	-
lu2.	-ul	›	-
lai3>	-ial	›	-
lau3>	-ual	›	-
la2>	-al	›	-
ll1.	-ll	›	-l
mui3.	-ium	›	-
mu*2.	-um	›	- beze změn
msi3>	-ism	›	-
mm1.	-ram	›	-m
nois4j>	-sion	›	-j
noix4ct.	-xion	›	-ct
noi3>	-ion	›	-
nai3>	-ian	›	-
na2>	-an	›	-
nee0.	-een	›	-een
ne2>	-en	›	-
nn1.	-nn	›	-n

Pravidlo	Komentář změn		
	koncovka		změna
pihs4>	-ship	›	-
pp1.	-pp	›	-p
re2>	-er	›	-
rae0.	-ear	›	-ear
ra2.	-ar	›	-
ro2>	-or	›	-
ru2>	-ur	›	-
rr1.	-rr	›	-r
rt1>	-tr	›	-t
rei3y>	-ier	›	-y
sei3y>	-ies	›	-y
sis2.	-sis	›	-s
si2>	-is	›	-
ssen4>	-ness	›	-
ss0.	-ss	›	-ss
suo3>	-ous	›	-
su*2.	-us	›	- beze změn
s*1>	-s	›	- beze změn
s0.	-s	›	-s
tacilp4y.	-plicat	›	-ply
ta2>	-at	›	-
tnem4>	-memt	›	-
tne3>	-ent	›	-
tna3>	-ant	›	-
tpir2b.	-ript	›	-rib
tpro2b.	-orpt	›	-orb
tcud1.	-duct	›	-duc
tpmus2.	-sumpt	›	-sum
tpec2iv.	-cept	›	-ceiv
tulo2v.	-olut	›	-olv
tsis0.	-sist	›	-sist
tsi3>	-ist	›	-
tt1.	-tt	›	-t
uqi3.	-iqu	›	-
ugo1.	-ogu	›	-og
vis3j>	-siv	›	-j
vie0.	-eiv	›	-eiv
vi2>	-iv	›	-
ylb1>	-bly	›	-bl
yli3y>	-ily	›	-y
ylp0.	-ply	›	-ply
yl2>	-ly	›	-
ygo1.	-ogy	›	-og
yhp1.	-phy	›	-ph
ymo1.	-omy	›	-om
ypo1.	-opy	›	-op
yti3>	-ity	›	-
yte3>	-ety	›	-
yti2.	-ity	›	-i
yrtsi5.	-istry	›	-
yra3>	-ary	›	-
yro3>	-ory	›	-
yfi3.	-ify	›	-
ycn2t>	-ncy	›	-nt
yca3>	-acy	›	-
zi2>	-iz	›	-
zy1s.	-yz	›	-ys

Příloha C. - Sekvenční diagram přechodu mezi pohledy při provádění stemmingu.



Příloha D. – Anglický stop list obsahující 174 slov.

a
about
above
after
again
against
all
am
an
and
any
are
aren't
as
at
be
because
been
before
being
below
between
both
but
by
can't
cannot
could
couldn't
did
didn't
do
does
doesn't
doing
don't
down
during
each
few
for
from
further
had

hadn't
has
hasn't
have
haven't
having
he
he'd
he'll
he's
her
here
here's
hers
herself
him
himself
his
how
how's
i
i'd
i'll
i'm
i've
if
in
into
is
isn't
it
it's
its
itself
let's
me
more
most
mustn't
my
myself
no
nor
not

of
off
on
once
only
or
other
ought
our
ours
ourselves
out
over
own
same
shan't
she
she'd
she'll
she's
should
shouldn't
so
some
such
than
that
that's
the
their
theirs
them
themselves
then
there
there's
these
they
they'd
they'll
they're
they've
this
those

through
to
too
under
until
up
very
was
wasn't
we
we'd
we'll
we're
we've
were
weren't
what
what's
when
when's
where
where's
which
while
who
who's
whom
why
why's
with
won't
would
wouldn't
you
you'd
you'll
you're
you've
your
yours
yourself
yourselves