MATEMATICKO-FYZIKÁLNÍ FAKULTA
UNIVERZITY KARLOVY

DIPLOMOVÁ PRÁCE

Morfologické značkování textu: automatická disambiguace

Jiří Mírovský

PRAHA 1998

Vedoucí práce: Dr. Jan Hajič

Děkuji vedoucímu diplomové práce, Dr. Janu Hajičovi, za cenné rady, náměty a konzultace a děkuji Mgr. Pavlu Květoňovi za totéž a navíc za ochotné a rychlé modifikace parseru na klíč.

Prohlašuji, že jsem tuto práci vypracoval samostatně s použitím uvedené literatury, a souhlasím s jejím zapůjčováním.

V Praze 20. srpna 1998 Jiří Mírovský

Obsah

Kapitola 1

Úvod

1.1 Motivace cíle

Strojové zpracování přirozeného jazyka (alespoň v textové podobě) je dlouhodobý cíl, který si kladou mnozí počítačově zaměření lingvisté již po dlouhá léta. Ukazuje se, že se jedná o problém spíše těžko než lehko zvládnutelný, jehož řešení je však vysoce žádoucí.

Využití je nasnadě: automatický překlad z jazyka do jazyka, automatická kontrola nebo i oprava pravopisu, komunikace s počítačem v přirozeném jazyce apod..

Pro obtížnost automatického zpracování přirozeného jazyka je potřeba celý problém řešit po menších částech.

Ve většině mně známých přístupů je prvním závažnějším krokem morfologická analýza.

Úkolem morfologické analýzy je přiřadit každému vstupnímu slovu jeho základní tvar (tzv. lemma) a informace o tvaru (tag) (viz dále), tento proces se nazývá *tagování*.

Potíž je v tom, že ze samotného slova často nelze tyto informace jednoznačně určit, např. slovo „mezi" může být předložka i podstatné jméno.

Přiřadit každému slovu ve vstupním souboru všechny možnosti lemmat a tagů není principiálně nijak složité (stačí velký slovník). Mnohem těžší je z těchto všech možností vybrat (na základě kontextu) tu správnou. Jeden možný přístup k řešení tohoto úkolu je obsahem této diplomové práce.

1.2 Motivace způsobu

Metody automatické disambiguace[1] se dělí do dvou odlišných směrů:

- metody analytické
- metody statistické

První přístup (analytický) byl použit například v projektu Grammar Corrector z roku 1997 [2], přístup druhý (statistický) zkoumala ve své diplomové práci například Barbora Hladká v roce 1994 [1], též později ještě s Janem Hajičem, viz [5].

Oba přístupy mají své nedostatky i své přednosti.

Parsing češtiny (použitý v Grammar Correctoru) je omezen bezkontextovou gramatikou, která na zachycení gramatiky českého jazyka nestačí, ale je efektivně[2] zpracovatelná (zásobníkovým automatem).
Výhodou použití gramatiky je možnost zachytit závislosti větných členů, stojících ve větě daleko od sebe.

N-gramový model (testovaný v diplomové práci B. Hladké) nezná omezení bezkontextové gramatiky, jeho nevýhodou ale je vysoká paměťová a časová náročnost, kterou je možno snížit jen za cenu menší spolehlivosti a vzdání se snahy zachytit závislosti slov stojících ve větě daleko od sebe.

Kombinací obou způsobů by možná bylo možno využít jejich výhod a potlačit jejich nevýhody. Zkoumáním toho se zabývá tato diplomová práce.

Obecně lze u mnohých metod zvýšit spolehlivost za cenu zvýšení prostorové nebo časové náročnosti. Je ale ještě jedna možnost. Některé aplikace nemusejí požadovat (jako svůj vstup) text otagovaný jednoznačně, ale stačí jim, aby poměr (počet tagů / počet slov) byl co nejmenší. Zkusme tedy upustit od požadavku jeden tag u jednoho slova a sledujme, jak roste spolehlivost tagování v závislosti na výsledném poměru (počet tagů / počet slov). I tímto se zabývá tato diplomová práce.

[1] disambiguace: určování správného lemmatu a tagu z více možných, zjednoznačňování tagování

[2] efektivně: dostatečně rychle a s malými prostorovými nároky (narozdíl od pojmu „efektivně" v teorii algoritmů)

1.3 Přesné zadání

Takto zní přesné zadání této diplomové práce:

Cíl: Morfologická disambiguace s preferencí spolehlivosti (recall).

Metoda (rámcově): Kombinace gramatiky (za použití CFG parsingu) a statistických dat (automatické učení).
Implementace v prostředí Unix (Linux/Solaris).

Data: čeština (projekty MŠMT 96151 / KAČK K216).

Realizace: Popis metody, algoritmů a dat.
Uživatelská a programová dokumentace k programům.
Grafy výsledné úspěšnosti na testovacích datech (recall / precision, nastavení statistických parametrů).

1.4 O tomto textu

Text této průvodní zprávy sestává z 5 kapitol:

1. ÚVOD - uvedení do problému
2. TEORETICKÁ ČÁST - podrobný popis problému tagování a zvoleného způsobu řešení
3. UŽIVATELSKÁ ČÁST - návod na použití programů této diplomové práce
4. PROGRAMÁTORSKÁ ČÁST - popis konkrétní realizace algoritmů, použitých datových struktur a formátů dat
5. PŘÍLOHY - tabulky výsledků testů, ukázky vstupů a výstupů aj.

V přílohách je též uveden malý slovníček základních pojmů, užívaných v dalším textu (příloha 5.3).

Kapitola 2

Teoretická část

2.1 Morfologická analýza - tagování

Tagování je proces, při kterém je každému vstupnímu slovu přidělena morfologická značka - tag, určující morfologické vlastnosti slova. Tj. např. slovu *hrad* ve větě *Navštívili jsme hrad* je přiřazena značka, nesoucí informaci, že dané slovo je podstatné jméno rodu mužského neživotného ve čtvrtém pádě jednotného čísla (a některé další informace, podrobný popis tagů naleznete v příloze 5.2.1).

Tento úkol se na první pohled zdá být jednoduchý, člověk „koukne a vidí". Ovšem počítač zde má problémy. Jak již bylo řečeno, ze samotného slova (bez kontextu) často není možno určit jednoznačně všechny morfologické kategorie. I u výše uvedeného příkladu, odhlédneme-li od kontextu, ze samotného slova *hrad* nepoznáme, zda je v prvním či čtvrtém pádě.

Program, který každému slovu ve větě bez přihlížení ke kontextu přiřadí všechna možná lemmata a všechny možné tagy[1], existuje.[2] Jmenuje se `ma` a jeho výstupem je text v SGML formátu, kde na každé řádce (až na výjimky) je jedno slovo a za ním všechny jemu příslušné možné značky. Popis tohoto formátu naleznete v příloze 5.2.6, příklad takového textu naleznete v příloze 5.2.2.

Na tomto (mírně upraveném) výstupu programu `ma` pracuje program `disamb` - předmět této diplomové práce.

Úspěšnost programu `ma` je teoreticky stoprocentní (tj. stoprocentní úspěšnosti nebrání žádná principiální překážka), v praxi se ale občas vyskytne

[1] tj. lemmata a tagy, které by toto slovo mohlo mít při různých kontextech

[2] Byl vyvinut a používá se na katedře lingvistiky na MFF UK

slovo, které v databázi programu `ma` dosud není. Pak je samozřejmě doplněno. Vyjímečně se chyba vyskytne i přímo v databázi programu `ma`. Je-li odhalena, je samozřejmě rovněž opravena.

Cílem morfologické analýzy je text (ve formátu SGML), kde u každého slova je právě jeden tag. Proces zjednoznačňování výstupu programu `ma` se nazývá *disambiguace*. Provádí se na základě kontextu, tedy s přihlédnutím ke zbytku věty, a je-li to nutné, s přihlédnutím k ostatním větám textu, v krajních případech s použitím znalostí o světě. To vše v případě, že zjednoznačňování provádí člověk. Počítač může mít jen těžko takový rozhled.

Metod, které může použít k disambiguaci počítač, je celá řada, jak jsem se již zmínil v úvodu práce.

Velmi oblíbené jsou metody statistické, využívající znalostí, získaných z velkého množství ručně jednoznačně otagovaných textů.[3] Jazykových modelů, využívajících takových statistických znalostí, je opět mnoho. Osobně jsem přišel do styku s n-gramovým modelem, který je použit v této diplomové práci, a dále s exponenciálním modelem vycházejícím z teorie maximální entropie. Druhý z těchto modelů je časově velmi náročný a pro praktické použití ve své základní podobě zřejmě nepoužitelný.

Další typ metod je možno nazvat analytickými. Nezakládají se na statisticky získaných znalostech, nýbrž na znalostech člověka, který se snaží přímo napsat pravidla, vyjadřující závislosti tagovacích značek na svém okolí, a na základě těchto pravidel potom vybírat správnou značku (či zahazovat špatné).

Jak jsem rovněž již předeslal v úvodu, v této diplomové práci jde o pokus kombinace dvou vybraných metod disambiguace, konkrétně parsingu češtiny pomocí bezkontextové gramatiky a dále n-gramového modelu.

2.2 Disambiguace pomocí parsingu

Disambiguace pomocí parsingu by se teoreticky dala popsat takto:
Z nejednoznačně otagované věty vybíráme jeden řetězec tagů po druhém[4] a jeden každý zkontrolujeme, zda patří do gramatiky. Tagy, které jsou součástí

[3] Ruční značkování je práce velmi nezajímavá a zdlouhavá, jak sám mám možnost se přesvědčovat, zvláště když člověk zjednoznačňuje text novin pět let starých a navíc výsledky dvaceti zápasů druhé ligy košíkové nebo článek reprezentující autorův názor, se kterým třeba nesouhlasím.

[4] Všechny možné posloupnosti tagů, každá reprezentuje jedno možné vzájemně jednoznačné přiřazení tagů slovům věty, ale jen jedno toto přiřazení je to pravé.

alespoň jednoho řetězce, patřícího do gramatiky, ve výstupu ponecháme, ostatní odstraníme.

V praxi není možno brát jeden řetězec po druhém, protože jich je (u delších vět) příliš mnoho, ale pro popis principu práce disambiguace tato představa vyhovuje.

V této diplomové práci je použit parser z projektu Grammar Corrector [2], používající bezkontextovou gramatiku a pro její parsování analýzu zdola nahoru.

Pro velikou neúspěšnost parsování celých vět (s gramatikou dodanou s parserem), zvolil jsem pro účely této diplomové práce možnost (kterou naštěstí parser podporoval) neparsovat celé věty, ale jen části vět, konkrétně jmenné fráze.[5]

Disambiguace pomocí parsingu v této diplomové práci vypadá tedy takto: Parser poskytne řetězce tagů zparsovaných jmenných frází. Slova věty, která jsou součástí některé z těchto frází, jsou zpracována takto: Jsou ponechány jen tagy a lemmata, které byly součástí alespoň jednoho řetězce tagů poskytnutého parserem. Slova věty, která nejsou součástí žádné z těchto frází, jsou ponechána beze změny (se všemi lemmaty a tagy).

Je vidět, že disambiguace pomocí parsingu ani v obecném případě, ani v případě této diplomové práce, nevede většinou k úplné disambiguaci (tedy ke zjednoznačnění tagování), ale jen ke zmenšení počtu tagů.

2.3 Disambiguace pomocí n-gramového modelu

Jazykové modely n-gramového typu se mohou lišit. Jejich společnou ideou je závislost pravděpodobnosti tagu na jeho malém okolí, velikosti n (nebo $n-1$). Tímto okolím může být např. tag předchozí, aktuální slovo, aktuální lemma, předchozí slovo, následující tag apod..

Ideou modelu použitého v této diplomové práci je, že pravděpodobnost tagu na určitém místě věty závisí na tagách předchozích, a to navíc jen$(n-1)$ předchozích tagách.

Nepřihlíží se tedy ke slovu, kterému tag patří. To umožní na poměrně malých trénovacích datech větší generalizaci. Na druhou stranu není pravda, že by se příslušné slovo úplně ignorovalo, neboť nejpravděpodobnější tag se

[5]jmenná fráze - rozvité podstatné jméno nebo to, co se syntakticky chová jako rozvité podstatné jméno. Např.: ,,budoucí salvadorský prezident", ,,boj o novou tvář" apod.. Zajímavá jmenná fráze (zparsovaná parserem) je uvedena v příloze 5.2.9.

vybírá pouze z množiny tagů, které byly danému slovu přiřazeny programem `ma`.

Zde užitý n-gramový model a vzorec pro určení pravděpodobnosti řetězce tagů věty délky m vychází z této aproximace (označené $\doteq$):

$$p(t_1 \dots t_m) = p(t_1) \cdot p(t_2|t_1) \cdot \dots \cdot p(t_m|t_1, \dots, t_{m-1}) \doteq$$
$$\doteq p(t_1) \cdot \dots \cdot p(t_{n-1}|t_1, \dots, t_{n-2}) \cdot \prod_{i=n}^{m} p(t_i|t_{i-n+1}, \dots, t_{i-1}) \tag{2.1}$$

(Vzorec je poněkud nepřehledný, jasnější bude z následujících vzorců pro 2-gramový a 3-gramový model.)

V praxi je potřeba omezit se na malé n, protože tabulky pravděpodobností by byly pro velká n příliš velké. Navíc při malém množství dostupných trénovacích dat by velké n stejně neznamenalo žádné zlepšení aproximace.

V praxi se tedy používají unigramy, bigramy a trigramy, tj. $n = 1$, $n = 2, n = 3$.

Samotné unigramy nejsou zajímavé díky opravdu špatné aproximaci.

Pravděpodobnost řetězce tagů se v případě bigramů určí dle tohoto vzorce (pro větu délky n):

$$p(t_1...t_n) = p_1(t_1) \cdot \prod_{i=2}^{n} p_2(t_i|t_{i-1}), \tag{2.2}$$

kde
p je definovaná pst řetězce tagů,
t_i je tag u i-tého slova věty, počítáno od 1,
p_1 je unigramová pst získaná na trénovacích datech,
p_2 je podmíněná bigramová pst získaná na trénovacích datech.

Podobně v případě trigramů:

$$p(t_1...t_n) = p_1(t_1) \cdot p_2(t_2|t_1) \cdot \prod_{i=3}^{n} p_3(t_i|t_{i-2}, t_{i-1}), \tag{2.3}$$

kde navíc
p_3 je podmíněná trigramová pst získaná na trénovacích datech.

2.3.1 Získání n-gramových rozdělení

Pravděpodobnostní rozdělení p_1, p_2, p_3 nejsou známa. Proto jsou použita trénovací data - ručně jednoznačně otagované soubory, na kterých jsou naučeny relativní četnosti a ty pak prohlášeny za hledané pravděpodobnosti.

Postup k získání relativních četností z trénovacích dat je uveden v části 3.6.

Konkrétní programové řešení je popsáno v části 4.2.2.

2.3.2 Vyhlazování n-gramových rozdělení

Jak je vidět ze vzorců 2.2 (resp. 2.3), nevyskytla-li se některá z dvojic $t_{i-1}t_i$ (resp. trojic $t_{i-2}t_{i-1}t_i$) v trénovacích datech, bude pst $p_2(t_i|t_{i-1})$ (resp. $p_3(t_i|t_{i-2}, t_{i-1})$) nulová a také výsledná pst $p(t_1...t_n)$ bude nulová. Přitom nepřítomnost takové dvojice či trojice v trénovacích datech neznamená, že taková posloupnost tagů je špatně. Zbavujeme se takto možnosti porovnat např. takové dvě posloupnosti tagů, kde u první nebyla v trénovacích datech vidět žádná z dvojic (resp. trojic) a u druhé pouze jedna z dvojic (resp. trojic). Druhé z těchto posloupností bychom jistě chtěli dát pst vyšší než té první. Podle výše uvedených vzorců ale obě dostanou pst nulovou.

Nabízí se řešení (tzv. vyhlazování): v případě, že $p_3(t_i|t_{i-2}, t_{i-1}) = 0$, použít $p_2(t_i|t_{i-1})$. Pokud i $p_2(t_i|t_{i-1}) = 0$, použít $p_1(t_i)$, a pokud i $p_1(t_i) = 0$, použít uniformní pst. Přičemž ale psti s delším kontextem chceme asi přikládat větší váhu než psti s kontextem kratším. Proto nebudeme definovat vyhlazenou pst např. pro bigramy přímo takto:

$$pv_2(t_i|t_{i-1}) = (p_2(t_i|t_{i-1}) + p_1(t_i) + p_0)/3,$$

kde
pv_2 je definovaná vyhlazená podmíněná bigramová pst,
p_i pro $i = 1, i = 2$ jsou psti získané na trénovacích datech,
p_0 je uniformní pst tagu,
protože takto bychom sice díky dělení trojkou získali opět pstní rozložení, ale všem pstem p_i pro $i = 0, \ldots, 2$ bychom přikládali stejnou váhu. Proto se použijí vyhlazovací koeficienty.

Vzorec pro výpočet vyhlazené podmíněné bigramové psti s vyhlazovacími koeficienty vypadá pak takto:

$$pv_2(t_i|t_{i-1}) = \lambda_2 p_2(t_i|t_{i-1}) + \lambda_1 p_1(t_i) + \lambda_0 p_0, \tag{2.4}$$

kde navíc
λ_i pro $i = 0, \ldots, 2$ jsou reálná čísla (vyhlazovací koeficienty), pro která platí $0 \leq \lambda_i \leq 1$ a $\sum \lambda_i = 1$. (Proto již není nutno dělit třemi.)

Pro trigramy je vzorec analogický:

$$pv_3(t_i|t_{i-2}, t_{i-1}) = \lambda_3 p_3((t_i|t_{i-2}, t_{i-1}) + \lambda_2 p_2(t_i|t_{i-1}) + \lambda_1 p_1(t_i) + \lambda_0 p_0. \quad (2.5)$$

Získání vyhlazovacích koeficientů

Pro získání vyhlazovacích koeficientů použijeme tzv. held-out data (značme H) - nová trénovací data[6], která nejsou součástí původních trénovacích dat.

Na held-out datech H natrénujeme vyhlazovací koeficienty. Cílem je maximalizovat pravděpodobnost held-out dat přes λ_i za dodržení uvedených omezujících podmínek pro λ_i, tzn.:

$$p_n(H) = \prod_{i=n}^{|H|} pv_n(t_i|t_{kontext}) \quad (2.6)$$

maximalizuji přes λ_i za předpokladů $\lambda_i \geq 0$ a $\sum \lambda_i = 1$.
$n = 2$ nebo $n = 3$ podle toho, zda použiji bigramy nebo trigramy.
$t_{kontext}$ je t_{i-1} nebo t_{i-2}, t_{i-1} opět podle toho, zda použiji bigramy nebo trigramy.

Tuto maximalizaci je v praxi nutno řešit numericky, použitím např. EM-algoritmu, jak popíši dále v části 4.2.2.

Konkrétní postup (pro uživatele) k získání a použití vyhlazovacích koeficientů popisuji v části 3.7.

2.3.3 Zahazování n-gramů s malou četností

Velmi mnoho trigramů bylo v trénovacích datech vidět jen jednou[7]. Jsou to trojice, které se sice vyskytly, nicméně právě jen jednou. Otázkou je, jakou důležitost máme přikládat takové informaci. V praxi může často jít o chybu nebo o málo používanou frázi nebo o specifikum těch konkrétních trénovacích dat, což nám může při použití na testovacích datech spíše uškodit než pomoci.

[6]běžně 1/10 rozsahu původních trénovacích dat, abychom si zbytečně nezmenšovali trénovací data, kterých bývá nedostatek

[7]Tzv. Zipfův zákon říká, že polovičku „věcí" (tj. v našem případě trigramů) uvidím v trénovacích datech jednou, čtvrtinu dvakrát, osminu třikrát atd.. (Míru platnosti tohoto „zákona" možno ověřit na konkrétním příkladě, viz 5.6)

Tomuto jevu se říká přetrénování. Pstní rozdělení, získané z trénovacích dat, je pak sice podrobné, ale nespolehlivé.

Možným řešením této situace je zahození trigramů s malými četnostmi. Pak můžeme sledovat, jak se mění spolehlivost trigramů v závislosti na tom, které četnosti zahodíme. Zda jen trojice s četností 1, nebo trojice s četností menší než nějaké malé přirozené n.

O totéž se můžeme pokusit u bigramů, u unigramů by to asi nemělo valný smysl.

Konkrétní postup (pro uživatele) k zahazování n-tic s malou četností je popsán v části 3.8.

2.3.4 Viterbiho algoritmus

Vybrat z otagované věty na základě vzorce 2.2, resp. 2.3, nejlepší řetězec tagů naštěstí neznamená brát jeden řetězec tagů za druhým, každý ohodnotit (pomocí tohoto vzorce) a pamatovat si doposud nejlepší nalezený. To by totiž u delších vět bylo časově nezvládnutelné, neboť množství řetězců prudce roste v závislosti na délce věty.[8] Namísto toho se používá Viterbiho algoritmus, který nalezne vždy nejlepší řešení, aniž by bral jeden řetězec tagů po druhém.

Viterbiho algoritmus je popsán např. v [3]. O jeho implementaci píši v části 4.2.3.

Princip Viterbiho algoritmu je tento: Algoritmus využívá té skutečnosti, že u n-gramového modelu závisí pravděpodobnost tagu pouze na $(n-1)$ tagách předchozích.

Takže, procházíme větu slovo po slovu. Odhlédněme od okrajových případů začátku věty a mysleme si, že jsme u $k-$tého slova věty, kde $k > n$. Podstatou algoritmu je, že v tuto chvíli beru v úvahu (a kdesi v datech udržuji) nikoliv řetězce tvořené všemi možnými posloupnostmi tagů od začátku věty do místa $k-1$, ale pouze řetězce končící všemi možnými posloupnostmi tagů od pozice $k-n+1$ do pozice $k-1$ a začínající pouze již jednoznačně danými posloupnostmi tagů od pozice 1 do pozice $k-n$, dávajícími s těmi konci nejlepší pravděpodobnost.

Přibráním dalšího tagu na konec posloupnosti totiž nemohu již ovlivnit výběr počáteční posloupnosti tagů.

Viterbiho algoritmus, který dává ne jeden, ale m nejlepších řetězců, liší se

[8] Např. při průměrném počtu 4 tagů na slovo by pro větu o délce 20 slov bylo těchto řetězců 4^{20}, což je přibližně 10^{12}.

pouze v tom, že si pamatuje ne jednu, ale m nejlepších počátečních posloupností.

2.4 Hlavní program `disamb`

Hlavní program provádí disambiguaci podle těchto kroků:

1. Věta (ze vstupního nejednoznačně otagovaného SGML souboru) je poslána parseru, aby se ji pokusil zparsovat (nalézt jmenné fráze).

2. V případě, že toto se podaří, parser vrátí všechny možnosti zparsování nalezených jmenných frází, tj. parser poskytuje jeden řetězec tagů za druhým.
   Tyto řetězce jsou sjednoceny a ve větě jsou ponechány jen tagy z tohoto sjednocení.

3. Dále je použit Viterbiho algoritmus na nalezení určitého počtu nejlepších řetězců tagů (dle nastavených parametrů) z potenciální množiny všech řetězců tagů dané věty, s použitím n-gramového modelu.

4. Tyto nejlepší řetězce tagů jsou sjednoceny a věta je zapsána na výstup se slovy otagovanými právě tagy z tohoto sjednocení.

Pomocí parametrů je možno nastavit, zda bude použit parser a zda bude použit n-gramový model. Je tedy možno zkoumat výsledky použití jen parseru, jen n-gramového modelu nebo kombinace obou.

2.5 Co tedy bylo uděláno

- Byla implementována disambiguace pomocí parsingu, za použití převzatého parseru a převzaté gramatiky.

- Byla implementována disambiguace pomocí n-gramového modelu, a to konkrétně bigramy a trigramy. Též samozřejmě učení n-gramů na trénovacích datech.

- Vyhlazování n-gramů EM-algoritmem, zahazování n-gramů malých četností.

- V případě použití n-gramů možno nastavit výsledný poměr (počet tagů / počet slov).

- Kombinace různých nastavení parametrů výše uvedených i dalších byly otestovány a zaznamenány do tabulek.

Co z toho je nového?

- Disambiguace pomocí parsingu jmenných frází.
- Zahazování n-gramů malých četností.
- Použití n-gramů s nastavitelným poměrem (počet tagů / počet slov).
- Velké množství tabulek s výsledky testů.

2.6 Výhled do budoucna

Možné pokračování

Jak se ukázalo, trigramy vítězí nad bigramy o to přesvědčivěji, čím více roste objem trénovacích dat. Při jejich podstatně větším objemu věřím, že trigramy by získaly nad bigramy ještě víc navrch. Také úspěšnost disambiguace se s nárůstem trénovacích dat stále zvyšuje. A zahazování n-gramů malých četností by bylo jistě zajímavější při podstatně větších trénovacích datech.

Tedy **první úkol do budoucna**: několikanásobně zvětšit objem trénovacích dat.

Během testování jsem se také potýkal s problémem, že ručně otagované soubory byly místy otagovány špatně. Těžko pak dosahovat nějakých absolutních úspěchů při porovnávání automatického otagování s tímto ručním. Na odstranění tohoto problému se na lingvistické katedře (ÚFAL, MFF UK) pracuje. Data, která jsem použil v této diplomové práci, byla pracovní verzí dat budoucích. Na druhou stranu, mluvíme nyní o nějakých setinách, maximálně desetinách procenta úspěšnosti, čili o nepříliš zajímavých číslech.

Jedním z největších problémů byla gramatika češtiny. Jelikož parser byl převzat z projektu Grammar Corrector, gramatika s ním dodaná tomu odpovídala. Propouštěla jen malé procento vět, zhruba asi jedno procento. To proto, že jejím úkolem bylo především nezparsovat gramaticky špatnou větu. Pro účely této diplomové práce by byla potřeba gramatika zaměřená jinak - zparsovat větu tolika způsoby, aby alespoň jeden byl správný. Nezahodit tedy žádný správný tag.

Samozřejmě, ideální by bylo zparsovat větu právě jedním správným způsobem, ale to je pro bezkontextovou gramatiku vyloučené, navíc v mnohých případech se jedná o otázku nikoliv jen syntaxe, ale též sémantiky, takže velký problém někdy i pro člověka.

Situaci s gramatikou jsem vyřešil tím, že jsem se rozhodl parsovat jen jmenné fráze, které byly v gramatice zpracovány docela pěkně. Bohužel jejich úspěšnost také nebyla stoprocentní a jakýkoliv můj pokus to změnit vedl k podivným výsledkům a hlavně k mnohonásobnému prodloužení doby práce parseru.

Zde vidím největší možnost budoucího zlepšení, a tedy **druhý úkol** (nejspíše pro lingvistu) **do budoucna**: napsat gramatiku, šitou na míru potřebám tagování. Tedy rychlou, propouštějící vše správné a co nejméně toho špatného, ne nutně parsující celé věty.

Jiným problémem byl zdroj trénovacích dat - noviny. Novinářský styl bývá velmi šroubovaný, sám jsem mnohdy musel opakovaně číst větu, abych správně odlišil podmět a předmět a nakonec pochopil, co věta vlastně říká. Navíc v novinových článcích často byly zprávy ze sportu, což znamená celé odstavce heslovitě zapsaných výsledků sportovních utkání. Zajímavé by bylo učit a testovat program na textech knižního stylu a obsahu, myslím, že i v tomto případě by se výsledky zlepšily. (Nebo by alespoň nebyly stejné a bylo by zajímavé podívat se, jaké by byly.)

Tedy **úkol č.3 do budoucna**: Až budou k dispozici taková lepší data, vyzkoušet program na nich.

Co zachovat

Program `disamb` si nečiní žádné ambice na praktické využití v budoucnu. Jeho účel je hlavně vědecký, experimentální. Kdybych jej programoval znovu, nyní, když už vím napřed o všech překážkách a problémech, které jsem předtím objevoval postupně během jeho vytváření, určitě bych spoustu věcí udělal jinak a zřejmě lépe. Neznamenalo by to zlepšení výsledků, ale zcela jistě podstatné zpřehlednění programu a jeho jednotlivých částí a možná i ještě nějaké zrychlení, i když na rychlost jsem kladl velký důraz.

Pro budoucí praktické použití myslím, že funkce pracující s tabulkou indexů tagů jsou celkem použitelné, i když k dokonalosti (alespoň v mých očích) mají také daleko.

Za nezahození ale určitě stojí implementace tabulek n-gramů a funkcí s těmito tabulkami pracujících. Tyto tabulky a funkce se mi jeví jako praktické, rychlé[9], spolehlivé a přehledně naprogramované.

[9] Oproti tabulkám stejného významu, které byly součástí projektu Grammar Corrector, jsou tyto minimálně stokrát rychlejší.

A samozřejmě hlavní výsledek této diplomové práce - výsledky testů, které jsem učinil a sepsal[10] - by se neměl zahodit, protože většina z těchto testů nebyla předtím ještě učiněna, či pokud ano, nebyla zaznamenána či zveřejněna.

[10] výsledky okolo dvou set spuštění programu `disamb` s různými parametry

Kapitola 3

Uživatelská dokumentace

3.1 Hardwarové požadavky

Pro vlastní program (kompletní instalaci) včetně dat (ovšem trénovací sgml soubory nepočítaje) a pro následný běh programu postačí jistě 10MB volného místa na disku.

Pro pohodlný běh programu (se současnými naučenými tabulkami) bohatě postačí 16MB operační paměti, ne ovšem výrazně méně, aby program nemusel stále swapovat. Toto číslo je ovšem velmi přibližné, při jistém nastavení parametrů[1] může program potřebovat paměť mnohem větší.

Program nemá žádný grafický výstup, postačí tedy textový režim.

3.2 Softwarové požadavky

- operační systém Linux nebo Unix
- překladač jazyka C
- program `make`
- jazyk Perl
- shell

Program byl vyvinut v operačním systému Linux RedHat 5.0, s jádrem 2.0.32, překládán překladačem `gcc` 2.7.2.3 pomocí programu GNU Make

[1]konkrétně při nastavení vysokého poměru (počet tagů/počet slov) v souboru `moje/Parametry.txt`

3.76.1, perlové skripty psané v Perlu 5.004_04, shellové skripty jsou psány pro Bash.

Co se týče verze Perlu, perlové skripty jsou velmi jednoduché a obsahují jen základní příkazy Perlu, tedy se domnívám, že budou fungovat v libovolné verzi Perlu, v nejhorším případě není žádný problém přepsat je do jiného jazyka, jsou opravdu triviální.[2]

Co se týče verze programu `make`, všechny soubory `Makefile` kromě těch v adresáři parser jsou opět velmi jednoduché a neměly by vyžadovat žádné novější vlastnosti příkazu `make`.

Rovněž na překladač jazyka C není kladen žádný zvláštní požadavek, programy (u adresáře parser bez záruky) jsou psány v normě Posix.

K operačnímu systému: program byl testován na Linuxu RedHat 5.0, na Linuxu RedHat 5.1 a dále na Unixu Solaris a opět se nedomnívám, že by měl zvláštní nároky na operační systém. Jediné, co potřebuje, jsou roury, což je důvod jeho problémové (nikoliv nemožné) případné přenesitelnosti na operační systémy nedisponující tímto nástrojem.
Problémy byly při kompilaci parseru (tedy převzatého programu) na Unixu Solaris. Mé vlastní programy šly i zde přeložit bez problémů.

K jádru: zůstaneme-li na Linuxu, postačí nám jistě i jádra řady 1.2.xx.

3.3 Instalace

Instalace krok za krokem:

1. Ujistěte se, že splňujete hardwarové požadavky popsané v části 3.1 a softwarové požadavky popsané v části 3.2.

2. Do vhodného adresáře rozbalte archiv `disamb.tgz` příkazem
   `gunzip disamb.tgz ; tar -xvf disamb.tar`
   nebo rovnou `tar -zxvf disamb.tgz`
   Program se rozbalí do podadresáře `disamb`, ve kterém vznikne adresářová struktura popsaná v části 4.1.

3. Chcete-li instalovat demonstrační trénovací (včetně heldout) soubory, rozbalte tamtéž a stejným způsobem archiv `datatren.tgz`.

4. Chcete-li instalovat demonstrační testovací soubory (doporučeno :-), rozbalte taktéž archiv `datatest.tgz`.

[2] Pro Perl jsem se rozhodl proto, že úloha by byla v jazyce C řešitelná zbytečně složitě a shellovský skript by byl příliš pomalý. A též proto, že jsem si chtěl Perl vyzkoušet.

5. Chcete-li instalovat naučené tabulky četností ngramů (opět doporučeno), rozbalte také archiv `datatabs.tgz`.

3.4 Hlavní program - program `disamb`

Hlavní program se nachází v adresáři `moje` a jmenuje se `disamb`.

Překládá se v tomto adresáři příkazem `make`. (Má-li být při práci použit parser, je nutno jej také přeložit, a to v adresáři `parser` rovněž příkazem `make`.) Schéma jeho spouštění:

```
disamb IN_SGML [IN_SGML_J OUT_TEST] ,
```

kde `IN_SGML` je vstupní SGML soubor určený pro disambiguaci a
kde na `stdout` jde disambiguovaný výstupní SGML soubor a
kde na `stderr` jdou chybové hlášky programu `disamb` a
kde do souboru `logs/parsererr.log`[3] jde chybový výstup parseru a
kde `IN_SGML_J` je vstupní SGML soubor totožný s `IN_SGML`, ale již (ručně) víceméně zdisambiguovaný, v jeho přítomnosti program vypíše statistiky o úspěšnosti vlastní disambiguace do souboru `OUT_TEST`.

Jak právě popsáno, program `disamb` může běžet ve dvou režimech.

První režim - prostá disambiguace, program je tedy používán pro disambiguaci nejednoznačně otagovaného souboru, spouští se s jedním argumentem - jménem tohoto souboru. Disambiguovaný výstupní soubor jde na `stdout`.

Druhý režim - testovací, používaný pro testování činnosti programu. Na vstup jsou předloženy dva SGML soubory, jeden nejednoznačně otagovaný, určený k disambiguaci, druhý tentýž, ale zdisambiguovaný. Program provede disambiguaci prvního souboru a v porovnání s druhým souborem vypíše statistiky o úspěšnosti své činnosti do souboru daného třetím parametrem.

Činnost programu `disamb` je bohatě paremetrizovatelná, nastavení parametrů se provádí editací souboru `moje/Parametry.txt`, jak podrobněji popsáno v části 3.4.3.

Program je možno spouštět paralelně několikrát. Co se týče změn nastavení parametrů v právě výše uvedeném souboru, tak program čte tyto parametry jednorázově ihned po svém spuštění a poté již tento soubor nepotřebuje, není tedy třeba obávat se těchto problémů. Parser zapisuje chybové a stavové hlášky do souboru, součástí jehož názvu je systémový čas (s

[3] s připojeným časem k názvu, možno změnit v parametrech volání fce `VolejParser` v souboru `moje/disamb.c`

přesností na sekundy) pouštění parseru, takže jediná kolize, která by mohla v tomto směru nastat, je, kdyby byl program `disamb` spuštěn několikrát v jedné sekundě.

Ještě poznámka: Není-li v případě testovacího režimu předložený ručně otagovaný soubor úplně disambiguován, pak slova s více tagy jsou ignorována, myšleno v závěrečných statistikách.

3.4.1 Vstup programu `disamb`

Pro přehlednost to shrňme a pro úplnost doplňme. Program `disamb` požaduje 1 nebo 3 parametry na příkazové řádce, podle toho nepoběží nebo poběží v testovacím módu. Jsou to tyto parametry:

- vstupní nejednoznačně otagovaný SGML soubor, určený k automatické disambiguaci
- vstupní jednoznačně otagovaný SGML soubor, určený k testování úspěšnosti disambiguace
- cesta k výstupnímu souboru pro zápis informací o úspěšnosti disambiguace

Dále je vstupem soubor `moje/Parametry.txt`, ve kterém je možno nastavit činnost programu `disamb`. (Soubor je popsán v příloze 5.2.7.)

Dále program ke své činnosti potřebuje obsah adresáře `parser`, pokud je volání parseru nastaveno. V tom případě potřebuje také tabulku indexů tagů `data/nove.tagy`.

A nakonec, je-li nastaveno použití n-gramů (Viterbiho algoritmu), potřebuje program tabulky četností n-gramů a některé další tabulky, tedy:

- `data/ngramy/*.tab` ... tabulky četností n-gramů
- `data/ngramy/konst.txt` ... sumy četností n-gramů potřebné pro výpočet pravděpodobnosti z četnosti
- a opět `data/nove.tagy` ... abecedně seřazeny všechny možné tagy - tabulka indexů tagů

3.4.2 Výstup programu `disamb`

Pro přehlednost to také shrňme: program `disamb` má 2 nebo 3 nebo 4 výstupy, podle toho, zda není nebo je použit parser a zda neběží nebo běží v testovacím módu, tzn., zda mu nebyl nebo byl předložen testovací jednoznačně otagovaný SGML soubor. Ve všech případech má program `disamb` tyto první dva výstupy, navíc třetí v případě použití parseru:

- `stdout` - výstupní zdisambiguovaný SGML soubor
- `stderr` - chybové hlášky, informace o průběhu disambiguace, případné ladící informace
- `logs/parsererr.log`[4] - chybový a stavový výstup parseru

V případě testovacího módu ještě čtvrtý výstup - daný třetím parametrem volání programu `disamb` - soubor s informacemi o úspěšnosti tagování. Jeho formát je podrobně popsán v příloze 5.2.8.

Spouštěcí skripty

Pro účely testování programu `disamb` na připravených testovacích souborech není nutno vypisovat všechny parametry na příkazovou řádku. V adresáři `moje` jsou připraveny tři skripty (pro tři testovací soubory, popsané v 5.1.1), kde jsou již parametry nastaveny. Tyto skripty jsou `testlb`, `testvb` a `testm2`.

Spouštěcí skripty berou testovací soubory z adresáře `data/sgmlsoubory/testovaci`. Standardní a chybový výstup jsou přesměrovány do souborů `log1[lb|vb|m2]` a `log2[lb|vb|m2]`, soubor statistik o úspěšnosti tagování je pojmenován `logtest[lb|vb|m2]`, všechny tyto výstupní soubory jsou v adresáři `logs`.

Spustíme-li spouštěcí skript s parametrem, je tento připojen k názvům právě popsaných výstupních souborů.

3.4.3 Soubor `Parametry.txt`

V souboru `Parametry.txt` se nacházejí parametry, které program `disamb` čte po svém spuštění.

Program čte určitý pevně daný počet řádek souvislého počátečního úseku souboru, ostatní řádky jsou ignorovány a mohou tedy sloužit pro poznámky či zálohy parametrů.

Soubor je možno editovat a parametry měnit. Nesmí se měnit pořadí jednotlivých řádek, řádky musejí začínat parametrem, následovaným alespoň jednou mezerou a případně komentářem, řádek nesmí být delší než 80 znaků. Mezi řádky nelze vkládat prázdné řádky.

[4] nastavitelné v hlavičce volání fce `VolejParser` v programu `moje/disamb.c`, k názvu je připojen i aktuální čas

Příklad obsahu souboru `Parametry.txt` s vysvětlením méně jasných parametrů naleznete v příloze 5.2.7.

3.5 Příprava trénovacích a testovacích dat

Jak již řečeno, program `disamb` požaduje vstupní data a poskytuje výstupní data ve formátu SGML (viz 4.3.1). Převod běžného textu do SGML formátu (a jeho opatření lemmaty a tagy) provádí program `ma`, který není součástí této diplomové práce.

Ovšem součástí této diplomové práce jsou drobné filtry, které výstup programu `ma` ještě trochu upravují. Podrobně o tom píši v části 4.2.1.

3.6 Učení tabulek četností n-gramů

V adresáři `podpurne/uceni/ngramy` se nachází soubor `ucsengramy.c`. V něm jsou nastaveny některé konstanty, které je možno před kompilací změnit. Jedná se o tyto definice:

```
#define TTABFILE "../../../data/ngramy/tindex.tab"  /* jmeno
                              souboru s tabulkou indexu tagu */
#define TABTRIGRAMY "../../../data/ngramy/Trigramy.tab"
                            /* soubory pro ukladani cetnosti */
#define TABBIGRAMY "../../../data/ngramy/Bigramy.tab"
#define TABUNIGRAMY "../../../data/ngramy/Unigramy.tab"
#define KONST "../../../data/ngramy/konst.txt" /* soubor
                              s konstantami pro vypocet psti */
```

`TTABFILE` je cesta k tabulce indexů tagů (viz 4.3.3).

`TABTRIGRAMY` až `TABUNIGRAMY` jsou cesty k tabulkám četností trigramů až unigramů, do kterých se budou ukládat naučené četnosti.

`KONST` je cesta k souboru (viz 4.3.3) s konstantami pro výpočet pravděpodobnosti z četnosti, tedy k souboru, ve kterém jsou uvedeny sumy četností trigramů až unigramů.

Program pro učení četností n-gramů se překládá příkazem `make`. Spouští se pak takto:

`ucsengramy.x soub1 soub2 ... soubN`,
kde soub1, ..., soubN jsou cesty k učícím SGML souborům (viz 3.5).

Program bere své jednotlivé argumenty a na nich se učí četnosti n-gramů. Za každou naučenou větu vypíše tečku na `stdout`. Vypisuje též jméno nalezeného souboru, než se z něj začne učit.

Existují-li již soubory pro ukládání četností, četnosti z těchto nových souborů jsou přiučeny, tedy přičteny.

Existuje-li již soubor s konstantami pro výpočet pravděpodobností z četností, sumy nově naučených četností jsou přičteny. Jinak je vytvořen nový.

Není-li trénovací soubor zcela disambiguován (tedy existuje v něm slovo s více než jedním tagem), jsou naučeny všechny možnosti dané touto nejednoznačností.

Krok za krokem, jak naučit/přiučit četnosti n-gramů:

1. Vytvořte trénovací soubory, viz 3.5.
2. V adresáři `podpurne/uceni/ngramy` v souboru `ucsengramy.c` se přesvědčete, že cesty k výše uvedeným souborům jsou dobře nastaveny.[5]
3. Pokud učíte nové četnosti (tedy nepřiučujete), smažte případně existující soubor `KONST` a umístěte do nastavených cest prázdné soubory (soubory délky nula) s četnostmi n-gramů.[6]
4. Zkompilujte program příkazem `make`.
5. Spusťte program příkazem `ucsengramy.x soub1 ... soubN`, kde `soub1` až `soubN` jsou cesty k jednotlivým trénovacím souborům.

Hotovo.

3.7 Vyhlazovací koeficienty

V adresáři `podpurne/uceni/lambdy` je program `vyhlazuj.c`, který z tabulek četností n-gramů u trénovacích dat a z tabulek četností n-gramů u heldout dat za použití EM-algoritmu spočte vyhlazovací koeficienty pro bigramy i trigramy a zapíše je do určeného souboru. Během výpočtu vypisuje na standardní chybový výstup výsledky jednotlivých iterací.

Krok za krokem, jak spočítat vyhlazovací koeficienty:

[5] Defaultní nastavení by mělo být v pořádku.

[6] Při zachování defaultních jmen stačí soubory zkopírovat z adresáře `data/ngramy/prazdne`.

1. V programu `vyhlazuj.c` (v adresáři `podpurne/uceni/lambdy`) nastavte správné cesty k souborům četností n-gramů pro heldout data i pro trénovací data a nastavte epsilony - ukončující podmínky pro iterační algoritmus.

2. Zkompilujte program příkazem `make`.

3. Spusťte program příkazem `vyhlazuj`. Program na stderr vypisuje průběh iterací a nakonec do souboru `data/ngramy/lambdy.txt` vypíše spočtené vyhlazovací koeficienty.

4. Chcete-li, aby se spočtené lambdy použily pro disambiguaci, přepište je do souboru `moje/Parametry.txt`.

Hotovo.

3.8 Zahazování n-gramů s malou četností

V adresáři `podpurne/uceni/ngramytabfiltr` jsou tyto skripty, psané v Perlu:

`filtr2.pl` *... program pro zahazování bigramů*

`filtr3.pl` *... program pro zahazování trigramů*

`suma2.pl` *... program pro sečtení četností bigramů*

`suma3.pl` *... program pro sečtení četností trigramů*

Všechny tyto programy vyžadují na standardním vstupu tabulku Bigramy.tab, resp. Trigramy.tab.

Program filtr[23].pl na standardní výstup vypíše tabulku [23]-gramů s pouze těmi dvojicemi, resp. trojicemi, jejichž četnost je větší nebo rovna konstantě, uvedené přímo v programu (proměnná N). n-tice s menšími četnostmi se zapomínají.

V takto vzniklé nové tabulce n-gramů se ovšem změnila suma četností (potřebná pro výpočet relativních četností), která je nyní nekonzistentní s číslem uvedeným v souboru `data/ngramy/konst.txt`.

Také lambdy, vyhlazovací koeficienty, je potřeba po zahození n-gramů s malými četnostmi znovu spočítat.

Postup krok za krokem je následující:
Chceme např. z tabulky trigramů vyházet trojice s četností menší než 3.

1. Ve skriptu `podpurne/uceni/ngramytabfiltr/filtr3.pl` inicializujeme proměnnou N na 3.

2. Přejmenujeme soubor data/ngramy/Trigramy.tab např. na Trigramy.tab.old.
3. Zadáme příkaz (v adresáři data/ngramy) ../../podpurne/uceni/ngramytabfiltr/filtr3.pl <Trigramy.tab.old >Trigramy.tab. Program vytvoří novou tabulku Trigramy.tab a na stderr vypíše novou sumu četností trigramů.
4. Touto novou sumou nahradíme dosavadní sumu trigramů v souboru data/ngramy/konst.txt (číslo na 3. řádku).
5. Spočteme nové vyhlazovací koeficienty - postupujte dle návodu v části 3.7

Hotovo.

Kapitola 4

Programátorská dokumentace

4.1 Adresářová struktura

Označme # kořenový adresář programu. Pak adresářová struktura vypadá takto:

`#/data/ngramy/*.tab` *... tabulky četností n-gramů*

`#/data/ngramy/konst.txt` *... počty naučených n-gramů*

`#/data/ngramy/tindex.tab` *... tabulka indexů tagů*

`#/data/ngramy/heldout/*` *... tab. četností n-gramů a jejich počty pro heldout data*

`#/data/ngramy/prazdne/*.tab` *... prázdné tabulky četností n-gramů*

`#/data/sgmlsoubory/jednoznacne/` *... soubory pro učení n-gramů*

`#/data/sgmlsoubory/testovaci/` *... testovací soubory*

`#/data/b2800a.f2o` *... převodní tabulka staré tagy - nové tagy*

`#/data/nove.tagy` *... tabulka indexů tagů*

`#/docs/` *... dokumentace v latexu (tento dokument) apod.*

`#/logs/*/` *... zálohy různých výstupů z programu*

`#/logs/log1` *... standardní výstup programu (disambiguovaný soubor)*

`#/logs/log2` *... chybový výstup programu*

`#/logs/logtest` *... testovací výstup programu (informace o úspěšnosti)*

`#/logs/parsererr.log` *... chybový výstup parseru*

`#/moje/` *... hlavní adresář programu*

`#/moje/*.h` *... hlavičkové soubory jednotlivých komponent programu*

`#/moje/*.c` *... zdrojové soubory jednotlivých komponent programu*

`#/moje/test[lb|vb|m2]` *... spouštěcí skripty*

`#/moje/Parametry.txt` *... parametry pro běh programu*

`#/parser/` *... převzato z projektu Grammar Corrector*

`#/podpurne/konverse/` *... konverzní programy pro úpravu sgml souborů*

`#/podpurne/konverse/src/` *... zdrojové texty těchto konverzních programů*

`#/podpurne/uceni/tagytab/` *... interface tabulky indexů tagů*

`#/podpurne/uceni/ngramytab/` *... interface tabulky četností n-gramů*

`#/podpurne/uceni/ngramytabfiltr/` *... zahazování n-gramů malých četností*

`#/podpurne/uceni/ngramy/` *... zdrojové texty programu pro učení n-gramů*

`#/podpurne/uceni/ngramy/ucsengramy.x` *... program pro učení n-gramů*

`#/podpurne/uceni/lambdy` *... zdrojové texty programu pro výpočet vyhl. koeficientů*

`#/podpurne/uceni/lambdy/vyhlazuj` *... program pro výpočet vyhl. koeficientů*

4.2 Programy

4.2.1 Příprava datových SGML souborů

Výstup programu `ma` (viz příloha 5.2.2) není ještě přímo vstupem programu `disamb`. Jeho výstup je upraven několika jednoduchými filtry (z adresáře `podpurne/konverse`).

Formát datových souborů potřebných pro učení četností n-gramů je jiný (liší se o použití jednoho filtru) než datových souborů pro vstup programu `disamb`.

Oba formáty používají poziční třináctiznakové tagy (viz příloha 5.2.1). Prvním filtrem, který je aplikován na výstup programu `ma`, jsou tedy staré tagy převedeny na tagy nové. Tento převod je vzájemně jednoznačný, jednomu starému tagu odpovídá právě jeden tag nový.

Filtr pro tuto transformaci se jmenuje `stare-nove`. Na `stdin` očekává SGML soubor se starými tagy, na `stdout` vypíše tento soubor s novými tagy. Pro svou práci používá převodní tabulku `./b2800a.f2o`.

Použitím tohoto filtru jsou soubory ve formátu pro vstup programu `disamb` (viz příloha 5.2.3). Pro získání souborů pro trénování četností n-gramů je potřeba použít ještě jednoho filtru, který upraví řádky s delimitery (např. čárka, tečka, otazník) tak, aby se delimitery „tvářily" jako běžná slova (viz příloha 5.2.5).

Tento filtr se jmenuje `d2WWW` a opět převádí soubor ze `stdin` do souboru na `stdout`.

Několik poznámek k upřesnění formátu trénovacích a testovacích SGML souborů viz 4.3.1.

4.2.2 Trénování

Trénování četností n-gramů

Datové struktury a funkce, potřebné pro trénování n-gramů, jsou naprogramovány v souboru `ucsengramy.c` a v dalších pomocných souborech v adresáři `podpurne/uceni/ngramy`.

Pomocné funkce pro učení četností z trénovacích SGML souborů jsou zčásti převzaty z projektu Grammar Corrector.

Vstupní soubory jsou čteny větu po větě, každá věta se ukládá do struktury `Phrase`, a to voláním funkce `DejVetu`.

Získání četností z takto načtené věty a jejich zaznamenání do tabulek provádí funkce `NaucSeVetu`.

O implementaci tabulek pro ukládání četností n-gramů více v následující části.

Tabulky četností n-gramů

Datové struktury a funkce pro práci s tabulkami četností n-gramů jsou nadefinovány a popsány v souborech `ngramytab.h a ngramytab.c` v adresáři `podpurne/uceni/ngramytab`.

Zvlášť jsou funkce pro unigramy, bigramy a trigramy (a jiné nejsou).[1]

O těchto tabulkách se zmíním podrobněji, neboť se domnívám, jak jsem již jinde napsal, že stojí za to, zůstat zachovány a v budoucnu třeba v jiných programech být použity.

Tabulka unigramů je jednoduše realizována polem, kde index tagu odpovídá indexu pole a obsah příslušného pole znamená četnost tagu s tímto indexem. V tomto poli jsou tedy uvedeny i tagy s nulovou četností. Přístup je samozřejmě v konstantním čase.

Tabulka bigramů je složitější. Index prvního tagu z bigramové dvojice je opět indexem do pole, prvkem tohoto pole je však struktura, obsahující ukazatele na dvě pole. První z těchto dvou polí je vzestupně uspořádané pole indexů druhých tagů bigramů (začínajících oním prvním tagem) s nenulovou četností, druhé pole na odpovídajících indexech obsahuje četnosti těchto bigramů. Vyhledávání indexu prvního tagu je opět v konstantním čase, indexu druhého tagu v logaritmickém čase, neboť realizováno půlením intervalu. Zjištění četnosti je pak otázkou přímého přístupu do pole, čili opět konstantní čas. Tedy přístup k četnosti bigramu je v logaritmickém čase.

Tabulka trigramů je opět složitější, ale princip se nemění, jen je učiněn ještě jeden krok (shodný jako u bigramů k indexu druhého tagu) k indexu třetího tagu. Čas přístupu k četnosti trigramu je tedy opět v logaritmickém čase.

Četnosti jednotlivých n-gramů je možno zvyšovat po jedné (pro účely učení) nebo uložit do souborů a později najednou ze souborů přečíst (pro účely užívání).

O souborech uchovávajících tabulky četností n-gramů je pojednáno v části 4.3.2.

[1] Rozšíření na 4-gramy by znamenalo další programování, ovšem již velmi rutinní, podle vzoru rozšíření bigramů na trigramy.

Trénování lambd - vyhlazovacích koeficientů

Získání vyhlazovacích koeficientů analyticky je prakticky nemožné pro obtížnost tohoto úkolu. Proto je potřeba použít numerickou metodu. Algoritmus použitý zde je iterativní a nazývá se EM-algoritmus.

Následující tabulka algoritmus popisuje.

VSTUP:	pstní rozdělení (pro trigramy $p_0, \dots, p_3$) získaná z trénovacích dat pstní rozdělení $\tilde{p}$ získané na heldout datech epsilon - ukončovací podmínka
VÝSTUP:	vyhlazovací koeficienty (lambdy)
KROK1: KROK2: KROK3:	nastav počáteční hodnoty lambd (pro trigramy $\lambda_0 \dots \lambda_3$) v cyklu přes všechny prvky jazyka spočítej nové lambdy (viz dále) liší-li se některá z nových lambd od staré o více než epsilon, pokračuj krokem 2. Jinak konči.

KROK2 (pro trigramy):
Pro $i = 0, \dots, 3$ spočti

$$c_i = \sum_{t_1,t_2,t_3 \in T_H} \tilde{p}(t_1, t_2, t_3) \frac{\lambda_i p_i(t_3|t_1, t_2)}{\sum_{j=0}^{3} \lambda_j p_j(t_3|t_1, t_2)},$$

kde
T_H je množina trojic tagů, vyskytnuvších se za sebou v held-out datech,
$\tilde{p}$ je trigramová nepodmíněná pst (relativní četnost) na held-out datech
a kde např. $p_1(t_3|t_1, t_2)$ je $p_1(t_3)$, tzn. nadbytečný kontext se ignoruje a je uveden jen pro jednoduchost zápisu.

Nové lambdy pak pro $i = 0, \dots, 3$ spočti takto:

$$\lambda_i = \frac{c_i}{\sum_{j=0}^{3} c_j}$$

4.2.3 Tagování

Struktura uchovávající větu

Datové struktury a funkce pro načtení, uchování a zápis věty jsou implementovány v souborech `SGML.h` a `SGML.c` v adresáři `moje`.

Věta je uchovávána ve spojovém seznamu s prvky typu `RadekSGML`, na tento spojový seznam ukazuje globální proměnná `vetaS`.

Každý prvek tohoto seznamu reprezentuje jeden řádek vstupního sgml souboru, což většinou (a tyto případy jsou podstatné) znamená slovo s lemmaty a tagy.

Lemmata příslušející k jednomu slovu jsou uchovávána rovněž ve spojovém seznamu, s prvky typu `Lemma`. Rověž tak tagy příslušející k jednomu lemmatu, s prvky typu `Tag`.

Spojový seznam `vetaS` (a jeho podseznamy pro lemmata a tagy) jsou vytvořeny s načtením první věty vstupního sgml souboru a při dalších větách dochází k alokaci další paměti pouze v případě, že dosavadní vytvořený seznam je malý. (Takto je ostatně zacházeno s většinou spojových seznamů programu `disamb`, které se opakovaně plní různými daty, domnívám se, že tak dochází k úspoře času, který by jinak byl tráven neustálým alokováním a uvolňováním paměti.)

Struktura `Lemma` obsahuje integerový atribut `pouzito`, struktura Tag atribut `pouzit`. Tyto atributy jsou při načtění věty nastaveny na nulu. Během disambiguace, má-li lemma (resp. tag) být ponecháno (ponechán) ve výstupním souboru, je tento atribut nastaven na nenulovou hodnotu. (Je-li tato hodnota 1, znamená to, že lemma (tag) bylo (byl) zparsováno (zparsován), je-li tato hodnota 2, znamená to přítomnost v jednom z řetězců vybraných Viterbiho algoritmem.)

Při zápisu věty do výsledného souboru jsou zapsány jen lemmata a tagy s příslušně nastaveným atributem `pouzit[o]`.

Načtení věty ze souboru do struktury `vetaS` provádí funkce `CtiVetuSGML`, její zápis do souboru funkce `ZapisVetuSGML`.

Viterbiho algoritmus

Viterbiho algoritmus je implementován v souborech `Ngramy.c` a `Ngramy.h`, ale datové struktury, ve kterých jsou uchovávány řetězce, jsou definovány v souboru `SGML.h` (to vše v adresáři `moje`), neboť jsou součástí celé struktury, ve které je uchovávána věta.

Jedná se konkrétně o struktury `Viterbi` (pro bigramy) a `Viterbi3` a `ViterbiTri` (pro trigramy), které tvoří spojový seznam, reprezentující řetězce tagů, vznikající během průběhu Viterbiho algoritmu.

Funkce, pracující na těchto datových strukturách, jsou naprogramovány zvlášť pro bigramy a pro trigramy. Názvy funkcí pro bigramy začínají slovem `Viterbi` a neobsahují slovo `Trigramy`, názvy funkcí pro trigramy začínají rovněž slovem `Viterbi` a obsahují slovo `Trigramy`.

Klíčovými jsou funkce `ViterbiPstBigramy` a `ViterbiPstTrigramy`, které volají ostatní funkce, realizující nejprve vyhledání požadovaného počtu nejlepších řetězců a poté aplikaci těchto nalezených řetězců na lemmata a tagy hlavního spojového seznamu `vetaS`.

Volání parseru

Volání parseru je naprogramováno v souborech `VolejParser.c` a `VolejParser.h` v adresáři `moje`.

Parser je spuštěn jako nezávislý synovský proces a pro data přicházející od parseru je vytvořena roura.

Chybový výstup parseru je přesměrován do souboru. Jeho název je vytvořen před voláním parseru z funkce `main` (v souboru `disamb.c`) a jeho součástí je aktuální čas, aby mohl být program spuštěn vícekrát najednou a nedocházelo ke kolizi při zápisu parseru do chybového výstupu. Parser se volá funkcí `VolejParser`.

Zpracování výstupu parseru

Datové struktury a funkce potřebné pro zpracování výstupu parseru (příklad výstupu viz příloha 5.2.9) jsou naprogramovány v souborech `Retezce.c, Retezce.h, Ngramy.c, Ngramy.h`, vše opět v adresáři `moje`.

Jmenné fráze jsou čteny z roury od parseru a ukládány do spojového seznamu, na který ukazuje proměnná `retez` a jehož prvky jsou struktury `Retezce`. Každý tento prvek udržuje jednu frázi. Jedna fráze je opět spojový seznam, jehož prvky jsou struktury `Retezec`. Každý tento prvek obsahuje jedno lemma a jeden tag dané jmenné fráze.

Při načítání frází z roury jsou ignorovány fráze, které (samozřejmě v rámci jedné věty) již přišly dříve,[2] dále fráze, které jsou (co se týče lemmat, tagy jsou v tomto případě zanedbány) vlastním podřetězcem některé fráze, která přišla dříve. (Případ, že podřetězec přišel dříve než nadřetězec, je řešen později.)

Po načtení všech jmenných frází jedné věty jsou tyto po skupinkách příbuzných frází (tj. shodná lemmata, různé tagy) brány a aplikovány na větu (tedy jsou nastaveny atributy `pouzit[o]` u příslušných lemmat a tagů ve spojovém seznamu `vetaS`). Během sestavování skupinky příbuzných frází jsou zahazovány zbylé fráze, které jsou vlastními lemmatovými podřetězci jiných frází.

[2] což se stává velmi často, parser může jeden řetězec zparsovat více způsoby

Po aplikování všech skupin frází jsou lemmata a tagy slov nezasažených žádnou z frází (slov, která nebyla součástí žádné jmenné fráze) též přibrány k těm, které mají být ponechány ve výstupu. Parametrem nastavitelně je/není také tak naloženo s nezparsovanými lemmaty zparsovaných slov.

Načtení řetězců z roury od parseru a aplikaci těchto řetězců na `vetaS` provádí funkce `ZpracujRetezce`.

Funkce `main`

Hlavní funkce programu `disamb` se nachází v souboru `moje/disamb.c`.

Zde se nejprve načtou parametry ze souboru `Parametry.txt`, poté (dle nastavených parametrů) je alokována potřebná paměť, načteny tabulky n-gramů, zavolán parser, otevřeny ostatní potřebné soubory.

Následuje hlavní cyklus, kde je čtena věta po větě a zpracovávána.

Po hlavním cyklu následuje uvolnění naalokované paměti a nakonec případný výpis statistik o úspěšnosti tagování.

4.2.4 Ostatní programy

Práce s tabulkou indexů tagů

Tabulka indexů tagů (viz 4.3.3) je zpracována funkcemi ze souborů `tagytab.c` a `tagytab.h` v adresáři `podpurne/uceni/tagytab`. Zde jsou také popsány datové struktury, se kterými se pracuje.

Tabulka slouží k převodu tagu na index a zpět. Všechny tabulky četností n-gramů používají indexy tagů, nikoliv tagy.

Tabulka je (po startu programu `disamb`) nejprve načtena do paměti do pole. Jeden prvek pole odpovídá jednomu tagu, index v poli je též indexem tagu. Index 0 odpovídá neznámému tagu.

Vyhledávání v poli podle indexu je záležitost velmi jednoduchá, probíhá v konstantním čase. Vyhledávání podle tagu je realizováno půlením intervalu, tedy probíhá v čase logaritmickém.

4.3 Datové soubory

4.3.1 Upřesnění formátu SGML souborů

K upřesnění formátu sgml souborů:

Vstupní sgml soubory (určené k disambiguaci) nesmějí obsahovat prázdné řádky, poslední řádek musí končit znakem `newline`.

Dále (pokud má být použit parser) nesmějí obsahovat věty nulové délky, tj. věty, neobsahující řádky začínající <d> nebo <f>.

Parseru vadí i definece jazyka (v záhlaví sgml souboru) typu <csts lang=cz>, ale snáší <csts cz>.

Porušení sgml formátu a výše uvedených upřesnění má za následek buď legální (ovšem předčasné) skončení programu s výpisem chybové hlášky, nebo v horším případě spadnutí programu.

4.3.2 Soubory četností n-gramů

Soubory uchovávající tabulky četností n-gramů jsou tři, jejich implicitní umístění a pojmenování je:

- `data/ngramy/Unigramy.tab` ... tabulka četností unigramů
- `data/ngramy/Bigramy.tab` ... tabulka četností bigramů
- `data/ngramy/Trigramy.tab` ... tabulka četností trigramů

Formát těchto souborů je následující. Na každém řádku je jeden n-gram a jeho četnost. N-gram je reprezentován n indexy tagů (viz 4.3.3) v jejich přirozeném pořadí, za n-gramem následuje jeho četnost.[3] Výhodou tohoto formátu je jeho čitelnost a snadná přímá editovatelnost.

V souborech jsou zaznamenány jen ty n-gramy, které mají nenulovou četnost. Díky tomu jsou relativně malé.

O funkcích a datových strukturách pracujících s tabulkami četností je pojednáno v části 4.2.2.

4.3.3 Ostatní datové soubory

Soubor indexů tagů

Soubor indexů tagů `data/nove.tagy` je textový soubor, kde na každém řádku je jeden 13-timístný tag. Tagy jsou vzestupně seřazeny dle abecedy. Pořadí tagu v tomto souboru (počítáno od 1) je jeho indexem.

Implementace práce s tímto souborem je popsána v části 4.2.4.

[3] Tedy např. v souboru `Bigramy.txt` řádek `15 1447 1186` znamená, že dvojice tagů `15 a 1447` byla v trénovacích datech viděna 1186 krát, což je velmi mnoho. Nahlédneme-li na příslušné řádky souboru `data/nove.tagy`, zjistíme, že se jedná o dvojici tagů AAFP1—-1A– a NNFP1—–A–, tedy přídavné jméno následované podstatným jménem, oboje v ženském rodě, množném čísle a prvním pádě.

Tento soubor indexů tagů je vytvořen z převodního souboru `data/b2800a.f2o` vynecháním sloupce starých tagů, seřazením tagů a vynecháním případně se opakujících tagů.

Soubor sum četností n-gramů

Soubor sum četností n-gramů `data/ngramy/konst.txt` je textový soubor, vznikající automaticky při učení četností n-gramů, obsahující celkové počty n-gramů načtených při učení. Na prvním řádku je celkový počet unigramů, na druhém bigramů, na třetím trigramů. Tato čísla jsou potřeba při vypočítávání pravděpodobnosti z četnosti výskytu n-gramu.

4.4 Převzaté programy

Jak již několikrát zmíněno, ne všechny programy použité v této diplomové práci jsem programoval já. Některé byly převzaty z projektu Grammar Corrector, obhájeném loni na ÚFALu na MFF UK.

Jedná se o tyto programy:

- celý adresář `parser`
- program `stare-nove.c` z adresáře `podpurne/konverse/src/`.
- některé programy z adresáře `podpurne/uceni/ngramy`

Také mnohé datové soubory jsou převzaté. Jsou to:

- především všechny trénovací a testovací SGML soubory
- převodní tabulka `b2800a.f2o` v adresáři `data`.

Kapitola 5

Přílohy

5.1 Výsledky

5.1.1 Použité SGML soubory

Jako trénovací soubory jsem měl k dispozici (a použil jsem) novinové texty, poskytnuté mi katedrou ÚFAL, MFF UK. Jedná se konkrétně o texty Lidových novin (soubory `a2800l*`), Českomoravského Profitu (soubory `a2800c*`), Mladé Fronty (soubory `a2800m*`) a Vesmíru (soubory `a2800v*`).

Celkové množství dat pro učení četností n-gramů bylo 22,76 MB ve 20 SGML souborech, což představuje 34 305 vět, 592 088 slov a 598 274 tagů.[1]

Heldout data jsem použil v množství 28 (menších) souborů (celkem 2,31 MB), což bylo 3 146 vět, 57 144 slov a 59 004 tagů.

A konečně jako testovací data jsem použil tři soubory o celkové velikosti 0.95 MB[2], což bylo 1 423 vět a 11 314 slov.

Trénovací a testovací data přesněji

Data pro učení četností n-gramů sestávala z 9,00 MB textů Lidových novin, 7,09 MB textů Českomoravského Profitu, 2,41 MB textů Mladé Fronty a 4,26 MB textů Vesmíru.

Jednalo se přesně o tyto soubory: `a2800l1, a2800l2, a2800la, a2800lc, a2800ld, a2800lf, a2800lg, a2800lh, a2800c1, a2800c2, a2800ca,`

[1] Z rozdílu mezi počtem slov a tagů je vidět, že ani trénovací ručně disambiguované soubory nebyly otagovány zcela jednoznačně.

[2] Týká se ručně jednoznačně otagovaných dvojníků nejednoznačně otagovaných testovacích souborů.

a2800cb, a2800cc, a2800cd, a2800ce, a2800m1, a2800ma, a2800v1, a2800va, a2800vb.bez000.

Heldout data sestávala z 1,11 MB textů Lidových novin a 1,21 MB textů Mladé Fronty.

Jednalo se přesně o tyto soubory: a2800lb.00007, a2800lb.00010 až a2800lb.00016, a2800le.00000 až a2800le.00004, a2800m2.00000 až a2800m2.00015 bez a2800m2.00012.

Testovací data sestávala z těchto tří souborů:

soubor	velikost (v kB)	počet vět	počet slov	označme jej
a2800lb.00000_00006	512,73	754	12126	Testlb
a2800vb.00000	322,95	441	7658	Testvb
a2800m2.00012	115,93	228	2530	Testm2

(Velikosti souborů uváděné v tabulce jsou velikosti ručně jednoznačně otagovaných dvojníků těchto testovacích souborů.)

Soubor Testlb má počáteční poměr (počet tagů / počet slov) $= 3,723$, soubor Testvb má tento poměr $= 3,595$ a soubor Testm2 má tento poměr $= 3,405$.

S výjimkou testování závislosti úspěšnosti disambiguace na velikosti trénovacích dat, používal jsem tabulky naučené na všech trénovacích datech, jak popsáno výše.

S výjimkou testování závislosti úspěšnosti disambiguace na zahozených n-gramech malých četností a s výjimkou testování závislosti úspěšnosti disambiguace na velikosti trénovacích dat, používal jsem pro trigramový model tabulku trigramů se zahozenými trigramy četnosti 1 a tabulku bigramů se zahozenými bigramy četnosti 1, pro bigramový model tabulku bigramů bez zahazování bigramů malých četností.

Při testování závislosti úspěšnosti disambiguace na velikosti trénovacích dat jsem používal tabulky se všemi n-gramy, tedy bez zahazování n-gramů malých četností.

Tabulka 5.1: Parser s ponechanými nezparsovanými lemmaty

soubor	min. délka	recall	úplnost	precision	tagů teď/pův (v %)	+ trigramy
Testlb	1	98,02	1,837	0,534	49,3	91,27
	2	98,26	1,887	0,521	50,7	91,47
	3	98,36	2,353	0,418	63,2	91,36
	4	98,64	2,867	0,344	77,0	91,29
	5	99,00	3,151	0,314	84,6	91,54
Testvb	1	97,65	2,106	0,464	58,6	89,74
	2	97,94	2,170	0,451	60,3	89,96
	3	98,46	2,627	0,375	73,0	90,21
	4	98,84	3,069	0,322	85,3	90,34
	5	99,26	3,294	0,301	91,6	90,36
Testm2	1	98,30	1,875	0,524	55,1	91,23
	2	98,71	1,956	0,524	57,4	91,54
	3	99,11	2,529	0,392	74,3	91,54
	4	99,19	2,908	0,341	85,4	91,66
	5	99,51	3,089	0,322	90,7	91,66

5.1.2 Parser samotný a parser kombinovaný s trigramy

Jak bylo předpokládáno, parser samotný (bez použití n-gramů) nezajistí úplnou disambiguaci. V tabulkách 5.1 a 5.2 uvádím výsledky práce parseru samotného a v posledním sloupci práce parseru spojeného s trigramy nastavenými na úplnou disambiguaci (tj. jeden tag na slovo).

V tabulce 5.1 jsou výsledky pro případ ponechávání nezparsovaných lemmat u zparsovaných slov, v tabulce 5.2 jsou výsledky pro případ neponechávání nezparsovaných lemmat u zparsovaných slov.

Pro obě tabulky: první sloupec udává testovací soubor, druhý sloupec udává minimální délku uvažovaných zparsovaných řetězců, další tři sloupce jsou recall, úplnost a precision (viz 5.3), předposlední sloupec udává, kolik procent tagů zůstalo ve výstupním souboru oproti souboru vstupnímu a poslední sloupec udává úspěšnost disambiguace (recall) při doplňujícím použití trigramů nastavených na úplnost=1.

Při srovnání s dalšími tabulkami se ukazuje, že n-gramy (od jisté velikosti trénovacích dat) jsou úspěšnější než parser.

Tabulka 5.2: Parser se zahozenými nezparsovanými lemmaty

soubor	min. délka	recall	úplnost	precision	tagů teď/pův (v %)	+ trigramy
Testlb	1	96,90	1,735	0,559	46,6	90,45
	2	97,70	1,788	0,546	48,0	91,16
	3	97,90	2,281	0,429	61,3	91,10
	4	98,35	2,822	0,349	75,8	91,13
	5	98,82	3,120	0,317	83,8	91,44
Testvb	1	96,16	2,015	0,477	56,0	88,78
	2	97,35	2,083	0,467	57,9	89,87
	3	98,10	2,567	0,382	71,4	90,18
	4	98,63	3,032	0,325	84,3	90,31
	5	99,13	3,277	0,303	91,1	90,31
Testm2	1	97,17	1,792	0,542	52,6	90,55
	2	98,38	1,882	0,523	55,3	91,46
	3	98,99	2,486	0,398	73,0	91,50
	4	99,15	2,882	0,344	84,6	91,62
	5	99,51	3,072	0,324	90,2	91,66

Tabulka 5.3: Zahazování n-gramů - spolehlivost (recall) jednoznačného tagování (trigramy)

bigramy≥	trigramy≥	Testlb (v %)	Testvb (v %)	Testm2 (v %)
1	1	92,34	90,72	92,33
1	2	92,36	90,74	92,37
1	3	92,22	90,70	92,41
1	4	92,16	90,55	92,25
1	5	92,15	90,47	92,17
2	1	92,03	90,73	92,33
2	2	92,16	90,77	92,37
2	3	91,99	90,69	92,21
2	4	91,95	90,59	92,13
2	5	91,94	90,52	91,98
3	1	92,02	90,65	92,41
3	2	92,09	90,65	92,29
3	3	91,80	90,51	91,86
3	4	91,83	90,44	91,66
3	5	91,85	90,36	91,70

5.1.3 Zahazování n-gramů malých četností

Výsledky zahazování n-gramů s malou četností ukazují tabulky 5.3 a 5.4 (pro trigramový model) a tabulka 5.5 (pro bigramový model).

První tabulka ukazuje spolehlivost (recall) disambiguace pro různé kombinace zahazování n-gramů a pro různé testovací soubory.

Druhá tabulka ukazuje, jaké vyhlazovací koeficienty pro různé kombinace zahazování n-gramů spočítal EM-algoritmus.

Třetí tabulka ukazuje totéž co první dvě, ale pro bigramový model.

Ukazuje se, že při této velikosti trénovacích dat opravdu přesvědčivě působí pouze zahození trigramů s četností 1. Při podstatně větších trénovacích datech by to možná dopadlo jinak.

Nakonec (spíše pro zajímavost) uvádím tabulku 5.6 s údaji, jak se mění soubory četností n-gramů v závislosti na zahozených n-gramech.

Tabulka 5.4: Zahazování n-gramů - vyhlazovací koeficienty trigramového modelu

bigramy≥	trigramy≥	λ_3	λ_2	λ_1	λ_0
1	1	0,276	0,615	0,106	0,003
1	2	0,300	0,586	0,111	0,003
1	3	0,286	0,600	0,111	0,003
1	4	0,263	0,622	0,112	0,003
1	5	0,246	0,639	0,112	0,003
2	1	0,256	0,609	0,131	0,004
2	2	0,283	0,572	0,141	0,004
2	3	0,271	0,583	0,142	0,004
2	4	0,249	0,604	0,143	0,004
2	5	0,233	0,620	0,143	0,004
3	1	0,246	0,600	0,150	0,004
3	2	0,272	0,561	0,162	0,005
3	3	0,261	0,569	0,166	0,004
3	4	0,240	0,589	0,166	0,005
3	5	0,224	0,604	0,167	0,005

Tabulka 5.5: Zahazování n-gramů, recall & vyhlazovací koeficienty - bigramy

bigramy≥	Testlb	Testvb	Testm2	λ_2	λ_1	λ_0
1	91,51	89,02	90,43	0,891	0,107	0,002
2	91,18	88,95	90,40	0,858	0,138	0,004
3	91,11	88,89	90,40	0,832	0,163	0,005

Tabulka 5.6: Zahazování n-gramů - soubory četností

	počet různých n-gramů	$\sum$ četností	velikost souboru (v B)
trigramy≥ 1	178442	571777	2926705
trigramy≥ 2	57532	450867	959353
trigramy≥ 3	32637	401077	548764
trigramy≥ 4	22547	370807	382547
trigramy≥ 5	17294	349795	295378
bigramy≥ 1	34764	574945	407752
bigramy≥ 2	19021	559202	226778
bigramy≥ 3	13608	548376	164363

5.1.4 Závislost spolehlivosti disambiguace na její úplnosti

Výsledky jednoho z nejzajímavějších experimentů uvádím v tabulkách 5.7 a 5.8, a to závislost spolehlivosti disambiguace (recall) na dosažené míře úplnosti disambiguace.

V tabulkách (první je pro trigramy, druhá pro bigramy) uvádím v prvním sloupci testovací data, v druhém sloupci nastavený parametr ,,min. pomer (pocet tagu / pocet slov)" ze souboru `Parametry.txt`, v dalším sloupci pak skutečný dosažený tento poměr, ve čtvrtém sloupci výslednou spolehlivost disambiguace při tomto dosaženém poměru, v předposledním sloupci precision a v posledním sloupci procentuální podíl ponechaných tagů ve výstupu oproti tagům na vstupu.

Nastavení parametru (druhý sloupec) na vyšší hodnoty než 1,3 by znamenalo další extrémní zvýšení časových a prostorových nároků Viterbiho algoritmu. (Při všech nastaveních byl současně omezen maximální počet vybíraných řetězců Viterbiho algoritmem na 150.)

Tabulka 5.7: Trigramy: závislost spolehlivosti disambiguace (recall) na úplnosti disambiguace

soubor	nastavený parametr	úplnost	recall	precision	tagů teď/pův (v %)
Testlb	1,0	1,000	92,16	0,922	27,0
	1,05	1,060	93,97	0,887	28,6
	1,1	1,211	96,59	0,798	32,6
	1,15	1,408	97,98	0,696	38,0
	1,2	1,622	98,55	0,608	43,7
	1,25	1,782	98,83	0,555	48,0
	1,3	1,895	99,08	0,523	51,1
Testvb	1,0	1,000	90,77	0,901	27,9
	1,05	1,071	92,78	0,866	29,9
	1,1	1,247	95,87	0,769	34,8
	1,15	1,454	97,48	0,670	40,6
	1,2	1,644	98,11	0,597	45,9
	1,25	1,785	98,52	0,552	49,9
	1,3	1,893	98,73	0,522	52,9
Testm2	1,0	1,000	92,37	0,924	29,7
	1,05	1,068	94,15	0,882	31,8
	1,1	1,214	96,40	0,794	36,1
	1,15	1,386	97,79	0,706	41,2
	1,2	1,569	98,50	0,628	46,7
	1,25	1,719	98,77	0,575	51,1
	1,3	1,836	99,29	0,541	54,6

Tabulka 5.8: Bigramy: závislost spolehlivosti disambiguace (recall) na úplnosti disambiguace

soubor	nastavený parametr	úplnost	recall	precision	tagů teď/pův (v %)
Testlb	1,0	1,000	91,51	0,915	27,0
	1,05	1,059	93,32	0,881	28,5
	1,1	1,205	96,24	0,799	32,5
	1,15	1,395	97,6	0,700	37,6
	1,2	1,609	98,22	0,610	43,4
	1,25	1,767	98,60	0,558	47,6
	1,3	1,879	98,89	0,526	50,7
Testvb	1,0	1,000	89,02	0,890	27,9
	1,05	1,071	91,36	0,853	29,9
	1,1	1,238	94,78	0,766	34,6
	1,15	1,431	96,89	0,677	40,0
	1,2	1,600	97,64	0,610	44,67
	1,25	1,736	98,02	0,565	48,5
	1,3	1,838	98,35	0,535	51,3
Testm2	1,0	1,000	90,43	0,904	29,7
	1,05	1,065	92,33	0,867	31,7
	1,1	1,205	95,26	0,791	35,8
	1,15	1,376	96,44	0,701	40,9
	1,2	1,550	97,35	0,628	46,1
	1,25	1,703	97,98	0,575	50,7
	1,3	1,813	98,89	0,545	53,9

Tabulka 5.9: Použité množiny trénovacích dat

označení	velikost (v MB)	počet vět	počet slov	počet tagů
Train20	22,76	34 305	592 088	598 274
Train18	20,18	30 094	522 298	528 332
Train16	17,81	26 306	459 112	464 864
Train13	14,97	21 976	383 166	388 301
Train10	12,17	17 625	310 502	313 247
Train8	9,25	13 611	238 164	239 433
Train5	6,06	8 956	157 073	158 112
Train3	3,84	5 658	100 396	100 853

5.1.5 Závislost spolehlivosti disambiguace na velikosti trénovacích dat

Zajímavé jsou též výsledky uváděné v tabulkách 5.11 a 5.12, kde uvádím závislost spolehlivosti (recall) úplné disambiguace na velikosti trénovacích dat (první tabulka) a vyhlazovací koeficienty poskytnuté EM-algoritmem pro různě velká trénovací data (druhá tabulka).

V obou tabulkách jsou údaje pro bigramový i trigramový model, názvy sloupců s údaji pro trigramový model začínají číslicí 3, názvy sloupců pro bigramový model začínají číslicí 2.

O jaká trénovací data se jedná, popisuji v tabulce 5.9, informace o na těchto datech naučených četnostech uvádím v tabulce 5.10. (K tabulce četností: sloupce 2 až 4 udávají množství různých n-gramů naučených na trénovacích datech.)

Překvapilo mne, že i při velkém zmenšení velikosti trénovacích dat poskytují trigramy i bigramy stále velmi pěkné výsledky.

Dále je příjemné, že spolehlivost trigramů s nárůstem velikosti trénovacích dat stále roste, těším se na některé budoucí výsledky s trénovacími daty třeba dvakrát nebo desetkrát většími.

Tabulka 5.10: Tabulky četností n-gramů v závislosti na množině trénovacích dat

tr. data	unigramů	bigramů	trigramů	$\sum$ unigramů	$\sum$ bigramů	$\sum$ trigramů
Train20	1 273	34 764	178 442	598 274	574 945	571 777
Train18	1 234	33 119	167 555	528 332	509 077	509 971
Train16	1 224	32 063	160 307	464 864	449 138	453 532
Train13	1 182	29 743	145 226	388 301	376 302	384 393
Train10	1 125	24 023	100 031	313 247	298 751	284 696
Train8	1 068	20 578	82 193	239 433	227 067	215 391
Train5	1 019	16 893	61 533	158 112	150 184	142 902
Train3	956	13 324	44 153	100 853	95 620	91 071

Tabulka 5.11: Závislost spolehlivosti disambiguace na množině trénovacích dat

tr. data	3 Testlb	3 Testvb	3 Testm2	2 Testlb	2 Testvb	2 Testm2
Train20	92,34	90,72	92,33	91,51	89,02	90,43
Train18	92,20	90,68	92,25	91,51	88,95	90,43
Train16	92,09	90,60	92,41	91,55	88,85	90,71
Train13	91,93	90,28	92,02	91,42	88,76	90,40
Train10	91,70	90,32	92,06	91,17	88,78	90,67
Train8	91,68	89,88	91,94	91,07	88,29	90,67
Train5	91,32	89,61	90,32	90,87	88,34	89,76
Train3	90,63	89,15	90,08	90,33	88,16	89,76

Tabulka 5.12: Závislost spočtených vyhlazovacích koeficientů na množině trénovacích dat

tr. data	$3\,\lambda_3$	$3\,\lambda_2$	$3\,\lambda_1$	$3\,\lambda_0$	$2\,\lambda_2$	$2\,\lambda_1$	$2\,\lambda_0$
Train20	0,276	0,615	0,107	0,002	0,891	0,107	0,002
Train18	0,264	0,624	0,109	0,003	0,888	0,110	0,002
Train16	0,255	0,631	0,111	0,003	0,886	0,111	0,003
Train13	0,232	0,644	0,121	0,003	0,876	0,121	0,003
Train10	0,243	0,610	0,143	0,004	0,854	0,142	0,004
Train8	0,218	0,616	0,161	0,005	0,835	0,160	0,005
Train5	0,173	0,632	0,189	0,006	0,807	0,188	0,005
Train3	0,134	0,631	0,227	0,008	0,766	0,226	0,008

5.1.6 Srovnání s předchozími pracemi ([1] a [5])

Srovnání s pracemi [1] a [5] je problematické, neboť v obou těchto pracech byla uvažována jiná množina tagů (více podrobná), takže jejich výsledky musejí být z tohoto důvodu horší.

Pozitivním výsledkem, potvrzujícím tuto teorii, je, že skutečně při redukované množině tagů vycházejí výsledky lepší.

Jisté zlepšení zřejmě přináší i použití výstupu programu `ma` jako vstupu programu `disamb` oproti uvažování pravděpodobnosti výskytu tagu podmíněné výskytem slova. (Což je výhodou zřejmě ale jen u relativně malých trénovacích dat (za relativně malá považuji i data, která jsem používal já).)

Rozdílný výsledek oproti výsledkům práce [5] jsou přesvědčivě lepší procenta spolehlivosti u trigramů oproti bigramům.

5.2 Ostatní

5.2.1 Popis 13-ti místných tagů

(převzat z [2])

Tag je třináctice znaků, z nichž každá pozice zastupuje přesně vymezený morfologický jev. Pozice v tagu jsou následující:

1. slovní druh. N=podstatná jména, A=přídavná jména, P=zájmena, C=číslovky, V=slovesa, D=příslovce, R=předložky, J=spojky, T=částice, I=citoslovce.

2. slovní poddruh.
3. rod. M=mužský životný, I=mužský neživotný, F=ženský, N=střední, X=kterýkoliv, Y=M nebo I, H=F nebo N, Z=M nebo I nebo N.
4. číslo. S=jednotné, P=množné.
5. pád.
6. vnitřní rod zájmen.
7. vnitřní číslo zájmen.
8. osoba. (1,2,3,X=některá)
9. čas sloves. P=přítomný, F=budoucí čas, R=minulý, H=R nebo P, X=R nebo P nebo F.
10. stupeň přídavných jmen nebo příslovcí.
11. polarita slova. A=kladné, N=negace.
12. nedokumentováno
13. tvar. (1-10) - archaismy, nespisovné výrazy apod. jsou zde označeny.

Na všech pozicích může stát pomlčka s významem nevyplněné hodnoty.

5.2.2 Příklad části SGML souboru - výstupu programu `ma`

```
<s id="S/1994/J/Inf/x/x/lnd94103:001-p1s7">
<f cap>Předkladatel<l>předkladatel<t>NMS1A
<f>zákona<l>zákon<t>NIS2A
<f>poslanec<l>poslanec<t>NMS1A
<f cap>Viktor<l>Viktor<t>NMS1A
<f cap>Dobal<l>Dobal<t>NMS1A<l>dobalit<t>VMS2A
<f>hovoří<l>hovořit<t>VPS3A<t>VPP3A
<f>o<l>o<t>R4<t>R6
<f>tvrdém<l>tvrdý<t>ANS61A<t>AMS61A<t>AIS61A
<f>ultimátu<l>ultimátum<t>NNS6A<t>NNS3A
<D>
<d>.
```

Pozn.: Tento příklad je sice reálný, ale nepříliš ilustrativní. V češtině (v souborech, které jsem měl k dispozici) připadají průměrně 3 - 4 tagy na slovo ve výstupu programu ma*. Tento příklad jsem vybral proto, že je díky malému počtu tagů přehledný.*

5.2.3 Příklad části SGML souboru s pozičními 13-místnými tagy - vstupu programu disamb

```
<s id="S/1994/J/Inf/x/x/lnd94103:001-p1s7">
<f cap>Předkladatel<l>předkladatel<t>NNMS1-----A--
<f>zákona<l>zákon<t>NNIS2-----A--
<f>poslanec<l>poslanec<t>NNMS1-----A--
<f cap>Viktor<l>Viktor<t>NNMS1-----A--
<f cap>Dobal<l>Dobal<t>NNMS1-----A--<l>dobalit<t>Vi-S---2--A--
<f>hovoří<l>hovořit<t>VB-S---3P-AA-<t>VB-P---3P-AA-
<f>o<l>o<t>RR--4--------<t>RR--6--------
<f>tvrdém<l>tvrdý<t>AANS6----1A--<t>AAMS6----1A--<t>AAIS6----1A--
<f>ultimátu<l>ultimátum<t>NNNS6-----A--<t>NNNS3-----A--
<D>
<d>.
```

5.2.4 Příklad části disambiguovaného SGML souboru - výstupu programu disamb

```
<s id="S/1994/J/Inf/x/x/lnd94103:001-p1s7">
<f cap>Předkladatel<l>předkladatel<t>NNMS1-----A--
<f>zákona<l>zákon<t>NNIS2-----A--
<f>poslanec<l>poslanec<t>NNMS1-----A--
<f cap>Viktor<l>Viktor<t>NNMS1-----A--
<f cap>Dobal<l>Dobal<t>NNMS1-----A--
<f>hovoří<l>hovořit<t>VB-S---3P-AA-
<f>o<l>o<t>RR--6--------
<f>tvrdém<l>tvrdý<t>AANS6----1A--
<f>ultimátu<l>ultimátum<t>NNNS6-----A--
<D>
<d>.
```

5.2.5 Příklad části SGML souboru ve formátu pro trénování četností n-gramů

```
<s id="S/1994/J/Inf/x/x/lnd94103:001-p1s7">
<f cap>Předkladatel<l>předkladatel<t>NNMS1-----A--
<f>zákona<l>zákon<t>NNIS2-----A--
<f>poslanec<l>poslanec<t>NNMS1-----A--
<f cap>Viktor<l>Viktor<t>NNMS1-----A--
<f cap>Dobal<l>Dobal<t>NNMS1-----A--
<f>hovoří<l>hovořit<t>VB-S---3P-AA-
<f>o<l>o<t>RR--6--------
<f>tvrdém<l>tvrdý<t>AANS6----1A--
<f>ultimátu<l>ultimátum<t>NNNS6-----A--
<D>
<f>.<l>.<t>WWWWWWWWWWWWWW
```

5.2.6 Popis vstupního a výstupního formátu SGML

(převzat z [2])

Vstupní a výstupní soubory (obsahující otagovaný český text) programu `disamb` jsou ve formátu CSTS, speciální variantě SGML, užívané všeobecně na ÚFAL, MFF UK.

Oficiální popis CSTS formátu je k nahlédnutí v souboru docs/csts.dtd.

Pro rychlejší pochopení uvedu význam nejdůležitějších značek, které jsou v něm použity:

1.<csts cz> a </csts> musí být na začátku a konci dokumentu
2.<s id=„identifikator"> označuje začátek věty
3.<p> je začátek odstavce
4.<f> = forma, slovo v původním textu
5.<l> = základní tvar slova (lemma) - vztahuje se k nejbližšímu slovu vlevo
6.<t> množina morfologických značek (tag) - vztahuje se k nejbližšímu lemmatu vlevo

Ovšem domnívám se, že podrobnější zabývání se tímto formátem nepovede k užitku ohledně pochopení obsahu této diplomové práce.

Jen jedno důležité upozornění. Všechny vstupní SGML soubory programu `disamb` musejí mít poslední neprázdný řádek ukončen znakem `newline` a další tyto znaky nesmějí následovat.

5.2.7 Příklad souboru `Parametry.txt`

```
1 ... 0 znamena nepouzit parser, 1 znamena pouzit parser
0 ... pouzit Viterbiho? 0=ne, 1=ano (vzdy), 2=jen pri nezparsovani
1.0 ... min. pozadovany pomer (pocet tagu / pocet slov) u zparsovanych
vet
1.0 ... min. pozadovany pomer (pocet tagu / pocet slov) u nezparsovanych
vet
100 ... omezeny maximalni pocet vybiranych retezcu u zparsovanych vet
150 ... omezeny maximalni pocet vybiranych retezcu u nezparsovanych vet
0.0032060552 ... lambda2_0 (bigramy, koef. u uniformni psti)
0.1434342280 ... lambda2_1 (bigramy, koef. u unigramove psti)
0.8533597168 ... lambda2_2 (bigramy, koef. u bigramove psti)
0.0033555645 ... lambda3_0 (trigramy, koef. u uniformni psti)
0.1416048795 ... lambda3_1 (trigramy, koef. u unigramove psti)
0.5938233909 ... lambda3_2 (trigramy, koef. u bigramove psti)
0.2612161651 ... lambda3_3 (trigramy, koef. u trigramove psti)
3 ... ve Viterbiho algoritmu pouzit: 2=bigramy, 3=trigramy
2 ... minimalni delka podstatnych retezcu z parseru
0 ... 1=zahodit nezpars. lemmata u zpars. slov, 0=ponechat, i tagy

Odsud dal uz jen poznamky:
```

Významy jednotlivých parametrů jsou jasné z komentářů, přesto, co se týče lambd, tak například
`lambda2_0` je vyhlazovací koeficient u uniformní pravděpodobnosti u bigramů,
`lambda3_2` je vyhlazovací koeficient u bigramové pravděpodobnosti u trigramů.
A co se týče posledního parametru, ten říká, zda se u zparsovaných slov mají ponechat nebo zahodit nezparsovaná lemmata a jejich tagy.

K parametrům o požadovaném poměru (počet tagů / počet slov):
Rozumné je volit tento parametr v intervalu $< 1.0, 1.3 >$, přičemž horní hranici intervalu je možno volit i větší, ale pozor na zvýšené prostorové a časové nároky Viterbiho algoritmu! (Rozhodně je velmi vhodné omezit nastavením příslušného parametru maximální počet vybíraných řetězců Viterbiho algoritmem!)

S tímto parametrem je naloženo takto: Toto číslo je umocněno na délku věty (v počtech slov) a tolik nejlepších řetězců je pak vyhledáváno Viterbiho algoritmem. Pro velmi dlouhé věty je možno příslušnými dalšími parametry nastavit maximální počet těchto řetězců (tedy omezit jejich počet shora).

Výsledný poměr (počet tagů / počet slov) bývá většinou vyšší, než je nastavený parametr. (Výjimkou je nastavení na 1.0, kdy výsledný poměr je 1.0.) To je způsobeno tím, že dva řetězce tagů se od sebe mohou lišit (a většinou liší) o více než jeden tag.

5.2.8 Soubor s informacemi o úspěšnosti tagování

V tomto souboru jsou zapsány informace o nastaveních získaných ze souboru `Parametry.txt`, informace o otevřených souborech (soubor pro disambiguaci a testovací soubor) a dále informace o úspěšnosti tagování jednotlivých vět a slov.

Každé větě předchází její identifikace (ze vstupního SGML souboru), také identifikace odpovídající věty z testovacího SGML souboru (měly by být shodné).

Následuje informace, zda věta byla nebo nebyla zparsována. Tzn. zda od parseru přišel alespoň jeden řetězec tagů minimální požadované délky (dle nastavených parametrů).

Poté následují jednotlivá slova věty, vždy slovo z prvního souboru a vedle něj odpovídající slovo z testovacího souboru.

Pod nimi jsou tři čísla. První znamená počet tagů u daného slova před disambiguací, druhé znamená počet tagů u daného slova po disambiguaci, třetí (které může být pouze 0 nebo 1) říká, zda správný tag (tj. tag u daného slova v testovacím souboru) se nachází mezi tagy ponechanými u tohoto slova. Tedy zda disambiguace tohoto slova byla úspěšná či ne (1=úspěch, 0=neúspěch).

Na konci souboru s informacemi o úspěšnosti tagování jsou celkové statistiky o úspěšnosti tagování, tj. kolik bylo ve vstupním souboru vět, slov, tagů, kolik tagů bylo po disambiguaci ponecháno, procento úspěšnosti, procento úplnosti disambiguace, a to vše pro všechny věty dohromady a též zvlášť pro věty zparsované a nezparsované.

Pozor, v případě použití parseru a Viterbiho algoritmu najednou, průběžné i závěrečné informace o původním počtu tagů u slov a v souboru znamenají informace o počtu tagů po aplikaci parseru!

Příklad části souboru s informacemi o úspěšnosti tagování

```
Vety: S/1994/J/Inf/x/x/lnd94103:001-p1s7 S/1994/J/Inf/x/x/lnd94103:001-p1s7
```

```
parserret=2: nezparsovano

Předkladatel Předkladatel
1 1 1
zákona zákona
1 1 1
poslanec poslanec
1 1 1
Viktor Viktor
1 1 1
Dobal Dobal
2 1 1
hovoří hovoří
2 1 1
o o
2 1 1
tvrdém tvrdém
3 1 1
ultimátu ultimátu
2 1 1
```

5.2.9 Příklad jmenné fráze zparsované parserem

Příkladem formátu, ve kterém parser předává hlavnímu programu zparsované jmenné fráze, může být tato obzvláště dlouhá jmenná fráze:

```
RR--2-------- od
AAIP2----1A-- konkrétní
NNIP2-----A-- čin
RR--4-------- na
NNFS4-----A-- záchrana
AGIP2-----A-- vymírající
NNIP2-----A-- druh-1
RR--4-------- po
AAIS4----1A-- společenský
NNIS4-----A-- tlak
RR--4-------- na
NNNS4-----A-- navrácení
AANS2----1A-- územní
```

```
NNNS2-----A-- plánování
RR--2-------- do
NNIS2-----A-- resort
NNNS2-----A-- ministerstvo
AANS2----1A-- životní
NNNS2-----A-- prostředí
```

Přeložena zpět do srozumitelného zápisu: *od konkrétních činů na záchranu vymírajících druhů po společenský tlak na navrácení územního plánování do resortu Ministerstva životního prostředí.*

Uvádím ji zde ne proto, že by tento vnitřní formát byl nějak důležitý, ale spíše pro vyjímečnost zpracování této jmenné fráze parserem.

Je pěkné, že ji parser takto správně zparsoval (zasvěcení si mohou zkontrolovat), ale bohužel, parsování věty obsahující tuto jmennou frázi trvalo přibližně hodinu, tedy více, než zbylých 754 vět onoho testovacího souboru dohromady.

To je způsobeno tím, že tato jmnenná fráze obsahuje spoustu jmenných podfrází, které parser také zparsoval, a mnohé z nich jistě několikrát (použitím jiných pravidel gramatiky či v jiném pořadí).

A samozřejmě s tímto správným zparsováním parser poskytl nepřeberné množství špatných zparsování této jmenné fráze.

5.3 Slovníček pojmů

V této příloze se čtenář může najednou seznámit se základními pojmy, užívanými a porůznu vysvětlovanými v ostatních částech tohoto textu. Řazení není abecední, ale významové, čímž je myšleno, že pojmy předcházející jsou možná použity při vysvětlování pojmů následujících.

lemma: Základní tvar slova. Např. u slova ,,hradu" je lemma ,,hrad", u slova ,,běželi" je lemma ,,běžet".

tag: Morfologická značka. Obsahuje informace o slově a jeho tvaru, např. u slova ,,peckou" říká, že se jedná o podstatné jméno ženského rodu jednotného čísla a že je v sedmém pádě. Podrobný popis tagů užívaných v této diplomové práci je v příloze 5.2.1.

tagování: Opatření textu lemmaty a tagy, tj. každému slovu je přiřazeno lemma (či několik lemmat) a tag (či několik tagů).

disambiguace: Zjednoznačňování tagování, tj. ubírání lemmat a tagů u slova.

úplná disambiguace: Zjednoznačnění tagování, tedy ponechání právě jednoho lemmatu a jednoho tagu u slova.

SGML formát: Obecně užívaný formát textových souborů obsahujících řídící znaky. CSTS (varianta SGML) je formát vstupních a výstupních souborů obsahujících české otagované věty, používaný v této diplomové práci. Trochu blíže o tom v příloze 5.2.6.

n-gram, n-gramy: N-tice tagů následujících po sobě, užíváno též jako pojmenování metody využívající n-gramový model, viz 2.3.

řetězec tagů: Posloupnost tagů, reprezentující možné přiřazení tagů slovům věty, tedy jedno možné zjednoznačnění tagování.

úplnost disambiguace: Poměr (počet tagů / počet slov) v SGML souboru. Udává míru úplnosti disambiguace, hodnota 1 tedy znamená, že ke každému slovu je přiřazen právě jeden tag, hodnota vyšší než jedna znamená průměrné množství tagů na slovo.

recall: Poměr (počet slov s přítomným správným tagem (a lemmatem) / celkový počet slov). Tento poměr udávám vždy v procentech a znamená spolehlivost disambiguace.

precision: Poměr (počet slov s přítomným správným tagem (a lemmatem) / celkový počet tagů). Dá se spočítat z předchozích dvou.

Literatura

[1] Barbora Hladká: *Počítačové vybavení pro zpracování velkých českých textových korpusů,* (diplomová práce, Praha 1994)

[2] Pavel Květoň: *Dokumentace k projektu Grammar Corrector,* (analytická část (parsing češtiny), Praha 1997)

[3] Frederick Jelinek: *Statistical Methods for speech recognition*, kap.5, (MIT Press, Cambridge 1998)

[4] Frederick Jelinek: *Statistical Methods for speech recognition*, kap.4, (MIT Press, Cambridge 1998)

[5] Jan Hajič, Barbora Hladká: *Probabilistic and rule-based tagging of an inflective language - a comparison*, (ÚFAL, 1996)