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

Bakalářská práce

Stemmer pro češtinu

Plzeň 2012 Petr Zápotocký

Prohlášení

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

V Plzni dne 10. května 2012

Petr Zápotocký

Abstract

This work is focused on usage and development of stemmers. Stemmers are an important preprocessing tool for many advanced Natural Language processing tasks. The first part of the work covers theoretical knowledge, including stemming algorithms. The second part is an implementation of the chosen algorithm. At the end, the reader should be able to construct own stemmer.

Obsah

1 Úvod

V dnešní technologicky vyspělé době je čím dál častější potřeba pracovat s textem na vyšší úrovni a nekončíme tudíž jen u pouhého čtení. Potřebujeme s textem dále pracovat, vyhledávat souvislosti a využít tak všechny jeho vlastnosti a nesený význam. To ovšem není vůbec snadná záležitost. Zejména pak u flektivních (ohebných) jazyků, jakým je například čeština. Ke zpracování takovýchto textů se pak používají různé systémy, které se soustředí na určitý aspekt. Patří mezi ně například lemmatizátory nebo právě stemmery, které mají za úkol určit kořen daného slova. Stemmerů se pak tedy využívá například při vyhledávání, kdy chceme nalézt výsledky založené jen na kořeni slov.

Druhým cílem této práce byla implementace vybrané stemmingové metody a následné testování. Znamená to tedy práci s jakýmkoliv textem, který je následně vyhodnocen. V tomto případě je stemmingová metoda použita na odřezávání předpon (prefixů), což je velice dobrý základ pro jakýkoliv stemmer a zároveň je při tom dobře vidět celý princip.

Po přečtení této práce by měl čtenář mít přehled o stemmingu a jeho metodách. Díky tomu by si měl také udělat obrázek o náročnosti případné implementace a případných rozdílech vzhledem k různým jazykům. Díky implementované stemmingové metodě by měl také čtenář dostat takové základní informace, aby byl snadněji schopen zhotovit svůj vlastní stemmer založený na stejném nebo podobném principu.

2 Slovotvorba

Základem při práci s češtinou a se slovy je znalost slovotvorby. Díky tomu se pak snáze orientuje při tvorbě stemmeru, ale i lemmatizátoru a také při hledání optimálních algoritmů, které by splnily všechny požadavky. Jak je uvedeno v [Dr. Marek Nekula(2003)], slovotvorba se zabývá formou a významem jednoslovných pojmenování vzniklých na základě pojmenování už existujících, popř. procesem jejich vzniku. V závislosti na způsobu, jakým určité slovo vzniklo ho můžeme rozčlenit na určité části. Celou slovotvorbu pak můžeme rozdělit na dvě základní metody, *derivaci* a *kompozici*. Pod derivací si můžeme představit odvozování a pod kompozicí se skrývá skládání slov.

2.1 Kořen

Základem každého slova je *kořen* (viz [Dr. Marek Nekula(2002)]). Je to část slova, která je dále nečlenitelná a zároveň nesoucí základní význam. Při tvoření slov se ke kořeni připojují různé afixy. Pokud se jedná o připojení afixu před kořen, mluvíme o prefixu (předponě). V opačném případě se jedná o sufix (příponu).

2.2 Prefix (předpona)

V [Dr. Marek Nekula(2002)] je uvedeno, že prefix patří k tzv. slovotvorným formantům. Prefix rozšiřuje základové slovo a tím tak vytváří slovo nové, které ovšem nemění slovní druh. Takto nově vytvořené slovo, ale má oproti původnímu pozměněný význam. Oproti sufixu (příponě) je prefix jednodušeji rozpoznatelný. Prefixací nazýváme proces, kdy se prefixy při tvorbě slov připojují k základům.

2.3 Sufix (přípona)

Sufixy samozřejmě jako prefixy také patří mezi slovotvorné formanty a spolu s nimi jsou souhrnně označovány jako afixy. Sufix je zase opět jako prefix

součástí odvozeného slova a následuje za kmenem slova. Je to tedy jakýsi opak předpony, ovšem v tomto případě je přípona svázána s původním slovem těsněji a mění se tím také význam a morfologická povaha (slovní druh) základní slova. Samozřejmě stejně jako existovala u předpon prefixace, existuje podobný proces i u přípon a nazývá se logicky sufixace. Je to vlastně také odvozování nových slov ze slov základních, ale zde tedy může docházet narozdíl od prefixace ke změně slovního druhu. Vše je také uvedeno v [Dr. Marek Nekula(2002)].

2.4 Koncovka

Součástí přípony může být také koncovka, která však může být za slovním základem i sama o sobě. Většinou se připojuje až na samý konec původního slova. Existuje také speciální případ sufixu nazývaný postfix, který je připojen až za koncovku a zůstává v nezměněném tvaru [Dr. Marek Nekula(2003)].

2.5 Slovotvorný základ

Slovotvorný základ musíme chápat jako slovo nebo kořen, který předcházel vytvoření slova nového. Může se tak jednat pouze o kořen nebo naopak delší až rozsáhlé slovo, které ovšem stále funguje jako základ pro vytvoření nového slova, které je jakousi jeho nadstavbou (viz [Dr. Marek Nekula(2003)]).

2.6 Další slovotvorné postupy

Existuje celá řada dalších speciálních slovotvorných postupů, pomocí kterých jsou tvořena slova (viz [Dr. Marek Nekula(2003)]). Prvním z nich je například *konekt*. Ten je použit při tvorbě slov pomocí kompozice a spojuje k sobě členy, které následně tvoří nové slovo (například život-o-pis). Pokud je přední člen následné složeniny pozměněn, jedná se o *spřežku*. Pokud jsou slova tvořena zmíněnými postupy, nastává někdy situace, že se mění různé hlásky. Jedná se o *hláskovou alternaci*. Zvláštním prípadem jsou potom alternace *vznikové* a *zánikové*. To jsou případy, kdy se při vzniku nového slova ubere nebo naopak přidá určitá hláska (pes - ps-í / teplo - tepel-ný).

Jsou tu ale také metody, které úplně mění slovní druh a celý význam slova. Takovému slovotvornému postupu se říká *mutace*. Pokud se ale jedná pouze o změnu slovní druhu při zachování původního významu, nazývá se tento způsob *transpozice*. Dále se pak ve slovotvorbě využívá *modifikace*, která pouze například připojením určitého prefixu upřesňuje určitým způsobem význam původního slova opět při zachování slovního druhu. Nakonec se samozřejmě česká slovní zásoba musí vypořádat s přejatými cizími slovy. Pokud se tato cizí slova mají bez problémů používat, potřebují upravit pomocí slovotvorného afixu. Tomuto způsobu se říká *adaptace*.

3 Metody stemmingu

Metody stemmingu se v základu rozlišují na dva přístupy. Jeden z přístupů je založen na bázi znalosti morfologie daného jazyka. Znamená to, že se snažíme všechna slova zanalyzovat na základě specifických pravidel daného jazyka. Druhý přístup prosazuje statistický model, který se snaží najít jakousi databázi všech možných kořenů a podle ní následně volit správnou možnost. Většinou se však používá model, který se skládá z těchto dvou různých přístupů v různém poměru.

Kvůli dostupné technologii se zpočátku více využívalo přístupu, který využíval morfologické znalosti jazyka, jelikož statistické metody jsou více náročné na hardwarový výkon. V dnešní době však již technika natolik pokročila, že se tohoto způsobu začíná také více využívat. Obecně však oba tyto základní přístupy ke stemmingu dále dělíme dále na jednotlivé metody založené na základě různých algoritmů. Přehled těchto algoritmů byl čerpán z [Smirnov(2008)].

3.1 Triviální algoritmus

Tento jednoduchý algoritmus se v reálných systémech moc nepoužívá, ale slouží spíše jako odrazový můstek dalších vylepšených algoritmů (více popsáno v [Harman(1991)]). Tento princip je založen na jednoduchém ořezávání slov v daném místě. Dalším možným přístupem je tzv. "S"-stemmer, který pracuje s anglickými podstatnými jmény v množném čísle a modifikuje je podle daných specifických pravidel na stejná podstatná jména ovšem v čísle jednotném *(3.1)*.

3.2 Lovinsové algoritmus

Julie Beth Lovinsová byla vůbec první, kdo zveřejnil popis stemmeru (více v [Lovins(1968)]). Tento stemmer zkoumal anglický jazyk a definoval 294 konců slov, každý spojený s jednou z 29 podmínek, plus 35 transformačních pravidel. Každé z procházených slov končících na určitou část splňující tyto podmínky a pravidla bylo o tuto část zkráceno. Například pro anglické slovo

```
if slovo končí na "ies", ale nekončí na "eies" nebo "aies" then
  ies = y
else
  if slovo končí na "es", ale nekončí na "aes", "ees" nebo "oes" then
    es = e
  else
    if slovo končí na "s", ale nekončí na "us" nebo "ss" then
      s = NULL
    end if
  end if
end if
```

"nationally" byly nalezeny dvě možné koncovky "ationally"a "ionally". První možnost byla ovšem vyloučena na základě pravidla o minimální délce kmenu. Tato délka byla v anglickém jazyce nastavena na tři znaky. Za kmen byla tedy považována část "nat". Transformační pravidla se uplatňovala například u zdvojených souhlásek.

3.3 Porterův algoritmus

Tento algoritmus uplatňuje transformační pravidla. Každý krok se skládá z pravidel ve formě <podmínka> <sufix> -> <nový sufix>. Například pro pravidlo, které se zabývá změnou koncovky "eed"na "ee", je podmíněno obsahem alespoň jedné samohlásky ve zbytku slova. Například u anglických slov "agreed"a "feed"se v prvním případě provede změna na "agree", zatímco druhé slovo zůstane kvůli dodržení zmíněné podmínky nezměněno. Tento algoritmus je velmi výstižný a čitelný pro programátora. Je v něm zapracováno kolem 60 pravidel. Je také efektivní z hlediska výpočetní složitosti. Jsou zde ale také určité nedokonalosti (například "police"/"policy"), které se můžou řešit pomocí dodatečného slovníku ([Lovins(1968)]).

3.4 Paice-Huskův algoritmus

Je to iterační algoritmus s jednou tabulkou, která obsahuje kolem 120 pravidel indexovaných podle posledního písmena přípony. V každé iteraci se algoritmus snaží najít aplikovatelné pravidlo na poslední znak slova. Pokud takové

pravidlo neexistuje, algoritmus končí. Podmínka konce je také závislá na délce slova. Pokud slovo začíná na samohlásku a případném odtržení by zbývaly pouze dvě písmena, algoritmus opět končí. To samé platí pro slova, která začínají souhláskou a zbývaly by při odtržení pouze tři znaky ([Paice(1990)]).

3.5 Dawsonův algoritmus

Tento algoritmus lze považovat za vylepšení Lovinsové algoritmu. Základem jsou tedy také specifická pravidla a podmínky spojené s konkrétními konci slov. Vše je popsáno v ([Dawson(1974)]). Tento alogritmus ovšem využívá opravdu velice propracovaného seznamu anglických přípon spolu s transformačními pravidly. Jedná se asi o 1200 záznamů. Přípony jsou uloženy v obráceném pořadí a indexovány délkou a posledním znakem. Pravidla pak definují, jestli může být nalezený sufix odstraněn. Tento algoritmus se pro svou složitost nestal zcela oblíbeným.

3.6 N-gram algoritmus

Do teď byly zmiňovány algoritmy založené na odsekávání afixů (předpony a přípony) na základě různých pravidel a podmínek. Existují však také statistické metody, které nepracují na základě znalosti slovotovrby daného jazyka, ale využívají ke své práci různé počty výskytů. Díky tomu mohou tedy být tyto metody nezávislé na konkrétním jazyce a mohou být v takřka nezměněé podobě využívány i v jiném jazyce, než pro který byly primárně navrženy. Tento velký problém se závislostí dané stemmingové metody na konkrétním jazyce se snaží řešit právě i tento algoritmus ([James Mayfield(2003)]), který byl úspěšně testován na osmi evropských jazycích. Celý princip je založen na prozkoumávání daného textu a následného vyhodnocování počtu výskytů N-gramů zkoumaného slova. N-gramem se v tomto případě stává n po sobě jdoucích znacích daného slova, kdy se navíc počítá prázdný znak na začátku a na konci slova. Myšlenka je totiž založena na předpokladu, že neměnné unikátní kořeny se v textu vyskytují méně často než různé afixy, které se vážou i na různá jiná slova a kořeny. Použití bylo například demonstrováno u anglického slova "juggling"a 4-gramu, což bylo prezentováno na konherenci CLEF 2002 (Conference and Labs of the Evaluation Forum), viz [Martínez(2006)]. Je zde totiž patrný nesrovnatelně vysoký výskyt klasické anglické přípony

”ing”, což dokazuje jasnou klasifikaci této přípony, aniž bychom předem znali nebo používali jakékoliv morfologické pravidlo daného jazyka.

3.7 HMM Algoritmus

Tento stemmingový algoritmus patří také do staistického přístupu. Je založen na HMM (v přkladu skrytých Markových modelech). Stejně jako u většiny statistckých stemmingových metod zde není potřeba hlubší znalosti daného jazyka, tudíž tento algoritmus ani není založen na předdefinovaných gramatických vlastnostech. Algoritmus HMM je založen na bázi konečného automatu a danými přechody na základě pravděpodobnostní funkce. Každý znak slova zde představuje stav konečného automatu. Na základě rozdělení na dva stavy (kořeny, kořeny a přípony) je postupně nacházena nejpravděpodobnější cesta v grafu konečného automatu (viz [Melucci Massimo(2003)]).

3.8 YASS Algoritmus

Tento algoritmus se zabývá problémem se shlukováním za předpokladu předem neznámého počtu shkluků. Většinou se v těchto případech využívá hierarchické metody, která počítá vzdálenosti mezi jednotlivými elemnty daného shluku. Ve výsledných shlucích se pak nachází hledaný kořen, který je vytvořen jádrem tohoto shluku. Funkce pro udělování penalizace vzhledem ke vzdálenosti je také v [Majumder Prasenjit(2007)] uvedena pro dva řetězce a vypadá následovně (3.1 a 3.2).

$$p_i = \begin{cases} 1 & \text{když } x_i = y_i \quad 0 \leq i \leq \min(\text{m, n}) \\ 0 & \text{jinak} \end{cases} \tag{3.1}$$

Samotná vzdálenost se pak počítá takhle.

$$D(X, Y) = \sum_{i=0}^{n} \frac{1}{2^i} p_i \tag{3.2}$$

Pro roztřídění jednotlivých elementů do správných shluků musí být zvolen určitý práh. To je však celkově velice závislé na hardwarovém výkonu, stejně

jako většina statistických stemmingových metod.

3.9 Stemming založený na korpusu

Tento případ se snaží zbavit problémů, které jsou spojené s odřezáváním dané části slov, které jsou si syntakticky velice podobné, ovšem sémanticky zcela rozdílné. Mohou se pak nesrpávně sdružovat velice podobné výrazy, které jsou ovšem svým významem velice rozdílné a stemmingový přístup ke každému slovu také. Tento algoritmus se proto snaží omezovat toto sdružování slov na základě myšlenky, že se některá slova nevyskytují ve stejném typu textu spolu. Další motivací tohoto algoritmu je rozdíl úspěšností jednotlivých alogritmů na různých typech textu. Může se totiž stát, že určitá metoda bude velice úspěšná na výročních zprávách společností a naopak selže na sportovních reportážích. Z těchto důvodů je proto tento algoritmus založen na používání různých korpusů určených pro dané tématické texty. Více je uvedeno v [Jinxi Xu(1998)].

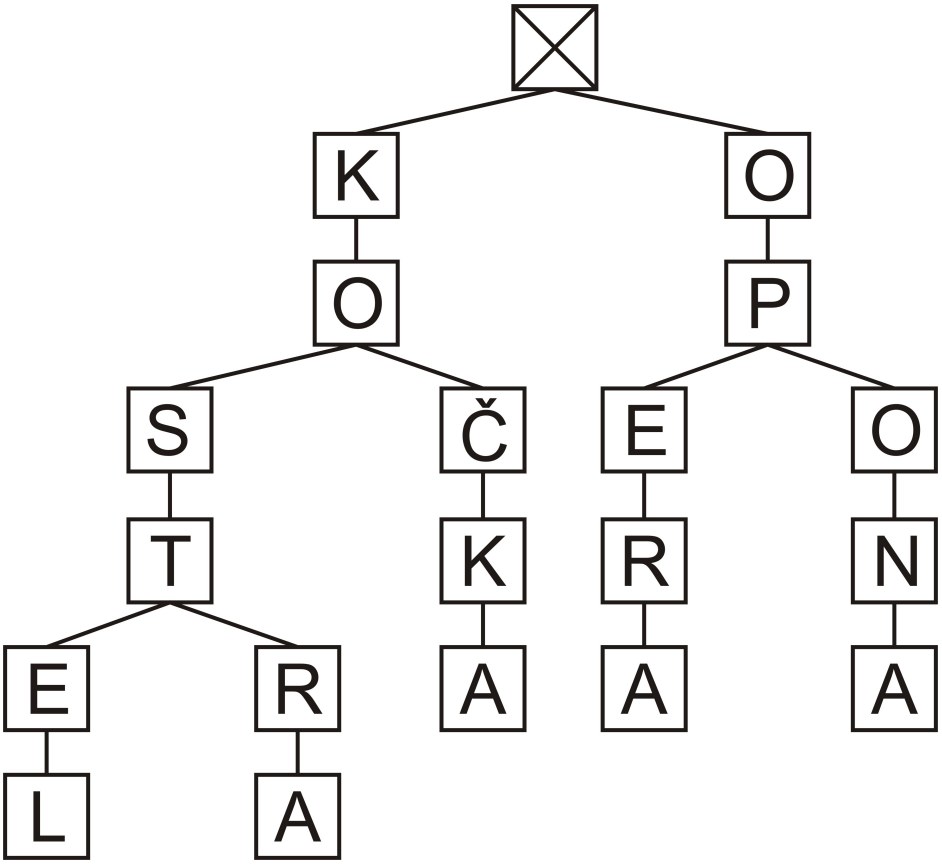
4 Trie

Pro uložení všech předpon je z důvodu dobré komprese dat a také z důvodu lepšího prohledávání použita datová struktura *trie*. Tento datový typ je hojně používán u elektronických slovníků, kdy jsou všechna slova uložena právě pomocí *trie*. Díky tomu je tak vyhledávání v těchto slovnících velice snadné, efektivní a méně náročné. Zároveň ale také veškerá slova obsažená ve slovníku zdaleka nezabírají takový paměťový prostor, který by obsadily při klasickém uložení jednoho slova za druhým. Více o těchto strukturách je možné najít v [Wróblewski(2004)].

4.1 Klasická trie

Trie je vlastně stromová struktura, která v sobě samozřejmě jakožto každá stromová struktura obsahuje uzly a hrany. *Trie* tedy také obsahuje všechny tyto klasické rysy stromové struktury a její základní princip spočívá v tom, že se stejná písmena nebo části slova od jeho začátku při jeho vícenásobném výskytu neukládají znovu a znovu. Od tohoto jedinečného principu se pak odvíjí zmíněná účinnost komprese a také rychlost vyhledávání. Nejlépe je však celý princip vidět na příkladu. Klasická *trie* používaná pro ukládání a vyhledávání textu vypadá následovně *(Obr. 4.1)*.

Jak je tedy patrné z obrázku, tento typ *trie* je tvořen pomocí základního prázdného uzlu. Z tohoto hlavního uzlu se pak odkazuje na další uzly v nižší úrovni. Tyto odkazy jsou na obrázku zobrazeny jako hrany. Celým stromem se postupuje od shora dolů. V první úrovni pod hlavním uzlem jsou tedy vedena všechna jedinečná a použitá počáteční písmena jako uzly. K těmto prvním písmenům se vytvoří další úroveň uzlů, kde se bude odkazovat na všechna druhá písmena v pořadí vázaná na dané písmeno první. Takto vznikne tolik větví, kolik různých druhých znaků navazuje na znak předchozí. Tento postup pokračuje do té doby, než se vyčerpají všechny možnosti následujících znaků, které plynou ze zadaných řetězců.



Obrázek 4.1: Nástin trie na základě slov: kostra, kočka, kostel, opera, opona.

4.2 Další verze trie

Je třeba zmínit také *komprimovanou trie*. Tato stromová struktura je principiálně stejná jako trie klasická. Neexistují zde ovšem tzv. redundantní uzly. To jsou takové uzly, které mají pouze jednoho následovníka. Jednoduše řečeno je to případ, pokud na daný znak existuje pouze jediný znak pokračující. V tomto případě se tedy do jednoho uzlu spojí dva či více znaků, které splňují tuto podmínku. Pro kompletnost by zde také měla být zmínka o *sufixové trie*. Zde se strom pro daný řetězec vytváří ze všech sufixů od nejkratšího k nejdelšímu.

5 Maximum entropy modeling

Maximum entropy modeling, česky řečeno modelování s maximální entropií, je způsob, pomocí kterého se dají klasifikovat informace z různorodých zdrojů. Problém s touto klasifikací je založen na potenciálně velkém počtu příznaků, které mohou být celkem složité a využívají předem získaných znalostí o důležitosti a očekávaném výskytu jednotlivých specifických vlastností při následné klasifikaci. Každý tento příznak následně odpovídá určitému specifickému omezení v rámci specifikace. Ze všech modelů vyhovujících daným omezením se vybere model s maximální entropií. V ideálním případě by měly být zadány všechny potenciálně důležité informace před začátkem klasifikace a následně ponechány trénovacímu procesu pro vytvoření nejlepšího modelu.

Pro každou danou sadu příznaků se počítá očekávaný výskyt jednotlivých příznaků na základě trénovací množiny. Příznaky jsou tedy binární funkce v následujícím tvaru *(5.1)*.

$$p(x,y) = \begin{cases} 1 & \text{když } x > 0 \text{ a } y = 1 \\ 0 & \text{jinak} \end{cases} \tag{5.1}$$

kde x vektor vstupního prvku (v našem případě se zde jedná o aktuální slovo) a y je označení třídy (v našem případě se zde jedná o počet odtržených písmen).

Nyní je zapotřebí zjistit rozdělení pravděpodobnosti $p(y|x)$, kde x je aktuální slovo a y je označení třídy. Potřebujeme tedy zjistit maximální entropii všech rozdělení v souvislosti s $p(y|x)$.

Nejlepší rozdělení pravděpodobnosti se pak počítá pomocí následujícího tvaru *(5.2)*.

$$p(y|x) = \frac{1}{Z(x)} \exp \sum_{i=1}^{n} \lambda_j f_j(x,y) \tag{5.2}$$

$$Z(x) = \sum_{y} \exp \sum_{i} \lambda_i f_i(x,y)$$

Kde Z je pouze normalizační konstanta, která zajišťuje rozdělení prav-

děpodobnosti $p(y|x)$, n značí počet příznaků a λ je váha daného příznaku. V našem případě jsou tedy používány funkce $p(x, 0)$ a $p(x, 1)$. V nejjednodušším případě se zvolí třída s větší pravděpodobností.

6 Formát dat

Pro správný chod programu jsou zapotřebí patřičná vstupní data. Prefixy, trénovací data a testovací data.

6.1 Prefixy

K uložení všech předpon byla použita stromová datová struktura *trie*. Tato implementace byla poskytnuta panem Ing. Michalem Konkolem. Výpis předpon je připraven v textovém vstupním souboru, kde je každá předpona uložena na nové řádce. Z tohoto souboru se tedy následně ukládají postupně všechny předpony do trie.

6.2 Trénovací data

Trénovací data jsou zapotřebí k tzv. natrénování klasifikátoru. Jedná se o vstupní soubor, ve kterém jsou již označkována všechna slova patřičnou hodnotou. Tento trénovací textový soubor má proto specifickou strukturu. Na každé řádce se nachází jedno slovo z naší zvolené trénovací množiny slov a po mezeře následuje přirozené číslo (rozuměno v oboru informatiky - do množiny je započítávána i nula), které určuje počet písmen, jenž by měla být správně při stemmingu odtržena. Při načítání se tak z tohoto souboru postupně vytvoří datová struktura typu *List*, která je plněna speciálně vytvořenými objekty *Word*. Tento objekt obsahuje vždy svůj název (*name*), což je vlastně konkrétní slovo, a také patřičnou značku (*label*), která určuje počet odtrhnutých znaků. Ve výsledku tak máme k dispozici pole objektů typu *Word*, ve kterém jsou uložena námi označkovaná trénovací data.

6.3 Testovací data

Samozřejmostí je také načtení testovacího souboru, který je předložen klasifikátoru a následně vyhodnocen. Text z tohoto souboru je po načtení zklasifikován a označkován. Jedná se tedy o klasický textový soubor, ve kterém je

uložen námi požadovaný text pro klasifikaci v klasickém textovém formátu. Tento vstupní text je programem postupně načítán a jsou z něj parsována jednotlivá slova, která se podobně jako u trénovacích dat ukládají do datové struktury *List* ve formě objektů typu *Word*. Ovšem zde se narozdíl od trénovacích dat nevyužívá zápisu značek, jelikož samozřejmě testovací text označkován není. Nicméně se využívá stejného způsobu načítání a proto jsou tyto značky implicitně nastaveny na hodnotu 0. Tato hodnota se poté při klasifikaci upravuje dle výsledků klasifikátoru.

7 Klasifikační metoda

Celý klasifikátor funguje na principu učení a následné klasifikace neoznačených dat. Jsou mu proto nejprve předloženy trénovací data, ze kterých se ještě před tím vytvoří objekty jedntolivých slov *Word* s udanou značkou počtu odtržených znaků. Tyto objekty jsou pak postupně zpracovávány klasifikátorem, který se postupně toto označení učí a připisuje mu různou pravděpodobnost. Po tomto procesu se naučenému klasifikátoru nechá zpracovat a zklasifikovat neoznačená testovací data.

7.1 Tvorba příznaků

Vlastnosti klasifikátoru jsou určeny zejména vytvořenými příznaky, na základě kterých se pak klasifikátor učí rozpoznávat správný počet odtržených znaků. Jednotlivé příznaky patří pod záštitu třídy *FeatureExtractorManager*, kam se postupně přidávají jako nově vytvořené objekty. Každý příznak jednotlivě se tvoří implementací rozhraní třídy *FeatureExtractor*. Toto rozhraní tvoří tři metody. Jednou z nich je metoda *train(TrainningInstanceList instance)*, která slouží k jakémusi trénování na již prošlém textu a zpřesňuje tak další přidělování příznaků. Druhou použitou metodou je *extractFeature(InstanceList instances, FeatureVectorGenerator generator)*, která vlastně přesně definuje daný příznak. Algoritmus, který je zde používán tak udává přesný účinek daného příznaku a odvíjí se od něj celková účinnost stemmeru. Specifická místa slova závislá na daném příznaku se zde označkují indexem 1. Třetí doplňující metodou je *getNumberOfFeatures()*, která vrací počet možností, které můžeme označkovat číslem jedna v daném příznaku.

7.2 Trénování klasifikátoru

Trénování klasifikátoru je samozřejmě závislé na vytvořených trénovacích datech. Tato data se musí přetvořit v objekt třídy *TrainingInstanceList*. Probíhá to tak, že se postupně čtou objekty třídy *Word*, ze kterých se tvoří pole slov a přiřazených značek. Toto následně obsahuje vytvořený *TrainingInstanceList*. Ke trénování klasifikátoru je pak nutný také tzv. trenér, který

je vytvořen na principu maximální entropie a používá vytvořený *FeatureExtractorManager*.

7.3 Použití klasifikátoru

V první řadě se tedy musí klasifikátor natrénovat. Poté na řadu přichází práce s neoznačeným testovacím textem. Podobně jako při trénování musí být testovací data přetransformována do používaného *BasicInstanceList*, což je vlastně pouze list slov s nulovými značkami. Nyní se může přistoupit k samotné klasifikaci. Z klasifikátoru pak dostáváme vyhodnocené značky přiřazené testovacím slovům.

8 Příznaky - Features

Učení klasifikátoru je určeno zejména jeho příznaky (features), které udávají výsledný směr učení a také se od nich odvíjí celkový výsledek klasifikace. Obecně tyto vlastnosti pracují na principu indexování pozic daného slova. Na základě určité vlastnosti se poté na vybraných pozicích objevuje index 1, který udává právě určitý specifický příznak spojený s touto pozicí. Pokud se některý z indexů nemění, je implicitně nastaven na hodnotu 0, což zaručuje, že daná pozice slova není spojena s žádným specifickým příznakem.

8.1 Slovník - Dictionary

Všechny řetězce vyvtořené ze vstupních označkovaných dat jsou uloženy do datové struktury typu *HashSet*. Tato struktura je zvolena z důvodu rychlého vyhledávání, zda se slovo ve slovníku vyskytuje. V této struktuře se totiž nevyskytují stejné položky vícekrát. Uložené řetězce jsou vytvořeny odtržením požadovaného počtu znaků od původního slova. Účinnost tohoto příznaku je pak založena na případném výskytu již označeného slova, což nám dává jistotu správnosti odtržení. Tedy pokud jsou trénovací data označkována správně. V souvislosti s tímto příznakem se kontroluje případný výskyt zbylého podřetězce vzniklého odtržením předpony. Pokud je nalezena jakákoliv shoda s nějakým podřetězcem, je na dané pozici změněn index na hodnotu 1.

8.2 Předpony - Prefix

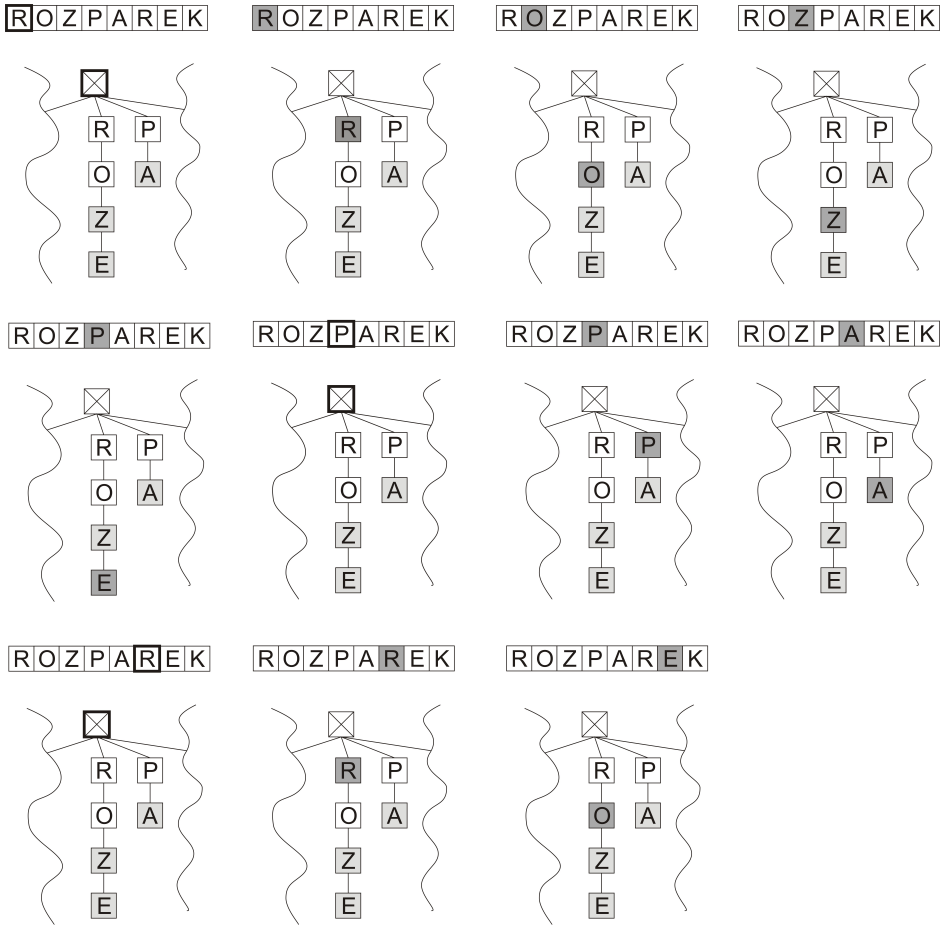
Celá datová struktura *trie* vytvořená z prefixů je použita k vyhledávání všech možných předpon obsažených v daném slově. Načtené slovo se prochází znak po znaku od začátku do konce. V závislosti na tomto postupu se prochází zmíněná *trie*. Vyhledávání totiž probíhá následovně. Začíná se samozřejmě prvním znakem slova. Zde nastává první kontrola souběžně s *trie*. Pokud se tento znak v této struktuře nachází, pokračujeme v kontrole dalšího znaku v pořadí. V opačném případě se začíná testovat od kořene *trie*. Vždy, když je objevena shoda s jakoukoliv předponou, je dané místo označeno jako odtržitelné. Ve výsledku tento použitý příznak prosazuje odtržení na místech

existujících českých předpon. Příkladem může být analýza slova rozprava na obrázku *(Obr. 8.1)*, kde je vždy vidět část struktury prefixů. Tmavě šedé označení ukazuje právě provnávaný znak a světle šedé označení zvýrazňuje ukončení použitelné předpony, která začíná u kořene trie. Je vidět, že se tedy vždy začíná od kořene. Poté se vždycyk kontroluje, zda existuje potomek shodný tímto znakem. V principu to znamená testování, jestli existuje předpona začínající na dané písmeno. Pokud však existuje další písmeno shodné s potomkem v trie zanoříme se o úroveň níže a odkazujeme nyní na toto písmeno, za kterého vedou další odkazy na možná pokračování dosavadního řetězce. V každé této fázi se také ověřuje, jestli dosud dosažená část předponou. Pokud dalšímu znaku procházeného slova není nalezen další odkaz na písmeno v nižší úrovni a vezmeme poslední dosaženou předponu. V daném místě se v trie odkážeme znovu na kořen a začíná se znovu s hledáním případné další předpony. Na konci obrázku je vidět, že další písmeno už se s potomkem neshoduje a dosavadní část předponou není. V takovémto případě celý algoritmus končí a u slova rozparek dostaneme dvě možné předpony (roz, pa). Časová složitost tohoto algoritmu je $O(n^2 \times x)$, kde x značí délku analyzovaného slova a n značí počet analyzovaných slov.

Zde bych chtěl jen dodat, že byla souběžně vyvíjena i další metoda vyhledávání všech možných předpon, která byla založena na testování shody předpon a všech možných podřetězců daného slova. Znamenalo to tedy, že se v daném řetězci vzal první znak a porovnal se se všemi možnými předponami. V dalším kroku se vzaly první dva znaky a provedlo se stejné porovnání. Tento postup následoval až do té doby, než se nekontrolovalo celé slovo. Poté se první znak slova vymazal a celý postup se opakoval znovu a znovu, než bylo smazáno i poslední písmeno původního slova. Případné shody určitého podřetězce s předponou byly ukládány do matice. Pozice v matici závisela na pořadovém čísle předpony a pořadovém čísle průchodu slova. Ve výsledku se ovšem tato metoda ukázala jako ne zcela ideální.

8.3 Délka slova - Size

Tato vlastnost je založena na maximálním možném počtu odtržitelných znaků. V principu to znamená, že je označován poslední možný (desátý) odtržitelný znak. Pokud klasifikované celé slovo této délky ani nedosahuje, počítáme za poslední odtržitelný znak poslední znak celého slova. Ve výsledku nám tedy tento příznak zajišťuje, aby nebylo odtrženo více znaků než je reálně v češtině možné. Tato situace může nastat, pokud by se například slovo skládalo z



Obrázek 8.1: Příklad rozboru slova ”rozparek”.

předpon, které by se klidně všechny mohly odtrhnout, ovšem reálně by se jednalo jen o shodu tvarů těchto prefixů a ne přímo o předpony. Samozřejmě je tu podmínka dostatečné délky slova, při které je tento příznak účinný.

9 Testování

Testování je založeno na principu porovnávání stejného textu, který máme označkován. Tento text je zklasifikován klasifikátorem a jsou tím pádem vytvořeny také patřičné značky. Ve výsledku se pak porovnává vždy k danému slovu původní značka (vytvořená ručně) se značkou určenou klasifikátorem. Podle počtu shodných značek se pak určuje výsledná přesnost celého algoritmu.

Při testování dané implementace stemmingové metody byly označkovány čtyři typy různých textů. Rozumějme texty z různých zdrojů a také s různým zaměřením. Všechny tyto texty byly následně ručně označkovány dle českých jazykových pravidel tak, aby odpovídaly správnému oddělení předpon. Ze všech textů tedy ve výsledku vznikl zdrojový soubor, který obsahoval všechna slova daného textu s příslušnou značkou označující počet odtržených písmen. Velikost těchto testovacích souborů byla v rozmezí sta až tisíce slov.

Samotné testování probíhalo následovným způsobem. Všechny testovací soubory měly svoji kopii, ovšem bez ručně přiřazených značek. Je zde tedy nutné podotknout, že při práci klasifikátoru má neoznačené slovo implicitně nastavenou značku na 0, což znamená žádný odtržený znak. Musíme si tedy představit dvojí stejné soubory, kde první má za každým slovem značku (číslo) a druhý nemá za slovy nic (klasifikátor pracuje s nulou). V principu byl klasifikátoru vždy předložen neoznačený text, který byl následně zklasifikován a označen. Nyní tedy existoval textový soubor, který měl původní (ručně přidělené) značky a nový, který obsahoval značky přiřazené klasifikátorem. Na základě odlišností těchto dvou souborů mohlo být založeno testování účinnosti.

Přesnost dané stemmingové metody tedy spočívala v porovnávání jednotlivých slov a jim přidělených značek na základě ručního značkování a na základě přidělení klasifikátoru. U každého slova byly tedy porovnány tyto dvě značky a pokud se vyskytla shoda, byla zvýšena hodnota úspěšné klasifikace o jedničku. Po ukončení klasifikace byl proveden výpočet přesnosti následujícím způsobem.

$$p = \frac{\sum good}{\sum word} \tag{9.1}$$

Kde p značí přesnost, good značí počet slov se správně určenými značkami a word značí celkový počet slov.

Tato přesnost však byla ještě dále zjemňována, jelikož v úplně prvním provedeném testu se klasifikátor učil z jednotného označkovaného textu a následná klasifikace a porovnávání probíhalo na textech jiných. Zde se přesnost pohybovala okolo 0,66. Následně byl však použit jiný testovací přístup. K celému průběhu testu byl používán pouze jeden textový soubor, který se ovšem rozdělil na části. Určitá část sloužila k učení klasifikátoru a zbylá část byla následně zklasifikována a porovnána s původními (ručně dosazenými) značkami. Zde se výsledky pohybovaly v závislosti na textu a rozložení velikosti části pro učení a pro klasifikaci od 0,55 do 0,71.

Někomu se může zdát dosažená přesnost spíše menší. Ovšem z mého pohledu je to vcelku dobrý výsledek. Musíte si totiž uvědomit, že účinnost byla závislá pouze na správné nebo špatné klasifikaci. Pokud by klasifikace probíhala pouze na odtržení žádného nebo jednoho písmena, nebyly by to až tak dobré výsledky. V našem případě však mohlo být odtrženo až deset písmen. Znamená to tedy, že přesnost byla vztahována na jednu pravděpodobnostní třídu z jedenácti. Pokud tedy mělo být odtrženo například pět znaků a klasifikátor jich odtrhl šest, je to velice dobrý odhad, ovšem vzhledem k výpočtu patří tento výsledek mezi špatné. A v takovém světle už výsledky vypadají opravdu dobře.

V závěru testování bych chtěl jen podotknout, že největší chyby v klasifikaci byly tvořeny nesprávným odtrháváním znaků v případě, že v originálním textu byla nula. Pokud už nějaké slovo bylo označeno ručně jiným číslem než nula, klasifikátor toto číslo v převážné většině případů odhalil správně.

10 Závěr

Pokročilá práce s textem je zejména u flektivních jazyků složitější, jelikož takovéto ohebné jazyky ovlivňuje časování a skloňování. Mezi takovéto jazyky patří tedy také čeština a je tedy zřejmé, že například stemmery by měl každý, kdo chce s českým textem pracovat na vyšší úrovni, alespoň znát. V lepším případě by však měl mít přehled o jejich principech a v ideálním stavu by je měl umět i upravovat, vytvářet a vyvíjet. Cílem této práce je tedy seznámení čtenáře s činností a principem stemmeru.

V první části čtenář nalezne základní informace o české slovotvorbě, jelikož bez těchto znalostí je další postup zcela zbytečný a neúčinný. Je zde následně také vysvětlen základní princip stemmeru a jsou zde popsány různé použitelné přístupy, na kterých může být stemmer založen. Tato první část také následně obsahuje jakýsi přehled stemmingových metod se základním vysvětlením jejich principu a použití. Druhá část je zaměřena na důležité konstrukce a aspekty použité v následné implementaci daného stemmeru. Třetí část této práce je pak zaměřena na samotnou implementaci zvolené metody, která je ve výsledku otestována. Díky tomu by se měl čtenář zorientovat v implementaci konkrétního algoritmu ve stemmeru, což by mu mělo usnadnit další práci.

S prací jsem spokojen, jelikož splňuje všechny požadavky, které byly na začátku kladeny. Považuji ji za úspěšnou ale také proto, že jsem se v této oblasti velice dobře zorientoval a mohu případně svoje nabyté vědomosti do budoucna využívat.

Literatura

[Dawson(1974)] DAWSON, J. Suffix Removal and Word Conflation. *ALLC Bulletin*. 1974, s. 33–46.

[Dr. Marek Nekula(2003)] DR. MAREK NEKULA, K. *Příruční mluvnice češtiny*. Brno : NLN, s.r.o., 2003. ISBN 80-7106-134-4.

[Dr. Marek Nekula(2002)] DR. MAREK NEKULA, K. *Encyklopedický slovník češtiny*. Brno : NLN, s.r.o., 2002. ISBN 987-80-7106-484-8.

[Harman(1991)] HARMAN, D. How effective is suffixing? *Journal of the American Society for Information Science*. 1991, 42, 1, s. 7–15.

[James Mayfield(2003)] JAMES MAYFIELD, P. M. Single n-gram stemming. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28 - August 1, 2003, Toronto, Canada*, s. 415–416. ACM, 2003. doi: http://doi.acm.org/10.1145/860435.860528. ISBN 1-58113-646-3.

[Jinxi Xu(1998)] JINXI XU, W. B. C. Corpus-Based Stemming using Cooccurrence of Word Variants. *ACM Transactions on Information Systems*. 1998, 16, s. 61–81.

[Lovins(1968)] LOVINS, J. B. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*. 1968, 11, s. 22–31.

[Majumder Prasenjit(2007)] MAJUMDER PRASENJIT, K. M. M. YASS: Yet another suffix stripper. *ACM Trans. Inf. Syst.* 2007, 25, 4.

[Martínez(2006)] MARTíNEZ, F. d. M. A. An evaluation of conflation accuracy using finite-state transducers. *Journal of documentation*. 2006, 62, s. 328–349. ISSN 0022-0418. doi: http://dx.doi.org/10.1108/00220410610666493.

[Melucci Massimo(2003)] MELUCCI MASSIMO, O. N. A novel method for stemmer generation based on hidden markov models. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, s. 131–138, New York, NY, USA, 2003. ACM. doi: 10.1145/956863.956889. Dostupné z: http://doi.acm.org/10.1145/956863.956889. ISBN 1-58113-723-0.

[Paice(1990)] PAICE, C. D. Another stemmer. *SIGIR Forum*. November 1990, 24, 3, s. 56–61. ISSN 0163-5840. doi: 10.1145/101306.101310. Dostupné z: http://doi.acm.org/10.1145/101306.101310.

[Smirnov(2008)] SMIRNOV, I. Overview of Stemming Algorithms. [online], 2008. Dostupné z: http://the-smirnovs.org/info/index.php.

[Wróblewski(2004)] WRóBLEWSKI, P. *Algoritmy : Datové struktury a programovací techniky*. Brno : Computer Press, 2004. ISBN 80-251-0343-9.